MASTER

Modeling and computation of vernacular regions

Meulemans, W.

Award date:
2010

Link to publication
Master’s Thesis

Modeling and Computation of Vernacular Regions

by

Wouter Meulemans

Supervisor

prof. dr. M.T. de Berg

Eindhoven, August 18, 2010
A vernacular region is a region without a precise or administrative boundary. A formalization is needed to model these regions for automated usage. This allows for more intelligent treatment of search queries like “hotels in the British Midlands”. A common approach is to use the internet as a source of information, using search engines to relate exact geographical locations to vernacular regions. The question arises how to find the (approximate) boundary of a region, given a set of points that are likely to be inside or near the region. However, current models are not satisfactory. In this thesis, a new geometric model is described and algorithms are developed to compute this model. The model’s performance is evaluated in an experimental study.
Contents

Abstract i

1 Introduction 1

2 Problem analysis 3
  2.1 Properties of vernacular regions 3
  2.2 Existing models 5

3 A new model 9
  3.1 A new model 9
  3.2 Properties 11
  3.3 Algorithms 16
    3.3.1 Finding the core 16
    3.3.2 Finding a core without holes 17
    3.3.3 Calculating an egg-yolk model 23
    3.3.4 Filtering outliers and detecting holes 23

4 Extending the model for context 25
  4.1 Extending the model 25
  4.2 Properties 27
  4.3 Algorithms 29
    4.3.1 Computing distances 30
    4.3.2 Outline and offsetting 33

5 Evaluation 35
  5.1 Theoretical evaluation 35
  5.2 Practical evaluation 36
    5.2.1 Obtaining real data 36
    5.2.2 Kernel Density Estimation 37
    5.2.3 Results and comparison 38

6 Conclusions 47

Bibliography 51

A Experiments 53
Chapter 1

Introduction

In everyday language, people tend to use vernacular regions to describe a place of interest. Such a vernacular region is a region without a precise or administrative boundary. Other terms used for these regions are vague regions or imprecise regions. Examples of vernacular regions are “downtown Chicago” and “the British Midlands”. Whereas humans often can interpret such regions intuitively, this is not straightforward for a computer: a formalization is needed to model these regions such that for example search engines can give better results on queries like “hotels in the British Midlands”. A common approach is to use the internet as a source of information, using search engines to relate exact geographical locations to vernacular regions. The question arises how to find the (approximate) boundary of a region, given a set of points that are likely to be inside or near the region.

Previous work. There is a large collection of work available on vernacular regions. There are roughly four categories of research, though most of the work belongs to multiple.

The first category deals with user studies: how do people deal with imprecise regions and what do they expect? Montello et al. [32] perform a study asking people in Santa Barbara to delineate “downtown” on a map. One of their observations is that the “core” of the region is not necessarily the geometric center. The vagueness of “downtown” seems to be intuitive to most interviewed people when asked to delineate with 50% and 100% certainty. Recently, Davies et al. [9] also performed a user study, concluding that the best way to model an imprecise region depends on the type of region and the expectations of the user.

In the second category, researchers try to find a way to represent a vernacular region. Common approaches are fuzzy boundaries [20, 23, 25, 35], crisp boundaries [3, 36] or intermediate shapes such as the egg-yolk model [8, 15, 21, 37]. Each approach has its own advantages and disadvantages. Although a fuzzy set has the most expressive power, it is also more difficult to store than the polygonal shape of a crisp boundary and an egg-yolk model. The advantage of a crisp boundary is that, in presence of negative samples, the problem can be reduced to geometric separability, a well-studied problem in computational geometry [14, 38]. Unfortunately, this problem is NP-complete for multiple settings using a polygonal separator (refer to the work of Seara [38] for an overview).

The third category focuses on methods to obtain sample data from the web and how well these perform. Typically, data is obtained by grounding results from search engines to geographical locations [20, 25, 34, 35, 41]. This approach uses trigger phrases, such as “* in the British Midlands”, to find items that are likely to be in the intended vernacular region. The effectiveness of this approach is studied by Jones et al. [25] and Grothe and Schaab [20]. Data
obtained this way often contains numerous outliers, either by incorrect information on the web or by incorrect grounding. Purves et al. [35] identify this problem and also observe that the search engine used may have influence on the results: English documents about Scotland are likely to be located in the area of dense population and political influence, while French and German documents are typically tourist information and cover more area of Scotland.

The fourth category deals with finding the desired model, such as a fuzzy boundary or an egg-yolk model, for a given set of samples. A common approach applied in literature is Kernel Density Estimation (KDE) [20, 25, 35]. This method assigns a certainty to every point in the plane by calculating the influence of nearby samples on that point using a kernel function. The advantage of this approach is its simplicity. However, incorporating geographical context into the model is not straightforward. Moreover, it heavily relies on the parameterization of the kernel function. Grothe and Schaab [20] apply a Support Vector Machine (SVM) and compare this approach to the KDE method, concluding that an SVM performs better. Alani et al. [1] apply Voronoi diagrams to delineate the imprecise region, when the input also contains negative samples. Although they obtain good results, they rely heavily on a database containing information on vernacular regions. For some countries this data may be available, but it seems a severe restriction for worldwide application. The dual of a Voronoi diagram, a Delaunay triangulation, has also been successfully applied to delineate vague regions [3, 36]. However, this approach again relies on negative samples. Arampatzis et al. [3] and Reinbacher et al. [36] also apply a model based on \( \alpha \)-shapes (for details on these shapes, refer to the work of Edelsbrunner et al. [13]). An \( \alpha \)-shape is used to find an initial estimate of the region and this region is adapted based on the negative samples.

There is also a large collection of work available on shape recognition, a related area. Given a set of points, an approximate shape is calculated. However, often an assumption is made on the density of the input points. An example of this is the dot pattern as assumed by Chaudhuri et al. [6]. Such a pattern is a near-uniform distribution of points in the intended region or object to be found. It is important for methods that attempt to detect vernacular regions to be able to deal with strongly varying point density. Samples are often distributed by population density or tourist attractions: large cities and places that tourists often visit typically have more samples. Besides a density assumption, outliers are often also not considered. For example, the \( \chi \)-shapes presented by Duckham et al. [12] do not include outliers.

Our contributions. In Chapter 2, we analyze vernacular regions and existing models. We conclude that the current models are not satisfactory. In particular, there is room for improving models by including geographical context. In Chapter 3, we define a new model based on shortest paths and present ways of calculating the region. We present an \( O(n \log n) \) algorithm to find a region without holes based on \( n \) samples. The model is defined such that it allows for easy integration of geographical context and Chapter 4 deals with this extended model and corresponding algorithms. In Chapter 5, we evaluate the new model and present an experimental study based on real data obtained from geotagged photos on Flickr (www.flickr.com). We present our conclusions and ideas for future work in Chapter 6.
In this chapter, we analyze vernacular regions and existing models. This is done in sections 2.1 and 2.2 respectively. Based on this analysis, we conclude that there is room for improvement.

2.1 Properties of vernacular regions

To find a model for vernacular regions, we first look at what we are trying to model. How do we represent the vagueness of a region? Are there different types of vague regions? Can we describe their shape in an abstract way? What kind of geographical data could influence the vernacular region? These are the kind of questions we study in this section.

Fuzzy boundary. Cohn and Gotts [8] distinguish intrinsic vague regions and ignorance vague regions. The former are regions that have no exact boundary, and often there is some “degree of membership”. The latter are regions that have a precise but unknown boundary, and often one talks about a “certainty of membership”. We do not make this distinction, as it is mostly up to interpretation: in both cases, we expect a “fuzzy” boundary for the imprecise region. To capture this in a model, several methods have been proposed. One straightforward approach is to assign each point in the plane a value that indicates the degree or certainty of membership, directly modeling the fuzzy boundary. We can use a threshold value to generate a crisp (polygonal) boundary for the vernacular region. However, this often is too strict as it only classifies points as being either inside or outside of the intended region. By using more thresholds, one can regain some of the strength of the fuzzy boundary, while making it easier to store. Often, two thresholds are used, dividing the plane in three parts, “certainly inside”, “certainly outside” and “potentially inside”. This model is referred to as the egg-yolk model [8]. Figure 2.1 shows a simple example for each of these three boundary types.

![Figure 2.1: Three ways to model the boundary, starting with the most expressive one.](image-url)
Chapter 2. Problem analysis

Types of regions. We distinguish three types of regions: urban, national and international regions. The type depends on what is expected to contain the intended region. Urban and national regions are typically contained within a city or country respectively. Examples of these are “downtown Chicago”, “the London suburbs” and “the British Midlands”. One can expect these to be within the administrative boundaries of the exact region. The third type, international regions, are typically not contained within the borders of one country and represent natural regions, such as “the Alps” or “the Rocky Mountains”.

We also identified a fourth type of region, which we do not consider: property-based regions. Examples of this type are “places where the African Elephant lives” or “places with high average income”. These regions are not considered as their extent is often disconnected, whereas the other three types are usually connected.

Shapes of regions. What kind of shape do we expect a vernacular region to have? Arampatzis et al. [3] assume such regions to be connected and close to convex. Long and thin “tendrils” are unlikely to occur and typically indicate outliers. There are several ways to formalize this. Arampatzis et al. consider the perimeter length and the squared perimeter length divided by the area. Another possible measure is fatness [19], often used to specify how realistic an input polygon is.

The assumptions made by Arampatzis et al. [3] seem reasonable for the three types of regions we consider, although connectedness is not always assumed [15]. Whereas Arampatzis et al. do not allow holes in their regions, we do not want to exclude this beforehand. Schneider and Hagen [37] also allow for holes in imprecise regions. For example, the suburbs of a city can completely surround the city center. However, this center does not belong to the suburbs.

Geographical context. Some vernacular regions have a boundary that is (partially) determined by geographical data. For example, “southern Limburg” has a crisp boundary in the south, the border of the Netherlands, whereas the northern boundary is fuzzy in nature (see Figure 2.2). So, what kind of geographical context might be relevant for determining vernacular regions?

Administrative boundaries, such as country or province borders, could be relevant as mentioned in the example of “southern Limburg”. Another category of relevant context information is infrastructure, such as the main road network and the railway network. For example, in urban areas these could influence the region: the city center and the suburbs might be separated by a ring road, such as the Boulevard Périphérique in Paris. The last category of relevant information we wish to include are the rivers and shorelines. Vague regions often do not extend over seas or large lakes and hence the shorelines can be used to limit the extent near the coast. Also, rivers often cause fairly crisp boundaries. For example, the river Rhine is often considered to be the boundary of the “southern parts of the Netherlands”. All this data can be effectively represented as a collection of line segments, potentially subdividing the plane. It may result in partially crisp boundaries for a vague region. In some situations, this may be desirable as indicated by Davies et al. [9].

However, there is also context information that is less suitable for representation by line
2.2. Existing models

segments. One can think of terrain elevation. When searching for “the Alps”, one will find points mostly located in the mountains (having a high terrain elevation) and hence regions with low terrain elevation are unlikely to be part of “the Alps”. Also, the type of terrain could be included, for example urban area, agricultural area and natural area. When searching for “downtown Chicago”, one would mostly find points located in urban areas and hence points that are for example in agricultural area are likely to be outliers. However, this information assigns properties to regions in the plane, distinguishing them from the earlier information. For this reason, we will not include this in our model, leaving incorporation of these other types of context up to future work.

Input. Input is commonly obtained from the web, by querying search engines for the vernacular region in combination with trigger phrases. By grounding the found references to precise places, one obtains a number of sample points which should be in the intended vague region. However, due to incorrect information on the web or grounding errors, this data may have a lot of outliers and the model should take this into account. In some cases, each sample is assigned a weight or certainty, based on the search results (by using for example term or document frequency). Other approaches also include negative samples, points that should not be in the imprecise region. However, such negative points are often hard to obtain accurately and hence it is undesirable to assume such points in the input.

2.2 Existing models

In this section, we relate the properties from Section 2.1 to existing models. We distinguish input demands and output properties. Five models are compared: Kernel Density Estimation, Support Vector Machine, Voronoi Diagram, Delaunay Triangulation and \( \alpha \)-Shape.

Models. The Kernel Density Estimation model (KDE) \cite{20, 25, 35} assigns a certainty to each point in the plane by using a kernel function and radius. These two parameters combined determine the strength of influence for each sample in the input. Outliers are implicitly dealt with by parameterizing the kernel function and radius. The resulting boundary is fuzzy.

The Support Vector Machine model (SVM) \cite{20} uses the machine learning technique of support vectors to try and learn the model underlying the samples. This approach allows for classification of new points as being either inside or outside of the vernacular region and thus results in a crisp boundary.

The Voronoi Diagram model (VD) \cite{1} creates the Voronoi diagram on the input samples and classifies each cell as either inside or outside the region. The classification is based on a database containing the relevant information. For the resulting region to be bounded, negative samples are required.

The Delaunay Triangulation model (DT) \cite{3, 36} requires both positive and negative samples. The largest connected component of positive samples in the Delaunay triangulation is delineated by a simple polygon to find a crisp boundary for the region. To obtain a reasonable shape and filter outliers, some points can be reclassified.

With the \( \alpha \)-Shape model \cite{3, 36} an \( \alpha \)-shape is used to find an initial estimate of the region. Because an \( \alpha \)-shape may have multiple components, the largest is chosen as a delineation of the region. If there are negative samples inside the estimated region, the perimeter is changed such that these points lie outside. The shape is explicitly controlled by disallowing
perimeter changes that would raise the perimeter (or ratio between the perimeter and area) above a certain threshold. Negative samples that cannot be put outside the region this way are considered to be outliers.

**Input demands.** Table 2.1 gives an overview of the currently existing models and their input demands. We distinguish four different input demands:

- **Negative**: does the model use negative samples? Methods that do not require negative samples are preferable.
- **Outliers**: is the model able to deal with outliers? If it deals with outliers, is this done explicitly or is it inherently in the method (implicit). Since outliers typically are present in real data, these must be handled, either implicitly or explicitly.
- **Context**: does the model use geographical context? While a model should not rely on the context, it may help to find a better approximate region.
- **Parameters**: on how many parameters does the model depend? While parameters are hard to avoid completely, it is undesirable to have too many parameterizations.

<table>
<thead>
<tr>
<th>Model</th>
<th>Negative</th>
<th>Outliers</th>
<th>Context</th>
<th>Parameters</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDE</td>
<td>No</td>
<td>Implicit</td>
<td>No</td>
<td>2</td>
<td>[20, 25, 35]</td>
</tr>
<tr>
<td>SVM</td>
<td>No</td>
<td>Explicit</td>
<td>No</td>
<td>2</td>
<td>[20]</td>
</tr>
<tr>
<td>VD</td>
<td>Required</td>
<td>No</td>
<td>No</td>
<td>0</td>
<td>[1]</td>
</tr>
<tr>
<td>DT</td>
<td>Required</td>
<td>Explicit</td>
<td>No</td>
<td>1</td>
<td>[3, 36]</td>
</tr>
<tr>
<td>$\alpha$-Shape</td>
<td>Yes</td>
<td>Explicit</td>
<td>No</td>
<td>2</td>
<td>[3, 36]</td>
</tr>
<tr>
<td><em>desired</em></td>
<td>No</td>
<td>Implicit or explicit</td>
<td>Yes</td>
<td>Few</td>
<td></td>
</tr>
</tbody>
</table>

**Output properties.** Table 2.2 gives an overview of the currently existing models and their output properties. We distinguish four different output properties:

- **Boundary**: does the model result in a fuzzy or crisp boundary or an egg-yolk model? A crisp boundary is undesirable as it does not model the vagueness of the region’s boundary.
- **Connected**: does the model always result in a connected region? Typical vernacular regions are connected and this property could be used to find better boundaries.
- **Shape**: does the model take the shape of a region into account? Some models may not take shape into account allowing for arbitrary shapes. However, arbitrary shapes are unlikely to occur and models may attempt to obtain a reasonable shape. This can be done in multiple ways, depending on what is considered reasonable.
- **Holes**: does the model allow for holes in a region? A support for holes is desirable.

Note that the Delaunay Triangulation and the $\alpha$-Shape models could easily be adapted to allow for holes and disconnected regions. Instead of selecting the largest component of the triangulation or $\alpha$-shape, it is also possible to work with all components. However, in the corresponding papers, the authors assume that a vernacular region is simply-connected, justifying their approach.
2.2. Existing models

Table 2.2: An overview of the output properties of existing models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Boundary</th>
<th>Connected</th>
<th>Shape</th>
<th>Holes</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDE</td>
<td>Fuzzy</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>[20, 25, 35]</td>
</tr>
<tr>
<td>SVM</td>
<td>Crisp</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>[20]</td>
</tr>
<tr>
<td>VD</td>
<td>Crisp</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>[1]</td>
</tr>
<tr>
<td>DT</td>
<td>Crisp</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>[3, 36]</td>
</tr>
<tr>
<td>$\alpha$-Shape</td>
<td>Crisp</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>[3, 36]</td>
</tr>
<tr>
<td>$desired$</td>
<td>Fuzzy or egg-yolk</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Conclusions. We observe that none of the models uses geographical context. This certainly leaves room for improvement. The only models that use the connectedness property are the Delaunay Triangulation and $\alpha$-Shape model. However, the Delaunay Triangulation model requires negative samples to work and gives a crisp boundary. The $\alpha$-Shape model can also operate without negative samples, but is not suitable for point sets with varying density. We conclude that none of the existing models meets our requirements. Therefore, we will define a new model using the connectedness property as a starting point. The next two chapters deal with this model and corresponding algorithms.
In this chapter, we develop a new model without using any geographical context. This ensures that the new model does not rely heavily on the context information, as this may not always be available or sufficiently detailed. We present the new model in Section 3.1, followed by the derivation of some mathematical properties in Section 3.2. Algorithms for the computation of the model are the topic of Section 3.3.

3.1 A new model

In the following paragraphs, we develop a new model for vernacular regions. We assume that the input is a set $S$ of unweighted, positive samples in $\mathbb{R}^2$.

**Connectedness.** Because we assume that a vernacular region is connected, we can use this property to define a new model. The question is, what do we know if two samples, say $p$ and $q$, are in the vernacular region $\mathcal{R}$? For the region to be connected, there must be some path from $p$ to $q$ contained within $\mathcal{R}$.

We could define that the line from $p$ to $q$ is in region $\mathcal{R}$. However, this allows for convex regions only, which is undesirable. To resolve this problem, we look for shortest paths using a different distance measure instead. The distance measure we use is the squared Euclidean distance. When two samples $p$ and $q$ are in the region $\mathcal{R}$, we now require that the shortest path (going from sample to sample using the squared Euclidean distance) is also in the region $\mathcal{R}$. As the triangle inequality no longer holds, there may be a path shorter than going straight from $p$ to $q$, visiting one or more intermediate samples. Figure 3.1 shows two simple examples.

![Figure 3.1](image_url)

**Figure 3.1:** Each edge is labeled with its squared Euclidean distance. (a) All shortest paths are going straight from one node to the other. (b) The shortest path from $p$ to $q$ now goes via sample $r$ as $4^2 + 2^2 < 5^2$. 

Figure 3.2: Using squared Euclidean distance allows for non-convex shapes. (a) A set of samples located in Gelderland. (b) The graph representing the core of Gelderland. It is the union of all shortest paths between two samples.

We can now define the core of the region as the union of the shortest paths between any two samples. This results in a graph with the samples as vertices and the edges being links between samples that have no shortcut via another sample. The main advantage of this model is that it can capture non-convex shapes without parameterization or geographical context and it is able to adapt to varying density. An example is given in Figure 3.2.

It is possible to generalize this model. The distance between two samples \( p \) and \( q \) then becomes \( d^t(p, q) \), the Euclidean distance raised to the power \( t \) for some \( t \geq 1 \). The parameter \( t \) then controls how easily a shortest path is detoured along other samples. With \( t = 1 \), the shortest path is always going straight from \( p \) to \( q \) resulting in a convex core. Higher values of \( t \) make the model prefer paths with shorter steps over paths with larger steps, even if this means taking more steps. This allows for non-convex shapes. If \( t \) becomes high enough, the resulting core is the minimal spanning tree (assuming points in general position).

**Outliers.** Sample data may contain a large number of outliers and thus the model should be able to deal with these erroneous samples. Outliers are characterized by laying far away from (most of) the other points: the points in the intended region will have relatively short distances between them, while outliers will have far greater distances to other points. We could remove all edges from the graph that are longer than, say, two times the average edge length. This can be parameterized to allow for a more strict or a more relaxed handling of outliers, depending on the user preferences. However, we need to be careful to keep the graph connected. More details on how outlier filtering is done can be found in Section 3.3.4.

**Holes.** Since we do not wish to exclude the presence of holes in vernacular regions, we need a way to detect these. We can use a similar approach as with outliers. Holes will typically be larger than the average face of the graph. Hence, all faces that have an area larger than two times the average face area are considered holes. Again, this is parameterized to deal with holes flexibly, depending on user preferences. Note that to be able to talk about faces, the graph representing the core should be planar. This is indeed the case as shown in Section 3.2.
3.2 Properties

**Fuzzy boundary.** The core defined by the squared Euclidean distances results in a crisp boundary, whereas a fuzzy boundary (or egg-yolk model) would be desired. A straightforward way to obtain a fuzzy boundary is to define the certainty at a point \( p \) in the plane as the Euclidean distance of \( p \) to the outline of the core. An egg-yolk model can be found by setting a threshold distance \( \delta \). Any point in the plane with a distance to the core greater than \( \delta \) is considered outside the region. The core itself is the “yolk”. All other points form the “egg white”, the uncertain area. Figure 3.3 shows a potential fuzzy boundary around the core found for Gelderland (see Figure 3.2 for the core).

![Figure 3.3: A fuzzy boundary for Gelderland based on the core in Figure 3.2(b).](image)

3.2 Properties

In this section, four lemmas are presented based on the squared Euclidean distance. From these lemmas, it follows that the Delaunay triangulation is a superset of the core for point sets in general position. A linear postprocessing step can be used to resolve most problems with four cocircular points. However, this procedure may not be sufficient in degenerate cases.

The first lemma we present is a corollary of the Law of Cosines. It is related to when a direct path (a straight line) is shorter than a path with one intermediate node.

**Lemma 1** Let \( p, q \) and \( r \) be three points in \( \mathbb{R}^2 \). Then \( d^2(p, q) \leq d^2(p, r) + d^2(r, q) \) if and only if \( \angle prq \leq \frac{1}{2} \pi \).

![Figure 3.4: Situation sketch for Lemma 1.](image)

**Proof.** Refer to Figure 3.4. Let \( \varphi = \angle prq \). From the Law of Cosines, we know that the following equation holds:

\[
d^2(p, q) = d^2(p, r) + d^2(r, q) - 2 \cdot d(p, r) \cdot d(r, q) \cdot \cos \varphi
\]

As \( \varphi \) must be between 0 and \( \pi \), \( \cos \varphi \) is nonnegative if and only if \( \varphi \leq \frac{1}{2} \pi \). As distances are always nonnegative, we can conclude that \( d^2(p, q) \leq d^2(p, r) + d^2(r, q) \) if and only if \( \varphi = \angle prq \leq \frac{1}{2} \pi \). \( \square \)
Based on this lemma, we observe that the model resembles the Gabriel Graph [18]. The edge set of the Gabriel Graph is defined as all the edges $pq$ for which the open disc with diameter $pq$ is empty. By Lemma 1, the core can only contain an edge $pq$ if this disc is empty. However, the new model can exclude more edges and thus is a subset of the Gabriel Graph. The main reason to use the new model instead of the Gabriel Graph is that the new model is based on paths instead of samples. For the Gabriel Graph, the removal of an edge is supported by the presence of one point in the circle. However, in the new model, an entire path supports the removal of an edge; this path does not even need to go through the circle. For an example refer to Figure 3.5, where a high density path is shown between samples $p$ and $q$. Depending on whether the path passes through the circle $C$, edge $pq$ is present in the Gabriel Graph. However, in the new model this is irrelevant: the length of the path is shorter than $|pq|^2$ and thus the edge is not in the core, in neither of the examples. One can argue that removing “long” edges may alleviate the problem. However, such an approach cannot cope easily with varying density. Moreover, it also introduces another (unnecessary) parameter.

![Figure 3.5](image_url)

Figure 3.5: (a) The path of white points contains a point in the circle $C$, so $pq$ is not in the Gabriel Graph. (b) The path of white points does not contain a point in $C$, hence $pq$ is present in the Gabriel Graph.

For power $t$ approaching infinity, the model resembles the relative neighbourhood graph [40]. This graph is similar to the Gabriel Graph, but a lune is used instead of a disc. The lune of two points, $p$ and $q$, is the intersection of two discs with radius $|pq|$, one centered on $p$ and the other on $q$ (refer to Figure 3.6(a)). This resembles the new model with an infinite power, since for any point $r$ within the lune, it holds that $|pr| < |pq|$ and $|rq| < |pq|$ and thus $|pr|^t + |rq|^t < |pq|^t$ holds for sufficiently large $t$. The $\beta$-skeleton is a generic framework of which the Gabriel Graph and the relative neighbourhood graph are special instances (with $\beta = 1$ and $\beta = 2$ respectively). In this framework, a more generic lune is defined based on the value of the parameter $\beta$. This parameter determines the center of each disc and its radius. For $\beta = 1$, the center of both discs is the center of the line segment $pq$. For higher values of $\beta$, the centers of the discs move in opposite direction towards points $p$ and $q$; with $\beta = 2$, the centers coincide with $p$ and $q$. The radius of the discs is defined such that the disc centered near $p$ passes through $q$ and vice versa. A lune with $\beta = 1.5$ is depicted in Figure 3.6(b). This $\beta$-skeleton was first defined by Kirkpatrick and Radke [27]. An overview of the $\beta$-skeleton, Gabriel Graph and relative neighbourhood graph is given by Jaromczyk and Toussaint [24]. However, the new model does not coincide with the $\beta$-skeleton for values of $\beta$ between 1 and 2; the lune for a given value of $\beta$ does not correspond to the area where a single sample is sufficient to provide a shorter path for a fixed value of $t$. For every point $r$ on the boundary of this area, the boundary condition $|pq|^t = |pr|^t + |rq|^t$ holds. Using the definition of the
3.2. Properties

Figure 3.6: (a) The lune for $\beta = 2$, as used by the relative neighbourhood graph. (b) The lune for $\beta = 1.5$. The center of each disc is indicated using a white point.

lune, Pythagoras’ theorem and the boundary condition, it is straightforward to express $\beta$ in terms of the power $t$ at the point where the boundaries of the discs intersect. This expression is $\beta = 2 \cdot \sqrt{1/4}$. However, for a given value of $t > 2$ and the corresponding value of $\beta$, the boundary condition $|pq|^t = |pr|^t + |rq|^t$ does not hold on other parts of the boundary of the lune. Hence, the lune does not correspond to the area where a single sample is sufficient to provide a shorter path for a fixed value of $t$. Nonetheless, this area is a subset of the lune with $\beta = 2$ and a superset of the lune (disc) with $\beta = 1$.

For a point set in general position, in particular without four cocircular points, the Gabriel Graph is a subset of the Delaunay triangulation. As the core of the new model is a subset of the Gabriel Graph, the new model is automatically planar and a subset of the Delaunay triangulation as well. However, when there are four cocircular points, the Delaunay triangulation is not unique and the core need not be planar. These two issues are addressed below. Two lemmas are presented. The first shows that only four cocircular points may violate the planarity of the core. The second shows that every edge is still part of some Delaunay triangulation. Note that this does not imply that there is a single Delaunay triangulation that contains all edges of the core.

**Lemma 2** For points in general position, the graph induced by the union of all shortest paths using squared Euclidean distance is planar. The only possible violation of planarity are four points forming a rectangle.

**Proof.** To derive a contradiction, suppose two edges of the graph, $uv$ and $xy$, intersect. Since these edges are (part of) a shortest path, we know that it is not shorter to go by node $x$ on the path from $u$ to $v$: $d^2(u, v) \leq d^2(u, x) + d^2(x, v)$. From Lemma 1, we conclude that $\angle uxv \leq \frac{\pi}{2}$. Similarly, we can conclude that $\angle uyv \leq \frac{\pi}{2}$, $\angle xuv \leq \frac{\pi}{2}$ and $\angle xvy \leq \frac{\pi}{2}$. As $uxvy$ forms a quadrilateral, we know that the sum of these angles is exactly $2\pi$. The only way the constraints are fulfilled is when all angles are $\frac{\pi}{2}$; then the four nodes form a rectangle. □
Lemma 3 Any edge along a shortest path using squared Euclidean distance on a point set \( S \) is an edge of some Delaunay triangulation of \( S \).

Proof. Suppose edge \( pq \) is an edge along a shortest path on \( S \) using squared Euclidean distance. From Lemma 1, we know that \( pq \) can be the shortest path from \( p \) to \( q \) only if there is no point \( r \in S \) such that \( \angle prq > \frac{1}{2}\pi \). Hence, the circle with diameter \( pq \) is empty, as any point in the circle would make an angle greater than \( \frac{1}{2}\pi \) with \( p \) and \( q \). Because this circle is empty, \( pq \) is part of some Delaunay triangulation of \( S \). \( \square \)

As every edge in the core is part of some Delaunay triangulation, it seems reasonable to assume that the edges of the core are a subset of a Delaunay triangulation. Since we do not assume general position, this triangulation is not unique, but we argue here that we can always choose an appropriate one. The only room for choice in the triangulation is when four or more points are cocircular and the circle defined by these points contains no other points. From Thales’ Theorem and Lemma 1, we know that a diagonal can be part of the core only if that diagonal corresponds to the diameter of the circle through the four points. This gives us three cases:

- Neither of the diagonals corresponds to the diameter. It is now irrelevant which of the edges is in the Delaunay triangulation as neither is in the core.

- Exactly one diagonal corresponds to the diameter. This diagonal could be part of the core, while the other cannot. Hence, the diagonal corresponding to the diameter should be in the Delaunay triangulation.

- Both diagonals correspond to the diameter. In this case the four points form a rectangle and thus are the special case of Lemma 2. It is possible that both should be in the core. However, by choosing an arbitrary one in the Delaunay triangulation, we can ensure that the resulting core is planar. Moreover, this choice does not have a big impact on the final core: the faces in the triangulation are isomorphic and hence if one choice leads to those faces being holes, so does the other. In degenerate cases, it is possible that the triangulation affects whether the chosen diagonal is in the core or not. An example is given in Figure 3.7. In this example, there is no shorter path for edge \( pq \). For edge \( rs \) the chain of points offers a shorter path than the edge itself. Hence, \( pq \) should be in the core while \( rs \) should not. However, these points form a square and either edge can be chosen for the Delaunay triangulation.

From the cases above we can conclude that mostly in the second case, the choice of Delaunay triangulation is relevant. We can easily postprocess a triangulation in \( O(n) \) time to deal with this special case, where \( n \) is the number of points. This is done by inspecting each edge and its incident triangles and flipping the edge if necessary.

We observe that the problem with a degenerate case can occur only for exactly four cocircular points. By the lemma below, we know that no diameter can be part of a shortest path when there are five or more cocircular points.

Lemma 4 Let \( p \) and \( q \) be samples in \( S \) and diametrical points of circle \( C \). If there are two or more points in \( S \setminus \{p, q\} \) on \( C \) on the same side of \( p \) and \( q \), then \( pq \) cannot be the shortest path using squared Euclidean distance.
3.2. Properties

Figure 3.7: A point set where the choice for Delaunay triangulation matters. The dashed line represents a series of points with a distance of 1 between neighbours. Each edge is labeled with its squared Euclidean distance. Euclidean distances are indicated at the side.

Proof. Let \( r \) and \( s \) be two points on \( C \) and on the same side of diameter \( pq \). Without loss of generality, assume that \( r \) is before \( s \) on the arc from \( p \) to \( q \). See Figure 3.8 for a situation sketch. We now show that \( d^2(p,q) > d^2(p,r) + d^2(r,s) + d^2(s,q) \), proving that there is a shorter path from \( p \) to \( q \).

Because \( pq \) is a diagonal, we know by Thales’ Theorem that \( \angle prq \) equals \( \frac{1}{2} \pi \). Hence, by the Law of Cosines, we also know that \( d^2(p,q) = d^2(p,r) + d^2(r,q) \). Hence, we have to show that \( d^2(r,q) > d^2(r,s) + d^2(s,q) \) to complete the proof. Because \( rq \) is not a diagonal and \( s \) lies on the smaller circle arc from \( r \) to \( q \), we know that \( \angle rsq > \frac{1}{2} \pi \). Hence, by Lemma 1 we can conclude that \( d^2(r,q) > d^2(r,s) + d^2(s,q) \).

We make one last observation concerning the core. Let \( f \) be the face of the core that contains the line segment between two samples \( p \) and \( q \). Walking along \( f \) from \( p \) to \( q \) need not result in a path that is shorter than \( pq \), when using the squared Euclidean distance. However, there can still be a path between the samples that is shorter but does not completely coincide with the boundary of \( f \). Refer to Figure 3.9 for an example.

Figure 3.8: Situation sketch for Lemma 4.

Figure 3.9: Both paths around face \( f \) are longer than \( pq \). The thick line is a shorter path.
3.3 Algorithms

In this section, we deal with computing the vernacular region using the new model. The complete algorithm consists of three parts, described individually in the upcoming sections. Section 3.3.1 deals with computing the core on a given set of samples. In Section 3.3.2 we present a more efficient algorithm when we do not use hole detection. How to obtain a fuzzy boundary is the topic of Section 3.3.3. Finally, Section 3.3.4 deals with outlier filtering and hole detection methods.

3.3.1 Finding the core

Because the core we wish to find is a subset of the Delaunay triangulation, we can start by computing this triangulation on the set of samples. This takes $O(n \log n)$ time and $O(n)$ space [29].

The question remains how to eliminate the edges that are not part of a shortest path between two samples. This can be done by finding the shortest path distance between any two adjacent samples in the triangulation. If an edge $e$ between two samples is not the shortest path, surely no shortest path between two other samples uses $e$ as replacing it with the shorter path yields an even shorter path for the other samples.

A simple approach is to use Dijkstra’s algorithm repeatedly. Because we have a planar graph to begin with, this algorithm has a worst-case execution time of $O(n^2 \log n)$ by using Fibonacci Heaps [17]. In practice this algorithm may be faster as only a limited number of nodes needs to be inspected to conclude that some edge is the shortest path or not. An advantage of this algorithm is that it is fairly simple to implement, compared to other methods with theoretical better bounds.

Cabello [4] solves the $k$-many distances problem for arbitrary planar graphs. This problem asks to find the shortest path distances between $k$ given pairs. In the case that $k = O(n)$ and the pairs are known in advance (as is the case for us), this algorithm solves the problem in $O(n^{4/3} \log n)$ time and $O(n)$ space. This problem is more general than the problem we need to solve and one could try to eliminate or adapt certain parts of the algorithm to obtain a better worst-case execution time. There are three main differences:

- If no outlier filtering is done beforehand, we deal with a triangulation instead of an arbitrary planar graph. However, this does not seem to improve the execution time for the algorithm or give rise to adaptations. Otherwise, triangulating the graph, giving the new edges infinite weight, would improve the algorithm for any planar graph.

- All pairs are neighbours in the graph. While this allows us to eliminate part of the data structure, the bottleneck of the computation remains the same.

- We use the squared Euclidean distance as edge weights rather than arbitrary (nonnegative) weights. However, this distance measure still allows for arbitrarily long detours. Therefore, this difference does not help in directly improving the algorithm presented by Cabello.

Combining the Delaunay triangulation algorithm with the algorithm for $k$-many distances results in an algorithm that executes in $O(n^{4/3} \log n)$ time and $O(n)$ space. The question arises whether it is possible to find the core even faster. In the next section, we develop an algorithm to find the outline of the core in $O(n \log n)$ time.
3.3.2 Finding a core without holes

When we wish to find a core that contains no holes, we can use a faster algorithm. In this case, we are interested only in the outline of the core. We can now devise an algorithm that takes $O(n \log n)$ time and $O(n)$ space, based on an algorithm presented by Klein [28]. Klein’s algorithm creates a data structure in $O(n \log n)$ time and space that allows for querying the shortest-path distance between a node on the outer face and another node in $O(\log n)$ time. The algorithm makes use of Top Trees, a specialized data structure for dynamic trees that allows for efficient updates and path or subtree queries. Top Trees have been originally presented by Alstrup et al. [2] and are built on Topology Trees [16]. The first stand-alone, worst-case logarithmic bound implementation of Top Trees are given by Werneck [43]. His work also provides a good overview on similar and related structures. In this section, we briefly describe Klein’s algorithm and the required changes to find the outline of the core and prove its correctness and time bound.

Definitions and notation. In Klein’s work, there are three basic structures that are used. There is an input graph $G$ with outer face $f_{out}$, a rooted spanning tree $T$ and a dual tree $T^*$. The dual $T^*$ is defined as the dual of $G$ excluding any edges that are dual to an edge in $T$ (refer to Figure 3.10). This makes $T^*$ again a tree. The dual tree is rooted at $f^*_{out}$, the dual vertex of the outer face. The tree $T$ defines a distance assignment to vertices, denoted by $d_T(v)$. More precisely, $d_T(v)$ is the distance from the root of $T$ to $v$ along the tree. The actual path is denoted by $T[v]$.

Figure 3.10: A graph with spanning tree $T$ and its dual tree $T^*$.

Let $l(uv)$ denote the length (or weight) of a directed edge $uv$ of $G$. We define the reduced length of $uv$ as $l(uv) + d_T(u) - d_T(v)$. We call an edge relaxed if its reduced length is nonnegative and unrelaxed if it is negative. An edge is tight if its reduced length is exactly zero. We observe that any edge in $T$ is tight. Moreover, if all edges are relaxed, $T$ is a shortest-path tree. We call an edge a leafmost unrelaxed edge if it is an unrelaxed edge and all its descendants in $T^*$ are relaxed. We can relax an unrelaxed edge $uv$ by removing the incident edge on $v$ currently in $T$, adding $uv$ and updating the distances accordingly.

We also need a definition for when a (directed) path $P$ in $G$ is left of (or analogously right of) some other path $Q$ in $G$ with the same initial vertex. We give only an intuitive definition here; for a formal definition, refer to the work of Klein [28]. Suppose we are “walking” along the common parts of $P$ and $Q$ and $uv$ is the last edge of some common part and $vp$ and $vq$ are the next edges for $P$ and $Q$ respectively. We call $P$ left of $Q$ if $uv$ is always in between $vp$ and $vq$ in counterclockwise order around $v$ for any common part. An example is given in Figure 3.11.
Let $T'$ and $T''$ be two trees with the same root. We call $T'$ left of $T''$ if $T'[v]$ is left of $T''[v]$ for every vertex $v$. We call a tree a rightmost shortest-path tree if it is a shortest path tree such that every other shortest-path tree is left of the rightmost. A less strict tree is called a right-short tree. Such a tree need not be a shortest-path tree. However, for every vertex $v$, it must hold that any path right of (but not equal to) $T[v]$ is longer than $T[v]$ itself.

Klein’s algorithm. The idea of the algorithm is to maintain a rightmost shortest-path tree while walking along the outer face in clockwise direction. Using persistence methods by Driscoll et al. [11], these trees are stored in $O(n \log n)$ space. The high-level algorithm is given below. Note that its presentation is different from Klein’s to make it more uniform with the adaptation we will present later.

**Algorithm 1 ShortestPaths($\mathcal{G}$)**

**Require:** $\mathcal{G}$ is a planar graph with $s$ vertices on $f_{\text{out}}$, $r_0$, $r_1$, ..., $r_{s-1}$ in clockwise order.

**Ensure:** A data structure to query shortest-path distances.

1: $T \leftarrow$ rightmost shortest-path tree rooted at $r_0$
2: $i \leftarrow 1$
3: **while** $i < s$ **do**
4: remove the edge of $T$ entering $r_i$
5: add $r_ir_{i-1}$
6: **while** there is an unrelaxed edge **do**
7: relax a leafmost unrelaxed edge
8: **end while**
9: $i \leftarrow i + 1$
10: **end while**

The initialization can be done in $O(n \log n)$ time by using Dijkstra’s algorithm on vertex $r_0$ with an additional postprocessing step where a right-first search tree is built on the tight edges of $\mathcal{G}$. The loop moves the root of $T$ around the outer face. Lines 4 and 5 move the root to the next vertex. Although this may cause $T$ to be a shortest-path tree no longer, Klein proves that $T$ is a right-short tree throughout the algorithm. The loop in lines 6 and 7 adapt $T$ such that it is a shortest-path tree once more. As $T$ remains right-short, it is now a rightmost shortest-path tree.

For efficiency, it is important that a leafmost unrelaxed edge is selected for relaxation. Klein proves that each edge is relaxed at most once by using this scheme. One relaxation step...
3.3. Algorithms

takes $O(\log n)$ time by using a Top Tree for the dual tree $T^*$. Another Top Tree is used for $T$ to allow for $O(\log n)$ time queries for distances between the root of $T$ and any other vertex in the graph.

An erroneous adaptation. A first idea would be to use the union of the trees constructed in Klein’s algorithm. However, this need not result in the right core, even when just considering the outline. This is caused by the nature of shortest-path trees. We can determine a shortest path between two nodes only if one is an ancestor of the other in the tree. If they reside on different branches, we cannot conclude what the shortest path is, but at most give an upper bound on the length. As only the nodes on the outer face are used as roots of the trees, we cannot guarantee that we have seen the shortest path between two internal nodes during the algorithm. An example is given in Figure 3.12.

Figure 3.12: (a) An example point set with Euclidean distances as indicated. Each adjacent pair of small points has distance 1, except at the lower left and right corners, which have a distance of $\sqrt{2}$. (b) The rightmost shortest-path tree for each point on the outer face of the Delaunay triangulation. (c) The desired core has one edge more on the outline.

The outline of the core. To devise an $O(n \log n)$ time and $O(n)$ space algorithm for the outline of the core, we now adapt Klein’s algorithm. We are no longer interested in the full query structure, reducing space complexity to linear as we need only the current tree $T$ and its dual $T^*$. We observe that it is not needed to know the vertices on the outer face $(r_0, \ldots, r_{s-1})$ in advance. So we can adapt the outer face along the way, as long as it does not influence previous results. We walk along the outer face, constantly asking whether the edge to the next vertex on the outer face is a shortest path. If it is, we continue like Klein’s algorithm. If it is not, we remove the edge from the graph. A high-level overview of the algorithm is given below. The FindNext procedure is illustrated in Figure 3.13.

The given algorithm assumes that each vertex is visited only once when walking around the outer face: its clockwise neighbour is then unique. This need not be the case in general. To make the algorithm robust, we need the incoming edge for procedure FindNext rather than only the vertex. In addition, the guard of the loop on line 5 requires a slight change. If $r_i = r_0$ holds, it should stop only if in addition, the next vertex on the outer face of $r_i$ is equal to $r_1$. The second clause indicates that the next vertex to be treated has also been treated before and we are really back at the initial situation. This addition does not affect the time complexity of $O(n \log n)$.
**Algorithm 2** CoreOutline(Г)

**Require:** Г is a planar graph.

**Ensure:** The outer face of Г is the outline of the core.

1: \( r_0 \leftarrow \) some node on outer face  
2: \( T \leftarrow \) rightmost shortest-path tree rooted at \( r_0 \)  
3: \( i \leftarrow 1 \)  
4: \( r_1 \leftarrow \text{FindNext}(Г, T, r_0) \)  
5: **while** \( r_i \neq r_0 \) **do**  
6: remove the edge of \( T \) entering \( r_i \)  
7: add \( r_ir_{i-1} \)  
8: **while** there is an unrelaxed edge **do**  
9: relax a leafmost unrelaxed edge  
10: **end while**  
11: \( i \leftarrow i + 1 \)  
12: \( r_i \leftarrow \text{FindNext}(Г, T, r_{i-1}) \)  
13: **end while**  
14: **return** \( r_0, \ldots, r_{i-1} \)

**Algorithm 3** FindNext(Г, T, r)

**Require:** \( r \) is on the outer face of Г and the root of shortest-path tree \( T \).

**Ensure:** Any too long edge in clockwise direction on the outer face starting at \( r \) is removed.

**Ensure:** The return value is the next vertex on the (new) outer face.

1: \( r' \leftarrow \) clockwise neighbour of \( r \) on outer face  
2: **while** \( d_T(r') < l(rr') \) **do**  
3: remove edge \( rr' \)  
4: \( r' \leftarrow \) clockwise neighbour of \( r \) on outer face  
5: **end while**  
6: **return** \( r' \)

![Figure 3.13](image-url)  

Figure 3.13: An example of the FindNext procedure. (a) The edge \( rr'_a \) is not the shortest path and is removed. (b) Again, \( rr'_b \) is not the shortest path. (c) Now \( rr'_c \) is the shortest path, so the procedure returns \( r'_c \).

Note that we cannot simply check whether the next edge is part of the current shortest-path tree as we are dealing with a rightmost shortest-path tree while walking clockwise around the outer face. Due to this, the edge is not in the tree if there is another path that is exactly as long.
3.3. Algorithms

When an edge is removed from $G$ in FINDNEXT, the trees $T$ and $T^*$ must be updated accordingly. We can easily see that $T$ does not change as the removed edge is not part of $T$. Updating $T^*$ is done as follows. Let $e$ be the edge we are removing from the original graph. The edge $e$ has two incident faces, the outer face $f_{\text{out}}$ and an internal face $f$. As $e$ is removed, $f$ becomes part of the outer face and thus the dual should be updated such that all edges incident to $f^*$ (the dual vertex of $f$) are now incident to $f_{\text{out}}^*$. This can be done using the cut and link operations of Top Trees and each call takes $O(\log n)$ time. The number of calls made depends on the degree of $f^*$ and thus updating the dual tree takes $O(\deg(f^*) \log n)$. Since the graph is planar, we know that $O(\sum_{f} \deg(f^*)) = O(n)$ and hence the total time spent on updating the dual tree during the algorithm is $O(n \log n)$. Therefore, the time complexity of the algorithm remains the same.

What remains to be done is justifying that the approach is still correct. The main reason is that the edge being removed is never part of tree $T$, not even in its intermediate right-short state. This is stated and proven in the following lemma.

**Lemma 5** Let $e$ be an edge that is removed at some point during the algorithm. Then $e$ is never part of tree $T$.

**Proof.** There are three ways for $e$ to become part of $T$ in the algorithm. However, we will see that none of these ways are valid when $e$ is removed during the algorithm. Let $p$ and $q$ denote the endpoints of $e$ and assume that $p$ is before $q$ in clockwise order along the current outer face. Because $e$ is removed at some point (when FINDNEXT is called for $p$), we know that $e$ is not a shortest path between $p$ and $q$. Now the three cases are as follows:

- Edge $e$ is part of the initial shortest-path tree. However, $e$ is not the shortest path between its endpoints. Therefore, $e$ cannot be part of the initial shortest-path tree.

- Edge $e$ is made part of $T$ when the root changes from $p$ to $q$. However, $q$ cannot be the next vertex on the outer face when vertex $p$ has been dealt with as $e$ is removed during the processing of vertex $p$.

- Edge $e$ is a leafmost unrelaxed edge at some point during the algorithm. Lemma 6 states that this cannot be the case.

As none of the three possibilities to add edge $e$ to tree $T$ can occur, we know that $e$ is never part of $T$. □

**Lemma 6** Let $e$ be an edge that is removed at some point during the algorithm. Then $e$ is never a leafmost unrelaxed edge.

**Proof.** We first observe that the right-shortness invariant that holds in Klein’s algorithm is still valid in the new algorithm. Klein has proven that selecting and relaxing a leafmost unrelaxed edge maintains the invariant, as well as moving the root to the next vertex on the outer face. Both claims are still valid, even after removal of the edges by using the current outer face, rather than the initial outer face. Hence, the only way to falsify the invariant is during the FINDNEXT procedure when edges are removed. However, we observe that the removed edge is left of any path in the tree. Hence, if the tree was right-short before removal, it is still right-short after. Since we start with a right-short tree, the removal of edges maintains the right-shortness invariant.
Let $p$ and $q$ denote the endpoints of $e$ and assume that $p$ is before $q$ in clockwise order. Edge $e$ has two directed edges that can be a leafmost unrelaxed edge, $pq$ and $qp$. Note that points $p$ and $q$ do not necessarily have to be on the initial outer face. There can be a chain of removed edges leading to $p$ and $q$ being on the current outer face. However, as these edges have been removed, none of these can be part of a shortest path.

The directed edge $pq$ cannot be a leafmost unrelaxed edge, as the intermediate tree $T$ no longer satisfies the right-shortness invariant in this case. Refer to Figure 3.14(a). The shortest path from $p$ to $q$ is right of the edge $pq$. Hence, if $pq$ is part of a tree, then this tree is not right-short. A leafmost unrelaxed edge can be selected during the algorithm to be relaxed, making it part of the tree. Therefore, $pq$ cannot be a leafmost unrelaxed edge as it would allow the edge to become part of the tree, violating the right-shortness invariant.

For edge $qp$, we need a different argument as the shortest path is now left of the edge. We prove this by induction on the number of edges that are removed to make $e$ part of the outer face at some point. Refer to Figure 3.14(b). In the base case, $e$ is on the initial outer face; no edges need to be removed. Let $T'$ be the tree after relaxing $qp; T'[p]$ is then $T'[q]$ followed by edge $qp$. We observe that path $T[p]$ is left of path $T'[p]$ as $T[p]$ does not use edge $qp$. Since $qp$ is unrelaxed, we know that $d_T(q) + l(qp) < d_T(p)$. But then tree $T$ was not right-short, contradicting the invariant.

For the induction step, suppose $z$ edges are removed to make $e$ part of the outer face, where $z > 0$ (refer to Figure 3.14(c)). For all these edges, we know that at most $z - 1$ edges need to be removed to make it part of the outer face. Hence, by the induction hypothesis, we also know that none of these edges are ever part of the tree $T$. So again, we can apply the same reasoning as for the base case: path $T[p]$ is left of path $T'[p]$ and the assumption that $qp$ is unrelaxed violates the right-shortness invariant.

From the analysis above, we conclude that an edge $e$ that is removed during the algorithm is never a leafmost unrelaxed edge due to the right-shortness invariant of the tree.

By Lemma 5, we know that an removed edge can never influence the structure $T$, even before its removal. Hence, any edge removed during the algorithm cannot affect any of the proofs given by Klein [28]. And therefore, the same results still hold.
To be able to deal with outliers appropriately, outlier filtering should be done before executing this algorithm: every vertex on the outer face is on the outer face in the end result, while everything inside of the core remains unchanged. Hence, outliers may prevent the algorithm from dealing with the actual core.

### 3.3.3 Calculating an egg-yolk model

To calculate an egg-yolk model from a (crisp) core, we look at the region that can be reached within a threshold distance $\delta$ from the core. This means we wish to calculate the Minkowski sum of the core (a polygon) and a circle of radius $\delta$. This is also known as polygon offsetting. Refer to the work of Kalmanovich and Nisnevich [26] and the work of Wein [42] for methods to compute the offset explicitly.

### 3.3.4 Filtering outliers and detecting holes

Outlier filtering can be done before or after computing the core. Either way, we use the same ideas but with a different set of edges. Filtering can be done by removing all edges that are longer than the mean multiplied by some constant factor. Let $\tau > 1$ be the constant factor and let $\mu_E$ denote the mean edge length over a set edges $E$. We can define the filter process in two ways, either with a static or a dynamic mean. However, we will see that the static-mean approach is flawed.

A static mean implies that first $\mu_E$ is calculated for the initial set of edges $E$. If filtering is done beforehand, $E$ consists of the edges in the Delaunay triangulation. Otherwise, $E$ consists of the edges of the current core. The edges of the (actual) core $C$ are defined as the following set:

$$C = \{ e | e \in E \land |e| \leq \tau \cdot \mu_E \}$$

However, this static mean can give undesirable results as illustrated in Figure 3.15. The further point $p$ is to the left, the higher the value of $\mu_E$ becomes. If $p$ is moved far enough to the left, $q$ and $r$ are no longer considered outliers by this procedure. This is because the mean is static and each outlier has effect on the final result.

![Figure 3.15: A point set with its Delaunay triangulation. Intuitively, points $p$, $q$, and $r$ are outliers.](image)

To counter this effect, we can use a dynamic mean. Instead of calculating a fixed mean, we now look for a core $C$ that satisfies the following equation:

$$C = \{ e | e \in E \land |e| \leq \tau \cdot \mu_C \}$$

However, this equation can have multiple solutions as adding edges or removing edges affects the value of $\mu_C$. We wish to find the maximal solution, the set with highest cardinality. To obtain this maximal solution, we start with an initial estimate of $C = E$, all available edges. While $|e_l| > \tau \cdot \mu_C$ holds for the longest edge $e_l \in C$, we remove the edge from $C$ and update
\( \mu_C \) accordingly. Note that this procedure ends with at least one edge, as then \(|e_l| = \mu_C \) must hold.

This procedure can be executed in \( O(n \log n) \) time by sorting the edges on length. Using this dynamic mean approach, edges that are considered outliers do not effect the mean in the final solution and hence the problem related to Figure 3.15 does not occur.

We must guarantee a connected graph, whereas the computed result might be disconnected. When removing an edge \( e \), we check if the result is still connected. If it is, we can continue like normal. If it is not, the edges are split into two components. If the smaller one is considered “too small”, the entire component is considered an outlier and thus removed as well. If the smaller one is not considered “too small” we do not remove \( e \). We still have to continue the process as one of the components may consist of edges that are all removed at some point. In such case, the removal of \( e \) no longer splits the edges and thus it can be removed after all. We call a component too small if it has less samples than a certain number, say \( \vartheta \).

Another step in removing outliers is to remove edges with an endpoint of degree 1, until no such edge exists. This can be done in \( O(n) \) time. After computing the core, we may wish to find holes. These holes are “outlier faces” and we use the same approach as for edges, using the area of faces in combination with a dynamic mean. For holes we may use a different value for \( \tau \), say \( \tau_h \). In addition, we also use clusters of faces rather than samples, introducing a new parameter \( \vartheta_h \). If the set of non-hole faces is found, there may be multiple components of faces (two faces are connected if they share an edge). While there are multiple components having more than \( \vartheta_h \) faces, we again add faces by increasing area until one large component remains. Note that a core may have multiple components inherently. Hence, we add faces only if this can in fact connect a separated component. Only the largest component is maintained at the end of the process. All empty faces are then merged, to prevent the core from containing parts that are infinitely thin (single edges).
Chapter 4

Extending the model for context

In this chapter, we revise the new model to incorporate geographical context. In Section 4.1, we extend the model, followed by an overview of mathematical properties in Section 4.2. The corresponding algorithms are the topic of Section 4.3.

4.1 Extending the model

In this section, we extend the model presented in Section 3.1 to take geographical context into account. The context is modeled as a set of obstacles $O$, where each obstacle is a closed line segment. The type of obstacle (river, road, etcetera) could be used to differentiate costs between types, such as crossing a country border is more costly than crossing a river. However, we do not make this distinction and simply use a set of unlabeled line segments.

**Squared Euclidean distance revisited.** To incorporate the geographical context, we no longer use the squared Euclidean distance as distance measure between two nodes but rather the square of the shortest, obstacle-avoiding path length. We still use the squared distance to make it a clean generalization of the model without geographical context. Moreover, context information may not be available or detailed enough and the model would result in convex shapes when using Euclidean distance. But even with detailed geographical context, using just the shortest path distance may give undesirable results: the shortest path may go through very sparsely sampled space, while a slightly longer path may go through very densely sampled space (see Figure 4.1). Intuitively, the latter seems to be more related to the intended region. Hence, just the shortest path distance is insufficient even in the presence of obstacles. This problem could also be resolved by thresholding the path distance: whenever the shortest path between two samples is longer than the threshold, the path is left out completely, assuming that the connectedness will be handled by other samples in between. However, this does not automatically guarantee that the region is connected. Moreover, the effectiveness of this method heavily depends on the threshold parameter.

It is possible that the context causes a shortest path that is too long: consider two samples on different sides of a river. The shortest path would have to go around the entire river, while it is more natural to just cross the river. The situation is even worse when using administrative boundaries, as these usually subdivide the plane. In this case, there need not even be a shortest path between two samples. To avoid this problem, it should also be possible to pass through the obstacles, albeit at some additional cost. This means we deal with *soft obstacles*, obstacles we can pass, in contrast to *hard obstacles*, those we may never
Chapter 4. Extending the model for context

Figure 4.1: (a) The shortest path from \( p \) to \( q \) goes through space with no samples nearby, above the obstacle. (b) A longer path from \( p \) to \( q \) goes through space with samples nearby, below the obstacle. This path is more likely to be in the intended region than the shortest path.

pass. An alternative would be to allow for multiple cores, but this violates the property of connectedness.

Formally, we define the distance between two samples, \( p \) and \( q \), as follows. Let \( \mathcal{P} \) denote the set of all paths between \( p \) and \( q \), \( |P| \) the Euclidean length of path \( P \) and \( k_P \) the number of intersections between path \( P \) and the set of obstacles \( \mathcal{O} \). Also, let \( \kappa \) denote the additional cost of one crossing. We then express the distance between \( p \) and \( q \) as:

\[
d^t_O(p, q) = \min_{P \in \mathcal{P}} \left\{ |P|^t + \kappa \cdot k_P \right\}
\]

The subscript \( \mathcal{O} \) indicates that we include the set of obstacles in the distance definition. Observe that if \( \mathcal{O} \) is empty or causes no crossings on the shortest path, this definition results in the Euclidean distance raised to the power \( t \). Using the squared Euclidean distance, \( t = 2 \), the expression above results in:

\[
d^2_O(p, q) = \min_{P \in \mathcal{P}} \left\{ |P|^2 + \kappa \cdot k_P \right\}
\]

**Fuzzy boundary revisited.** Similarly to the distance between samples, we use the shortest-path distance rather than the Euclidean distance to find the fuzzy boundary. Again, we could consider using soft obstacles instead of hard obstacles. This allows finding regions that should cross obstacles, but only by a bit. However, this may not be such a loss: when a vernacular region is divided into multiple (reasonably sized) parts by roads, rivers or boundaries, it seems likely that each of these parts has a reasonable number of samples and all should be part of the core. For an egg-yolk model, we would compute the reachable region when starting to walk anywhere in the core for at most a distance \( \delta \). For a fuzzy boundary, we can define the certainty at some point in the plane to be inversely dependent on the shortest path distance from the point to the core.
4.2. Properties

First of all, we observe that in the presence of obstacles, the core need not be a planar graph, even for points in general position. A simple example is given in Figure 4.2. Although these points are not in general position, we can imagine shifting each point slightly such that this is the case and the effect remains the same.

![Figure 4.2: The shortest path from u to v and from p to q cross, even when using the squared length of the shortest obstacle-avoiding path.](image)

In absence of geographical context, we have seen that the edges of the core are a subset of the Delaunay triangulation. Now that we have context information, this is no longer the case, but we can still take a similar approach. We wish to find a graph \( G \) such that whenever an edge is in the core, it is also in \( G \); the edges of \( G \) are a superset of the edges of the core. An obvious solution is using the complete graph. However, this results in a non-planar graph with \( \Theta(n^2) \) edges and state-of-the-art algorithms are just slightly faster than cubic time (for example, see the work of Chan [5] for an \( O(n^3/\log n) \) time algorithm). So can we think of a graph that has less edges?

In a worst-case situation, the core itself has a quadratic number of edges. Hence, it is not possible in general to find a superset of the core edges that is not quadratic in the number of samples. One of such situations is given in Figure 4.3(a). First, suppose that passing through the context information has a cost such that it is never preferable to cross these boundaries for shortest free-space paths between two nodes. Above and below the obstacle are roughly \( \frac{1}{2}n \) points. All of these points are on the boundary of a circle centered on one of the endpoints. Now, the distance between any two points, one above and one below, is equal. Therefore, we must conclude that all paths between a point above and one below are in the core. The core has \( \Theta(n^2) \) edges, though these overlap such that only a linear number is visible. Note that the points do not have to be exactly cocircular, small distortions are possible depending on the radius of the circle and the distance between neighbouring points. Figure 4.3(b) shows an even worse scenario where the core is a complete graph and also has \( \Theta(n^3) \) intersections.

However, the worst-case situation is rather unrealistic and is unlikely to occur when working with real vernacular regions. Usually, a lot of sample points are together in a free area of the plane, while the actual paths that are hindered by the geographical context are limited. We will now see how we can exploit these features to find a more sparse graph for realistic inputs.

We observe that the presence of obstacles can make edges longer, but not shorter. Therefore, if a certain convex area is free of objects, we can still use the Delaunay triangulation for the samples in this area. This is formalized by the following lemma.
Figure 4.3: (a) An example with a quadratic number of edges in the core. (b) A subset of the core is shown, but these edges already cause $\Theta(n^4)$ intersections.

**Lemma 7** Let $A$ be a convex, obstacle-free area in $\mathbb{R}^2$ and let $S$ be a set of samples. Let $S_A$ denote $S \cap A$ and let $p$ and $q$ be two points in $S_A$. If the edge $pq$ is not part of some Delaunay triangulation of $S_A$, then there is no shortest path between two points in $S$ that contains edge $pq$.

**Proof.** Assume that $pq$ is not part of any Delaunay triangulation of $S_A$. We now know that circle $C$ with $pq$ as diameter must contain another point $r \in S_A$ in its interior. Moreover, because all three points are in $S_A$ and thus in the convex, obstacle-free area $A$, we know that the shortest path distance between each pair of points is simply the squared Euclidean distance. Hence, by the Inscribed Angle Theorem and Lemma 1, we know that going from $p$ to $q$ via $r$ is shorter than going directly from $p$ to $q$. Therefore, we can conclude that $pq$ cannot be part of any shortest path as replacing the edge by edges $pr$ and $rq$ would shorten the path. $\square$

The above lemma can be used to reduce the number of edges in the original graph as we could start by finding maximal sets with an empty convex hull. For each of these subsets, we calculate the Delaunay triangulation. We then know that in one set, a linear number of edges is used. However, it remains to connect these “Delaunay components” in a proper way. Unfortunately, we observe that for a complete solution, we cannot connect only the boundaries of these components: given two components, it is possible that two samples not on the boundary may need to be connected for the actual core. An example is given in Figure 4.4. Here we see that there can even be an arbitrary number of crossings in both components (depending on the number of points in the center in the example). The leftmost and rightmost point are still on the boundary, but we can add points further to the left and to the right of the construction without changing the triangulation on the given points.

Without geographical context, we have seen that a more efficient algorithm can be used when we are interested only in the outline of the core (see Section 3.3.2). If we wish to take a similar approach with context, we need a bound on the complexity of this outline. We observe that even if two vertices are not on the outer face, their shortest, obstacle-avoiding path can be partially on the outer face. An example is given in Figure 4.5. We think that between two neighbouring samples on the outer face, there can be only a low constant number of paths
Figure 4.4: There are two Delaunay components, one on the left side and one on the right. For small values of $\varepsilon$, the distance to the hole in the context data is nearly the same for all nodes, if the angle at the leftmost and rightmost node is $\frac{1}{2}\pi$. The shortest path between the leftmost and rightmost node is represented by a dashed line.

that add to the part of the outer face between the two samples. In the example, there is only one such path (the dashed line). This would bound the number of paths on the outer face to $O(n)$, where $n$ is the number of samples. Moreover, we think that a constant number of paths in the core use an endpoint of a certain obstacle. This would imply that one obstacle can influence only a constant number of paths on the outer face. This leads to the following conjecture.

**Conjecture 1** Let $S$ be a set of samples and $O$ a set of obstacle line segments. The number of edges on the outline of the core of $S$ is bounded by $O(|S| + |O|)$.

Figure 4.5: A set of samples such that there is a path partially on the outer face of the core, while its corresponding samples are not. This path is given with a dashed line. The rest of the core is indicated by gray lines. The obstacles are given in fat lines.

4.3 Algorithms

In this section, we describe the new algorithms for the extended model. The distance between two samples has changed, so we need an algorithm to compute these. This is the topic of Section 4.3.1. The conversion from a crisp boundary to an egg-yolk model or fuzzy boundary is also more complex than before and this is dealt with in Section 4.3.2. Other algorithmic aspects, such as filtering the outliers and computing the core, remain unchanged apart from using the new distance measure.
4.3.1 Computing distances

The definition of distance between samples has changed to a minimum over all paths. For this reason, we need an algorithm to compute these distances. Given a set of obstacles $O$, we wish to find shortest, obstacle-avoiding path between two samples $p$ and $q$. Recall that the obstacles are soft obstacles; it is possible to cross them at an additional cost. The following definition of distance was given in Section 4.1:

$$d^2_O(p, q) = \min_{P \in \mathcal{P}} \left\{ |P|^2 + \kappa \cdot k_P \right\}$$

Note that it is possible to have points in common with an obstacle without causing intersections. For example, when a path avoids an obstacle, it may coincide with one of its endpoints. It is also allowed to “follow” an obstacle without causing intersections.

Calculating the minimal-length path is not easily reduced to a known problem. One candidate is the shortest-path problem in weighted subdivisions as studied for example by Mitchell and Papadimitriou [31]. We could model the obstacles as very thin faces in the subdivision. However, there will be small deviations in the path as Snell’s Law of Refraction is satisfied at face boundaries as argued by Mitchell and Papadimitriou. Also, this does not calculate the shortest path as we defined it, as it uses the Euclidean distance rather than the squared Euclidean distance. We could take the square of the resulting path length, but then the resulting path minimizes the following expression instead:

$$|P|^2 + 2 \cdot |P| \cdot \kappa \cdot k_P + \kappa^2 \cdot k_P^2$$

As we can see, crossings are penalized more strongly than intended. Also note that a path that optimizes $|P| + \kappa \cdot k_P$ does not necessarily optimize $|P|^2 + \kappa \cdot k_P$, so computing the Euclidean length explicitly afterwards does not resolve the problem.

Other variations, such as the shortest-path problem on three-dimensional surfaces, have the same limitations. In the special case that the crossing cost is infinite, we can treat the obstacles as hard obstacles and use known algorithms. By using the algorithm of Hershberger and Suri [22], we can solve the problem for all pairs of samples in $O(n(n + m) \log n)$ time, where $n$ is the number of samples and $m$ is the number of obstacles.

Since known algorithms cannot be applied directly, we need to develop a new algorithm to compute the distances in the presence of soft obstacles. For arbitrary positive crossing cost, we can use Dijkstra’s algorithm on a special input graph to obtain an execution time of $O(n^2 \cdot m \cdot k_{\text{max}} + n \cdot m^2 \cdot k_{\text{max}})$, where $k_{\text{max}}$ is the maximal number of intersections for a straight line between two samples. We assume that $k_{\text{max}}$ is positive, as the shortest path between two samples is always a straight line when there are no intersections. The algorithm and corresponding analysis are the topic of the next two paragraphs.

Exact path computation. As mentioned, the algorithm is a repeated application of Dijkstra’s shortest path algorithm on a special input graph. It is executed once for each sample and during the execution, the shortest path from that source sample to any other sample is calculated. In this paragraph, we discuss the input graph and how to find the desired path after executing Dijkstra’s algorithm.

First, we need a graph to represent the obstacles. This obstacle graph, $G_O = (V_O, E_O)$, must allow for easy intersection computations, while allowing paths to coincide with line segments. The obstacles define a (non-simple) subdivision. For each corner of a face in the
subdivision – including those of degree one or angle greater than \( \pi \) – we create a vertex in the obstacle graph. Note that vertices of the obstacle graph may coincide. We call a pair of distinct vertices *neighbours* if they coincide and the corresponding faces are locally adjacent. Two faces are locally adjacent if they share a line segment that ends at the beforementioned vertices. We call a pair of vertices \((v, w)\) a *match* if they do not coincide and the line segment \(vw\) is locally incident on the vertices from their corresponding faces.

The directed edges of the graph have two associated values, their Euclidean length and the number of crossings of the edge. This makes an edge a four-tuple \((v, w, l, k)\), where \(v\) and \(w\) are its source and target respectively, \(l\) its length and \(k\) the number of crossings. For each of the neighbouring and matching pairs, we create an edge. The complete edge set is defined as:

\[
\mathcal{E}_O = \{ (v, w, 0, 1) \mid v, w \in \mathcal{V}_O \land \text{neighbour}(v, w) \} \\
\cup \{ (v, w, |vw|, k_{vw}) \mid v, w \in \mathcal{V}_O \land \text{match}(v, w) \}
\]

In the formula above, we use \(k_{vw}\) to indicate the number of obstacles crossed by line segment \(vw\), excluding any obstacles that have an endpoint coinciding with either vertex \(v\) or \(w\). Note that, for now, each directed edge has a corresponding edge in opposite direction having the same length and crossing number. The construction is illustrated in Figure 4.6.

![Figure 4.6](image_url)

Figure 4.6: The obstacles are indicated by fat gray lines. Solid black lines represent the edges of the obstacle graph. (a) These four pairs of neighbouring vertices cause a circular connection around a vertex of the subdivision. (b) A matching pair for which the line segment crosses one obstacle. (c) A simple example of a complete obstacle graph.

To find the paths from a source sample \(s \in \mathcal{S}\) to all other samples, we now augment the obstacle graph \(\mathcal{G}_O\). The resulting graph is the *augmented graph* \(\mathcal{G}_A = (\mathcal{V}_A, \mathcal{E}_A)\).

\[
\mathcal{V}_A = \mathcal{V}_O \cup \mathcal{S} \\
\mathcal{E}_A = \mathcal{E}_O \\
\cup \{ (s, r, |sr|, k_{sr}) \mid r \in \mathcal{S}\setminus\{s\} \} \\
\cup \{ (s, v, |sv|, k_{sv}) \mid v \in \mathcal{V}_O \land \text{match}(s, v) \} \\
\cup \{ (v, r, |vr|, k_{vr}) \mid r \in \mathcal{S}\setminus\{s\} \land v \in \mathcal{V}_O \land \text{match}(r, v) \}
\]

However, we cannot use Dijkstra’s algorithm on this graph due to the non-additive behaviour of the path length, caused by the squared Euclidean length. We observe that if we fix the number of crossings, the additional distance is a constant as well. What is left to minimize is the squared Euclidean length. This in turn is equivalent to minimizing the Euclidean length.
of the path. We can represent this fixation of crossing numbers by introducing layers in the graph. The result is a layered graph, \( G_L = (V_L, E_L) \). This graph is defined as follows:

\[
V_L = \{ v_i \mid v \in V_A \land 0 \leq i \leq k_{\text{max}} \}
\]

\[
E_L = \{ (v_i, w_{i+k}, l) \mid (v, w, l, k) \in E_A \land 0 \leq i \leq k_{\text{max}} - k \}
\]

There are exactly \( k_{\text{max}} + 1 \) layers, the first being layer 0 and the last layer \( k_{\text{max}} \). Each layer \( i \) contains a copy of the vertices from \( V_A \) and represents them for all paths having exactly \( i \) intersections. An edge \( (v, w, l, k) \in E_A \) now connects the vertices for each layer, taking into account the number of crossings of the edge. If \( k \) equals zero, it connects vertices in the same layer. If \( k \) is at least 1, it causes edges going from a lower layer to a higher layer. Observe that there are no edges that go to a lower layer. An example of the augmented graph and the corresponding layered graph is given in Figure 4.7.

![Layered Graph Example](image)

**Figure 4.7:** Vertices corresponding to samples are shaded. The value of \( k_{\text{max}} \) is 2. (a) An augmented graph with source \( s \) on the left. The dashed lines are edges having one or more crossings. (b) The layered graph has 3 layers. Layer 0 is on the left, layer 1 in the middle and layer 2 on the right. All dashed lines go from a lower layer to a higher one.

To compute the paths from source \( s \), we apply Dijkstra’s algorithm on vertex \( s_0 \). Let \( P_{r,i} \) denote the shortest path to sample \( r \) in layer \( i \) and \( |P_{r,i}| \) its total Euclidean length. To find the optimal path to sample \( r \), we look for the path that minimizes \( |P_{r,i}|^2 + \kappa \cdot i \):

\[
d^2_{\text{opt}}(s, r) = \min_{0 \leq i \leq k_{\text{max}}} \left\{ |P_{r,i}|^2 + \kappa \cdot i \right\}
\]

We observe that in layer \( k_{\text{max}} \) only samples are required as any minimal path with \( k_{\text{max}} \) intersections must be a straight line between samples. Also, for source \( s \), only vertex \( s_0 \) is required; its copies can be ignored. However, these small optimizations give no asymptotical improvements.

**Analysis.** Let \( n \) denote the number of samples, \( m \) the number of obstacles and \( k_{\text{max}} \) the maximal number of intersections. The layered graph constructed has \( O(n \cdot k_{\text{max}} + m \cdot k_{\text{max}}) \) vertices and \( O(n \cdot m \cdot k_{\text{max}} + m^2 \cdot k_{\text{max}}) \) edges. One execution of Dijkstra’s algorithm takes \( O(|E_L| + |V_L| \log |V_L|) \) time using Fibonacci heaps. Hence, the execution time is \( O(n \cdot m \cdot \)
4.3. Algorithms

\(k_{\text{max}} + m^2 \cdot k_{\text{max}}\), as it is dominated by \(|E_L|\). As we need to execute Dijkstra’s algorithm for each sample, the total execution time is \(O(n^2 \cdot m \cdot k_{\text{max}} + n \cdot m^2 \cdot k_{\text{max}})\).

With a straightforward implementation, the memory usage for the graph is \(O(n \cdot m \cdot k_{\text{max}} + m^2 \cdot k_{\text{max}})\), the number of edges. However, we observe that the factor \(k_{\text{max}}\) can be eliminated by storing the edges more cleverly. This factor originates from the need to create up to \(k_{\text{max}}\) copies of each edge, one starting from each layer. However, the length and number of crossings of this edge is the same for all copies and thus one would suffice. We can do this as follows. Each vertex with crossing number zero stores all its copies in a list, and each of the copies has a reference back to this zero-crossings vertex. Also, that particular vertex stores its outgoing edges to the target vertex with crossing number zero. To follow an edge from some vertex in the graph, we access the list stored in the zero-crossings version in constant time. This gives us the zero-crossing version of the target. By accessing its list of copies, we now find the right target, again in constant time. Hence, this representation does not influence the execution time, but reduces the memory usage for the graph to \(O(n \cdot m + m^2)\). The complete solution may be as large as \(O(n^2 \cdot m)\) and thus the total memory usage is \(O(n^2 \cdot m + m^2)\).

4.3.2 Outline and offsetting

Given the core of the region, we must still find its outline and holes and also offset these to obtain an egg-yolk model. To find the outer face, we can use an algorithm to compute a face in an arrangement of line segments. Both Chazelle et al. [7] and De Berg et al. [10] present an \(O(k\alpha(k) \log k)\) algorithm for \(k\) line segments, where \(\alpha(k)\) is the inverse of the Ackermann function. This inverse is an extremely slowly growing function. As we have seen, the complexity of the core can be quadratic in the number of samples and thus the algorithm finds the outline of the core in \(O(n^2 m \alpha(n^2 m) \log(nm))\) time, where \(n\) is the number of samples and \(m\) the number of obstacles.

Detecting holes gives rise to another problem. Since the core is no longer planar, we cannot use the concept of face directly. However, holes still are large empty areas when the core is “drawn”. Hence, we could make the graph planar by inserting vertices on intersections and using the same method as before. While this approach is conceptually reasonable, the graph may become quite large. In a worst-case scenario, the core has \(O(n^4)\) intersections (see Figure 4.3(b)). Replacing all intersections in such a core results in \(O(n^4)\) faces. Hence, it takes \(O(n^4 \log n)\) time to detect holes using a dynamic mean (as described in Section 3.3.4).

Offsetting the core to obtain an egg-yolk model is more complex in the presence of context. For hard obstacles, the problem of finding shortest obstacle-avoiding paths is well studied. For example, Mitchell [30] and Herschberger and Suri [22] have studied the problem of computing the shortest-path map for a given point in the presence of polygonal obstacles. We can use the algorithm of Herschberger and Suri [22] to find the egg-yolk, using a generalization from point source to polygonal source. In particular, we are interested in the “wavefront” (the reachable area) at distance \(\delta\).

For soft obstacles, there seems to be no algorithm readily available. It might be possible to adapt the algorithm of Herschberger and Suri [22], by handling wave-obstacle collisions differently. We observe that if a wavefront hits an obstacle, it can continue on the other side at increased distance. Locally, it is always best to cross the obstacle in a straight line. Hence, if the wavefront continues on the other side, the source remains the same. On collision, we could then duplicate the wavelet and increase the distance on the source. However, the details and verification of such an approach are still an open problem.
Chapter 5

Evaluation

In this chapter, we evaluate the new model using our observations from Chapter 2. We also present a small experimental study based on real data.

5.1 Theoretical evaluation

Tables 5.1 and 5.2 show the input demands and output properties as described in Section 2.2 for the new model. For the input properties, we have missed our target on having “few” parameters. However, our parameters are well-separated, each having its own purpose. This makes the parameterization manageable as each separate step in the computation uses but a few parameters. Below is a list of the parameters.

1. **Euclidean power** \( t \geq 1 \): The Euclidean distance is raised to the power \( t \). Throughout the thesis, we assumed this parameter to be 2, resulting in the squared Euclidean distance.

2. **Crossing cost** \( \kappa \geq 0 \): The additional cost for crossing an obstacle.

3. **Outlier filter** \( \tau > 1 \): The factor used to classify edges as outliers.

4. **Sample cluster** \( \vartheta \geq 1 \): If a disconnected component has less than \( \vartheta \) samples during the outlier filter process, all these samples are considered to be outliers.

5. **Hole filter** \( \tau_h > 1 \): The factor used to classify faces as holes.

6. **Face cluster** \( \vartheta_h \geq 1 \): If a disconnected component has less than \( \vartheta_h \) faces, all these faces are considered holes.

7. **Offset distance** \( \delta \geq 0 \): The distance used to create an egg-yolk model or fuzzy boundary for a found core.

Table 5.1: A comparison of the input properties of the new model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Negative</th>
<th>Outliers</th>
<th>Context</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>desired</td>
<td>No</td>
<td>Implicit or explicit</td>
<td>Yes</td>
<td>Few</td>
</tr>
<tr>
<td>New model</td>
<td>No</td>
<td>Explicit</td>
<td>Yes</td>
<td>7</td>
</tr>
</tbody>
</table>
For the output properties, we have met all criteria. The output shape is implicitly controlled by using a higher or lower power for the Euclidean distance. The presented algorithms that use context can use any power of at least 1. The algorithms without context are valid only for powers of at least 2. A higher power makes tendrils more likely, while a lower power increases the “convexity” of the core. We also observe that the resulting boundary is fuzzy (or an egg-yolk model), but not completely. The distance to the core is directly related to the certainty; in the egg-yolk model, the width of the “egg white” is fixed. However, for a real fuzzy boundary this relation does not have to be so strict. Likewise, in an egg-yolk model, the width and shape of the egg white may vary. An example of a “real” egg-yolk model is given in Figure 5.1.

Table 5.2: A comparison of the output properties of the new model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Boundary</th>
<th>Connected</th>
<th>Shape</th>
<th>Holes</th>
</tr>
</thead>
<tbody>
<tr>
<td>desired</td>
<td>Fuzzy or egg-yolk</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>New model</td>
<td>“Fuzzy”</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

5.2 Practical evaluation

First, we describe what data we used and how it was preprocessed. We then present the Kernel Density Estimation used for comparison, followed by the actual results and a discussion.

5.2.1 Obtaining real data

The sample points are obtained from Flickr (www.flickr.com) by searching photos geotagged with the region of interest. This provides us with sample data in WGS84 format, coordinates given in longitude and latitude. In order to use our algorithms, we need to project these onto the plane. There are various way to do this, all with different properties (refer to the work of Snyder [39] for an overview). The projection we used is the Mercator Projection, one common in cartography (again, refer to the work of Snyder [39] for details). This projection causes area (and, depending on angle, distance) distortion proportional to the distance to the equator. The advantage of this projection is that it is straightforward and aligns with common practice for maps. The latter ensures that the generated samples and the background data align.

Often, photos have exactly the same coordinates as they were taken at approximately the same location. We filter out these duplicate locations as Delaunay triangulations are not well defined for such a point set. One way to retain the information of duplicates is to use a certainty, based on the number of photos at one location. Naturally, locations with a lot of photos are more likely to be part of the intended region than locations with only a few. Higher certainty could then decrease the distance between samples, making them more likely to be part of the core. However, the properties we derived for the algorithms without context would no longer hold. Therefore, we do not use this approach and simply discard the duplicate samples. However, a support for weighted point sets would be desirable.

The data sets are summarized in Table 5.3. Pictures of the sets are presented along with the results in Section 5.2.3 (figures 5.2 to 5.7). For each set, we list the number of photos found,
the number of samples in the actual input and the context data available. The search results
(except for the Irish coast) have been pre-filtered to be near the expected area (for example,
photos for the Alps in North America are discarded). The reasons for the chosen data sets
are as follows. South Holland is a province in the Netherlands and thus allows for an absolute
test to see if we can actually find the province. For the same reasons, we have selected the
Swiss cantons of Valais and Graubünden. The data set of the Alps provides us with a large
international region on which the context data (country borders) should have little effect.
Southern Limburg is a national region that is partially crisp. The Parisian suburbs is an
urban region, in which a hole is to be expected. However, the number of photos is very small
and they seem to be very biased to the eastern side of Paris and thus the results are not
satisfactory. However, the setting of Paris provides us with a nice test for a region that is
dependent on context, but not all of it: some roads and the Seine should be crossed, while
other roads should not. Therefore, we have manually added additional samples and outliers
around Paris to obtain a more reasonable distribution and a better test case. The Irish coast
data set provides us with another region that should contain a hole. The data set here is
still biased to the large cities of Dublin and Belfast, but there are enough samples to obtain
decent results.

Table 5.3: An overview of the input used for the experiments. The first three data sets are
based on administrative regions, while the others are based on actual vernacular regions.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Photos</th>
<th>Samples</th>
<th>Context</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graubünden</td>
<td>3265</td>
<td>871</td>
<td>Canton borders of Switzerland</td>
<td>598</td>
</tr>
<tr>
<td>Valais</td>
<td>5343</td>
<td>2114</td>
<td>Canton borders of Switzerland</td>
<td>598</td>
</tr>
<tr>
<td>South Holland</td>
<td>9209</td>
<td>3887</td>
<td>Province borders of the Netherlands</td>
<td>2007</td>
</tr>
<tr>
<td>Southern Limburg</td>
<td>602</td>
<td>256</td>
<td>Province borders of the Netherlands</td>
<td>2007</td>
</tr>
<tr>
<td>The Alps</td>
<td>22113</td>
<td>5923</td>
<td>Country borders of Europe</td>
<td>1993</td>
</tr>
<tr>
<td>Parisian suburbs</td>
<td>196</td>
<td>167</td>
<td>The Seine and main roads around Paris</td>
<td>338</td>
</tr>
<tr>
<td>Paris manual</td>
<td>-</td>
<td>623</td>
<td>The Seine and main roads around Paris</td>
<td>338</td>
</tr>
<tr>
<td>Irish coast</td>
<td>25060</td>
<td>11160</td>
<td>Shorelines of Ireland and Great Britain</td>
<td>968</td>
</tr>
</tbody>
</table>

5.2.2 Kernel Density Estimation

We compare the results of our model to the results of Kernel Density Estimation. To this end,
we use an unweighted Gaussian kernel function. The Gaussian kernel function is commonly
used (for example, see Grothe and Schaab [20]). Using a radius $r$, the certainty at a location,
$C(x)$, is expressed in the kernel function $K(d)$ as follows:

$$C(x) = \frac{1}{|S| \cdot r} \sum_{s \in S} K\left(\frac{|x - s|}{r}\right)$$

$$K(d) = \frac{1}{\sqrt{2\pi}} e^{-\frac{d^2}{2}}$$

To obtain an egg-yolk model from Kernel Density Estimation, we cut the KDE surface
using two parameters, $\alpha$ and $\beta$. The “yolk” is the set of all points for which $\beta \leq C(x)$; the
“egg white” is the set of all points for which $\alpha \leq C(x) < \beta$. 
5.2.3 Results and comparison

For each of the input data sets, we present the following figures: a visualization of the input data set, the result of Kernel Density Estimation, and the result of the new model with and without using geographical context. For the parameterizations used to obtain these results, refer to Appendix A.

Administrative regions. We first discuss the administrative regions. Figures 5.2 to 5.4 show the results.

When we look at the results of the model without context, it captures the shape of the cantons Graubünden and Valais decently (Figure 5.2 and 5.3 respectively). The former has a low sample density in the southwest corner, causing it to be filtered out. This part can also be considered a small tendril, while we have based our modeling on the assumption that regions do not have tendrils. Hence, this behavior is as expected. The data set for South Holland (Figure 5.4) contains a large number of outliers, in particular, a dense cluster of over a hundred samples located in southern Limburg. Using proper parameterization, the model still manages to pick up on the main part of the Dutch province.

When we compare the results of the new model with those of the KDE method, we see that the new model is better at capturing shape. The KDE method usually gives some sort of blob to represent the region. Especially when we are looking for areas that have (more or less) exact boundaries, the correct detection of shape is important.

The administrative regions have known boundaries and these are part of the context. This makes it relatively uninteresting to look at results using context, as we are already providing the answer. However, an automated system may not be aware of this. The results show that there are indeed parameterizations of the model such that we can precisely find the administrative boundaries. For South Holland, the model completely filters out the island just below the found region, while it is in fact a part of the province. This is caused by the subdivision of the provinces: the parameters must be set such that the large cluster in southern Limburg is filtered out. Since this island has a low number of samples and the province borders cause the samples to be located at high distance from the bulk of the samples, these are filtered out as well.
5.2. Practical evaluation

(a) Input  
(b) KDE result  
(c) Without context  
(d) With context

Figure 5.2: The experimental results for Graubünden.

(a) Input  
(b) KDE result  
(c) Without context  
(d) With context

Figure 5.3: The experimental results for Valais.
Figure 5.4: The experimental results for South Holland.
5.2. Practical evaluation

Vernacular regions. We now discuss results from some actual vernacular regions. Figures 5.5 to 5.9 show the results.

The results for southern Limburg (Figure 5.5) show a reasonable coverage of the southern parts of the province. The main advantage of including the context information is that the result can now be confined to the (Dutch) province of Limburg. For the core, the context provides small improvements. The result of the KDE method in southern Limburg is comparable to the results obtained with the new model.

The Alps (Figure 5.6) showcase some important aspects of the model. Whereas the result without context looks decent, it actually deteriorates when including context information. In a way, this makes sense as the Alps are not influenced by the context provided. However, it is desirable for the model to be able to cope with irrelevant context information. One solution here would be to use a lower crossing cost (close or equal to zero). But then the context – none of it – would be really taken into account, while in general this could be possible. There is an indication that the provided context here is not relevant: multiple faces of the subdivision (Austria, France, Italy and Switzerland) contain a large portion of the samples. This information may be useful for automatic parameterization. A low crossing cost should be used in the case described. If most of the samples lie within one face (or country, province, etcetera), a high crossing cost can be used. The KDE method gives a more smooth result for the Alps. While this looks quite nice, the coverage of the new model is better.

Figure 5.7 shows the results for the unmodified data set of the Parisian suburbs. As indicated before, there are just a few samples, mostly biased to the eastern side of Paris. If we forget that the data is about Paris, the model seems to find a reasonable region for the points present. However, it appears more useful to inspect the results of the manually generated data set. These results are depicted in Figure 5.8. The results for KDE and the new model without context are comparable. The core found using the KDE method covers a bit more of the city center, but the core of the new model covers less of the area close to, but outside the center. The new model actually detects a hole located on the city center, whereas the KDE method includes the entire area in the uncertain region. When we include the context, it neatly excludes the city center from the region, encircling the Boulevard Périphérique. We do observe that the core of the region is no longer all around the center. Instead it is split at the western side. This is caused by the outlier filter due to a low sample density in that area. This result indicates that the model is somewhat capable of handling context, part of which is relevant (such as the central ring road of Paris) and part of which is irrelevant. However, this data set is manually generated and although some manual outliers have been inserted, the results are an indication at best.

The final experiment is about the coast of Ireland. The results are shown in Figure 5.9. In contrast to the other data sets, this set did not use a very crude bounding box for its photos. The result is that there are samples practically all over the world, as far as southern Chile. Only the bulk of the input is shown, around Ireland and Great Britain. Without context, the model performs very well, the midlands of Ireland filtered out during the hole detection process. When we include context, the outlier filter process creates a gap in the shoreline in the southwest of Ireland. This is caused by the low sampling around the Shannon Estuary. No hole detection has been applied for this reason. It still manages to capture the general shape of the shoreline. In contrast, the KDE method seems unable to grasp the shoreline. It finds three major clusters: one near Dublin, one near Belfast and one just above the Shannon Estuary. The entire midland of Ireland is included in the uncertain area. Therefore, we conclude that the new model outperforms the KDE method in this experiment.
Chapter 5. Evaluation

Figure 5.5: The experimental results for southern Limburg.

Figure 5.6: The experimental results for the Alps.
5.2. Practical evaluation

Figure 5.7: The experimental results for the Parisian suburbs.

Figure 5.8: The experimental results for the Parisian suburbs using manually generated data.
Chapter 5. Evaluation

Figure 5.9: The experimental results for the Irish coast.
Raising the power. Here, we look at results for varying values of the parameter $t$. In all other experiments, this parameter has the value 2, resulting in the squared Euclidean distance. The input is the set of samples that are in the core of the Alps without context (Figure 5.6(c)). Refer to Figure 5.10. As expected, using a power of 1 results in the convex hull of the data set. When we use $t = \frac{3}{2}$, we get a shape that is no longer convex. It is a superset of the shape using $t = 2$, and a subset of the convex shape. Increasing the power further, we see that the shape keeps shrinking. The highest power used, $t = 64$, shows a shape that approaches a spanning tree as expected.

Figure 5.10: The core of the Alps using various values for the parameter $t$. 
**General observations.** From the process of manually parameterizing the experiments, we learned that it requires thorough knowledge of the algorithms to be able to get the correct parameters. Especially for outlier filtering, the parameter settings are quite sensitive. If clusters are allowed to be too large (a high value for $\vartheta$) or the filter strength is too high (a low value for $\tau$), the core tends to quickly deteriorate to a small cluster, having less than one percent of the total number of samples. In contrast, if clusters are not allowed to be large or the filter strength is low, there seems to be a tendency to remove little to nothing. A possible solution to this problem would be to have a minimal core size (say 50 percent of the total number of samples). This however introduces yet another parameter. Ideally, parameters would be chosen (semi-)automatically, for example by determining the variance of the samples.

The results of the Kernel Density Estimation are smooth shapes. For regions that are (partially) bounded by context, this seems undesirable. However, for shapes that are not strictly bounded, such as the Alps, a smooth shape seems more suitable than the jagged shape the new model produces. Since we have a polygonal core, we could consider applying polygon-smoothing methods to obtain a more smooth shape. As an example, Figure 5.11 shows an egg-yolk model for the Alps using a Bilaplacian smoother (see the work of Ohtake et al. [33] for details and other common smoothing methods). To integrate smoothing into the model, a method should be used that can take the context into account (unlike the Bilaplacian demonstrated).

![Figure 5.11: The core of the Alps smoothed using a Bilaplacian smoother (step size $\lambda = 0.3$).](image)
Chapter 6

Conclusions

In this thesis, we discussed vernacular regions. First, we analyzed desirable properties and existing models. We concluded that the existing models did not meet our criteria and a new model was developed based on connectivity. One of the main aspects of this model is its natural integration with context information. However, experimental results show that even without context information, the model is capable of finding a proper shape for a vernacular region.

We have also discussed the computation of the new model. When no context is used, we have seen that the defined core of the region is a subset of the Delaunay triangulation. This allowed us to develop an $O(n \log n)$ time algorithm, based on the algorithm by Klein [28], for the special case that we are interested only in the outline of the core. If we wish to detect holes in the region, the algorithm described by Cabello [4] can be used to compute the core in $O(n^{4/3} \log n)$ time.

With the inclusion of context, the Delaunay triangulation is no longer a superset of the core; in a worst-case scenario, the core is even a complete graph. The context information we used was a set of line segments. Each line segment was considered to be a soft obstacle, allowing crossings at additional cost. The introduction of soft obstacles caused the computation of the shortest path between two input samples to be non-trivial. We developed a so-called layered graph to be used with a shortest path algorithm. This graph has $O(n \cdot k_{\max} + m \cdot k_{\max})$ vertices and $O(n \cdot m \cdot k_{\max} + m^2 \cdot k_{\max})$ edges, where $n$ is the number of input samples, $m$ is the number of obstacles and $k_{\max}$ is the maximal number of obstacles on a straight line between any two samples. Using Dijkstra’s algorithm, the execution time is $O(n^2 \cdot m \cdot k_{\max} + n \cdot m^2 \cdot k_{\max})$.

**Future work.** It may be possible to improve the presented model. As indicated, the model has a lot of parameters. While this offers flexibility, it also makes finding a good result more difficult. Automatic parameterizations may help. This is mainly of interest for the crossing cost and the parameters of the outlier filter and hole detection.

Smoothing the resulting core gives more intuitive results. However, an integrated smoothing method should be able to take context into account. Such an obstacle-constrained smoothing method may greatly improve the (visual) quality of a vernacular region.

The new model described in this paper produces a rather constrained egg-yolk model, since it is just an offset of the (polygonal) core. An egg-yolk model has more expressive power than the new model uses. It may be possible to adapt the model to fully exploit the potentials of an egg-yolk model. For example, this could be done by having the offset distance depend on the size of the adjacent face: the bigger the face, the more uncertainty there is. Another
possibility is to use the density to locally define an offset distance.

Other ways to improve the model are the support for weighted point sets and other types of geographical context, such as terrain type and elevation. Modeling can be done by modifying the distance. It is particularly interesting to look at how these additions influence the computation and performance of the model.

We have introduced a distance measure for the new model based on soft obstacles. It may also be possible to use this distance with Kernel Density Estimation instead of the Euclidean distance. While the definition requires just a slight modification, exact computation is less straightforward. However, it may provide interesting results.

When context is included, some algorithmic aspects have the potential to be improved. First of all, we conjectured the complexity of the outline to be $O(|\mathcal{S}| + |\mathcal{O}|)$ (see Conjecture 1 in Section 4.2). This could allow for faster algorithms when we are interested only in the outline of the core, similar to the approach we used when we did not consider context in Chapter 3.

The main bottleneck in the computation is the algorithm to calculate the core. There are two options to improve the current approach. The first option is to reduce the size of the obstacle graph used to compute the actual distances. The other option is to reduce the number of pairs we need to check. As we have shown, this is quadratic in the worst-case scenario. However, for “realistic” inputs, the problems we described do not seem to occur. The Delaunay triangulation still is a decent first estimate, but it may need some additional connections near obstacles. However, the number of additional edges is far from quadratic.

Another interesting algorithmic problem is how to offset the core when we use soft obstacles. In the experiments, we used hard obstacles to create an egg-yolk model. However, soft obstacles may improve results when the region is not (strongly) influenced by the context information provided.

Acknowledgements

I would like to thank Ross Purves from the Universität Zürich for providing feedback on the new model as well as the data sets used during the experimental study. Also, I would like to thank Philip Klein from the Brown University for providing additional details on his work and ideas for the adaptation presented in Section 3.3.2. Furthermore, I would like to thank my supervisor, Mark de Berg, for his guidance and feedback.


Appendix A

Experiments

Table A.1 lists the parameterizations for each of the experiments presented in Section 5.2. The parameters are described in Section 5.1 and are listed below.

1. **Euclidean power** \( t \geq 1 \).
2. **Crossing cost** \( \kappa \geq 0 \).
3. **Outlier filter** \( \tau > 1 \).
4. **Vertex cluster** \( \vartheta \geq 1 \).
5. **Hole filter** \( \tau_h > 1 \).
6. **Face cluster** \( \vartheta_h \geq 1 \).
7. **Offset distance** \( \delta \geq 0 \).

To negate the effect of scaling, parameter \( \delta \) is given as a fraction of the diameter of the context. The crossing cost \( \kappa \) is given as a fraction of the diameter raised to the power \( t \). For the parameter \( \vartheta \), we also indicate its value as a percentage of the total number of samples. We note that for the data set of the Irish coast, hole filtering is used, but the face cluster parameter is omitted. This means that hole filtering is applied, but no connectivity checks are performed and faces are not merged.

Table A.2 shows the parameterizations used to obtain the egg-yolk model from Kernel Density Estimation. Again, to negate the effect of scaling, \( r \) is given as a fraction of the diameter. The parameters \( \alpha \) and \( \beta \) are given as a fraction of the maximal certainty, \( \max_{x \in \mathbb{R}^2} \{C(x)\} \).
Table A.1: The parameterizations for each of the experiments using the new model. Blank entries indicate unused parameters.

<table>
<thead>
<tr>
<th>Region</th>
<th>Output</th>
<th>Parameterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graubünden</td>
<td>Without context</td>
<td>$t = 2, \kappa = 10, \tau = 10 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.500, \tau = 10 %$</td>
</tr>
<tr>
<td>Valais</td>
<td>Without context</td>
<td>$t = 2, \kappa = 10, \tau = 0.5 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.500, \tau = 25 %$</td>
</tr>
<tr>
<td>South Holland</td>
<td>Without context</td>
<td>$t = 2, \kappa = 45, \tau = 3.9 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.500, \tau = 15 %$</td>
</tr>
<tr>
<td>Southern Limburg</td>
<td>Without context</td>
<td>$t = 2, \kappa = 20, \tau = 3.9 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.500, \tau = 10 %$</td>
</tr>
<tr>
<td>The Alps</td>
<td>Without context</td>
<td>$t = 2, \kappa = 10, \tau = 0.3 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.100, \tau = 20 %$</td>
</tr>
<tr>
<td>Parisian suburbs</td>
<td>Without context</td>
<td>$t = 2, \kappa = 10, \tau = 6.0 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.250, \tau = 10 %$</td>
</tr>
<tr>
<td>Paris manual</td>
<td>Without context</td>
<td>$t = 2, \kappa = 6, \tau = 3.2 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.025, \tau = 20 %$</td>
</tr>
<tr>
<td>Irish coast</td>
<td>Without context</td>
<td>$t = 2, \kappa = 10, \tau = 0.4 %$</td>
</tr>
<tr>
<td></td>
<td>With context</td>
<td>$t = 2, \kappa = 0.500, \tau = 700 %$</td>
</tr>
</tbody>
</table>

Table A.2: The parameterizations for the experiments using Kernel Density Estimation.

<table>
<thead>
<tr>
<th>Region</th>
<th>Parameterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graubünden</td>
<td>$r = 0.060, \alpha = 0.25, \beta = 0.60$</td>
</tr>
<tr>
<td>Valais</td>
<td>$r = 0.060, \alpha = 0.25, \beta = 0.60$</td>
</tr>
<tr>
<td>South Holland</td>
<td>$r = 0.060, \alpha = 0.25, \beta = 0.60$</td>
</tr>
<tr>
<td>Southern Limburg</td>
<td>$r = 0.030, \alpha = 0.25, \beta = 0.60$</td>
</tr>
<tr>
<td>The Alps</td>
<td>$r = 0.070, \alpha = 0.25, \beta = 0.60$</td>
</tr>
<tr>
<td>Parisian suburbs</td>
<td>$r = 0.050, \alpha = 0.25, \beta = 0.60$</td>
</tr>
<tr>
<td>Paris manual</td>
<td>$r = 0.060, \alpha = 0.25, \beta = 0.60$</td>
</tr>
<tr>
<td>Irish coast</td>
<td>$r = 0.0027, \alpha = 0.25, \beta = 0.60$</td>
</tr>
</tbody>
</table>