

Geometric Algorithms for Delineating Geographic Regions

Geometrische Algoritmen voor het
Afbakenen van Geografische Gebieden
(met een samenvatting in het Nederlands)

Proefschrift ter verkrijging van de graad van doctor aan de Universiteit Utrecht
op gezag van de rector magnificus, prof. dr. W.H. Gispen, ingevolge het besluit
van het college voor promoties in het openbaar te verdedigen op dinsdag
20 juni 2006 des middags te 2.30 uur
door

Iris Reinbacher

geboren op 28 februari 1975 te Knittelfeld, Oostenrijk

promotor: Prof. Dr. Mark H. Overmars
co-promotor: Dr. Marc J. van Kreveld

CONTENTS

1	Introduction	1
1.1	Geographic Information Systems and Computational Geometry . . .	2
1.1.1	Computational Geometry in GIS	4
1.2	Inaccuracy, Vagueness, and Imprecision	8
1.3	Geographic Information Retrieval	11
1.3.1	The SPIRIT Web search engine	12
1.4	Thesis Outline	15
2	Finding the North, East, West, and South of a Region	19
2.1	Criteria for a Good NEWS Partition	20
2.2	NEWS Partitions with Arbitrary Regions	24
2.3	Simply-connected NEWS Partitions	28
2.3.1	Computing a fair partitioning	29
2.3.2	Computing a maxmin and minmax area partitioning	35
2.4	Experiments	36
2.5	Extensions	40
2.6	Concluding Remarks	40
3	Delineating Imprecise Regions using Locations	41
3.1	Approaches to Delineate Imprecise Regions	43
3.1.1	Adaptation approach	43
3.1.2	Recoloring approach	45
3.2	Adaptation Approach in Detail	45
3.2.1	One blue point inside P	46
3.2.2	Several blue points inside P	48
3.2.3	Minimizing absolute angular change	55
3.3	Recoloring Approach in Detail	55
3.3.1	The angle-and-degree scheme	60
3.4	Experiments	60
3.5	Concluding Remarks	65

4	Computing Multiscale Gradient and Aspect Maps	67
4.1	Definitions for Local Slope	69
4.1.1	Uniform weighing over a neighborhood	70
4.1.2	Non-uniform weighing over a neighborhood	72
4.1.3	Maximum value in a neighborhood	73
4.1.4	Uniform weighing over a diameter of the neighborhood	73
4.2	Algorithms for Local Slope and Isolines	74
4.2.1	Uniform weighing method	76
4.2.2	Non-uniform weighing method	78
4.2.3	Maximum value method	79
4.2.4	Uniform weighing over diameter method	80
4.3	Experimental Results for Grid Data	81
4.3.1	Comparison of different methods	92
4.3.2	Comparison of different radii	93
4.4	Concluding Remarks	93
5	Scattered Relevance Ranking for Geographic IR	95
5.1	Basic Scattered Ranking Methods and Algorithms	97
5.1.1	Distance to query and angle to ranked	99
5.1.2	Distance to query and distance to ranked	99
5.1.3	Basic algorithms	99
5.2	Other Scattered Ranking Methods and Algorithms	101
5.2.1	Addition methods	101
5.2.2	The wavefront approach	102
5.2.3	Efficient algorithms for addition and wavefront model	103
5.3	Extensions of the Basic Ranking Methods	107
5.3.1	Staircase enforcement	108
5.3.2	Limited windows	109
5.3.3	Higher dimensions	111
5.4	Experiments	112
5.4.1	Basic ranking algorithms	112
5.4.2	Staircase enforced ranking algorithms	113
5.4.3	Ranking algorithms with limited windows	113
5.5	Concluding Remarks	117
6	Conclusions and Future Work	119
	Bibliography	132
	Samenvatting	133
	Acknowledgements	137
	Curriculum Vitae	139

1

INTRODUCTION

Everyone of us naturally uses geographical concepts in her or his daily conversation, for example: ‘I come from a little town in the middle of Austria. Now I live in the Netherlands, in an area called the Randstad. I also work there; my office is in the eastern part of Utrecht. My latest hobby is Aikido, I train either in a dojo in my neighborhood or in Amsterdam, which is north of Utrecht.’

Some of the geographical regions mentioned above have official, crisp boundaries, like Austria, the Netherlands, Utrecht (as a municipality), and Amsterdam. Usually, people can agree on these, although most of them would not be able to trace them exactly. Other geographical regions are used more loosely: they have no real boundaries that can be traced in any way, but people share a common idea about their location and extent. When people are asked to point out the middle of Austria, the Randstad, or the area west of Utrecht on a map, they would probably give similar boundaries to these regions, provided they are familiar with the concept of the Randstad, for example. Yet other regions have an extent that depends on each person individually. For example, my idea of the neighborhood I live in may be completely different from somebody else’s who lives in the same apartment building. Our different personal experiences and routines result in different boundaries of our neighborhood.

Geographical regions with imprecise boundaries have received considerable attention from researchers in linguistics as well as from researchers in geographic information science. Questions like ‘how many trees amount to a forest’, ‘where is the border between the mountain and the valley’, or ‘what is a hill’, stand at one end, algebraic frameworks to determine spatial relationships between imprecise regions mark the other end of the research spectrum. Giving crisp boundaries for imprecise regions is an important issue in geographic information systems (GIS) and geographic information retrieval (GIR). For example, a typical question to be answered by a GIS is ‘what is the average income of highly educated people living in the Randstad?’ Levels of education and income can be derived from census data; however, without crisp boundaries for the Randstad, this question cannot be answered. Another typical query involving imprecise regions is for example to search the Web for ‘3-star hotels in the north of Amsterdam’. Again, 3-star hotels are easy to determine, whereas not many websites will state that the hotel lies in the north of Amsterdam. These examples show that there is a need to derive reasonable crisp boundaries for imprecise regions.

This thesis introduces a number of geometric methods to derive crisp boundaries for imprecise geographical regions. These crisp boundaries should be ‘reasonable’ boundaries for regions that do not have any, like the middle of Austria, southern England or the Dutch Randstad. As an example of where the implementation of crisp boundaries is not possible, we name coastlines with a beach. Defining a crisp boundary between land and sea does not make sense, due to the varying tidal influence. We present the first geometric, algorithmic approach to the problem of delineating imprecise geographical regions. Furthermore, we present an—also geometric—approach to the ranking of documents with textual and spatial score, which can be used in geographic information retrieval. Advantages and drawbacks of this approach will be discussed in the corresponding chapters.

This chapter is structured as follows. We will give a brief introduction to geographic information systems and computational geometry in Section 1.1, together with an overview of applications of computational geometry in GIS. Section 1.2 introduces the notion of inaccuracy, vagueness, and imprecision of geographical data, and summarizes related research concerning uncertainty in GIS and computational geometry. Section 1.3 gives an overview of geographic information retrieval on the Web and presents the SPIRIT Web search engine. Most of the research presented in this thesis has been carried out within the SPIRIT project [94, 95]. We shall conclude this chapter with an overview of the contents and results of this thesis.

1.1 Geographic Information Systems and Computational Geometry

Geographic information systems (GIS) and computational geometry both are relatively young fields of science, which emerged in the mid sixties and seventies of the last century. This section gives an introduction to geographic information systems (GIS) and computational geometry. In Subsection 1.1.1 we will list the tasks performed by a GIS for which techniques from computational geometry are successfully applied.

A *geographic information system* (GIS) “can be seen as a system of hardware, software and procedures designed to support the capture, management, analysis, modeling and display of spatially referenced data for solving complex planning and management problems” [78], or in short, it is “a set of computer based systems for managing geographic data and using these data to solve spatial problems” [104].

Geographic information systems as they are known today emerged from a number of different sources dating from the nineteen-sixties. The Canadian geographic information system (CGIS) addressed the need for land and resource information for addressing transportation problems and computer mapping. The Oxford System for high quality digital cartography was a GIS focusing on automated map generation. The need to automate the access to and analysis of census data was another driving force in the development of geographic information systems. Nowadays, all these different sources are integrated into one system, the main tasks of which are the following:

- data acquisition and input
- data storage
- data manipulation
- data analysis
- data visualization

We will only briefly introduce these points here. A deeper discussion will follow in Subsection 1.1.1. There are several ways of *data acquisition and input*. New data can be collected directly in the field by an individual, or by using remote sensing methods. Existing paper maps can be digitized, or aerial pictures can be interpreted to derive for example vegetation data. This is referred to as photogrammetry. Furthermore, also data collected in the last census is GIS data. *Data storage* is an important task of a GIS. Modern data acquisition techniques provide a huge amount of data which needs to be stored in well-accessible structures. There is a main distinction between vector and raster data, and *data manipulation* is for example concerned with the conversion between these two types of data. *Data analysis* is often considered the most important task of a GIS, and it determines properties that are not represented explicitly in the data. Finally, the creation of meaningful output in the form of maps and diagrams is the task of *data visualization*.

Nowadays, GIS are used in various ways and have different target groups. Geographic information systems are currently used by governments, the military, business enterprises, researchers in geography, or any individuals. The uses include telecommunications, transportation planning and navigation, urban planning including market research and facility management, emergency management, land administration, environmental monitoring and assessment, health care, and many more. The different tasks performed by a GIS are discussed in more detail in the next section.

Introductory books to GIS include [27, 106]. An introduction to the algorithmic techniques applied in a GIS can be found in [165, 174].

Computational geometry is a discipline of computer science that has developed in the early nineteen-seventies as a subarea of algorithms research. It was motivated by application areas such as computer graphics and vision, where algorithms for spatial data were needed, and reached its status as an independent field of research in the mid eighties. Computational geometry aims to develop efficient algorithms and data structures for solving geometric problems. Since the emphasis lies on discrete mathematical problems involving mainly basic objects like points, line segments, polygons, polyhedra, etc., it is possible to focus on the combinatorial properties of a problem rather than dealing with numerical issues. Another goal of computational geometry is to provide basic software tools for application areas. The development of the libraries CGAL and LEDA [33, 101] is a step in this direction.

Typical questions solved by computational geometry are for example:

- What is the minimum area convex polygon enclosing a given set of points?
- Which pairs of a given set of line segments intersect?

- Are there more than two points of a given set that lie on a line?
- Does a given point lie inside a given polygon?
- Given a point, which is the nearest of a given set of locations?

In the design of algorithms, efficiency is measured in terms of time and memory use. The time and storage requirement for an algorithm depends on the computer and platform it is running on, as well as on the implementation. Therefore, the time and storage requirements for an algorithm are usually described asymptotically using O -('big-oh') notation. If an algorithm runs in $O(f(n))$ time on an input of n objects, this means that two constants, N and c , exist such that for any input size $n > N$, the running time of the algorithm is at most $c \cdot f(n)$ elementary operations (such as additions, comparisons, assignments, etc.). Lower bounds $\Omega(f(n))$ and asymptotically tight bounds $\Theta(f(n))$ are similarly defined, see [42] for details.

The first book devoted to computational geometry was written by Preparata and Shamos [137]. Nowadays, there are a number of textbooks on computational geometry, some have a general, algorithmic focus [46, 124, 127], others deal with the discrete-combinatorial aspects [114], and yet others are handbooks with numerous references [79, 145].

Computational geometry has numerous applications in various fields, such as engineering, visualization, robotics, motion planning, virtual environments, computer graphics and computer vision, computer aided design and manufacturing, operations research, pattern recognition, crystallography, computational biology, combinatorial optimization, cartography, integrated circuit design and many more. Computational geometry is also widely used in geographic information systems. In the next section, we will describe the computational tasks performed by a GIS where geometric algorithms are used.

1.1.1 Computational Geometry in GIS

In [164], a geographical information system is described as a system to “facilitate the input, storage, manipulation, analysis, and visualization of geographic data”. Computational geometry has applications in almost all tasks performed by a GIS, except for data acquisition. In the following paragraphs, we will list a number of these applications for each task of a GIS. As we are not attempting to be complete, we would like to suggest the surveys of de Floriani et al. [47] and van Kreveld [164] for further reading.

Data storage In a GIS, there are two types of geographical data, *spatial data* (objects) and *non-spatial data* (attributes). Spatial data is either two dimensional *object data*, representing, for example, the spatial location of buildings, railroads, and forests, or *field data*, where every point is assigned a value, such as height, annual precipitation, or average temperature. This data can be represented by two different models as shown in Figure 1.1. In the *raster model*, the chosen area is divided into equally-sized cells, each bearing an attribute value, e.g., for land cover or land usage. The raster model is usually called *regular square grid* (RSG) and

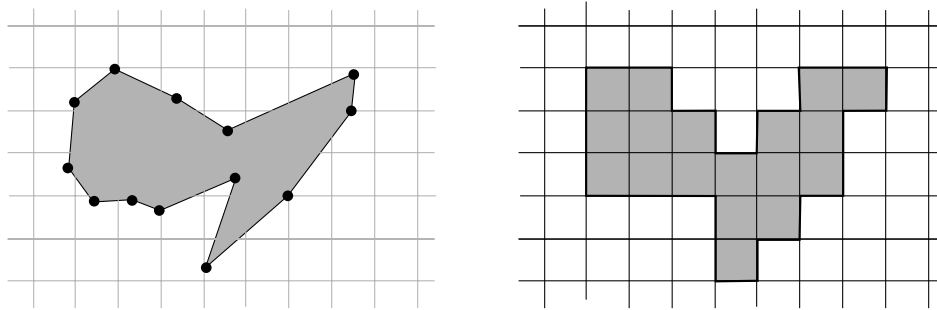


Figure 1.1: The same region in vector and raster representation.

each cell contains one value for each specified field variable. In the *vector model*, which is more commonly used in computational geometry, object data is represented by points, lines or polygons. As an example of a vector data representation, we name the well known *doubly connected edge list* (DCEL) introduced by Muller and Preparata [122].

The corresponding vector model for terrain data is the *triangulated irregular network* (TIN), composed of a number of scattered points with given height that are triangulated. Usually, the Delaunay triangulation is used to generate a TIN. However, it is possible that the Delaunay triangulation leads to undesired artifacts, like nonexistent dams in a valley for example. In this case, when there are known edges like valleys or ridge lines that should be preserved by the TIN, a constrained Delaunay triangulation is preferred [48].

The *R-tree* [84], and a number of other hierarchical data structures based on it, provides efficient access to geographical data and is probably the most widely used data structure in GIS. For terrain data, different data structures are used. We will only mention the *quadtree* [36, 156].

Data manipulation Both the raster and vector representations of spatial data have different advantages and disadvantages, [105]. The raster model has the simplicity of the structure and the operations on it as well as the direct access to information concerning a given point as an advantage. A disadvantage lies in providing only an approximate geometry that is highly dependent on the resolution of the grid. In the vector model, spatial information can be represented directly through its geometry, which leads to a high level of precision. The disadvantage of the vector model is that efficient data access and manipulation requires complicated search structures and techniques. Converting data from the raster to the vector model and vice versa is a data manipulation task in a GIS. See [47] for an overview. Data compression for dealing with the huge amounts of acquired data is also an important issue.

Data analysis Data analysis is often considered the most important task of a GIS. Here, as well, geometric algorithms are used to derive efficient solutions. Here is a brief survey of the types of problems and challenges that may arise.

Geographical data of the same region is usually organized in different *layers*, where each layer contains data with only one theme, such as road networks, land cover, hydrography, factory premises, elevation contours, etc. By overlaying these thematic layers, it is possible to find regions with the combined properties of different layers, e.g., land cover by elevation. *Map overlays* of two thematic layers are often based on red-blue line segment intersection algorithms and can be solved efficiently [35, 64, 132].

An important part of data analysis is the analysis of the neighborhood of a geographical entity. The computation of *buffer zones* around geographical features is equivalent to computing the Minkowski sum of the feature and a disk with given radius, centered at the origin [2, 96]. Map overlay and buffer computation play an important role in urban planning. For example, when planning a new railway line that must avoid urban areas while not running too close to the river, one would first compute appropriate buffer zones around the houses and the river. During the second stage, these two new layers would be overlaid, and on the resulting map, the potential area of the rail track can be determined. A related problem with applications in path planning is the *weighted region problem*. It consists of finding a cost-optimal path inside a region, where non-negative costs are assigned to each part of the region. The costs are the expenses of traveling one unit along a path inside this region, and they may be based on soil conditions, vegetation, etc. Obstacles that need to be avoided, such as nature preserves, can be modeled by assigning infinite costs. Optimal path planning inside weighted regions has applications in planning highways, railways, and pipelines, and has been studied in several papers [75, 113, 119]. Computing *visibility* is a well-studied area in GIS terrain analysis (see [126] for a survey) as well as in computational geometry [14, 128]. Visibility computations have applications such as finding the optimum placement of radio or telephone antennae such that there is reception throughout the whole terrain [19]. Visibility computations can also be used for planning *scenic paths*, where a number of vantage points are visible from the points on the path. The opposite of a scenic path is called a hidden or *smuggler's path*, which is not visible from any predefined viewpoint [126].

Computing the *drainage network* of a terrain provides information about water flow and resources, and areas that are likely to be flooded or have high risk of erosion. It is generally assumed that water flows downwards in the direction of steepest descent. It accumulates in streams and rivers that together form the drainage network of the terrain. A drainage network is a directed graph, where the arcs are directed towards the pits. Computing it is related to extracting point and linear features of the terrain [178]. Also the identification of *watersheds*, the area from which water drains into the same stream, provides information on the terrain [103]. Many other examples of geographical data analysis exist, see for example [49, 130].

Data visualization Geographical data is most often visualized on a map. A *map* is a two-dimensional representation of three-dimensional space, and for a long time maps were drawn by hand. In a GIS, maps can be precomputed and stored, but most often they are created on demand. Normally, the user chooses the data set and manner of visualization; therefore the output has to be automatically created online. In addition, geographic information systems allow the viewer to zoom in and out of a region, or to move to an adjacent region. Therefore, efficiency is crucial to



Figure 1.2: Different visualizations of the outcome of the 2004 US presidential elections, taken from [74]. Darker regions depict democrat majority. Left: A choropleth map. Right: A cartogram.

these kinds of applications.

When geographical data is processed, the *scale* of the output becomes an issue. The larger the scale of the map, the more detail can be shown. Decreasing scale makes a selection of the visualized features necessary. This is called *map generalization* [123, 172]. It includes problems of displacement, selection, aggregation, smoothing, simplification, and more.

Another well studied area in automated cartography as well as the algorithms field is *label placement* [38, 54, 159]. There are labels for points (e.g., cities), line objects (e.g., rivers), and polygonal objects (e.g., countries). Label placement is often considered to be an optimization problem with the following constraints: the labels must be legible and easy to relate to a spatial entity, they must not overlap each other, nor must they cover relevant features of the map. The placing of diagrams on maps, to show employment rates by age for example, is related to label placement [168].

Special purpose maps concentrate on showing one particular theme only and may use different ways of visualization, see Figure 1.2. *Choropleth maps* use a color scheme to represent a specified geographical variable per administrative region, such as birth rates per country. Other special purpose maps are for example *dot maps* (e.g., one dot representing 10.000 inhabitants), *schematic maps* (e.g., tram networks), *flow maps* (e.g., migration between countries), and *cartograms* [50, 55]. A cartogram shows values for regions by adjusting their area, such that it corresponds to the represented value. Naturally, a cartogram distorts geographic space. However, the regions should remain recognizable, so they should keep their shape, adjacency, and relative position as much as possible. Population cartograms are most common.

To give a terrain image a realistic appearance, *shaded relief maps* are used. They combine height and slope information of the terrain with a supposed placement of the sun in the northwest of the sky to produce shadows, providing clues to the terrain structure, and the illusion of depth.

1.2 Inaccuracy, Vagueness, and Imprecision

Geographical data as it is used in a GIS can never be absolutely correct. The reasons for this are manifold: Geographical data in a GIS is a generalized and incomplete representation of the real world. During the data acquisition and input phase, errors are not only due to the error-prone methods used, but are often inherent in the data itself. For example, remote sensing methods may be inaccurate, the decision as to whether a small number of free-standing trees should be incorporated as a forest in a GIS may be difficult, and the classification of an elevation as hill or mountain may depend on its surroundings.

Inaccuracy, vagueness, and imprecision of geographic objects has been a major field of research in the GIS community over the past years [26, 134, 180]. Note that these terms, and others like error, fuzziness, indeterminacy, uncertainty, reliability, and ambiguity, are often used interchangeably. There are different classes of fuzziness [66, 120]. In the following, we shall use a classification similar to the one in [120], and label the three classes inaccuracy, vagueness, and imprecision.

Fuzzy object description, which we will call *inaccuracy*, refers only to positional imprecision in the data points, which is due to measurement and computation techniques that are subject to error. For example, the remote sensing device may have an error margin, the (automated) interpretation of satellite images may include faults near boundaries, or the (manual) digitizing of analogue maps may be imprecise and lead to distortions of features. See for example [98, 180] for an overview of techniques for data acquisition and possible sources of inaccuracy. Besides these errors that occur during data acquisition and input, other sources of data inaccuracy are data generalization, or the conversion from one data model to another [104, 106]. Note that error propagation also plays a role [32, 90]. A bound on the measurement error of the data acquisition stage is usually given and referred to as data accuracy. It can be accounted for by either statistical methods, or by modeling the data points as disks with a fixed radius. However, the inaccuracy arising from the other sources remains difficult to formalize.

Fuzzy object definition, or *vagueness*, implies that no clear, generally accepted criteria can be formulated to determine whether an object belongs to a certain class. For example, how many trees comprise a forest? How large must a pond be to be called a lake? When does a stream become large enough to be called a river? The main question is to determine whether a certain object occurs at all in a specified region. It may also be difficult to determine geographical prepositions (across, over) and relationships (inside, next to). These linguistic terms may have no direct translation to other languages, or may even have a different meaning altogether [63, 160]. Not only language, but also perception, behavior and cognition play an important role in defining certain geographical objects, which are therefore inherently vague. Research on *spatial reasoning* aims to give crisp definitions of geographical objects, (see for example [20, 91, 108]). However, it seems to be difficult, if not impossible, to find generally accepted definitions for all geographical entities.

Fuzzy object geometry, which we will refer to as *imprecision*, deals with the spatial extent of a geographical object, such as a mountain range or a forest. In this context, we will assume that the classification of a certain area as forest is known and indisputable as such. In case of a clear break in land cover, (e.g., a lake, or in agriculture: crops vs. corn), the determination of a crisp boundary is relatively easy. However, most often there is a smooth transition between two types of land cover, for example between a forest and scrubland, which makes the delineation of the forest more difficult. In this context, determining a crisp boundary of a region that is imprecisely defined has received considerable attention over the past years [26, 102], and it is also the main topic of this thesis. Imprecision in boundaries is obviously intertwined with inaccuracy and vagueness, as the definition of an object as well as the position of the data points contained in the object are crucial for the determination of a crisp boundary.

To contrast and clarify these classifications, we will provide the following example: Assume we are given an area with a number of trees. *Inaccuracy* means that our knowledge of where each tree is precisely situated may be wrong. *Vagueness* implies that we do not know whether the area should be called a forest or not. Finally, provided that we have agreed upon calling our aggregation of trees a forest, *imprecision* means that we cannot be sure of its boundaries distinguishing it from its surroundings.

In this thesis, we will assume that the data points are accurate, and we will introduce several algorithmic methods to determine crisp boundaries for various types of imprecise regions. As stated above, crisp boundaries are necessary in GIS for data analysis purposes that use membership relations like inside or outside.

There are different approaches for dealing with the imprecision of a region. One method, for example, is to use bands around the region [138]. The boundary of the region may lie anywhere inside this band, which can be seen as a buffer zone with a fixed width around a point, a linear feature or a region. A different approach is to use bands with variable widths around a region. An example of this approach is given by Cohn and Gotts in [41]. They show the ‘egg-yolk’ representation of an imprecise region, where the region is defined by two concentric subregions, the inner ‘yolk’ and the outer ‘white’, both indicating different degrees of membership. A similar approach to delineate imprecise regions is the Realm/ROSE approach by Schneider [149]. Alani et al. [5] use a gazetteer to extract locations that are classified as inside or outside of a certain region. A Voronoi diagram of all locations is computed, and the edges of the Voronoi diagram that have adjacent cells with differently classified locations compose the boundary of the region. Purves et al. [40, 140] use the Web as a way to find locations inside an imprecise region. By applying a weight based on the document density, the extent of the imprecise region is defined as a density surface. A similar approach using the same methods to find locations is presented in Chapter 3.

Currently, GIS research focuses on delineating imprecise regions by applying *fuzzy set theory* [68, 102, 179]. Instead of assigning only the two boolean values 0 and 1 for membership, fuzzy set theory allows the assignment of membership values that cover the whole interval $[0, 1]$. The lower the assigned value, the less the described

object equals the definition of the set. Often, the classifications by fuzzy sets overlap; for example a patch of land at the edge of a forest may be classified as 40% forest and 60% scrubland. This makes it possible to define boundaries that enclose regions which have been classified as at least 50% forest, for example.

In the context of fuzzy geographical entities, computing the area, perimeter, or shape has recently received attention [67, 68], as well as the influence of scale on the representation of a fuzzy object [37, 65].

Despite the fact that imprecise regions have no crisp boundaries, it is still possible to determine spatial relations between them. For example, it is clear that the Dutch Randstad is not situated in the north of the Netherlands, and that the Vienna basin is northeast of the center of Austria. A classification into eight possible *topological relations* of two simple regions is: ‘disjoint, contains, inside, equals, meets, covers, covered by, and overlaps’ [60]. These relations can also be represented by a 3 by 3 matrix, where the rows and columns represent the features interior, boundary, and exterior of each region. These topological relations can be extended to imprecise regions that have an inner and an outer boundary [39], such as the egg-yolk model described above [41], or to imprecise regions modeled as fuzzy sets [81, 120].

For the modeling of cardinal directional relationships such as north of, southeast of, etc., we shall name the *cone-based directional system* [70]. There, one reference region lies at the center of a coordinate system that divides space into four (or more) cones, representing the North, East, West, and South, like on a compass rose [71]. By partitioning space in this manner, we can easily derive the cardinal direction of the reference region to any other region. Naturally, when the regions have shapes that are not (almost) convex, other systems will have to be applied [135, 147].

In computational geometry, the issue of inaccuracy has been dealt with either as computational or as data imprecision. The inaccuracy arising from error propagation, as well as from using number types with a limited number of decimals (e.g., float, double), is referred to as *computational imprecision*. See [177] for a survey. *Data inaccuracy* implies that the exact coordinates of the data points are not known. In this case, the data points are approximated by circles, squares, or convex polygons. Data inaccuracy gives rise to a number of questions, for example: how many convex hulls of a point set are possible [125, 129], or to what extent may the points be perturbed in order for the topology of the geometric structure (e.g., the Delaunay triangulation) to remain the same [1, 16, 83]. More related work on dealing with data inaccuracy with varying motivation is for example [22, 29, 45, 56, 80, 97, 112, 152].

In this thesis, we will present definitions based on geometric specifications for vague regions like the North, East, West, and South of a country. Furthermore, we will assume that the data points given are accurate, and therefore we will deal only with imprecision, i.e., fuzzy object geometries.

1.3 Geographic Information Retrieval

Traditional information retrieval is concerned with searching for whole documents or parts of documents based on their content, or searching within databases, whether stand-alone, or hypertext-networked such as the Internet, for text, sound, or images. *Geographic information retrieval* (GIR) includes all these areas with the restriction to spatially and geographically oriented indexing and retrieval that provides access to georeferenced sources of information. In the following, we will focus on GIR for documents on the Internet, although all the explanations apply to other fields of GIR as well.

Geospatial search engines allow the specification of both geographical location and textual terms in the query. The retrieved documents should be relevant with respect to the textual terms and the distance to the geographical location. The most common geographical queries are *point-in-polygon* (e.g., in which country lies Utrecht), *region queries* (e.g., Aikido dojos in Amsterdam), *distance and buffer zone queries* (e.g., swimming pools maximal 10 km outside Utrecht), and *path queries* (e.g., shortest route from Utrecht to Amsterdam). To retrieve a document that is relevant to a specific geographical location, it has to be georeferenced, i.e., its geographical context needs to be made accessible to a search engine.

Geotagging (also referred to as *georeferencing* or *geocoding*) web documents automatically has been a research topic for several years, see e.g., [6, 110, 171]. For any web page, it is possible to process the admin-c section of the `whois` entry or the IP-address of the computer hosting the page. However, often the administrator of a web page is not the same person as its owner, and nowadays it is possible to host web pages far from the places for which they are relevant, so these approaches lead to unsatisfactory results of the search. Extracting ZIP-codes from an address or area codes of phone numbers given on a web page and using business directories is more promising. It can also be useful to consider the link structure of the page to determine whether a page is of local or global interest (called the locality of the page), for example, compare a page of a local Aikido dojo to the global Aikido-FAQ [93].

It is also possible to process the metadata of a web page and to retrieve the geographic scope of the document from it. However, as current search engines do not use the meta tags of a html document for its ranking, there is only sparse geographical metadata available, and as long as this data is not employed, users will see no reason to include any. Furthermore, it is an unresolved issue whether the user can be trusted to provide accurate data in a meta tag rather than giving fraudulent information to reach a higher position in the ranking, and therefore increase the number of hits on her or his site.

Another way to automatically georeference a document is to extract the names of geographical entities (cities, regions, countries) from the document, and assign a spatial footprint to each of them using a gazetteer. A spatial footprint consists of the location, i.e., the coordinates, and the spatial extent of a named place. It is important to find the geographical location the document focuses on, and to eliminate non-relevant footprints, to achieve a better performance of the search.

When a query is sent to a GIR, the georeferenced documents are assigned a

textual and a spatial score, which represent the relevance of the document with respect to the given query. Assigning a textual score to a document is a standard procedure in information retrieval. First, the document collection is preprocessed to generate a term index from the full text. One of numerous methods to assign a score to a text document is the *vector space model* [15]. It consists of a high-dimensional space, one dimension for each term in the index. The query and all documents are represented by high-dimensional vectors with a non-zero entry for every term that appears in the query. The textual score of a document with respect to the query is the angular distance between their vectors. More details on text and information retrieval can be found in [15, 115].

Defining appropriate *spatial scores* for a document is a problem that differs from georeferencing. An approach with spatial indexing similar to the textual scoring described above is presented in [111]. Different ways of building a spatio-textual index have been proposed in [163]. Spatial scores are mainly computed with respect to the (Euclidean) distance of a location to the query, where increasing distance leads to a decreasing score [146]. Another method is to compute spatial relationships between regions, such as overlap, intersection, equality, containment, and directional. The spatial score for overlap then depends on the amount of overlap between two regions [100].

Once relevant documents have been identified, they have to be *ranked* according to their textual and spatial score. The balance between these two aspects is crucial and depends on the user's preference and the locality of the query. *Locality* is a measure of the sphere of influence of an entity in geographical space. For example, people may look for a Chinese takeout in their immediate neighborhood only, whereas they may be willing to drive longer distances to find a good offer for a new car.

There are a number of geospatial web search engines using different methods for georeferencing and ranking, for example GIPSY [176], GeoVSM [28], and BUSTER [170]. Most of the research in this thesis has been carried out within the framework of the SPIRIT project [94, 95], and the SPIRIT demo [158] uses the ranking procedures developed here. We will introduce the SPIRIT web search engine in the next section.

1.3.1 The SPIRIT Web search engine

SPIRIT is an acronym for '*SP*atially aware *I*nformation *R*etrieval on the *I*nterne*T*', and the goal of the SPIRIT project was to develop an internet search engine that is able to accomplish web searches for e.g., 'open air museums near Berlin', 'Aikido dojos in the east of Amsterdam', or 'Starbucks close to the Guggenheim museum'. All these queries are composed as '*something* spatially related to *somewhere*', where *something* is 'open air museums', 'Aikido dojos', and 'Starbucks', the spatial relations are 'near', 'in the east of', and 'close to', and the *somewhere* is 'Berlin', 'Amsterdam', and 'the Guggenheim museum'.

Conventional web search engines like Google, Yahoo, and others, are not capable of exploiting the spatial context of these queries, and with the common text-based retrieval, only web pages that contain all or most of the query terms will be found,

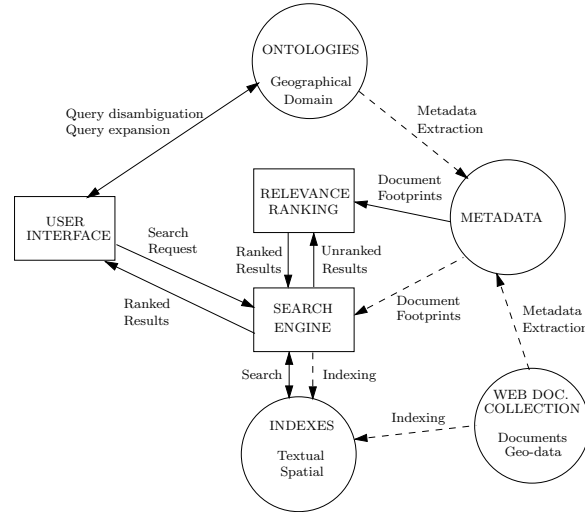


Figure 1.3: SPIRIT architecture, taken from [94]. Dashed arrows indicate preprocessing, solid arrows the data flow to process a query.

regardless of their relevance. Recent approaches to incorporate spatial knowledge in these search engines are, for example, based on the use of phone numbers or ZIP codes occurring on the web pages, and business directories to aid in determining the spatial context. However, this information may not be present on each relevant page, or it may not be sufficient to determine the appropriate location. For example, a ZIP code in the Netherlands narrows the location of an address down to a few houses in a certain street, whereas the ZIP codes in Austria can include a whole town or city district.

The demo of the SPIRIT Web search engine focuses on the tourism domain. Furthermore, the areas where geographical data has been incorporated are restricted to the United Kingdom, Switzerland, France, and Germany. SPIRIT consists of a number of components to retrieve relevant documents for a user's query. We will first provide a brief overview of the components necessary for preprocessing the data, and then see how they work together in answering a given query. More details can be found in [94]. The parts of the Web engine involved in preprocessing are:

- A geographical ontology and an ontology of the tourism-domain.
- A collection of web documents.
- A set of metadata derived from the web collection.
- A textual and a spatial index.

Further parts of SPIRIT are the user interface, the core search engine, and the relevance ranking component. These parts are only used online to process a query.

One major component of the SPIRIT search engine is the geographical and domain-oriented ontology, focussing on tourism [72]. It can be seen as a representation of the

semantics of terms and their relationship, providing a model of the terminology and the structure of geographical space, as well as a terminology of the tourism domain. This means that for each town, the geo-ontology contains not only its geographical location, but also its bounding box and the hierarchy it lies in. For example, a hierarchy of Cardiff is: Europe—Great Britain—Wales—southern Wales—Cardiff. Furthermore, the bounding box of Cardiff is stored as well as a note: Cardiff—capital of Wales. Other possible entries are Cardydd, the Welsh name of Cardiff. For the tourism domain, the ontology contains terms that are semantically related, for example: hotel, motel, bed and breakfast, and guest house are types of accommodation. The ontology plays an important role in all other parts of SPIRIT; it is used to extract the metadata from, and aids in spatial indexing of, the web collection for the preprocessing. In the online handling of a user query, the ontology is used for the disambiguation and expansion of the query (see below).

The SPIRIT demo uses a one Terabyte collection of web documents originating from a web crawl performed in autumn 2002, which amounts to 94 million web pages. Each web page that contains place names is associated with one or more spatial footprints, which are part of the metadata, and are derived using the ontology. Other parts of the metadata are extracted from the meta tags of the web page and contain name, spatial extent, keywords, contact and resolution [87].

There are two different indexes used in SPIRIT: a pure text-based index, as it is commonly used in text retrieval, and a spatio-textual index. Each term in the text index is associated with a list of documents containing the term. By only using this index, we get a similar text-only functionality as in a conventional search engine. The spatial footprints are used to create the spatial index, which is combined with the text index of the documents to form the spatio-textual index [163]. Spatial footprints are not only useful for the retrieval of relevant documents, but are also employed in distance computations to give a better ranking.

When a query like the ones above is entered into the demo of the SPIRIT Web search engine, SPIRIT performs the following tasks to return a set of relevant pages. SPIRIT, as depicted in Figure 1.3,

1. disambiguates and expands the query,
2. searches for the relevant web pages in the spatio-textual index,
3. ranks the retrieved results according to their relevance using the metadata,
4. and returns the list of ranked results.

When a user submits the query ‘hotels near Cardiff, UK’, it is first passed to the ontology for disambiguation—if the user chooses to—and expansion. Automated query disambiguation is based on population numbers and returns Cardiff, Wales in Great Britain for the geographic place name and a query footprint that contains the surroundings of Cardiff. Query expansion returns a list of possible other terms for accommodation such as motel, guest house, etc. This way the query becomes something like: ‘hotel or accommodation or motel or guest house inside the footprint of Cardiff and its surroundings’.

This expanded and disambiguated query is then sent to the search engine, which retrieves relevant documents using the spatio-textual index. The list of relevant

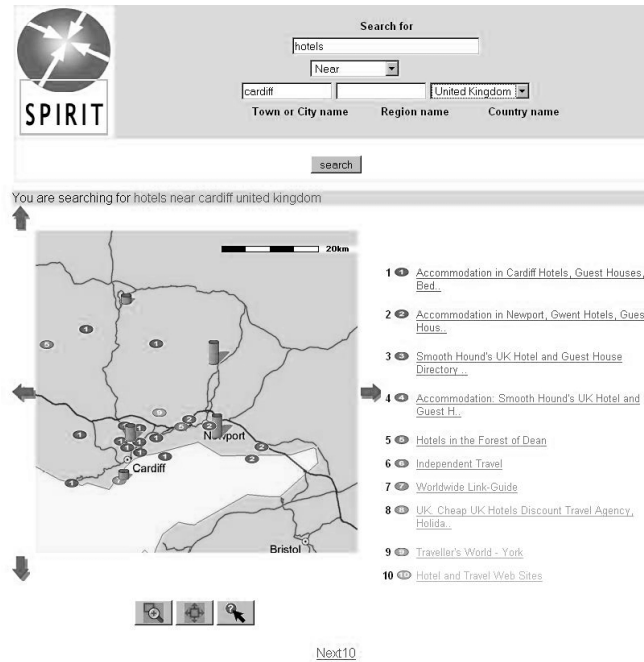


Figure 1.4: The top 10 SPIRIT results for the query ‘hotels near Cardiff, UK’.

documents is passed to the relevance-ranking component, where they are ranked according to their spatial and textual relevance. Details of how the ranking is done are described in Chapter 5 of this thesis. Finally, a ranking of the retrieved documents is returned to the user interface and displayed as a ranked list as well as a clickable map of the surroundings of Cardiff. See Figure 1.4 for the result of this search.

1.4 Thesis Outline

The contribution of this thesis to the field of imprecise regions in geographic information systems lies in introducing algorithmic methods to delineate imprecise geographic regions. Once we have found crisp boundaries for such regions, they can be used, for example, in geographic information retrieval to aid visualization and retrieval. Finding all campgrounds in the British Midlands for example, is only possible if the extent of the region is known. In the next three chapters we will introduce methods to delineate boundaries of various types of imprecise regions. In Chapter 5 we will present algorithmic methods to rank documents according to a textual and a spatial score.

In Chapter 2, we will look at the boundaries of North, East, West, and South (NEWS) of a country, a city, or any other region with given boundaries. As the partitioning of a region into its NEWS parts is often vague, we want to give crisp

boundaries between North, East, West, and South. This is useful in answering queries like ‘Aikido dojos in the east of Amsterdam’, where the term ‘in the east of Amsterdam’ is unlikely to be present on the web page. The partitioning of Amsterdam into its NEWS regions can be precomputed and stored to aid in finding all relevant documents.

We will provide three simple suggestions for partitioning a country outline into NEWS regions. The first suggestion is to divide the country by using two lines with slope $+1$ and -1 such that their intersection lies at the center of gravity of the country. It is easy to see that this will lead to regions of different sizes. The second suggestion is to partition the country by cutting off pieces of 25% of the country’s area by using horizontal and vertical lines.

The third suggestion—we will call it good NEWS—is a partitioning into four equal-sized parts, such that the sum of the distances of the four centers of gravity of the NEWS regions to the center of the whole polygon is maximized. This good NEWS partitioning has a simple shape and is unique, as shown in Section 2.2. However, it can make sense to consider only NEWS regions that are simply connected. For this, it is necessary to relax some of the criteria, such as the 25% share of each region. We will show in Section 2.3, that if a partitioning into four simply-connected NEWS regions with each having 25% of the total area exists, we can compute it efficiently. Otherwise, we can compute a simply-connected partitioning that maximizes the smallest region (or minimizes the largest) in cubic time. We have implemented all three basic suggestions for a NEWS partitioning and compare the outcomes in Section 2.4, using the outlines of ten European countries. The results of this chapter have previously appeared in [166].

In Chapter 3, we will delineate regions whose names are commonly used, but whose boundaries are known only approximately, like the British Midlands, the Dutch Randstad, or the Bible Belt in the United States. As these are imprecise regions without crisp boundaries, we could ask a number of people to draw the outline of what they think are the British Midlands on a map. Naturally, everyone will have a different idea of this region, so the outlines may all have the same core, but differ at the boundaries. A majority vote on the boundaries could be used to resolve the ambiguity.

To get data on a larger scale and for more imprecise regions, we performed a web search that provided us with a list of towns that seemingly lie within the British Midlands. A list of all towns in Great Britain was taken from a gazetteer. This allowed us to delineate boundaries of imprecise regions in a fully automated manner. Further details on the data acquisition step can be found in [8].

With the data at hand, we can abstract the problem further, and obtain the following: we are given a set of red (inside) and blue (outside) points from which we want to determine a reasonable polygon that represents the boundary of the imprecise region. However, the classification of the points to be red or blue may be incorrect. We describe two main approaches to determine the polygon in Section 3.1.

For the first approach, we start by computing an initial polygon using α -shapes based on the red points only. Then we try to adapt it such that for the final polygon all red points are inside and all blue points are outside. We give different algorithmic methods and results for initial polygons with one or more red and/or blue points

inside. These results are given in Section 3.2. For the second approach, we change the classification, i.e., the color, of a point if it is mainly surrounded by points of the other color. Different recoloring strategies and algorithms are presented in Section 3.3. Both approaches have been implemented, and we will provide the results on real life data in Section 3.4. The research presented in this chapter has been published in [143] and in [8].

In Chapter 4, we shall study imprecise regions in natural terrain data, using the slope. The slope value at each point of the terrain is usually divided into gradient, i.e., the steepness of the slope, and aspect, the cardinal direction in which the slope faces. We shall determine regions on a terrain having gradient values in a certain interval, or the same aspect values. These regions on their own do not make much sense; however, gradient and aspect are the most important values used in geomorphometry, and by combining them with curvature and height values on the terrain, it is possible to delineate landforms such as hills, valleys or mountain ranges. Hence, the presented methods help obtaining crisp boundaries for these imprecise landforms.

We shall introduce four different scale-dependent definitions of *local gradient* and *local aspect* for any point on a terrain in Section 4.1. The first two definitions average the given gradient or aspect values in a neighborhood around a point, applying uniform or non-uniform weighing. For the third definition, we choose the local gradient at a point to be the maximum gradient value to any other point within the neighborhood. The local aspect is then defined by the vector pointing to the point that realizes the maximum gradient. Finally, the fourth definition is a combination of the first and the third: The local gradient and aspect at a point are defined to be the maximum average gradient or aspect, respectively, over a diameter of the neighborhood.

We give efficient algorithms to compute the local gradient and aspect values at each point of a TIN terrain for each definition in Section 4.2. An implementation of our methods for grid data has been produced, and we compare the outcome, i.e., the resulting maps of constant gradient and constant aspect, of all methods for different neighborhood sizes. Most of this chapter will appear in [144].

As stated above, the methods for the delineation of imprecise regions described in Chapters 2 to 4 can be used for geographic information retrieval, for example to aid in area or point location queries. Chapter 5 deals with another main issue of geographic information retrieval, the relevance ranking of the retrieved documents. Since every document is assigned a spatial score and a textual score, we can treat each document as a point in two-dimensional space. When the query is represented by a point at the origin, the most simple ranking of the other points is the computation of the distance to the origin. In this case, points that lie clustered together in space represent documents with similar scores, and they are also clustered together in the ranking. However, if it is assumed that documents with similar scores have also similar contents, this outcome may be unsatisfactory for the user. Therefore, more sophisticated ranking methods should be used.

In Chapter 5, we shall introduce a number of geometrical, scattered ranking

methods. The idea is to produce a ranking such that consecutive documents in the ranking have high relevance with respect to the query, but also differ sufficiently in contents from each other. We will introduce two basic methods for scattered ranking, the angle-to-ranked and the distance-to-ranked method. In both methods, we choose the point that has maximum distance (Euclidean or angular) to any already ranked point to be next in ranking. We can apply additional weighing to give a preference to either the distribution or the proximity to the query. We also present additional methods based on the basic methods as well as extensions of them with respect to how the eligible points are chosen. It is possible that we have more than two scores, for example additional scores on metadata like the link structure of the document. In this case every document is represented by a point in multi-dimensional space, and we present the extension of our models to higher dimensions, where this is possible. Efficient algorithms to compute the ranking according to the different models are given in the according sections. We present experimental results for the different ranking schemes in Section 5.4.

The two basic scattered ranking methods for textual and spatial score are used, together with a standard text only ranking, in the SPIRIT demo [158]. The research presented in this chapter has been published previously in [167].

We shall provide a brief outlook on possible future research on imprecise regions in GIS and GIR in Chapter 6.

2

FINDING THE NORTH, EAST, WEST, AND SOUTH OF A REGION

This chapter addresses the issue of dividing a country or other geographical region into four subregions by compass directions. It will allow questions about ‘northern Germany’ or ‘eastern Spain’ to be answered properly in geographic information retrieval. Note that the specified regions do not have a well-defined boundary, but are used in a loose sense. A partitioning into regions like the ones we compute is also useful in geographic user interfaces, where the partitioning of a country is shown and the user can select a region by clicking.

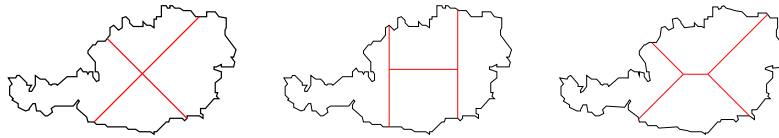


Figure 2.1: Three partitions of Austria by compass directions.

Algorithmically, this chapter describes how to determine — for a simple polygon P and a positive real A — all wedges with given apex angle and orientation that contain exactly area A of P inside. We solve both the standard version and one where the wedge is restricted to its simply-connected part inside P . The first version we solve in $O(n^2)$ time and the second version in $O(n \log n)$ time.

Related in computational geometry is research on area partitionings and continuous ham-sandwich cuts of polygons [21, 23, 24, 89]. Shermer [155] shows how to partition a simple polygon by a vertical line in two equal-area halves in linear time. Díaz and O’Rourke [51, 53] show how to compute, for a simple polygon P , a point p that maximizes $\min_{h \in H} \text{Area}(h \cap P)$, where H is the set of half-planes that contain p . They also show in [52] how to partition a convex polygon into equal-size parts using an orthogonal four-partition.

In Section 2.1 of this chapter we overview the most important criteria for a partitioning into North, East, West, and South (NEWS). This leads to three simple suggestions for good NEWS partitionings (shown in Figure 2.1). One of these (Figure 2.1, right) is a partitioning of a simple polygon P into four equal-size parts, such that the sum of the distances of the four centers of gravity (of the NEWS regions) to the center of gravity of P is maximized. Here the distances of West and

East are measured by x -coordinate only, and the distances of North and South by y -coordinate only. We show that such a partitioning has a simple shape and is unique.

In Section 2.2 we show that, for a given value A and simple polygon P , the set of all wedges with fixed apex angle and orientation that contain an area of size A of P has $\Theta(n^2)$ complexity, and we compute it optimally. We also develop an algorithm to compute the partitioning that maximizes the above-mentioned sum of distances of centers of gravity. For a simple polygon P with n vertices, it runs in $O(n \log n)$ time. The optimal partitioning, however, does not necessarily give connected regions for the North, East, West, and South. On the other hand, the algorithm also works for sets of polygons, like countries with islands. The efficiency remains $O(n \log n)$.

In Section 2.3 we discuss simply-connected partitionings into NEWS regions. We have to relax some of the criteria to obtain a problem statement that is reasonable, like the property that all four regions contain exactly 25% of the total area of P . We show that a partitioning like the right one in Figure 2.1 into four equal-size, simply connected regions can be computed in $O(n \log n)$ time, if one exists. This partitioning may be different from the one of Section 2.2. We also show that the simply-connected partitioning that minimizes the area of the largest region, or maximizes the area of the smallest region, can be computed in $O(n^3)$ time.

In Section 2.4 we show the output of our algorithm and compare it visually to the other two suggestions for partitionings done in Section 2.1, using ten country outlines. Section 2.5 describes three possible extensions.

2.1 Criteria for a Good NEWS Partition

There are many criteria one can use to partition a country into four regions by compass directions. Some criteria are especially relevant for the application in query answering of geographic information retrieval, whereas others apply more to geographic user interfaces. We list these criteria next:

1. The regions should be non-overlapping.
2. The union of the regions should fully cover P .
3. All regions should have the same portion of the area.
4. The relative orientation of any two points should be conserved (no point in North should be farther to the South than any point in South).
5. The regions should be simply-connected.
6. The partitioning should have a simple shape.
7. The length of the partitioning should be small.
8. The partitioning should be simply-connected.

Not all of these criteria can be satisfied at once. It appears to be difficult to choose criteria that would lead to ‘natural’ partitions for all countries. It is also unclear which criteria are conflicting or enforce each other. The fourth criterion contains the essence of the compass directions, and is therefore necessary. The first three criteria seem important too, especially for the user interface application. For the GIR application, some overlap between the regions may be desirable, because a

feature in the Northwest should be found if the query specifies either one of North and West.

We would like to develop a simple, efficient algorithm which works well for most countries. This means we either restrict ourselves to some of the criteria, or we find a solution which meets all to a certain extent. We choose to use purely geometrical definitions of the four regions. The experiments show that some of these definitions are often appropriate for a good partitioning. In any case they provide a good basis for an adaptation that is appropriate. We also note that a partitioning with some amount of overlap can be based on a partitioning with no overlap, for instance by growing the four regions. With the above criteria in mind, various algorithmic problems can be formulated to find a NEWS partition of a country. They are stated in the following three suggestions. The partitionings for these suggestions are shown in Figure 2.1.

Suggestion 1 Compute the center of gravity of the polygon and draw two lines with slope $+1$ and -1 through this point.

Suggestion 2 Use horizontal and vertical lines to iteratively cut the polygon into four regions which each cover 25% of the polygon's area.

Suggestion 3 Divide the polygon into four regular, equal-size regions such that the sum

$$\mathfrak{S} = \text{dist}_y(C_N, C_P) + \text{dist}_y(C_P, C_S) + \text{dist}_x(C_E, C_P) + \text{dist}_x(C_P, C_W)$$

is maximized. Here C_P denotes the center of gravity of polygon P , and C_N , C_E , C_W , C_S denote the centers of gravity of the North, East, West, and South region. dist_x and dist_y denote distance by x -coordinate and by y -coordinate. A closed region P is *regular*, if $\text{cl}(\text{int}(P)) = P$.

There are polygons that have an optimal partitioning where the center of gravity C_P has larger y -coordinate than the center of gravity C_N of the North region. Hence, in Suggestion 3 we have to be careful how to interpret distances since they may become negative: We interpret $\text{dist}_y(C_N, C_P)$ to mean $C_{N,y} - C_{P,y}$, the difference of the y -coordinates. In the sum \mathfrak{S} , the coordinates of C_P cancel out and we can equivalently maximize $\mathfrak{S} = \text{dist}_y(C_N, C_S) + \text{dist}_x(C_E, C_W)$.

The center of gravity can easily be computed in linear time [17], which solves the algorithmic part of Suggestion 1. For Suggestion 2, we let the x - and y -extent determine whether we split by horizontal lines first or by vertical lines first. We can apply the algorithm of Shermer [155] three times, which gives a linear time solution. In this paper, we will focus on finding a NEWS partitioning by an algorithm following the last suggestion. For now we will consider only the more general setting of partitioning the polygon into not necessarily simply connected NEWS regions (dealing with simply connected NEWS regions is the topic of Section 2.3). We will prove that a NEWS partitioning by Suggestion 3 can only lead to a partitioning with a particular shape described below.

We will denote with χ a construction made of a vertical line segment in the middle and two line segments with slope $+1$ and -1 at the top and at the bottom. Similarly, we denote with \succ the rotated shape with a horizontal line segment as middle part. We call the nodes where the three segments meet the *focal points*. In

a \succ partitioning, we have a West and an East focal point. In a χ partitioning, we have a North and a South focal point.

Lemma 1 *If an arbitrary simple polygon P is divided into four (not necessarily connected) parts, such that each part covers exactly 25% of the polygon's area and the sum $\mathfrak{S} = \text{dist}_y(C_N, C_S) + \text{dist}_x(C_E, C_W)$ is maximized, then it has inner boundaries shaped χ or \succ . Here C_N, C_E, C_W, C_S denote the centers of gravity of the NEWS regions. Furthermore, this partitioning is unique.*

Proof: In the proof of this lemma we make use of the following property of the center of gravity: *If a polygon P can be partitioned into two parts A and B , then C_P is the weighted average of C_A and C_B , with the areas of A and B as weights.*

We will prove in the first part that the inner boundary which separates the regions always consists of five straight line segments, one of them is vertical or horizontal and the other ones have slope $+1$ or -1 . In the second part of the proof we will show that in any NEWS partition according to the lemma statement, there are at most two different focal points.

First we prove that the middle part of the inner boundary is always a horizontal (vertical) line segment. Assume that the polygon P is divided into a North N and South S region only, such that both regions cover exactly 50% of the area and the sum \mathfrak{S} is optimal. Let $p_N \in \text{int}(N)$ and $p_S \in \text{int}(S)$ be two points inside N and S respectively, such that the y -coordinate of p_N is smaller than the y -coordinate of p_S . Let d denote the difference between the two y -coordinates. Let $0 < \gamma < d/2$. Because both points p_N and p_S lie in the interiors of the regions, there exist ϵ -disks ($0 < \epsilon \leq \gamma$) around these points which are fully contained in the according region. If we exchange the two ϵ -disks, C_S moves southward and C_N moves northward, increasing the sum \mathfrak{S} : a contradiction. We conclude that the boundary between North and South is horizontal. Exactly the same argument shows that in a partitioning into North, South, East, and West, there is a horizontal line with North above and South below it. Furthermore, by symmetry we know that East and West are separated by a vertical line.

Consider one of the other boundaries, say, the one between East E and South S . Let both regions cover exactly 50% of the area of P and assume $\mathfrak{S}' = C_{E,x} - C_{S,y}$ is maximal. Let $p_S \in \text{int}(S)$ and $p_E \in \text{int}(E)$ be two points that can be separated by a line ℓ with slope -1 , such that p_S lies above and p_E below ℓ . There exist ϵ -disks around the points that are fully contained in the according region. If we exchange the ϵ -disks, the values $C_{E,x}$ and $C_{S,y}$ will change. By construction, we get a total increase of \mathfrak{S}' . This shows that the boundary between South and East is a line with slope -1 , and the proof immediately extends to the sum \mathfrak{S} . Because of symmetry, the NW boundary also has slope -1 , whereas the NE and SW boundaries have slope $+1$. Again, the argument applies as well when partitioning P into four regions. This ends the first part of the proof.

In the second part we show that always three segments of the inner boundary meet at one focal point. Because of symmetry reasons we need to look at only one case; we choose to consider how the NS, NE, and SE boundaries meet. For now, we restrict ourselves to the case that each one of the NS, NE, and SE boundaries intersects $\text{int}(P)$. Given an arbitrary, simple polygon P and a NEWS partition such that the three supporting lines NS, NE, SE do not meet in a single focal

point. Assume that \mathfrak{S} is optimal for this partitioning. For simplification we put the coordinate system such that the NS boundary lies on the x -axis, the NE boundary lies on the line with equation $y = x + \delta$, and the SE boundary lies on the line with equation $y = -x + \delta$. If $\delta = 0$ then all three lines meet in one (East focal-) point (see Figure 2.2, left). Let $\delta > 0$ (the case $\delta < 0$ is symmetric). We choose a point $p_E \in \text{int}(E)$ very close to NE, a point $p_S \in \text{int}(S)$ very close to SE and a point $p_N \in \text{int}(N)$ very close to NS. By ‘very close’ we mean at a distance $d \leq \delta/4$. Around these three points we draw an ϵ -disk such that $\epsilon < d$, with the same ϵ for all three points. We name these disks $D_\epsilon(p_E), D_\epsilon(p_S), D_\epsilon(p_N)$. By choosing ϵ small enough, we ensure that none of these disks intersect the boundary of its region nor the lines $y = \pm x$ (see Figure 2.2, right). We now exchange the ϵ -disks as follows: We remove $D_\epsilon(p_E)$ from E and add it to N , we remove $D_\epsilon(p_N)$ from N and add it to S and we remove $D_\epsilon(p_S)$ from S and add it to E . We claim that (a) in this new NEWS partition all regions still have 25% of P 's area and that (b) the new \mathfrak{S}' is larger than the old \mathfrak{S} . (a) follows directly from our construction, and (b) is proved next. We assign coordinates to the three points as follows:

$$\begin{aligned} p_E &= (a, a + \gamma_1) & a > 0 \text{ and } 0 < \gamma_1 < \delta \\ p_S &= (b + \gamma_2, -b) & b > 0 \text{ and } 0 < \gamma_2 < \delta \\ p_N &= (-c, \gamma_3) & c > 0 \text{ and } 0 < \gamma_3 < \delta \end{aligned}$$

With this, we can compute the changes of the centers of gravity as follows:

$$\begin{aligned} \Delta C_{N,y} &= \epsilon' \cdot (a + \gamma_1 - \gamma_3) \\ \Delta C_{S,y} &= \epsilon' \cdot (-\gamma_3 - b) \\ \Delta C_{E,x} &= \epsilon' \cdot (b + \gamma_2 - a) \end{aligned}$$

with $\epsilon' > 0$, depending on ϵ and the area of P . It is easy to see that ϵ' is the same for all three equations, because N, S and E have the same area and the three (exchanged) disks have the same size.

If we sum up the three changes, we get $\Delta \mathfrak{S} = \epsilon' \cdot (\gamma_1 + \gamma_2 - 2 \cdot \gamma_3)$. By definition, γ_1, γ_2 are both greater than zero, and γ_3 can be chosen arbitrarily small. So, the total change $\Delta \mathfrak{S}$ is always greater than zero, and therefore $\mathfrak{S}' = \mathfrak{S} + \Delta \mathfrak{S} > \mathfrak{S}$. This is a contradiction to the assumption that \mathfrak{S} is optimal.

Now we consider what happens if one boundary, say, SE, does not intersect $\text{int}(P)$. If SE intersects $\partial(P)$ in the East, we can again find three disks whose exchange increases \mathfrak{S} . If SE does not intersect $\partial(P)$ in the East, we can move SE without changing the partition. Either we get that NS, NE, and SE meet at only one focal point, which is what we set out to prove, or SE reaches the boundary of P in the East. In the latter case we again find three ϵ -disks as before, which we can exchange to increase \mathfrak{S} .

To prove uniqueness, assume that two partitionings of the shape \succ exist that give four regions of area 25% of the area of P . Two different partitionings like \succ will always have one region, say, North, such that North of the one partitioning strictly contains North of the other partitioning. Since P is simply-connected, not both North regions can have exactly 25% of the area of P . Similarly, two partitionings, one of shape \succ and one of shape λ , will also have some region in the one partitioning for which there is proper containment in the corresponding region in

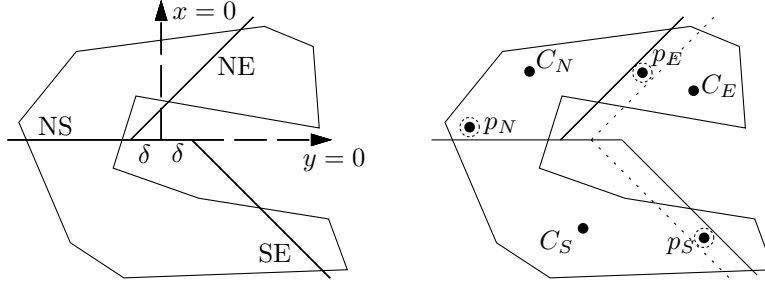


Figure 2.2: There are at most two focal points in any NEWS partition.

the other partitioning. □

2.2 NEWS Partitions with Arbitrary Regions

In this section we give an algorithm to compute all wedges with fixed apex angle and orientation that contain a given area A inside. We also show how to compute a NEWS partitioning of a simple polygon following Suggestion 3.

Definition 1 For a simple polygon P , we call a wedge of the form $(y \geq x+a) \cap (y \geq -x+b)$ that contains 25% of the area of P a North-wedge of P . East-, South-, and West-wedges are defined similarly.

Definition 2 The North-trace T_N is the locus of all points that are apex of a North-wedge. East-, South-, and West-traces are defined similarly.

The outline of an algorithm to compute a NEWS partitioning according to Suggestion 3 is as follows:

1. Compute the East-trace T_E and the West-trace T_W of polygon P .
2. Scan T_E and T_W simultaneously from top to bottom to determine if there is a pair of points $p_E \in T_E$ and $p_W \in T_W$ with the same y -coordinate and p_E to the right of p_W (or coinciding) and which gives a North area of 25% of P .
3. If such a pair exists, return the \succ partitioning.
4. Otherwise, compute the North-trace T_N and the South-trace T_S .
5. Scan T_N and T_S simultaneously from left to right, to determine a pair of points $p_N \in T_N$ and $p_S \in T_S$ with the same y -coordinate and p_N higher than p_S and which gives a West area of 25% of P .
6. Return the χ partitioning.

Before we describe the algorithm for computing a trace and finding the pair in more detail, we first show some properties of the traces. For convenience, we rotate the polygon by 45 degrees, so that a West-wedge is a quadrant of the form $(x \leq a) \cap (y \leq b)$.

Lemma 2 *After rotation, the West-trace is an infinite, continuous, piecewise quadratic curve consisting of $\Theta(n^2)$ pieces in the worst case, and any line with positive slope intersects the West-trace once.*

Proof: Let $p = (a, b)$ be a point on the West-trace, such that it is the apex of a West-wedge that contains 25% of the area of P . Since P is a simple polygon, the West-wedge must intersect the interior of P . Let $p' = (a', b')$ be any point with $a' > a$ and $b' > b$. Then a wedge with apex at p' contains strictly more than 25% of the area, thus p' cannot be on the West-trace. This proves the last statement of the lemma.

Assume next that we fix a subset of the edges of P that intersect the horizontal half-line of a West-wedge, and we fix another subset for the vertical half-line of that West-wedge. If the apex has a fixed y -coordinate, the area of intersection of the polygon and the West-wedge is a quadratic function in x [155]. Similarly, if the x -coordinate of the apex is fixed, the area of intersection is a quadratic function in y . It follows that the trace has the form $axy + bx^2 + cy^2 + dx + ey = f$ as long as the West-wedge intersects the same subset of edges of P . The values of the constants a, b, c, d, e , and f depend on P .

The quadratic upper bound on the number of pieces is trivial. The lower bound construction is shown in Figure 2.3. Proportions are chosen such that a wedge with 25% of the area contains four of the triangular teeth. The four long diagonal lines are actually very thin parts that hardly influence the area inside the wedge. Four wedges with 25% inside are shown dashed, and the trace is shown bold. \square

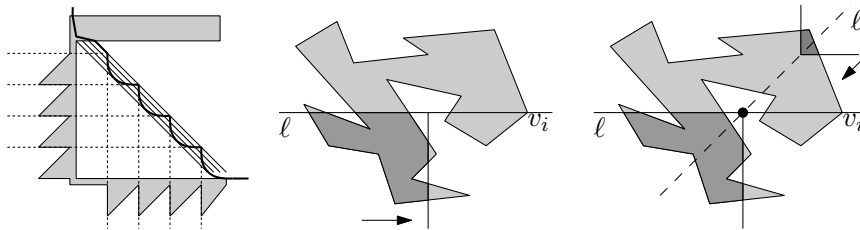


Figure 2.3: Left: A simple polygon that has a trace with quadratic combinatorial complexity. Middle and right: Illustration of the $O(n \log n)$ time partitioning algorithm.

We next describe a sweep algorithm that computes the West-trace for the rotated polygon. Since the algorithm can be used for any fixed area of P inside the wedge, we set $A = \text{area}(P)/4$ and give an algorithm to compute all West-wedge positions that have area A inside the intersection of P and the wedge. The sweep computes the trace from left to right.

We initialize by computing a vertical line that has area A of P left of it. Then we determine the highest point of P left of or on the vertical line. The x -coordinate of the vertical line and the y -coordinate of the highest point give the first break point p_0 of the trace. The initial part of the trace is a vertical half-line down to p_0 . We initialize two lists L_x and L_y with all vertices of P sorted by increasing x -coordinate

and decreasing y -coordinate, respectively. We remove the vertices from the lists up to the x - and y -coordinates of p_0 .

During the sweep we maintain three pieces of information: a balanced binary search tree on the edges of P that intersect the sweep line, the leaf in this tree that stores the edge of P vertically below the current position of the trace, and the equation of the curve that is now valid. One of three types of event can occur:

1. The vertical line through the current position of the trace (the sweep line) reaches a vertex of P .
2. The horizontal line through the current position of the trace reaches a vertex of P .
3. The trace reaches the boundary of P .

Which event will occur next can be determined in $O(1)$ time from the equation of the curve, the first element in each list, and the edges of P above and below the current trace position. Events of Type 1 and 3 may require an update of the tree or the pointer to the leaf with the edge below the trace. If the event is of Type 1 and the vertex of P is above the position of the trace, we do nothing else. If the event is of Type 1 and the vertex of P is below the position of the trace, we output the next break point and we update the equation of the curve. The equation of the curve is updated by subtractions and additions to the constants a, b, c, d, e , and f , depending on which edges of P are no longer intersected, and which edges of P start to be intersected by the West-wedge. For the second and third type of event similar actions are taken. Note that the sweep also continues if the position of the trace goes outside P . Later it may enter P again.

Since preprocessing takes $O(n \log n)$ time, events of Types 2 and 3 take $O(1)$ time, events of Type 1 take $O(\log n)$ time, and the trace has quadratic complexity, we conclude with the following theorem:

Theorem 1 *Given a simple polygon P with n vertices and a value A , the set of all positions of apexes of wedges with a fixed shape and orientation that contain a portion of area A of P inside can be computed in optimal $O(n^2)$ time.*

The NEWS partitioning algorithm is completed by computing a pair of points on the West- and East-trace such that 25% of the area of P is in the North (Step 2), or the symmetric situation (Step 5). This also takes $O(n^2)$ time, so we can compute a NEWS partitioning for Suggestion 3 in $O(n^2)$ time. Note that the running time for computing a trace is $O(n \log n + k)$, where k is the complexity of the trace. Typically, k is much less than quadratic, and in practice the algorithm is expected to be efficient.

We next give a worst case $O(n \log n)$ time algorithm, which improves the previous result. We only describe the case of a \succ partitioning. The idea is not to compute whole traces, but only a small part that has at most linear complexity. We will first determine between which two y -coordinates of vertices of P the horizontal half-line of the West-wedge lies (in the rotated polygon) for a solution that gives four regions with 25% of the area each. Similarly, we will find two consecutive x -coordinates

for the vertical half-line of the West-wedge. Then we compute the West-trace only between these x - and y -coordinates.

In more detail, the algorithm is as follows: First sort all vertices of P by x -coordinate in a list. Also sort all vertices of P by $x - y$ and by $x + y$ in two other lists. Then we start a binary search. Let v_i be a vertex of P with median y -coordinate, and consider a horizontal line ℓ through v_i (see Figure 2.3, middle). We will sweep the apex of a wedge on ℓ from left to right until the position where it is a West-wedge and contains 25% of the area of P . To this end, sort the intersections of ℓ and the boundary of P using Jordan sorting [92]. Merge them with the list of vertices sorted by x , to obtain a list of all events sorted by x -coordinate. Sweep the apex from left to right and maintain the area of P in the wedge. At every event point we update the function that describes the area of P inside the wedge. When the wedge has 25% of the area of P inside, the sweep stops; we found a West-wedge. The West focal point defines a slope +1 line ℓ' on which the East focal point should lie as well. We intersect P with ℓ' , and sweep a wedge with its apex on ℓ' from the upper right to the lower left until it is an East-wedge and contains 25% of the area of P inside. The events of this sweep are intersections of ℓ' with P , and crossings of the wedge boundary with the vertices of P . A sorted list of events is obtained by merging three sorted lists. These are the lists sorted on $x - y$ and $x + y$, and the intersections of ℓ' and P after Jordan sorting.

If the West-wedge and the East-wedge partly overlap, this is always in a square. In that case the part of P inside this square is both in West and East. To still be able to make the correct decision for the binary search, compute the area of North and of South, not including the square. North and South together have more than 50% of the area of P . If either North or South has area less than 25%, then we can still make the decision correctly. If both North and South have area larger than 25%, we can conclude by the proof of Lemma 1 that no \succ partitioning gives the desired result, and we compute a χ partitioning instead.

We have given the basis of a binary search that will give two consecutive y -coordinates y' and y'' of vertices of P between which the West focal point must lie. Similarly, we determine between which two x -coordinates x' and x'' the West focal point must lie. Each decision step of the binary search takes $O(n)$ time, and we take $O(\log n)$ steps in total. The three sorts on x , $x - y$, and $x + y$ prior to the binary search also take $O(n \log n)$ time.

We now know that the West focal point of the \succ partitioning must lie in the rectangle $[x' : x''] \times [y' : y'']$. Within this rectangle we compute the West-trace explicitly; we use the sweep algorithm given before. Since no vertex of P has its x -coordinate strictly between x' and x'' , nor its y -coordinate strictly between y' and y'' , we only have events of Type 3 in the sweep. We can have a linear number of such events if many edges of P intersect $[x' : x''] \times [y' : y'']$. However, we can not have more than $2n$ of them: the West-trace cannot intersect the same edge of P three or more times, because the subset of edges of P intersecting the half-lines of the West-wedge is the same for those three intersection points, and the function describing the trace is quadratic by Lemma 2. Hence, we compute the West-trace inside $[x' : x''] \times [y' : y'']$ in $O(n \log n)$ time overall.

In exactly the same way we can compute the relevant part of the East-trace in $O(n \log n)$ time. By sweeping a line with slope +1 over these two traces we get

all candidate North regions and determine if one has the desired area. If not, a \succ partitioning does not exist and we compute a λ partitioning instead, in the same way.

Theorem 2 *Given a simple polygon P with n vertices, we can determine a partitioning by \succ or λ where each region contains exactly 25% of the area of P in $O(n \log n)$ time. The partitioning maximizes the sum of distances of centers of gravity $\mathfrak{S} = \text{dist}_y(C_N, C_S) + \text{dist}_x(C_E, C_W)$.*

2.3 Simply-connected NEWS Partitions

In the previous section we obtained the optimal partition into NEWS regions in the case that we are only interested in the sum of center-of-gravity distances and the property that each subregion has 25% of the total area of P . The solution of that problem may give a partition into non-connected regions. In this section we study variations where the four subregions must be simply-connected. This implies that we must relax the 25% property, allow a different shape of partitioning, or accept that a solution does not always exist.

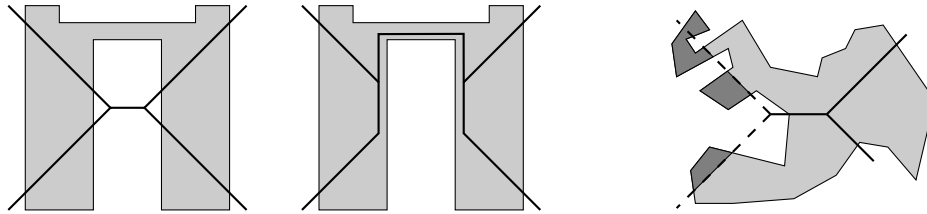


Figure 2.4: Left, middle: It is not interesting to consider simply-connected region partitions by relaxing the shape of the partitioning only. Right: When the West focal point lies outside P , it is not well-defined how to cut. Any one of the dark regions could be West.

Consider the polygon in Figure 2.4, left. It is partitioned according to the method in the previous section, giving a disconnected South. However, giving freedom to the shape of the partition would only yield the solution shown in the middle of Figure 2.4, where the two former parts of South are connected artificially by a very narrow passage. Therefore, we study problems where the shape of the partitioning is maintained, but the 25% property is relaxed. However, this makes it unclear what to optimize for the sum of distances of center-of-gravity. It seems we should maximize a weighted sum, where each distance is weighted by the area of the region. At the same time we still want the four NEWS regions to have area close to 25%. We choose to maintain the shape of the partition and try to obtain regions of area close to 25%. The shape of the partition will—in practice—cause the four regions to be taken from the appropriate parts of the polygon, which we previously formalized by the center-of-gravity distances. Hence, we consider the following problems:

Fair partitioning: Find a simply-connected partition into four regions by \succ or

χ such that each region has 25% of the area, or decide that no such partition exists.

Maxmin: Find a simply-connected partition by \succ or χ that maximizes the area of the smallest region.

Minmax: Find a simply-connected partition by \succ or χ that minimizes the area of the largest region.

We require in all cases that the partition itself is simply-connected inside P . Otherwise, we may get several options with a \succ or χ partition. For example, if a focal point is outside, one of the diagonal boundaries incident to it cannot be used. Furthermore, a diagonal boundary, for example between North and West, may intersect the polygon several times and there is a choice which cut to take. For example, in Figure 2.4, right, we must choose exactly one of the four dashed intersections of \succ and P .

Note that for some polygons we cannot guarantee any percentage of at least $\varepsilon\%$ with $\varepsilon > 0$ fixed for all regions, no matter how small ε is. Similarly, we cannot guarantee that all regions have at most $(50 - \varepsilon)\%$ of the area. The fair partitioning problem is a special case of both the second and the third problem, but we can solve it in $O(n \log n)$ time, whereas our solution of the latter two problems takes $O(n^3)$ time.

2.3.1 Computing a fair partitioning

For a fair split we wish to obtain four simply-connected regions, each with 25% of the area. Again we will compute the traces for the four wedges, but we have to adapt the algorithm considerably to incorporate simply-connectedness. Interestingly, the combinatorial complexity of a trace is linear now, and we can compute a trace in $O(n \log n)$ time. We again assume that P is rotated by 45 degrees and consider the computation of the West-trace. We will sweep a vertical line from left to right to compute all apex positions of wedges that have 25% of the area in the wedge. For any x -coordinate, there can be up to three such apex positions. We let $4A$ be the total area of P , so that we want to compute apex positions where the area of P in the corresponding wedge is exactly A .

Let ℓ be some position of the sweep line, and let ℓ_i be some connected component of $\ell \cap P$ (a vertical line segment). We consider apex positions on ℓ_i and observe that the area in the wedge decreases monotonically from top to bottom on ℓ_i . Assuming that no edge of P is vertical, the decrease is strict. The following four cases occur:

1. All wedges with apex on ℓ_i have area less than A .
2. All wedges with apex on ℓ_i have area greater than A .
3. There is one wedge with apex on ℓ_i that has area A .
4. None of the above, that is, the area decrease of the wedge when the apex goes from top to bottom on ℓ_i has a discontinuity where it jumps from larger than A to smaller than A .

The last case can occur at a y -coordinate of a vertex of P that is a local minimum or maximum of the exterior, see Figure 2.5, left. Corresponding to the four cases above, we store the following during the sweep:

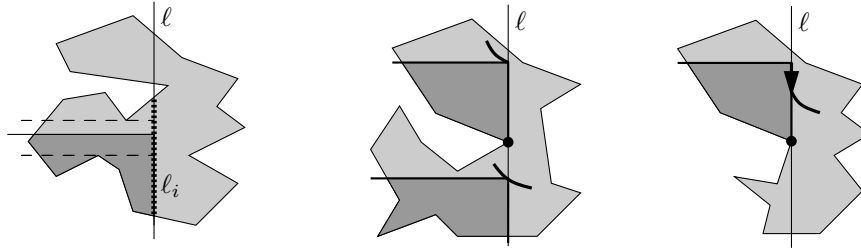


Figure 2.5: Left: Illustration of the area in a wedge with apex on ℓ_i (dark grey). There are discontinuities in the area decrease at the dashed lines. Middle and right: Two cases of the sweep algorithm.

1. We maintain the area in the wedge whose apex is at the highest point on ℓ_i , as a function of x .
2. We do not store anything.
3. We maintain the equation of the curve on which the apex of the area- A wedge lies.
4. We maintain the position on ℓ_i where the discontinuity that jumps over area A occurs. We also maintain the area of the largest wedge that has area less than A , as a function of x .

The curve of the third case is monotonically decreasing and is called the *trace*, like before. The position of the fourth case keeps its y -coordinate, that is, it follows a horizontal segment that we call *shift*.

To be able to know all events before they occur, we use a vertical decomposition VD and a horizontal decomposition HD of P . These are constructed during the preprocessing. For each edge in these decompositions, we also store the areas of the two subpolygons it induces. This part of preprocessing can easily be done in $O(n \log n)$ time (or even in linear time using linear time triangulation of polygons). For any trace or shift, we maintain in which trapezoid of VD and HD it lies at the current sweep position.

When the sweep line moves right, there are five possible events that can occur. In the following paragraphs we will analyze the four times five cases; note that not all combinations can occur, and that there is also some overlap. The largest number of different events occur when the sweep line reaches a vertex of P . In the following, we will call this a *vertex event*.

(i) The sweep line reaches a vertex of P . In several cases we have to perform a *scan down*, which is to find the y -coordinate at the current sweep line position where a trace or shift will start. We will always determine this y -coordinate from top to bottom. The *scan down* is described at the end of this paragraph.

Assume that some interval ℓ_i on the sweep line ℓ reaches a vertex v of P . Then v can be at the lower endpoint of ℓ_i , at the upper endpoint of ℓ_i , or in the middle. We distinguish the following situations:

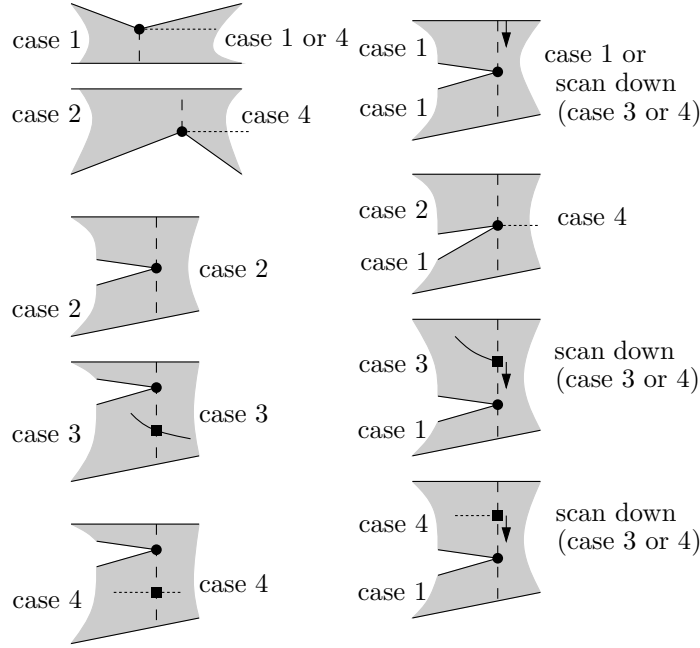


Figure 2.6: Vertex events in the sweep algorithm: Situations A, B, C, and D.

- A.** The edges of P incident to v are on opposite sides of the sweep line, and v is the upper endpoint of ℓ_i .
- B.** The edges of P incident to v are on opposite sides of the sweep line, and v is the lower endpoint of ℓ_i .
- C.** The edges of P incident to v are both on the left of the sweep line, and v is the lower endpoint of ℓ_i .
- D.** The edges of P incident to v are both on the left of the sweep line, and v is the upper endpoint of ℓ_i .
- E.** The edges of P incident to v are both on the right of the sweep line, and v is in the middle of ℓ_i .

These five situations must be combined with the four cases for ℓ_i before the event, as described above.

For Situation A, the only non-trivial case is Case 1, see the top left picture in Figure 2.6. The area function for a wedge with the apex at the highest point on ℓ_i may grow with a jump (if both edges incident to v go upward), in which case we can stay in Case 1 or get into Case 4. We can retrieve efficiently by how much the area jumps, because we precomputed the areas of all subpolygons arising from segments in the horizontal decomposition HD . So this is easy to test and handle. If we get into Case 4, the shift must start at the y -coordinate of the vertex v . Situation A for Cases 2, 3, and 4 cannot give a case transition, and we only need to update a trace equation or an area function and test for future events that may occur.

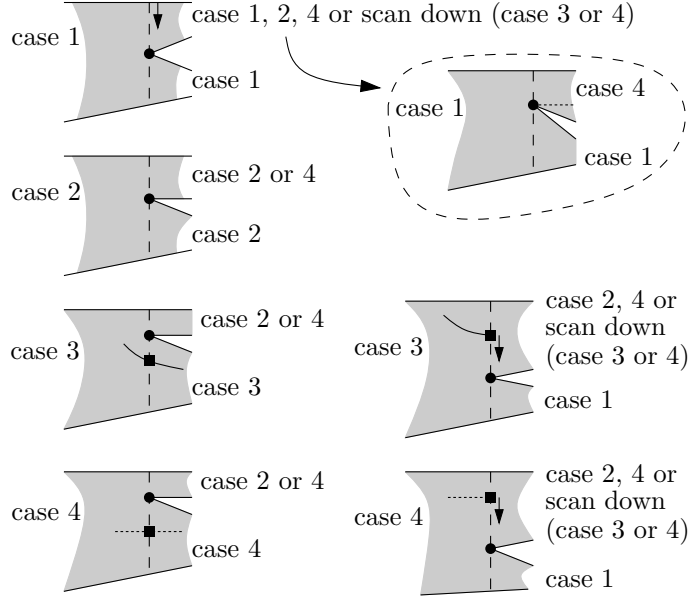


Figure 2.7: More vertex events in the sweep algorithm: Situation E.

For Situation B, the only non-trivial case is Case 2, see the second picture in the left column of Figure 2.6. Depending on the edge of P to the right of v , we may get a transition into Case 4. This is easy to handle. Situation B for Cases 1, 3, and 4 cannot give a case transition.

The three other situations combined with all possible cases are shown in seven pictures in Figure 2.6 and in six pictures in Figure 2.7. In some of the pictures of Figure 2.7 we see that after an event we can sometimes get into Case 2 or Case 4 for the upper new intersection of ℓ and P , for instance in the pictures in the left column. Which case we get into depends on the slope of the edge of P counterclockwise from the vertex that caused the event. If the slope is negative, we get into Case 4, otherwise in Case 2 (or in the top left picture, in Case 1 or we do a scan down).

Which case we get into after any of the events in the two figures can be determined because we have precomputed all areas of subpolygons of P arising from the edges in the horizontal and vertical decomposition. For example, consider Figure 2.7. Let e_d be the edge vertically down from the event vertex v in VD , let e_u be the edge vertically up from v in VD , and let e_l be the edge horizontally to the left in HD . For each of these edges we precomputed the areas of the two subpolygons that they induce. This allows us to determine from the previous case and the subareas what case we get into for each of the two new intersections of the sweepline with P . For example, consider the bottom right picture of Figure 2.7 and assume the edge of P counterclockwise from v goes upward, as drawn. Then the upper branch will be in Case 2 if the area of the subpolygon below e_l is greater than A . If this area is less than A , the upper branch will be in Case 3 or 4, which is determined by a scan down starting at the end of the shift.

Scan down It remains to describe the scan down. In the scan down, we start at a point on the boundary of P or at a trace or at a shift. In all cases we can determine the area in the wedge at the start of the scan down, and this area will be greater than A . We maintain the area inside the wedge as a function of the y -coordinate of the apex.

When the apex of the wedge sweeps down, the horizontal half-line of the wedge will cross horizontal edges of HD . If these are caused by a vertex v of P left of the apex, this is an event. There are three types of vertex events.

If one edge incident to v goes up and the other goes down from v , we only have to update the area function for the wedge.

If both edges incident to v go up from v , there is a sudden loss of area in the wedge. The amount is known from the areas of subpolygons in the horizontal decomposition. If the area in the wedge jumps over A , the scan down stops and a shift starts at this point. Otherwise, we update the area function and proceed with the scan down.

If both edges incident to v go down from v , there is a sudden loss of area in the wedge as well, and we do exactly the same.

Besides vertex events it can also happen that the area inside the wedge reaches A . In that case, we stop the scan down and a trace starts at this point.

In the events where the area function changes, we must update the event list to detect the events where the area inside the wedge reaches A in time.

(ii) The trace or shift reaches an edge of P . In this event, the only point or candidate point to give a desired wedge lies on the boundary of P and will go outside. Consequently, after the event, the whole interval ℓ_i will be in Case 1 (when the trace or shift leaves at the top of ℓ_i) or Case 2 (when the trace or shift leaves at the bottom of ℓ_i). We update the status structure accordingly. If we go into Case 1, we must determine the function giving the wedge area if the apex is on the top edge, and determine if and when a new trace can start (to predict event (iv) in time).

(iii) The area in the shift of a wedge becomes A . This can only happen in Case 4, and we get into Case 3 afterwards. We update the status structure and compute the equation of the trace. We also do tests to predict if events of Type (v) or Type (ii) will occur for the trace.

(iv) The area in the wedge with apex at the top of ℓ_i becomes A . This event can only occur for Case 1, and gives a transition into Case 3. We update the status structure and compute the equation of the trace. We also do tests to predict if and when events of Type (v) or Type (ii) will occur for the trace.

(v) The trace or shift crosses an edge of the vertical or horizontal decomposition. This event can only occur for Cases 3 and 4. We update the trapezoid in which the trace or shift is, and if necessary, compute the new equation of the trace or area function for the shift. When the trace crosses an edge of HD , this may give a transition from Case 3 into Case 4 (see Figure 2.8), in which case we do the necessary updates in the status structure. To obtain the area function for the shift, we subtract the ‘lost area’ in the subpolygon left and below the vertex that was hit. This area is precomputed as the area of some subpolygon in HD .

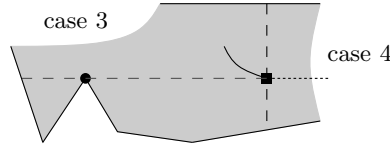


Figure 2.8: Type (v) event where a Case 3 to Case 4 transition occurs.

Then we do tests do predict when the next event of Types (ii), (iii), or (v) will occur for this trace or shift. All other situations to be handled are also Type (i) events, and these are treated separately.

Type (i) events are known in advance (some of these events are also of Type (v)). Type (ii) events are known on time since we maintain in which trapezoid of VD and HD any trace or shift lies, and a trapezoid gives at most two edges of P that can be reached. Types (iii) and (iv) events are known because we maintain the function that gives the area in the wedge, and we can determine for which x an area A is reached. Type (v) events are known because we know in which trapezoid the trace or shift is.

Correctness and running time follow from the following lemmas. Lemma 4 implies linear complexity of the traces.

Lemma 3 *Every wedge with area A inside will be found by the algorithm.*

Proof: The beginning of the trace on which the apex of such a wedge lies will be found during some event. So the event handling proves the correctness. \square

Lemma 4 *Any trapezoid of VD or HD is intersected by at most one trace or shift.*

Proof: First, observe that in the vertical decomposition VD there cannot be two points in one trapezoid on traces with the same y -coordinate. The wedge with apex at the higher point will have strictly larger area inside. Similarly, in the horizontal decomposition HD , there cannot be two points in one trapezoid on traces with the same x -coordinate.

Next, consider the case when two shifts go through the same trapezoid of VD . Any point on a shift has the property that a point on it gives a wedge area less than A , and a point just above it gives a wedge area greater than A . Then obviously, two shifts with different y -coordinate cannot enter the same trapezoid of VD . It would contradict the property that the wedge area is monotonically decreasing when the apex goes down vertically. For the same reason, a shift and a trace cannot enter the same trapezoid of VD . \square

The algorithm given above can be used for any value of A . Furthermore, it applies to any wedge with a given orientation and angle. We can simply rotate and shear-transform the polygon to bring it into the case that was described. We conclude:

Lemma 5 *Given a simple polygon P with n vertices and a value A , we can determine in $O(n \log n)$ time all locations of the apex of a wedge with fixed orientation*

and area A inside it, when the intersection of the polygon and the wedge is simply-connected.

After computing the West-traces and East-traces in the (still rotated) polygon P , we determine if there is a connecting line segment with slope $+1$ completely inside P that gives a North region of area A , and hence, a South region of area A as well. This can be done easily with a sweep over the polygon and the traces with a line of slope $+1$. If we fail, we also try the North trace, the South trace, and a connecting segment of slope -1 . If this also fails, there is no partitioning of P into four simply-connected regions by \succ or λ , where the partitioning itself is simply-connected inside P as well. We conclude with the following theorem:

Theorem 3 *For a simple polygon P with n vertices, we can determine in $O(n \log n)$ time if there is a simply-connected \succ or λ partitioning into four simply-connected regions that all have 25% of the area of P .*

2.3.2 Computing a maxmin and minmax area partitioning

To compute a partitioning that maximizes the smallest area region or minimizes the largest area region, we also assume that P is rotated by 45 degrees. We only describe the case where the North and South regions are adjacent in the partitioning.

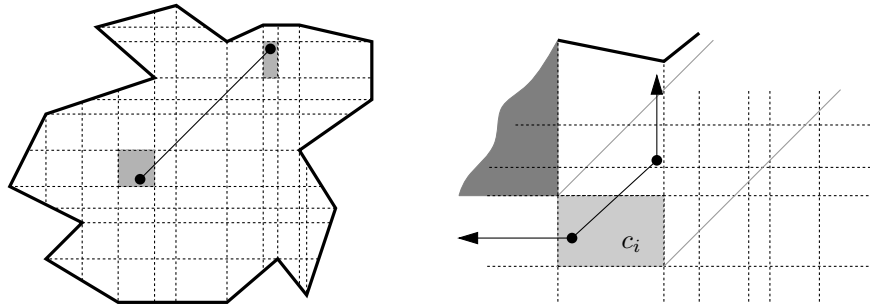


Figure 2.9: Left: The decomposition D induced by the horizontal and vertical decomposition, and a pair of aligned cells. Right: Area function for the North when the West focal point lies in c_i .

The algorithm begins with computing a horizontal decomposition HD and a vertical decomposition VD inside P . This gives a decomposition D of the interior of P into $O(n^2)$ cells (see Figure 2.9). For each vertex of D , we precompute and store the following: (i) The area of the West (East, North, and South) region if the West (East, North, and South, respectively) focal point would coincide with the vertex. (ii) The edges of P directly above, below, left, and right of it. We can easily precompute this information in $O(n)$ time per vertex, and $O(n^3)$ time overall. For every vertical edge of the decomposition D , we precompute the area in the cell left of it, and the area in the cell to the right of it. This is easy in linear time per vertical edge, or in quadratic time altogether. Next, we will consider pairs of cells of D that can be aligned diagonally, by which we mean that there is a line of slope $+1$ that

intersects both cells. One cell can be aligned with $\Theta(n^2)$ other cells, but we can show that there are only $O(n^3)$ alignments in total by a simple counting argument.

Lemma 6 *There are $O(n^3)$ pairs of cells in the decomposition that can be aligned diagonally.*

Proof: We imagine sweeping a line with slope +1 over the polygon and its decomposition D . We count a pair of cells as soon as they become aligned, which is when the sweep line starts intersecting the second cell of the pair. This happens at the top left vertex of the cell (assuming the cell is in the interior of D). At any top left vertex of a cell, only $O(n)$ new pairs are found, because the sweep line intersects only $O(n)$ edges of D . \square

When we consider a pair of aligned cells, we are interested in the functions that describe the areas of the West, East, North, and South regions as a function of the locations of the two focal points. Let these coordinates be (x_w, y_w) and (x_e, y_e) , where $y_e = y_w + x_e - x_w$. The function for West is a bivariate function in x_w and y_w . The functions for East, North, and South are trivariate functions in x_w, y_w , and x_e .

Assume that we have the four area functions for a pair of cells of the decomposition. Then we determine the exact values of the three unknowns that maximize the minimum of the four functions, or minimize the maximum. It will be the highest point on the lower envelope of four 3-dimensional patches in 4-space, or the lowest point on the upper envelope. We need to solve sets of three quadratic equations in three unknowns. In the standard model of computation for surface patches, these operations take $O(1)$ time.

It remains to show that we can obtain the appropriate functions in $O(1)$ time for a pair of cells. The area functions of West and East are easy and can be obtained directly from the precomputed information. For the North region, we will do this by choosing a cell for the West focal point and considering in which cells the East focal points can lie, see Figure 2.9. We treat the cells by looking at a row from left to right, and then going to the next higher row. Whenever we go from one cell to the next, we know which edges of P no longer intersect the North region boundary, and which edges of P have started to intersect the North region boundary. In this way we can determine the new quadratic function in $O(1)$ time per cell using the precomputed information. For the South region we do the same. We obtain:

Theorem 4 *For a simple polygon P with n vertices, we can compute a simply-connected \succ or \sphericalangle partitioning that minimizes the maximum area subregion or maximizes the minimum area subregion in $O(n^3)$ time.*

2.4 Experiments

We implemented the partitioning algorithms for all three suggestions to evaluate the performance on ten countries. We did not implement the versions producing simply-connected partitionings, so the partitionings produced for Suggestions 2 and 3 always give four regions with 25% of the area. The results can be found in Table 2.1.

The left column shows the partitionings for Suggestion 1, using the center of gravity. As expected, the resulting regions may have an area deviating significantly from 25%, like the East regions of Norway and Portugal and the West of Great Britain. Consequently, there are several parts that are not classified as east Portugal or west Great Britain, where intuition would do so.

The middle column shows the partitionings for Suggestion 2. If the x -extent is larger than the y -extent, the algorithm first computes a vertical line with 25% of the area left of it and a vertical line with 25% right of it. The remaining part is partitioned in equal-size regions by a horizontal line. If the y -extent is larger, the symmetric partitioning method is taken. Generally, the results correspond reasonably well to intuition. But the partitioning of a square-like shaped country as France, for instance, is highly dependent on the fact that the y -extent is slightly larger. If the x -extent were slightly larger, a completely different partitioning would be produced. As a consequence, the partitioning of France shows a West and East region that are unnaturally limited. Another unnatural situation appears in the northeast of Austria. Also observe that a large part of the West coast of Norway is in the East region.

The right column shows the partitionings for Suggestion 3. The partitionings of Austria and France are much better, but the partitionings of Norway and Italy are not ideal. In summary, the second and third suggestions both give good partitionings on most countries, although both fail on some. It depends on the global shape of the country which of the two suggestions is better.

Our suggestions did not take any perceptual aspects into account. The purely geometric and simple problem statements of the second and third suggestions must be adapted to work well in nearly all cases that may arise. There are many possible ways of doing this, but it is beyond the scope of this chapter to implement, compare, and discuss these.

Table 2.1. Comparison of three different algorithms for determining a NEWS division for ten european countries.

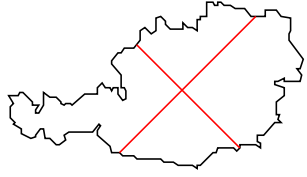
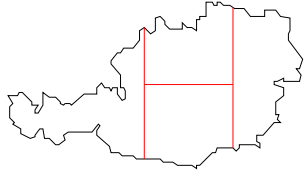
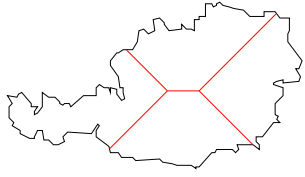
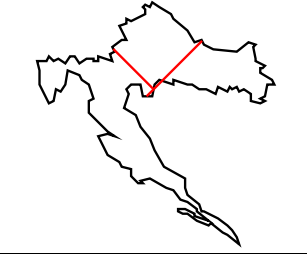
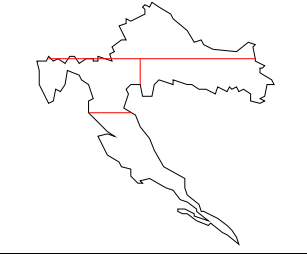
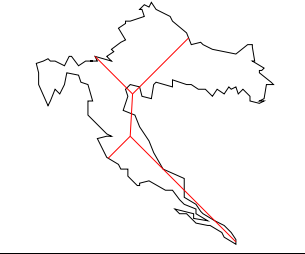

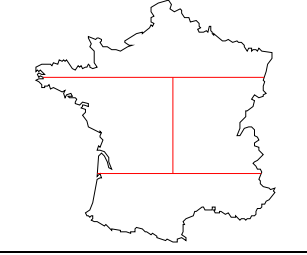
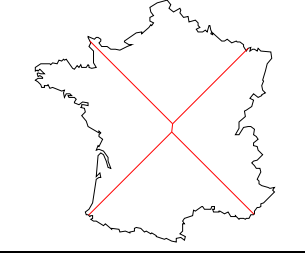
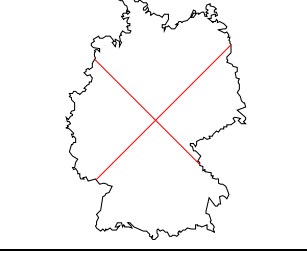
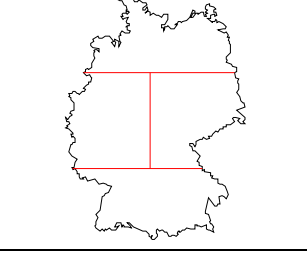
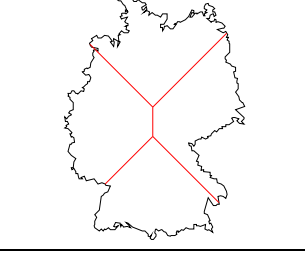
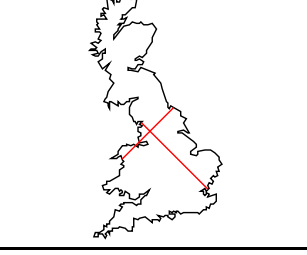
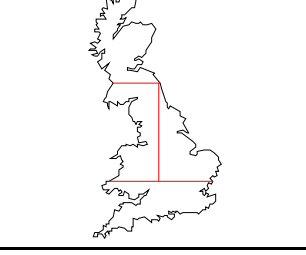
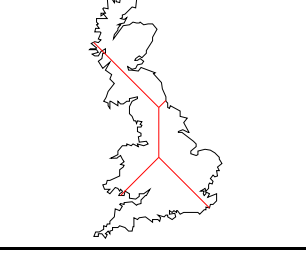
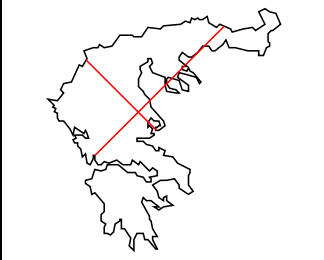
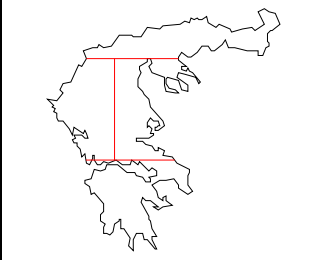
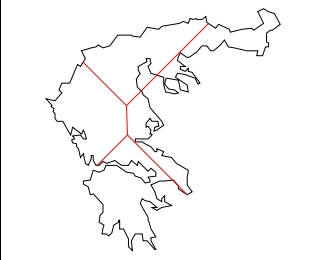


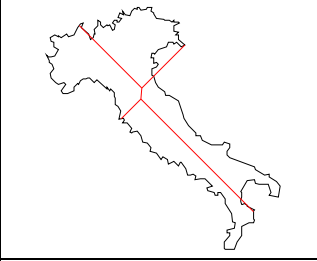
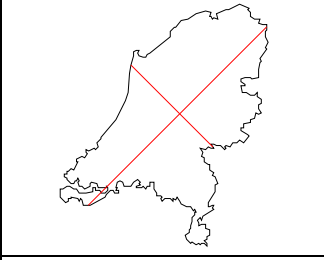
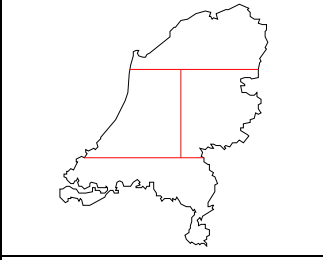
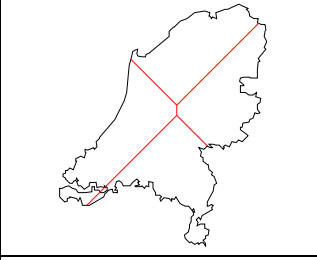
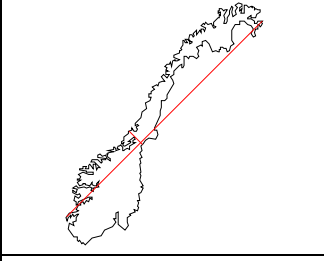
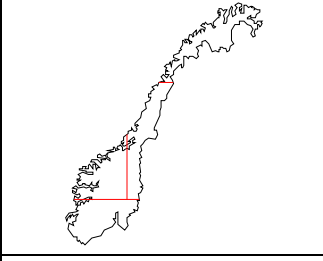
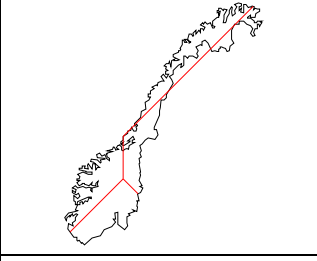
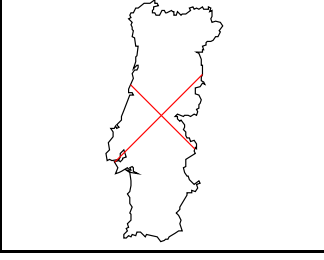
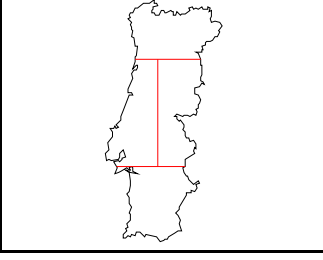
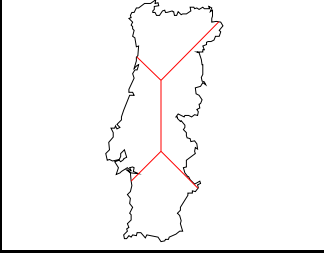
From top to bottom: Austria, Croatia, France, Germany, and Great Britain.		
		
		
		
		
		

Table 2.1. (Continued)

From top to bottom: Greece, Italy, The Netherlands, Norway, and Portugal.		
		
		
		
		
		

2.5 Extensions

In many cases it is useful to partition not only simple polygons, or to go beyond a partitioning into North, East, West, and South only. We briefly discuss three possible extensions.

Firstly, we consider non-connected polygons, which arise for countries that have one or more major islands. For all three suggestions, the algorithms can be adapted in a straightforward manner to incorporate this case. For the second suggestion, this follows from the fact that Shermer's algorithm [155] applies to disconnected polygons as well.

Secondly, one can define a center region of a country as well, which is important for spatial information retrieval. Suggestion 3 can be restated as partitioning a polygon into five regions, each with 20% of the area, such that the summed distances of the centers of gravity of North, East, West, and South to that of the original polygon is maximized (again, measured in x - or y -distance only). In this case, we can apply the same proof ideas as in Section 2.1 to show that the center region will be an axis-parallel rectangle and the boundaries between North, East, West, and South are lines with slope $+1$ or -1 that start at the corners of the central rectangle.

Thirdly, it is useful to define a degree of North, East, West, and South for any point in the country. For example, San Francisco is more in the West of the U.S. than Salt Lake City, although both can be considered to lie in the West. This is important for relevance ranking of the list of retrieved documents of a Web search.

2.6 Concluding Remarks

This chapter discussed how to partition a region into four subregions in a natural way, such that the subregions can be associated with the North, East, West, and South. The applications lie in geographic information retrieval and Web searching. After giving criteria for a good partitioning, we suggested and analyzed three possibilities. One of these led to interesting algorithms related to earlier research on partitioning (sectioning, cutting, dividing) a simple polygon into equal-size parts. The experiments show that two of the suggestions for partitioning give good results, albeit not perfect.

It is open whether the algorithms in this chapter can be improved: there is no $\Omega(n \log n)$ or cubic lower bound. Another interesting algorithmic question is how efficiently one can determine an orthogonal four-partition as in Suggestion 1, but one that minimizes the largest region, or maximizes the smallest region. Also the question of defining and computing a degree of North, East, West, and South is an interesting problem.

3

DELINEATING IMPRECISE REGIONS USING LOCATIONS

In the previous chapter, we have looked at the North, East, West, and South parts of a country and presented a method to delineate them. These regions are relatively simple to recognize and to delineate by straight lines. There are numerous other imprecise geographical regions like the British Midlands, the US Bible Belt or the Dutch Randstad, which have a more complex boundary and are less easy to delineate. Often, these boundaries are described like ‘above the rivers’ or ‘beyond the valley’, sometimes there are no visible or physical boundaries at all. In this chapter, we will delineate such regions by using place names marked as ‘inside’ or ‘outside’ as evidence.

To determine the locations that are inside or outside a certain imprecise region we can make use of the Web. The enormous amount of text on Web pages

Table 3.1: Trigger phrases used to identify geo-references

ID	Trigger phrase	Examples
in	* in [R]	Birmingham in the Midlands
which is	* which is [in in the * of] [R]	West Ham which is in London
is a	* is a [city county province region state town village] in [R]	Paris is a city in France
is direction	* is [in located in situated in] the [center centre north south east west north east south east north west south west] of [R]	Canterbury is located in the south east of England
such as	[cities towns villages counties provinces regions states] in [R] [such as including] *	Cities in the Midlands such as Birmingham
and other	* and other [cities towns villages counties provinces regions states] in [R]	Staffordshire and other counties in the Midlands

can be used as a source of data; the idea of using the Web as a geo-spatial database has appeared before [109, 121, 140]. Our approach is using so-called *trigger phrases*. For any reasonable-size city in the British Midlands, like Nottingham, it is quite likely that some Web page contains a sentence fragment like ‘... Nottingham, a city in the British Midlands,...’, or ‘Nottingham is located in the British Midlands...’. Table 3.1 lists the trigger phrases used for the data acquisition of this chapter, where ‘*’ matches anything, [R] represents the target region and [X|Y] matches X or Y. The sentence fragments found give locations that are most likely inside the British Midlands, and we used the SPIRIT ontology together with a gazetteer list from UK Ordnance Survey to assign coordinates to the extracted locations. These locations are considered ‘inside’ the British Midlands; we compute their bounding box, and enlarge it by 20% in each direction to get their surroundings as well. Again, we use the ontology and the gazetteer to determine all locations and their coordinates inside the enlarged bounding box, which were not found using the trigger phrases. We consider this set of locations to be ‘outside’ the British Midlands. Details of using trigger phrases to determine locations inside or outside a region to be delineated, as well as an evaluation of the method can be found in [8]. Obviously the process is not very reliable, and false positives as well as false negatives are likely to occur.

We have arrived at the following computational problem: given a set of ‘inside’ points (red) and a set of ‘outside’ points (blue), determine a reasonable polygon that separates the two sets. Imprecise regions generally are not thought of as having holes or a tentacle-shaped boundary, but instead as having a compact shape. Therefore, possible criteria for such a polygon are:

- The red points are inside and the blue points are outside the polygon.
- The polygon is simply-connected.
- The polygon has small perimeter.
- The polygon boundary has small absolute angular change.
- The polygon has large compactness ratio, circularity ratio, or form ratio, or it has small elongation ratio. These are shape measures for polygons used in geography [73, 130].

It is likely that some points are misclassified, and that a polygon with much better shape can be obtained if a small number of red points are allowed to be outside and a small number of blue points are allowed to be inside the polygon. A suitable balance is needed between the shape measure and the misclassification measure. The field of machine learning is largely devoted to the use of points of known classification (e.g., red and blue) to infer the classification of other points (see e.g., [43]). In machine learning applications, the feature space is usually high-dimensional and one is generally not constructing explicit region boundaries, as we are motivated to do in our geographical application. In pattern recognition, extraction of shapes from point patterns is also studied [162]. However, since we concentrate on a geographical application, we will consider only the polygon criteria listed above.

Related work in GIS has been conducted by Alani et al. [5], who use a gazetteer to extract locations that are classified as inside or outside of a certain region. A Voronoi diagram of all locations is computed, and the Voronoi cells are classified inside or outside, according to the locations they enclose. The region is the union

of all cells classified as inside, and its boundary is composed by the Voronoi edges that separate cells of different classification. However, this approach does not take into account that the data points may be classified wrongly. Concurrent with our approach, Purves et al. [40, 140] proposed another method which is also based on using trigger phrases to find locations inside imprecise regions on the Web. Instead of delineating polygons that separate points that are classified as inside or outside the imprecise region, they use weights like document density, to define the extent of the imprecise region as a density surface.

In computational geometry, red-blue separation algorithms exist of various sorts; see Seara [150] for a survey. Red-blue separation by a line—if it exists—can be solved by two-dimensional linear programming in $O(n)$ time for n points. Red-blue separation by a line with the minimum number of misclassified points takes $O((n + k^2) \log k)$ expected time, where k is the number of misclassified points [34]. Other fixed separation shapes, such as strips, wedges, and sectors, have also been considered [9, 150]. For red-blue separation by a polygon, a natural problem is the minimum perimeter polygon that separates the bichromatic point set. This problem is NP-hard (by reduction from Euclidean traveling salesperson [10, 57]); polynomial-time approximation schemes follow from the m -guillotine method of Mitchell [118] and from Arora’s method [13]. Minimum-link separation has also received attention [4, 12, 117].

In this chapter we present two general approaches to determine a reasonable polygon for a set of red and blue points. Based on these approaches we define and solve various algorithmic problems, some of which are of theoretical interest, others are of practical interest. The two global approaches are outlined in Section 3.1. The first approach starts by computing an α -hull of the red points, with a number of red and blue points inside this polygon. In a second step it tries to adapt the polygon to get the blue points outside while keeping the red points inside. We show that for a red polygon with n vertices and only one blue point inside, the minimum perimeter adaptation can be computed in $O(n)$ time. For the case of one blue and $O(n)$ red points inside, an $O(n \log n)$ -time algorithm is presented. If there are m blue points but no red points inside, an $O(m^3 n^3)$ -time algorithm is given. If there are m red and blue points inside, we give an $O(C^m \log^m \cdot n)$ -time algorithm, for some constant C . These results are given in Section 3.2. The second approach changes the color of points to obtain a better shape of the polygon. Different recoloring strategies and algorithms are presented in Section 3.3. The implementation and test results on several data sets for both approaches are given in Section 3.4.

3.1 Approaches to Delineate Imprecise Regions

This section describes our two approaches to obtain a reasonable boundary for a bichromatic set of points that may contain wrongly colored points. Let R be the set of red points and B the set of blue points.

3.1.1 Adaptation approach

In the adaptation approach we begin by computing on the red points only. A reasonable shape for the red points can for example be based on the α -shape [58, 59].

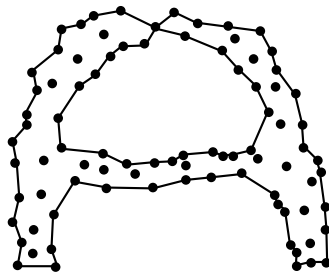


Figure 3.1: The α -shape of a set of points for a negative value of α .

The concept of α -shapes formalizes the intuitive notion of ‘shape’ for spatial point set data, and they have been studied and used extensively in geometric modeling, grid generation, medical image analysis, and visualizing the structure of earthquake data.

The formal definition of α -hulls, taken from [58], is:

Definition 3 An α -disk for $\alpha \in \mathbb{R}$ is

- a closed disk of radius $1/\alpha$ if $\alpha > 0$,
- a closed half-plane if $\alpha = 0$, and
- the closed complement of a disk of radius $-1/\alpha$ if $\alpha < 0$.

For a finite set S of sites in the plane, an α -disk is full if it contains S . The α -hull of S is the intersection of all full α -disks.

If $\alpha = 0$, the α -hull is the convex hull of the point set S . An α -shape is closely related to the α -hull, but it has straight edges. For every pair of points that give an α -disk that defines some part of the boundary of the α -hull, there is an edge between them in the α -shape. Figure 3.1 gives an example; a formal definition is complex and can be found in [58].

A reasonable choice for a polygon P to delineate an imprecise region enclosing the red points is the outermost set of edges of the connected component of the α -shape that contains the largest number of red points. This polygon is not necessarily simple because vertices may appear more than once on the boundary. We call such a polygon a *degenerate simple polygon*. The value of α has large influence on the shape; it should be chosen depending on the density of cities and towns in the region of interest. Using the component of the α -hull with most red points takes care of red outliers, which are isolated or appear in smaller components. These red points are likely to have been misclassified. This can happen due to ambiguity of place names, or by incorrect listing of a place name in a trigger phrase.

In the adaptation approach, we start with the polygon P based on the red points only. The polygon P may contain blue points, so we will change its shape to get them outside, but without bringing any red points outside. In practice we will bring blue points outside only if this does not change the shape of the polygon too much (for example, the perimeter increase is small). Blue points that remain inside are likely to have been misclassified; they correspond to places that are in the imprecise region, but are not mentioned in any trigger phrase.

The theoretical question of changing a polygon while increasing its perimeter as little as possible gives rise to various computational problems that we discuss in the next section. In all cases we are interested in a subpolygon $P^* \subseteq P$ that does not contain the blue points in the interior, nor any red points in the exterior, and has smallest perimeter among these (or the smallest absolute angular change). The polygon P^* will be simple and possibly degenerate. We will give different algorithms for various instances of the problem, e.g., for convex and simple polygons with one or more blue points inside only and with red and blue points inside, in Section 3.2. Experimental results are given in Section 3.4.

3.1.2 Recoloring approach

In the recoloring approach, we change the colors of points that are likely to have been misclassified. This is the case if many surrounding points have the other color. Let R be the set of red points and B the set of blue points. Compute the Delaunay triangulation of $R \cup B$. For each point, consider its color and the colors of its neighbors. If the point is largely surrounded by points of the other color, then we change its color, and continue. To formalize ‘largely surrounded’, we consider for a point the maximum angle of consecutive differently colored neighbors. If this angle is greater than some pre-specified value, say 230° , then we recolor. A more formal definition of the angle is given in Section 3.3. If the critical angle is smaller than 180° , then the method may not terminate so we always assume that it is at least 180° .

In general there are several points that can be recolored at some moment, and it is important in which order the recoloring is performed. Different orders can give different final colorings. We examine different recoloring schemes and prove bounds on the number of recolorings that can occur in certain schemes in Section 3.3. Experimental results of the recoloring approach are given in Section 3.4.

3.2 Adaptation Approach in Detail

In the adaptation approach, we start with a polygon P and adapt it until all blue points inside P are no longer inside, or until the shape would have to be changed too drastically. By choosing P initially as an α -shape, with α chosen such that e.g., 90% of the red points lie inside P , we can determine an appropriate initial shape and remove red outliers (red points outside P) in the same step. The parameter α can also be chosen based on ‘jumps’ in the function that maps α to the perimeter of the initial polygon that we would choose. Once P is computed, the remaining problem is to change P so that the blue points are no longer inside. The resulting polygon P^* should be contained in P and its perimeter should be minimum. In this section we discuss the algorithmic side of this problem. In practice it may be better for the final shape to allow some blue points inside, which then would be considered misclassified. Some of our algorithms can handle this extension.

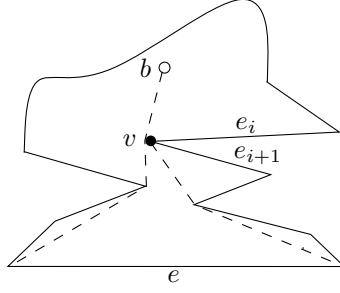


Figure 3.2: Illustration of the proof of Lemma 8.

3.2.1 One blue point inside P

First, we assume that there is only one blue point b inside P . The optimal, minimum-perimeter polygon P^* inside P has the following structure:

Lemma 7 *An optimal polygon P^* is a (possibly degenerate) simple polygon that (i) has b on its boundary, and (ii) contains all edges of P except one.*

Proof: If P has no blue point inside, then the optimal subpolygon $P^* \subset P = P$, as it must include all the red vertices of P . With one blue point b inside P , we have to adapt P by replacing k of its edges by a path inside P that excludes b of P . If $k = 0$, this path leads from a vertex v of P to b and back to v . The shortest of all such paths consists of two straight line segments \overline{vb} , \overline{bv} , where v is a vertex of P that is directly visible from b . The length of P^* is the length of P plus the length of \overline{vb} and \overline{bv} . P^* can be shortened by replacing one of the line segments and the edge $\overline{vv'}$ by a geodesic from a neighboring vertex v' of v to b . Therefore, it must hold that $k \geq 1$. It is easy to see that P^* cannot be optimal if $k > 1$, as it can be shortened by replacing one of the paths by an edge of P . Furthermore, if the new path around b does not contain b , it can always be shortened until it contains b . \square

We consider two versions of the problem: the special case where the interior of P contains only one point (namely b), and the general case where the interior of P contains b and a number of red points.

One blue and no red points in the interior of P

Let P be a red polygon with only one blue point b inside. Let $e = \overline{v_1v_2}$ be the edge of P that is not an edge of P^* . The endpoints v_1 and v_2 are connected by a path F via b inside the boundary of P . We denote the path that leads to the minimum-perimeter polygon P^* by F^* ; it consists of a shortest geodesic path between b and v_1 , and between b and v_2 , and is called a *funnel*. Note that these paths may have vertices at red points other than v_1 and v_2 ; such vertices are concave vertices of P , as shown in Figure 3.3 (left). In the optimal solution P^* , the edge e and the shortest path F^* have the following additional properties:

Lemma 8 (i) *The path F^* is a simple funnel.* (ii) *A funnel F with root b and base e can only be minimal if e is (at least partially) visible from b .*

Proof: We prove the first claim by contradiction (see Figure 3.2). Let P be a simple polygon with a point b inside. Let e and F^* be the edge and path that lead to the minimum perimeter polygon P^* , and assume that F^* has some edge on both shortest geodesic paths between b and v_1 and v_2 . Then the edges incident to b must also be the same. Let this shared segment be \overline{bv} . Vertex v is the endpoint of two edges e_i, e_{i+1} of P^* . The solution using e and F^* has perimeter strictly greater than the perimeter of P plus twice the length of \overline{bv} . However, using e_i instead of e and the geodesics to e_i 's endpoints yields a solution of length at most the perimeter of P plus twice \overline{bv} , a contradiction.

The second claim follows from the first claim. As the geodesics do not share any edges, there must be some opening angle $0 < \phi < \pi$ at b , and F^* is a funnel. The angle ϕ is determined by the two edges of the funnel that are incident to b . As the interior of F^* does not contain polygon edges, the base edge e is visible from b through ϕ . \square

We use the algorithm of Guibas et al. [82] to find the shortest path from the point b to every vertex v of the polygon. For every two adjacent vertices v_i and v_{i+1} of the polygon, we compute the shortest paths connecting them to b . The algorithm of Guibas et al. [82] can find all these paths in $O(n)$ time. For each possible base edge and corresponding funnel, we add the length of the two paths and subtract the length of the edge between v_i and v_{i+1} to get the added length of this choice. We obtain the following result.

Theorem 5 *For a simple polygon P with n vertices and with a single point b inside, we can compute in $O(n)$ time the minimum-perimeter polygon $P^* \subseteq P$, that contains all vertices of P , and has b on its boundary.*

One blue and several red points in the interior of P

In the general case we may also have red points inside P . Let R be the set of these red points, and assume that its size is $O(n)$. We need to adapt the algorithm given before to take the red points into account. We first triangulate the polygon P . Ignoring the red points R , we compute all funnels F from b to every edge e of the polygon. We get a partitioning of P into $O(n)$ funnels with disjoint interiors. In every funnel we do the following: If there are no red points inside F , we only store the length of the funnel without its base edge. Otherwise, we need to find a shortest path π_{\min} from one endpoint of the base edge to b and back to the other endpoint of the base edge, such that all red points in R still lie inside the resulting polygon P^* .

The shortest path π_{\min} inside some funnel F with respect to a set $R \cap F$ of red points consists of two chains which, together with the base edge e , again form a funnel F^* , see Figure 3.3 (middle). This funnel is not allowed to contain points of $R \cap F$ in its interior. We need to consider all possible ways of making such funnels, which involves partitioning the points of $R \cap F$ into two subsets. The red points of $R \cap F$ can only appear as reflex points on the funnel F^* , and therefore we can use

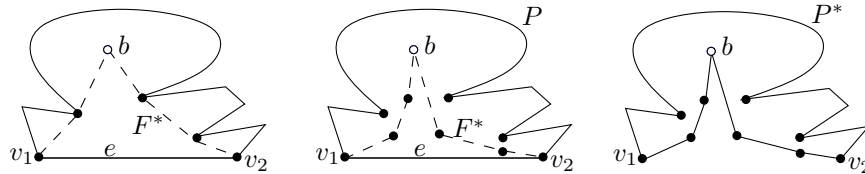


Figure 3.3: Left: The path F^* if $R = \emptyset$. Middle: The case $R \neq \emptyset$. Right: P^* for $R \neq \emptyset$.

an order on these points. For ease of description we let e be horizontal. Then F has a left chain and a right chain. The optimal funnel F^* also has a left chain and a right chain, such that all points of $R \cap F$ lie between the left chains of F and F^* and between the right chains of F^* and F . We extend the two edges of F incident to b , so that they end on the base edge e . This partitions F into three parts: a left part, a middle triangle, and a right part. In the same way as the first claim of Lemma 8, we can show that all points of $R \cap F$ in the left part must be between the left chains of F and F^* , and all points of $R \cap F$ in the right part must be between the right chains of F and F^* . The points of $R \cap F$ in the middle triangle are sorted by angle around b . Let the order be r_1, \dots, r_h , counterclockwise.

Lemma 9 *There is an index i such that the points r_1, \dots, r_i lie between the left chains of F and F^* , and r_{i+1}, \dots, r_h lie between the right chains of F^* and F .*

We iterate through the $h + 1$ possible partitions that can lead to an optimal funnel F^* , and maintain the two chains using a dynamic convex-hull algorithm [88]. Every next pair of chains requires a deletion of a point on one chain and an insertion of the same point on the other chain. We maintain the length of the path during these updates to find the optimal one.

As to the efficiency, finding all shortest paths from b to all vertices of P takes linear time for a polygon. Assigning the red points of R to the funnels takes $O(n \log n)$ time using either plane sweep or planar point location. Sorting the h red points inside F takes $O(h \log h)$ time, and the same amount of time is taken for the dynamic convex hull part. Since each red point of R appears in only one funnel, the overall running time is $O(n \log n)$.

Theorem 6 *For a simple polygon P with n vertices, a point b in P , and a set R of $O(n)$ red points inside P , we can compute in $O(n \log n)$ time the minimum-perimeter polygon $P^* \subseteq P$, that contains all vertices of P and all red points of R , and has b on its boundary.*

3.2.2 Several blue points inside P

When there are more blue points inside P , we use other algorithmic techniques. We first address the case of only blue points inside P and give a dynamic-programming algorithm. Then we assume that there are a constant number of blue and red points inside P , and give a fixed-parameter tractable algorithm.

Several blue and no red points in the interior of P

Let P be a simple red polygon with n vertices, given in clockwise order. Let B be a set of m blue points inside P . In this section, we present a polynomial-time algorithm to compute a (possibly degenerate) simple polygon, $P^* \subset P$, of minimum perimeter such that (1) no point of B is interior to P^* , (2) no point of P is exterior to P^* , and (3) $P^* \subseteq P$.

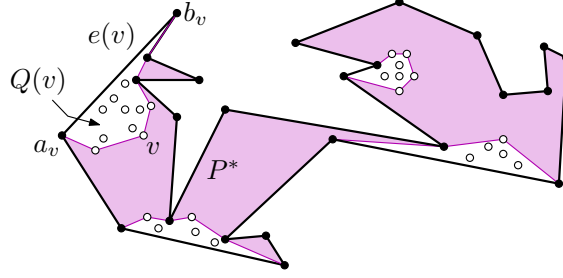


Figure 3.4: The (degenerate) simple polygon P^* is shaded and lies inside the black boundary of the input polygon P . The red points are vertices of P and of P^* , and the blue points (hollow circles) that were inside P are excluded from P^* , lying within pockets, each of which is formed by the relative convex hull of the blue points within it.

We begin by stating a structure lemma, which forms the basis for our algorithm (see Figure 3.4):

Lemma 10 *The polygon P^* is a (possibly degenerate) simple polygon whose vertices are either red points (of P) or blue points (of B). Each red point of P appears at least once (and at most twice) as a vertex of P^* . Each blue vertex, v , of P^* is necessarily a reflex vertex of P^* and is associated with an edge of P , $e(v) = a_v b_v$, which is the lid of the pocket, $Q(v)$, associated with v . $Q(v)$ is a simple polygon and is one of the connected components of the difference $P \setminus P^*$. The boundary of $Q(v)$ consists of the lid edge $e(v) = a_v b_v$, together with a polygonal chain of red and blue vertices; $Q(v)$ is the relative convex hull of the set $\{a_v, b_v\} \cup (B \cap Q(v))$.*

In general, P^* is a degenerate simple polygon, with ‘pinch points’ at red vertices that arise along the boundary of a relative convex hull of blue points within a pocket. Such pinch points appear twice in a boundary traversal of the vertices of P^* .

Let $\pi_P(u, v)$ denote the geodesic path within P from $u \in P$ to $v \in P$. We note that, by the defining property of relative convex hulls, each blue vertex v that appears on the boundary of a pocket $Q = Q(v)$ is *geodesically visible* to the endpoints, a_v and b_v , of the lid of the pocket Q : $\pi_P(v, a_v) \subset Q$ and $\pi_P(v, b_v) \subset Q$.

Since P^* is a simple polygon, it has a triangulation T , whose diagonals join pairs of vertices of P^* . (The existence of a triangulation holds even though P^* is potentially degenerate.) Our algorithm is a dynamic programming algorithm that searches for an optimal polygon P^* , together with a triangulation of it.

Our dynamic programming algorithm defines a value function, $f(\cdot)$, which assigns to each *subproblem* the cost of an optimal solution of the subproblem. In order to describe what goes into defining a subproblem, consider a diagonal, uv , of the triangulation T of P^* . We consider uv to partition P^* and T into the ‘done’ and the ‘yet-to-be-done’ portion of the triangulated optimal solution. The endpoints of uv are vertices of P^* , which may be red or blue. In the case that one or both are blue, we will also need to specify the identity of the lid of the pocket associated with the blue vertex; this permits us to partition the region P , within which the blue points are located that need to be excluded in the optimal enclosure P^* .

For a blue vertex u of P^* , we let $a_u b_u = e(u)$ be the lid of the pocket $Q(u)$, with a_u preceding b_u in clockwise order around P ; for a red vertex u , we define $a_u = b_u = u$, and $Q(u) = u$, for convenience. Then, the ‘done’ and the ‘yet-to-be-done’ portions of P are separated by a polygonal path, $\pi_P(b_u, u), uv, \pi_P(v, a_v)$, which is the concatenation of the geodesic path $\pi_P(b_u, u) \subset Q(u)$, the diagonal $uv \subset P^*$, and the geodesic path $\pi_P(v, a_v) \subset Q(v)$. The subproblem associated with the ‘state’ (b_u, u, v, a_v) is to compute a minimum-length red-blue separator (polygonal path) that starts at u , ends at v , lies inside P , encloses all red vertices of P going clockwise from b_u to a_v , and excludes the set B_{b_u, u, v, a_v} of blue points within P that lie to the left of the polygonal path $\pi_P(b_u, u), uv, \pi_P(v, a_v)$. We let $f(b_u, u, v, a_v)$ be the length of such a shortest separator; i.e., $f(b_u, u, v, a_v)$ is the *value* (or *cost*) associated with the subproblem.

For any diagonal, uv , of P^* in the triangulation T , there is a triangle, Δuvw , to the left of the oriented diagonal uv . We can view this triangle (i.e., the choice of vertex w) as the optimal choice in the dynamic program. By optimizing over all choices of w , we get a recurrence relation (the Bellman equations) that must be satisfied by the value function f .

We distinguish several cases, depending on the color of the endpoints u and v :

(1) u and v are both blue. Necessarily, u and v lie on different pockets, by the relative convexity of pockets, since a diagonal uv is, by definition, internal to P^* . There are several subcases according to the choice of w :

(a) w is a blue vertex on a different pocket, with lid $e(w) = a_w b_w$; see Figure 3.5. Then, there are two new subproblems, (b_u, u, w, a_w) and (b_w, w, v, a_v) . We define

$$f^{(1a)}(b_u, u, v, a_v) = \min_{w \in W^{(1a)}, a_w \in [b_u, a_v]} [f(b_u, u, w, a_w) + f(b_w, w, v, a_v)],$$

where $[b_u, a_v)$ indicates the half-closed interval of vertices of P in the clockwise listing from b_u to a_v , and $W^{(1a)}$ is the subset of B_{b_u, u, v, a_v} for which no blue point lies inside or interior to an edge of the triangle Δuvw . (We can tabulate in advance the set of all such triples of blue points.)

(b) w is a blue vertex on the same pocket as u ; see Figure 3.6. Then, there is one new subproblem, $(b_w = b_u, w, v, a_v)$. We define

$$f^{(1b)}(b_u, u, v, a_v) = \min_{w \in W^{(1b)}} [|uw| + f(b_u, w, v, a_v)],$$

where $W^{(1b)}$ is the subset of B_{b_u, u, v, a_v} for which no blue point lies inside or interior to an edge of the triangle Δuvw , and uw lies inside P .

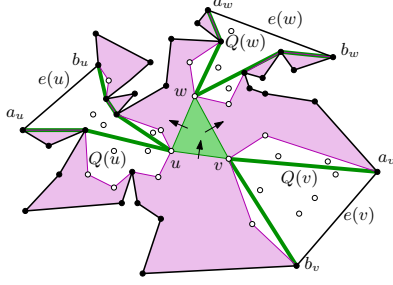


Figure 3.5: A step of the dynamic program: u and v are blue (hollow circles), and w is blue and lies on a different pocket.

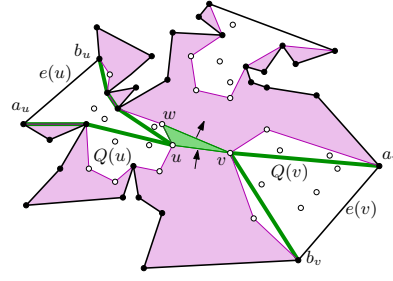


Figure 3.6: A step of the dynamic program: u and v are blue, and w is blue and lies on the same pocket as u .

- (c) w is a blue vertex on the same pocket as v . This case is analogous to case 1b and leads to a similarly defined function $f^{(1c)}$.
- (d) w is a red vertex, with both uw and wv being (internal) diagonals of P^* ; see Figure 3.7. Then, there are two new subproblems, (b_u, u, w, a_w) and (b_w, w, v, a_v) . We define

$$f^{(1d)}(b_u, u, v, a_v) = \min_{w \in (b_u, a_v) \cap W^{(1d)}} [f(b_u, u, w, w) + f(w, w, v, a_v)],$$

where (b_u, a_v) indicates the open interval of vertices of P in the clockwise listing from b_u to a_v , and $W^{(1d)}$ is the set of red points for which no blue point lies inside or interior to an edge of the triangle Δuvw .

- (e) w is a red vertex, with uw an edge of P^* , namely the first link in the geodesic path $\pi_P(u, b_u)$; see Figure 3.8. Let w' be the (red) successor of w in the geodesic path $\pi_P(u, b_u)$; possibly, $w' = b_u$. Then, there are two new subproblems, (b_u, w', w, w) and (w, w, v, a_v) . We define

$$f^{(1e)}(b_u, u, v, a_v) = \min_{w \in [b_u, a_v] \cap W^{(1e)}} [|uw| + f(b_u, w', w, w) + f(w, w, v, a_v)],$$

where $[b_u, a_v]$ indicates the closed interval of vertices of P in the clockwise listing from b_u to a_v , and $W^{(1e)}$ is the set of red points for which no blue point lies inside or interior to an edge of the triangle Δuvw .

- (f) w is a red vertex, with wv an edge of P^* , namely the first link in the geodesic path $\pi_P(v, a_v)$. This case is analogous to Case 1e and leads to a similarly defined function $f^{(1f)}$.

- (2) u and v have different colors (say, u is red and v is blue). There are subcases depending on the choice of w :

- (a) w is blue, lying on a different pocket as v ; see Figure 3.9. As in Case 1a, we obtain a value function $f^{(2a)}$ by optimizing over choices of w and lid $a_w b_w$, requiring triangle Δuvw to be blue-free (except at its vertices).

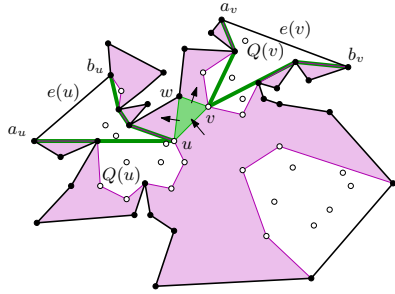


Figure 3.7: A step of the dynamic program: u and v are blue, and w is red, with uw and wv being (internal) diagonals of P^* .

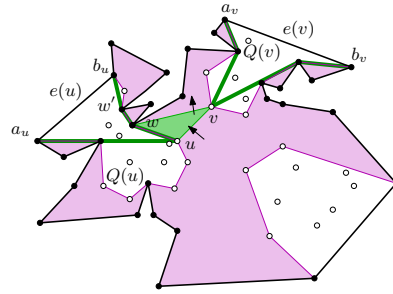


Figure 3.8: A step of the dynamic program: u and v are blue, and w is red, with uw an edge of P^* (the first link in the geodesic path $\pi_P(u, b_u)$).

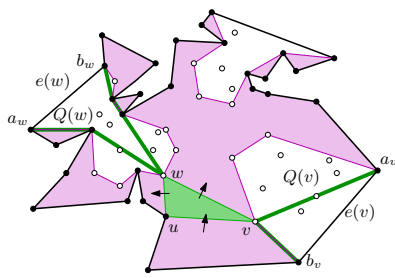


Figure 3.9: A step of the dynamic program: u is red, v is blue, and w is blue and lies on a different pocket.

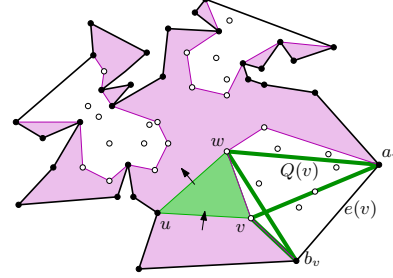


Figure 3.10: A step of the dynamic program: u is red, v is blue, and w is blue and lies on the same pocket as v .

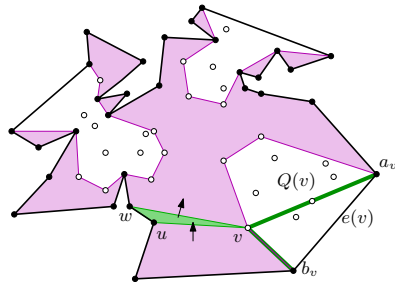


Figure 3.11: A step of the dynamic program: u is red, v is blue, and w is red. In this particular case, w is the clockwise successor of u around P , but it could be that uw is a diagonal of P^* , in which case a nonvacuous subproblem is specified by (u, u, w, w) .

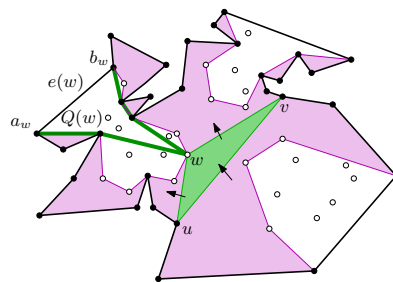


Figure 3.12: A step of the dynamic program: u and v are red, and w is blue.

- (b) w is blue, lying on the same pocket as v ; see Figure 3.10. As in Cases 1b and 1c, we obtain a value function $f^{(2b)}$ by optimizing over choices of w , requiring triangle Δuvw to be blue-free (except at its vertices).
 - (c) w is red, lying in the interval $(u, a_v]$; see Figure 3.11. As in Cases 1b and 1c, we obtain a value function $f^{(2c)}$ by optimizing over choices of w , requiring triangle Δuvw to be blue-free (except at its vertices). If w is the clockwise successor of u around P (as it is in the figure), then the value of the corresponding vacuous subproblem, (u, u, w, w) , is defined to be the Euclidean length, $|uw|$, of edge uw .
- (3) u and v are both red vertices; see Figure 3.12, where one of the two possible subcases (that in which w is a blue vertex) is shown. The corresponding subcases, 3a and 3b, are handled analogously to above, resulting in value functions $f^{(3a)}$ and $f^{(3b)}$, corresponding to whether w is blue or red.

Then, the overall optimization to compute f optimizes over all of the above subcases, 1a-1f, 2a-2c, and 3a-3b.

The correctness of the dynamic programming algorithm follows by induction.

The running time of the algorithm is $O(m^3 n^3)$, since there are $O(n^2 m^2)$ choices of state (b_u, u, v, a_v) and there are at most $O(mn)$ choices of actions (when we optimize over choices of w and a_w).

We summarize our result in the following theorem:

Theorem 7 *For a simple polygon P with n vertices and $m \geq 1$ blue points in P , we can compute in $O(m^3 n^3)$ time the minimum-perimeter polygon $P^* \subseteq P$ that contains all vertices of P and has the blue points either outside or on its boundary.*

Finally, we remark that, at a cost of increasing the complexity of the state space by a factor of k , and increasing the complexity of the action space also by a factor of k (in order to specify how to partition the ‘budget’ k), we can extend our results to optimize over red-blue separators that allow up to $k \leq m$ misclassified blue points remaining inside P^* , resulting in

Theorem 8 *For a simple polygon P with n vertices and $m \geq 1$ blue points in P , we can compute a minimum-perimeter polygon $P^* \subseteq P$, allowing up to $k \leq m$ misclassified blue points to remain inside P^* , in time $O((k+1)^2 m^3 n^3)$.*

Several blue and red points in the interior of P

We next give an algorithm that can handle m red and blue points inside a red polygon P with n vertices. The algorithm is fixed-parameter tractable: it takes $O(C^{m \log m} \cdot n)$ time, where C is some constant. Hence, if m is constant, this solution is more efficient than using dynamic programming.

Let R and B be the sets of red and blue points inside P , respectively, and let $m = |R| + |B|$. The structure of the solution is determined by a partitioning of $R \cup B$ into groups; see Figure 3.13. One group contains points of $R \cup B$ that do not appear on the boundary of P^* . In Figure 3.13, this group contains v, w, x , and y . For the other groups, the points are in the same group if they lie on the same

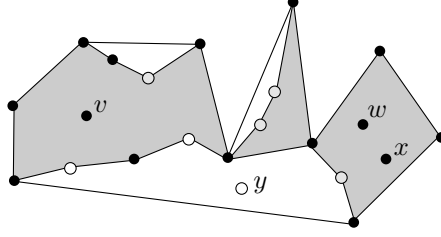


Figure 3.13: Structure of an optimal subpolygon when both blue (hollow) and red (solid) points are inside the original red polygon P .

chain that forms a pocket, together with some lid. On this chain, points from B must be convex and points from R must be concave for the pocket. Besides points from $R \cup B$, the chain consists of geodesics between consecutive points from $R \cup B$. For all points in the group not used on chains, all points that come from B must be in pockets, and all points that come from R may not be in pockets (they must lie inside P^*).

For a fixed-parameter tractable algorithm, we try all options for $R \cup B$. To this end, we generate all permutations of $R \cup B$, and all splits of each permutation into groups. The first group can be seen as the points that do not contribute to any chain. For any other group, we get a sequence of points (ordered) that lie in this order on a chain. We compute the full chain by determining geodesics between consecutive points, and determine the best edge of P to be replaced by this chain (we need one more geodesic to connect to the first point and one to connect to the last point of the chain). The best edge of P is the one with the smallest length increase. We repeat this for every (ordered) group in the split permutation, resulting in a candidate polygon $P^{(*)}$. Then we test if $P^{(*)}$ is (degenerate) simple, if the blue points of the first group are outside $P^{(*)}$, and the red points of the first group are inside $P^{(*)}$. If so, $P^{(*)}$ is one of the real candidates from which we want to find one with minimum perimeter.

Theorem 9 *For a simple polygon P with n vertices, and $m \geq 1$ red and blue points inside it, we can compute the minimum-perimeter subpolygon $P^* \subseteq P$ that contains all vertices of P and all red points, and has the blue points outside or on its boundary, in $O(C^{m \log m} \cdot n)$ time, for some constant C .*

Proof: Each of the $m!$ permutations of $R \cup B$ gives rise to 2^{m-1} splits, which amounts to $O(C_0^{m \log m})$. For each permutation and split we can compute P^* in $O(mn)$ time. Clearly, $O(C_0^{m \log m} \cdot mn) = O(C^{m \log m} \cdot n)$ for $C > C_0$. \square

Remark: The algorithm can easily be adapted to deal with up to k misclassified red and blue points in R and B within the same time bound.

Remark: If P is convex and there are only blue points inside P , we can improve the running time to $O(C^m \cdot n)$ by considering crossing-free partitions of B only. We can prove, in a similar way as in Lemma 8, that in an optimal solution, two chains

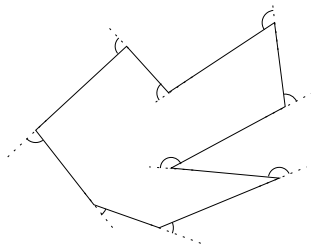


Figure 3.14: The absolute angular change of the simple polygon shown is the sum of the highlighted angles.

cannot intersect, and hence we need only consider partitionings into groups whose convex hulls do not intersect. There are only $O(C^m)$ crossing-free partitions [154].

3.2.3 Minimizing absolute angular change

Getting all blue points out of P and at the same time minimizing the perimeter length of the red region may in some cases lead to a highly detailed, fjord-like boundary, which is not desirable. One way to overcome this problem is, as already stated, to consider up to k blue points as misclassified and to allow them to remain inside the red region. Another way is to minimize not the perimeter length, but the total amount of ‘twists and turns’ of the boundary. We define the *absolute angular change* (or *total turn*, as it has been called in the study of bicriteria path optimization [11, 136]); see Figure 3.14.

Definition 4 *The absolute angular change of a polygon is the sum of the absolute value of the angle between the extension of an edge and its successor edge, when walking counterclockwise along the polygon.*

It is easy to see that the absolute angular change equals 2π in case of a convex polygon and is strictly larger than 2π in case of any other simple polygon.

Now the problem statement is as follows. We wish to determine the subpolygon $P^* \subseteq P$ that does not contain any blue points in the interior, nor any red points in the exterior and has smallest absolute angular change.

After an examination of the four possible cases (one or more blue points inside P with none or several red points inside P) we find that all the methods that were presented in Subsections 3.2.1 and 3.2.2 can easily be adapted to find the solution with the smallest absolute angular change. The running times for all methods remain the same.

3.3 Recoloring Approach in Detail

In the adaptation approach, we changed the boundary of the red region to bring blue points to the outside. However, if a blue point p is surrounded by red points, it may have been classified wrongly and recoloring it to red may lead to a more natural

boundary of the red region. Similarly, red points surrounded by blue points may have been classified wrongly and we can recolor them to blue.

In this section we present schemes for recoloring the given points, that is, assigning a new inside-outside classification. The starting point for our schemes is as follows: We are given a set P of n points, each of which is either red or blue. We first compute the Delaunay Triangulation $DT(P)$ of P . In $DT(P)$, we color edges red if they connect two red points, blue if they connect two blue points, and green otherwise. A red point is incident only to red and green edges, and a blue point is incident only to blue and green edges. To formalize that a point is surrounded by points of the other color, we define:

Definition 5 *Let the edges of $DT(P)$ be colored as above. Then the green angle ϕ of $p \in P$ is*

- 360° , if p is only incident to green edges,
- 0° , if p has at most one radially consecutive incident green edge,
- the maximum turning angle between two or more radially consecutive incident green edges otherwise.

We recolor points only if their green angle ϕ is at least some threshold value Φ . Note that if Φ has any value less than 180° , then it may be that a point is recolored red and blue indefinitely, with no termination. (A simple example consists of red points at $(-\epsilon, 1)$ and $(-\epsilon, -1)$, blue points at $(\epsilon, 1)$ and $(\epsilon, -1)$, for some small, positive ϵ , and a point at $(0,0)$, which repeatedly changes color if $\Phi < 180^\circ$.) So we assume in any case that $\Phi \geq 180^\circ$; a suitable value for the application can be found empirically. After the algorithm has terminated, we define the regions as follows. Let M be the set of midpoints of the green edges. Then, each Delaunay triangle contains either no point or two points of M . In each triangle that contains two points of M , we connect the points by a straight line segment. These segments define the boundary between the red and the blue region. Note that each point of M on the convex hull of the point set is incident to one boundary segment while the other points of M are incident to exactly two boundary segments. Thus, the set of boundary segments consists of connected components that are either cycles, or chains that connect two points on the convex hull. An alternative is to choose the separating polygon based on relative convex hulls. We define the perimeter of the separation to be the total length of the boundary cycles and chains. The intuition behind this approach is that the perimeter decreases with most recolorings and that we end up with a compact shape that separates the red and blue points. To get an even more compact shape in our experiments we defined the boundary segments via a point at a constant small distance from the red endpoint of each green edge. For green edges that were shorter than the fixed distance, we used the midpoint.

Before we present different recoloring schemes, we make the following basic observation.

Observation 1 *If we can recolor a blue point, then we do not destroy this option if we first recolor other blue points. We also cannot create possibilities for recoloring a blue point if we have only recolored red points before.*

We can now describe our first recoloring scheme, the preferential scheme. We first recolor all blue points with green angle $\phi \geq \Phi$ red, and afterwards, all red points

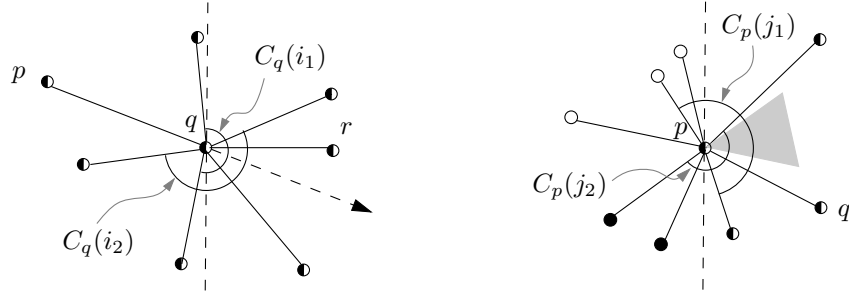


Figure 3.15: Illustrating Lemmas 11 and 12. Left: Either an edge or its opposite is in any coloring set. Right: If vertices to the left of p do not change color, but p does, then its color depends upon q . Points are shown solid (blue) or hollow (red) if they do not change color anymore; half-solid points indicate points whose color may still change.

with green angle $\phi \geq \Phi$ blue. It can occur that points that are initially blue become red, and later blue again. However, with our observation we can see that no more points can be recolored. Hence, this scheme leads to at most a linear number of recolorings. As this scheme gives preference of one color over the other, it is not fair and therefore not satisfactory. It would for example be better to recolor by decreasing green angle, since we then recolor points first that are most likely to be misclassified. Note that a red preferential scheme and a blue preferential scheme exist. They give the recolorings with the maximum number of red and blue points, respectively.

Another scheme is a true adversary scheme, in which an adversary may recolor any point with green angle $\phi \geq 180^\circ$. First, we show that the adversary scheme terminates after at most an exponential number of steps.

We observe that points on the boundary of the triangulation can be recolored at most once: such a point p is on the convex hull, and any edge sequence with angle greater than π must include both boundary edges incident to p . Thus, p can be recolored only to the color its two neighbors on the hull already have. Therefore, each point on the convex hull is recolored at most once.

Consider an edge \overline{pq} with p left of q . Unless q is the rightmost point on the convex hull, we can choose an *opposite edge* \overline{qr} such that there is no edge between \overline{qr} and the ray from q that goes in the direction opposite p , which is drawn dashed in Figure 3.15 (left). Fix a particular sequence of recoloring steps. If the color of point p is changed at step j , let $C_q(j)$ denote the *coloring set*, which is the maximal sequence of angularly consecutive green edges incident to q . Each coloring set for q must contain an edge or its opposite:

Lemma 11 *For any pair of an edge \overline{pq} with its opposite \overline{qr} , if point q changes color, then q receives either the color of p or of r .*

Proof: By the definition of an opposite edge, any coloring set $C_q(j)$ that does not contain \overline{pq} must include the opposite edge \overline{qr} for its angle to be at least π . \square

To show that any sequence of recoloring steps converges, we study the colors for a monotone chain: Starting with any edge $\overline{p_0 p_1}$, choose a *chain* of vertices p_0, \dots, p_m , ordered by x -coordinate, such that $\overline{p_i p_{i+1}}$ is opposite $\overline{p_{i-1} p_i}$, for all $0 < i < m$. This chain will end with p_m on the convex hull. Define the *color-change number* of this chain to be the number of integers $0 < i \leq m$ for which vertices p_{i-1} and p_i have different colors. Lemma 11 implies that recoloring of the interior vertices of this chain (i.e. vertices p_1, \dots, p_{m-1}) never causes the color-change number to increase.

We can start considering the sequence after the convex hull vertices have changed color. By starting a little inside the hull, we can show that the color-change number for some monotone chain must decrease.

Lemma 12 *Fix an integer $j > 0$ and let p be the point with smallest x -coordinate of all the points that are recolored in steps $\geq j$. Then point p is recolored a finite number of times.*

Proof: If p is recolored once, then the lemma is trivially true, so assume that p is recolored at least twice. Note that p cannot be on the convex hull.

We claim that p always receives the color of a point q to its right. But this would imply that the color-change number of a chain beginning with edge \overline{pq} decreases with each recoloring, so the maximum number of recolorings of p is the number of edges in such a monotone chain. Thus, it is sufficient to prove this claim.

Let j_1 and j_2 , with $j \leq j_1 < j_2$, denote the next two recoloring steps for p . The coloring sets $C_p(j_1)$ and $C_p(j_2)$ contain edges to neighbors to the left of p that have different colors, as shown in Figure 3.15 (right), since points to the left of p cannot change color by assumption. Thus, there is a wedge left of p between two different colors that cannot be contained in any coloring set $C_p(i)$ for steps $i \geq j$. All such coloring sets for p span the reflection of this wedge through p , which is shown shaded in Figure 3.15 (right). Any edge \overline{pq} that goes to the right inside this wedge, or is the next edge clockwise or counterclockwise of it, is contained in all coloring sets for p after step j , which means that p receives the color of q when recolored. \square

For the upper bound, observe that there are 2^n possible color assignments to points. As there is no cycling in the adversary scheme, that means it is not possible to return to a color assignment that has already occurred before, so the adversary scheme is finite. Therefore, the upper bound on the number of recolorings is $2^n - 1$.

Second we show that there exists a point set such that the number of recolorings is at least $\Omega(n^2)$. We arrange the n points as $n/3$ equilateral triangles that have the same center and orientation, but different sizes, so they contain each other. Each triangle initially only has points of one color, and the triangles have alternating colors from smallest to largest; see Figure 3.16. We call these equilateral triangles *layers*, to avoid confusion with the Delaunay triangles for this point set. The layers are numbered 1 and up from the inside out.

The point set is degenerate, and several Delaunay triangulations are possible. The freedom is only how to triangulate between two consecutive layers. We do so asymmetrically, as shown in the figure. For any layer i , one of its points has edges

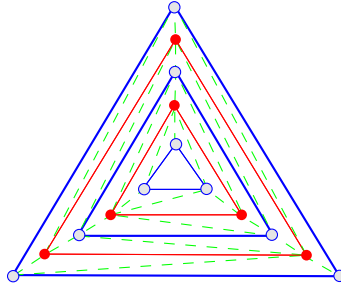


Figure 3.16: The construction for the quadratic lower bound of recolorings.

to all three points of the enclosing layer $i + 1$, and this point has green angle $> 180^\circ$. We can choose the relative sizes of the layers such that points on layers more to the inside always have the point with the largest green angle. Furthermore, we can make sure that after recoloring a point of a layer, the other two points will be recolored next since they will then have the largest green angle.

We choose the first (innermost) layer to be blue. By construction its points will be recolored first to be red (first the point with degree 5, then the point with degree 4, and then the point with degree 3), resulting in a first and second layer that are red. Now a point of the second layer has the largest green angle (the point that has edges to all three blue points of the third layer), and consequently the points of the second layer will be colored blue. Then the first layer will be colored blue again. In the next phase the first three layers will be colored red. Repeating this construction gives the lower bound of $\Omega(n^2)$. We can summarize:

Theorem 10 *Given any triangulation T of a finite number of points, each colored in one of two colors, then any sequence of recolorings is finite and the number of recolorings is at most $2^n - 1$. There exists a triangulated colored point set such that the number of recolorings is $\Omega(n^2)$.*

Finding a polynomial upper bound on the number of recolorings is difficult. The lower bound example shows that the total green angle need not decrease during every recoloring, so we cannot use this argument to bound the number of recolorings. Similarly, the number of green edges need not decrease. There are examples of recolorings that increase the number of points with green angle $\geq 180^\circ$. Recoloring a point with green angle $\geq 180^\circ$ can also increase the separation perimeter.

To select a practical recoloring scheme for implementation, we choose one that makes a reasonable choice on which point to recolor next, namely the one with the largest green angle. Furthermore, we only want to recolor a point if this leads to a decrease in the perimeter of the separating polygon. We call this the angle-and-perimeter scheme. Even with this choice, the best known upper bound on the number of recolorings is still $2^n - 1$.

To implement the algorithm efficiently, we maintain the subset of points that can be recolored, sorted by decreasing green angle, in a balanced binary search tree. We extract the point p with largest green angle, recolor it, and recolor the incident

edges. This can be done in time linear in the degree of p . We must also examine the neighbors of p . They may get a different green angle, which we must recompute. We must also test if recoloring a neighbor still decreases the perimeter length. This can be done for each neighbor in time linear in its degree. We summarize:

Theorem 11 *The running time for the angle-and-perimeter recoloring algorithm is $O(n + Z \cdot n \log n)$, where Z denotes the actual number of recolorings.*

3.3.1 The angle-and-degree scheme

In the angle-and-degree scheme we use the same angle condition as before, additionally we require that the number of green edges decreases. For any red (blue) point p , we define $\delta(p)$ to be the difference between the number of green edges and the number of red (blue) edges incident to p . We recolor a point if its green angle ϕ is at least some threshold Φ and its δ -value is larger than some threshold $\delta_0 \geq 1$. We always choose the point with largest δ -value, and among these, the largest green angle. In every recoloring step the number of green edges in $DT(P)$ decreases by $\delta(p)$, so we get a linear number of recolorings.

Theorem 12 *The running time for the angle-and-degree recoloring algorithm is $O(n^2 \log n)$.*

Remark: In practice, vertices in Delaunay triangulations have constant degree. In this case we obtain running times of the two recoloring schemes of $O((n + Z) \log n)$ and $O(n \log n)$.

3.4 Experiments

Using SPIRIT, we received four data sets with different numbers of red and blue points (in parentheses). The data sets are Eastanglia (14, 57), Midlands (56, 52), Southeast (51, 49), and Wales (72, 54). The red points were determined by Web searches using `www.google.uk` in September 2004 and trigger phrases such as ‘located in the Midlands’ and then extracting the names of the corresponding towns and cities in the search results as described in the introduction to this chapter and in [8]. Notice that Wales is not an imprecise region, but it can be used for the evaluation of our delineation methods.

We have implemented both the adaptation and the recoloring approaches using C++, and the libraries LEDA, CGAL and Qt. Although we made no attempt to minimize computation time, each of our tests took only a few seconds on an Intel-Xeon with a 2.80-GHz CPU and 2 GB memory under Linux-2.6. We show and discuss a few screen shots. Figure 3.17 features the adaptation method for two different values of α . The corresponding radius- α disk can be found at the right of each subfigure. Regarding the recoloring method we give an example of the angle scheme and of the angle-and-degree scheme; see Figure 3.18. In each figure blue points are marked by hollow circles and red points by black circles. Due to the α -shape that is used as initial region in the adaptation method, the area of the resulting region increases with increasing α ; see Figure 3.17. We found that good

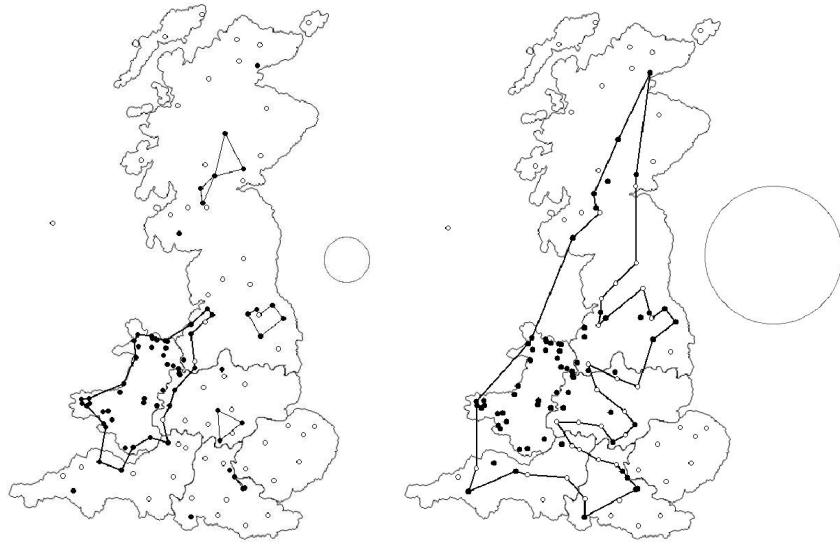


Figure 3.17: Regions for Wales computed by the adaptation approach. The α -values are shown as radius- α disks right of the figures.

values of α have to be determined manually. For smaller values of α , the α -shape is likely to consist of several connected components, which leads to strange results; see Figure 3.17 (left). Here, the largest component captures Wales quite well, but the other components seem to make little sense. For larger values of α the results tend to change very little, because then the alpha shape becomes similar to the convex hull of the red points. However, the value of α may not be too large since then outliers are joined in the α -shape and cause strange effects. For example, in Figure 3.17 (right) Wales has an enormous extent. When the initial value of α was well chosen, the results matched the region quite well for all our data sets, as far as this can be judged at all for imprecise regions.

For the angle scheme we found the best results for values of Φ that were slightly larger than 180° , say in the range 185° – 210° . Larger values of Φ severely restrict color changes. This often results in a large perimeter of the red region, which is not desirable. We compared the results of the angle scheme and of the preferential-blue and preferential-red scheme for $\Phi = 185^\circ$.

The preferential-red scheme recolors all eligible points from blue to red first and then all eligible points from red to blue. Only slight differences occurred on all four data sets.

The results strongly depend on the quality of the input data. Figure 3.18 shows this effect: Although Wales is contained in the resulting region, the region is too large. Too many points were classified falsely positive. The quality of the Eastanglia data was better, and the resulting region nearly matches the extent of Eastanglia (as generally considered).

For a small degree threshold, say $\delta_0 \leq 4$, the angle-and-degree scheme yielded nearly the same results as the angle scheme. This occurs because points having a



Figure 3.18: Region for Wales recolored by the angle scheme with $\Phi = 185^\circ$ (left) and for East Anglia recolored by the angle-and-degree scheme with $\Phi = 200^\circ$ and $\delta_0 = 4$ (right).

green angle larger than 180° are likely to have a positive δ -value. For increasing values of δ_0 the results depend less and less on the angle threshold Φ . If a point has a δ -value above 4, then its green angle is usually large anyway. However, if the main goal is not the compactness of the region, or if the input is fairly reliable, larger values of δ_0 can also yield good results; see Figure 3.18 (right), where only the two light-shaded points were recolored.

Comparing the adaptation and recoloring approaches shows that the two approaches behave similarly on the inner part, the ‘core’, of a point set. The main differences occur along the boundaries. The adaptation approach may produce more fjord-like boundaries than the recoloring approach. For example, compare Figure 3.17 (right) and Figure 3.18 (left).

Since Wales is a region with known administrative boundaries, we can compare it to the polygons we derived using our algorithms. In Table 3.2 we contrast the following data (related to the area of Wales): Wales correctly identified, Wales not identified, and regions incorrectly identified as Wales. On first glance the recoloring method performs better, as it identifies almost all of Wales correctly. However, this comes at the high cost of identifying regions twice as big as Wales itself incorrectly as Wales. The adaptation method yields the best result for $\alpha = 0.0068$, where the correctly identified part of Wales lies at 80% and the incorrectly as Wales identified region is only half the size of Wales. For smaller values of α both correctly and incorrectly identified regions decrease, whereas for larger values of α mainly the regions incorrectly identified as Wales increase. See Figures 3.19 and 3.20. Note, however, that a considerable part of the boundary of Wales is coast, and we do not have blue points in the water by default. This influences the results.

We now investigate how the parameters α and Φ of the adaptation and recoloring approaches, respectively, influence shape measures for polygons that are commonly used in geography [73, 130]. Such measures are compactness (A/r^2), circularity ratio (A/p^2), and form ratio (A/d^2), where A is the area of a given polygon P , r is the radius of the smallest enclosing circle of P , p is the perimeter of P , and d is the diameter of P . It turned out that the results of our experiments did not vary much between the different measures, so we only show how the compactness of the red region in our four data sets depends on α and Φ ; see Figure 3.21. In the left figure, the sudden jumps show where the number of points that are classified as outliers



Figure 3.19: The red regions of the input data of Wales (left) and the angle scheme with preference 'largest green angle' for $\Phi = 215^\circ$ (right).

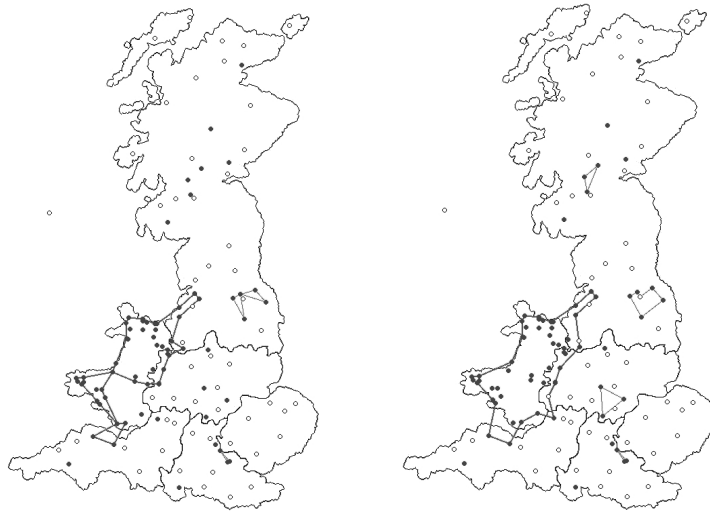


Figure 3.20: The red regions of Wales of the adaptation method for $\alpha = 0.0035$ (left), and $\alpha = 0.0068$ (right).

Table 3.2: Comparison of derived regions of Wales with administrative borders of Wales. Percentages are given with respect to the actual size of Wales.

	% of Wales identified	% of Wales not identified	% of incorrectly identified as Wales
Adaptation method			
$\alpha = 0.0035$	52.2	47.8	23.6
$\alpha = 0.0045$	62.3	37.7	33.1
$\alpha = 0.0068$	81.6	18.4	41.1
$\alpha = 0.008$	81.6	18.4	46.7
$\alpha = 0.0114$	83.5	16.5	103.1
Recoloring method			
$\Phi = 185$	97.1	2.9	243.7
$\Phi = 215$	96.3	3.7	190.1
$\Phi = 260$	97.2	2.8	240.8

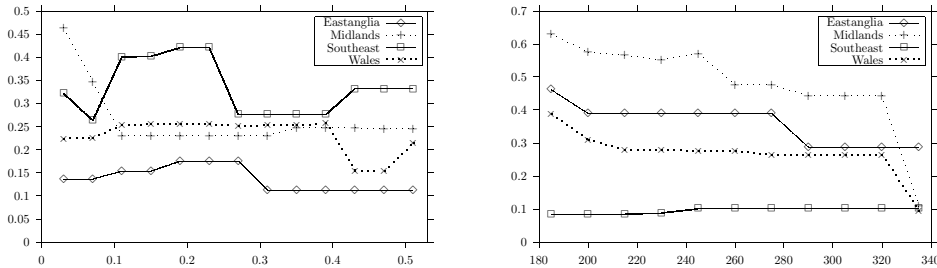


Figure 3.21: Compactness A/r^2 for various values of α (left) and Φ (right).

changes. Since the recoloring method can give rise to several red regions, we only counted the largest one in Figure 3.21 (right).

Finally, we investigate the performance of the angle scheme for the recoloring approach on random data. We are especially interested in the number of recolorings under the angle scheme, since we have not been able to show a polynomial upper bound on the number of recolorings. In order to imitate real instances we draw the point sets from the following distribution: from a given number n of points we draw $n/2$ blue points uniformly distributed from the unit square centered at the origin O , $n/8$ red points from a square of side length $7/8$ centered at O , and the remaining $3n/8$ red points from a circle of radius $1/4$ also centered at O . For an example with $n = 200$, see Figure 3.23 (left). The result of the angle scheme recoloring for $\Phi = 210^\circ$ is depicted in Figure 3.23 (right).

Figure 3.22 (left) shows how the number of recolorings (y -axis) depends on the angle threshold Φ (x -axis) for random data sets of size $n = 800$. Figure 3.22 (right) shows how the number of recolorings depends on the number of points (x -axis) for a

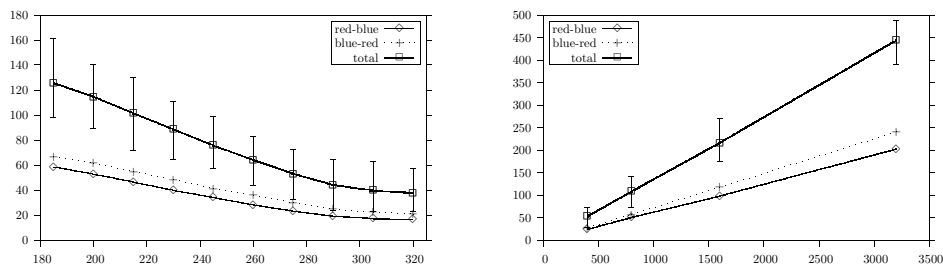


Figure 3.22: Number of recolorings as a function of Φ (left) and as a function of the number of points (right) respectively for random data.

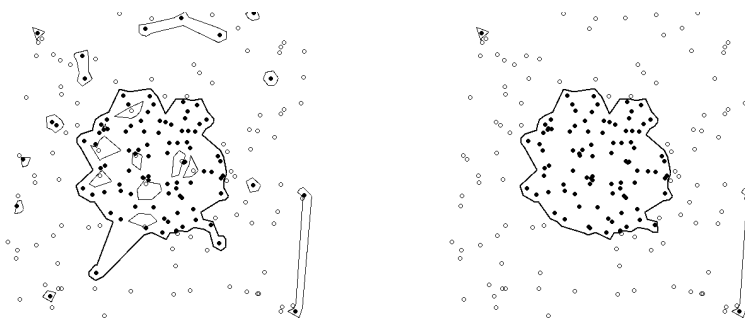


Figure 3.23: The red regions of a random point set before (left) and after (right) recoloring by the angle scheme for $\Phi = 210^\circ$.

fixed angle threshold $\Phi = 210^\circ$. Both figures show how often blue nodes were colored red, how often red nodes were colored blue, and the sum of these two numbers. All numbers were averaged over 30 random point sets of a given size or for a given threshold. The error bars show the minimum and the maximum total number of recolorings that occurred among those 30 point sets. It is interesting to see that the minimum, maximum, and average number of recolorings do not vary much, and that the number of recolorings seems to scale perfectly with the number of points.

The strength of the recoloring method is its ability to eliminate false positives provided that they are not too close to the target region. Since the differences between the various schemes we investigated seem to be small, a scheme that is easy to implement and terminates quickly can be chosen, e.g., the preferential-red or preferential-blue scheme.

3.5 Concluding Remarks

This chapter discussed the problem of computing a reasonable boundary for an imprecise geographical region based on noisy data. The methods presented here can also be used to delineate the extent of other imprecise regions, e.g., regions where a certain dialect is spoken, an ethnic minority dominates or a traditional custom prevails. Using the Web as a large database, it is possible to find cities

and towns that are likely to be inside and cities and towns that are likely to be outside the imprecise region. We presented two basic approaches to determine a boundary. The first approach formulated the problem as a minimum perimeter polygon computation, based on an initial red polygon and additional points that must be outside (blue) or stay inside (red). For the case of one blue point inside the red polygon we presented a linear time algorithm, and an $O(n \log n)$ time algorithm if there are also red points that must stay inside. If there are m blue points and no red points inside, we gave an $O(m^3 n^3)$ time algorithm, and a variation that allows misclassified points. For the case of m red and blue points inside, we presented a fixed-parameter tractable algorithm running in $O(C^{m \log m} \cdot n)$ time. This algorithm can also be adapted to deal with misclassified points.

The second approach involved changing the color, or inside-outside classification of points if they are surrounded by points of the other color. We proved a few lower and upper bounds on the number of recolorings for different recoloring criteria. An interesting open problem is whether the most general version of this recoloring method has a polynomial upper bound on the number of recolorings. In tests, the number of recolorings was always less than n .

We also presented test results of our algorithms, based on real-world data and random data. The real-world data included places obtained from trigger phrases for several British regions. Indeed the data appeared to be noisy, which is one reason why the boundaries determined were not always acceptable. Using post-processing, it is possible to improve the shape of the boundaries in various ways. Also, with the expanding of the Web, more reliable data may present itself automatically, when smaller towns also make their appearance on the Web and can be found using trigger phrases.

A natural extension is to assign weights or confidence values with red and blue points. Cities that appear many times in a trigger phrase are surely inside, and cities mentioned often on the Web, but never in a trigger phrase, are almost surely outside. For small towns not mentioned, the situation is less clear.

4

COMPUTING MULTISCALE GRADIENT AND ASPECT MAPS

This chapter is the last of this thesis dealing with the delineation of imprecise geographical regions, and it is different from the two previous chapters. The regions we will delineate in the following, are regions with gradient values in a certain interval and regions with constant aspect values, which do not make much sense when used stand-alone. However, for a geomorphologist, gradient and aspect present valuable clues about a terrain, and she or he will be able to derive the boundaries of landforms like hills, valleys, or mountain ranges.

Geomorphometry is concerned with the precise measurement and quantitative description of the shape of landforms. Given a terrain, the most important measures to classify landforms are slope, as well as profile curvature and plan curvature [157]. The value for slope at each point of the terrain is usually divided into *gradient*, i.e. the steepness of the slope, and *aspect*, the cardinal direction in which the slope faces. Using such measures and classifications, the goal is for example to derive drainage maps, specify areas in mountains that have high danger of avalanches, or study how a certain area has been formed.

Using some numerical value for gradient, and the classification convex or concave for plan and profile curvature, it is possible to identify landforms like convergent and divergent shoulders, footslopes, or crests, swales, and plains (see e.g. [61, 116, 133]). Slope can also be used to compute shaded relief maps, for irradiance mapping [27], and for parametric terrain classification.

Contour maps of terrains where each curve represents constant height values are very common. Especially when the original data is not available, they are used for example for digital terrain modelling [31, 44, 148]. However, as contour maps lack morphometric information between the contour lines, the outcome may not be satisfactory. Maps with curves representing constant gradient values — for simplicity we will call them *isogradients* — or areas of constant aspect (*isoaspects*) can aid in digital terrain modelling.

Other geomorphological features in terrains are *critical lines*. Critical lines are features where the slope or curvature changes abruptly, like ridges and valleys. They can be determined by identifying critical points such as maxima, minima, and saddle points of the terrain, and connecting them [141]. Critical points can be detected from their slope and curvature values [175], or using drainage network and catchment area delineation [178]. For applications like extracting volcano-tectonic features it is necessary to also find lines that define a break of slope without being a ridge or a

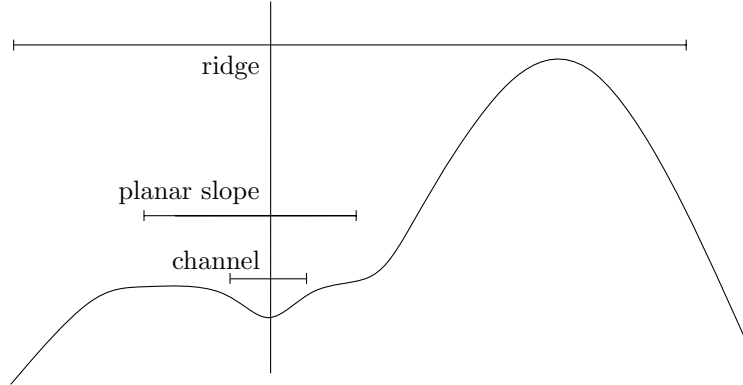


Figure 4.1: The same point can be classified differently at different scales, based on [65].

valley. These lines can be found with the method introduced in this chapter. The terrain can also be partitioned into areas having the same curvature, and the critical lines are identified as the boundaries of these areas [62]. Also for this application, maps that give scale-dependent isogradient lines can be useful. An important influencing factor of geomorphometry is at which scale we are studying the terrain. Visualization of a terrain at a large scale provides the most detail, and the smaller the scale, the more information is lost due to generalization. Another problem for the classification of landforms with respect to scale is that the morphometric class may change with different scales. See Figure 4.1 taken from Fisher et al. [65]. The scale of the terrain model also influences for example the area of a lake or the length of a seashore. This influence can be crucial when the investigated spatial object already has fuzzy boundaries [26, 37]. The book edited by Tate and Atkinson [161] presents research on scale related issues in GIS.

As a new method that yields isogradient lines and isoaspect maps, we introduce *local gradient* and *local aspect* for each point of a terrain. The basic idea is to define the local gradient for a point based on some neighborhood of the point. The size of the neighborhood can be chosen, which makes the definition scale dependent. Our definitions yield a continuously changing value of the local gradient value on for example TINs (Triangular Irregular Networks), whereas the standard definition on a TIN does not yield continuity. Continuity is important for the generation of isogradients.

Parametric terrain classification is concerned with the classification of landscapes by e.g., land features, surficial materials or geomorphological processes. For this application, it is important to determine *generalized* isogradient lines. These can be computed in various ways. Firstly, we can generalize the terrain and then derive isogradients from this. Alternatively, we can first derive the isogradients from the terrain and then simplify them using line simplification. Our definitions provide a third method.

This chapter is structured as follows: In Section 4.1 we will introduce four different scale dependent definitions for local gradient and local aspect for every point on a terrain. In the first two definitions for local gradient (aspect), we average all gradient (aspect) values in a certain neighborhood around a point. We can either apply uniform or non-uniform weighing for the averaging. These two definitions lead to algorithms to compute local gradient (aspect) for every point on a terrain with a running time of $O(mn)$. Here, m denotes the maximum number of edges intersected by the neighborhood of any point. In the third definition we choose the local gradient at a point to be the maximum gradient value to any other point inside the neighborhood. Local aspect is then defined by the vector to the point that realizes this maximum. We can compute local gradient and local aspect for the whole terrain in $O(n \cdot m^{3+\epsilon})$ time, where $\epsilon > 0$ is an arbitrarily small constant. The fourth definition for local gradient (aspect) at a point is the maximum average gradient (aspect) over a diameter of the neighborhood. This definition does not lead to an algorithm that can analytically derive a solution on a TIN, therefore we only present a heuristic to compute approximate isogradients and isoaspects in $O(nm)$ time. These results are given in Section 4.2 and apply for a square neighborhood around each point of a TIN. We have implemented all four methods for grid data and we compare the results for different sizes of the neighborhood in Section 4.3. Conclusions and an outlook on possible future work can be found in Section 4.4.

4.1 Definitions for Local Slope

Assume we are given a terrain T where every point has well defined values for slope. By default, slope is defined by a plane tangent to the surface at any given point. It consists of two components: the gradient, which is the maximum rate of change of altitude, and the aspect, the compass direction of the maximum rate of change. We will refer to these definitions as the *standard definitions* of gradient and aspect, and denote them by $\hat{F}_g(x, y)$ and $\hat{F}_a(x, y)$, respectively. Usually, gradient is measured in percent or degrees, and aspect in degrees, which are converted to a compass bearing. From these given values we want to derive a scale dependent contour map with lines or areas representing constant values of gradient or aspect. Throughout this chapter, we will call them *isogradients* and *isoaspects*, respectively.

We introduce the notion of *local slope*, composed of *local gradient* and *local aspect*, for each point p of the terrain T , using some neighborhood around p and denote them by $F_g(x, y)$ and $F_a(x, y)$. A natural choice for such a neighborhood can be derived from a disk in the xy -plane with some prespecified radius r , centered at p , which we will project vertically onto the terrain T and denote by D_r .

It is obvious that the choice of the size of the neighborhood influences the resulting isogradients and isoaspects and is therefore very important. If we choose a small radius r , such that we take only points close to p into account, we expect to get many, detailed isogradients. If we choose a large value for r , we expect to get few, more smooth isogradients.

The basic idea is to define for every point p that is the center of a disk D_r the local gradient and aspect depending on the points that lie inside D_r . We can do this in the following four ways:

1. Uniform weighing over the neighborhood D_r
2. Non-uniform weighing over the neighborhood D_r
3. Maximum value in the neighborhood D_r
4. Uniform weighing over a diameter in the neighborhood D_r

Note that standard gradient and aspect values need not be defined everywhere, e.g., on the edges and vertices of a TIN. In this case we can either exclude these points from the computation, or assign values from an adjacent triangle. All four definitions for local gradient lead to continuous functions $F_g(x, y)$ on the whole terrain.

The local aspect value at each point is a function $F_a(x, y) : \mathbb{R}^2 \rightarrow [0^\circ, 360^\circ) \cup \{\text{Flat}\}$, where the degree value is usually divided into a number of discrete classes (e.g., North, East, West, South). The additional value Flat is assigned to horizontal parts of the terrain, where the aspect is undefined. In this context, the aspect function is continuous at p , if the aspect at p is Flat, or for every point in an ϵ -neighborhood of p the aspect value changes only by a small amount δ_ϵ . The first two definitions given above lead to continuous functions $F_a(x, y)$ for the local aspect, the last two lead to not continuous functions.

We will first give the basic definitions and properties of local gradient and local aspect for a circular neighborhood D_r with radius r at any point p of the terrain.

4.1.1 Uniform weighing over a neighborhood

Gradient

In the uniform weighing over the given neighborhood D_r , we compute the average gradient sum of all points in D_r . The local gradient at a point p is defined by the following equation:

$$F_g(p) = \frac{1}{\text{area}(D_r)} \int_{p' \in D_r} \hat{F}_g(p') \, dx dy \quad (4.1)$$

Here, $p' = (x, y)$ is some point in the neighborhood D_r , and $\hat{F}_g(p')$ is the gradient according to the standard definition.

Gradient is usually considered to be a scalar value. However, in the standard definition, the gradient at p is derived from the tangent plane at p . Therefore, we have the choice whether we use the scalar or the vector gradient $\hat{F}_g(p')$ in Equation (4.1). The final, local gradient $F_g(p)$ will always be represented as a scalar value. The following example shows that this makes a difference. If we have two equally sized, adjacent regions with the same standard gradient value, say 10, and their outer normals pointing in opposite cardinal directions, say North and South, we will get as local gradient the value 10, when treating standard gradient as a scalar. Another way to look at it is to treat standard gradient as a 3-dimensional vector at each point on the terrain T , and to compute the vector sum of all 3D gradient vectors inside D_r . We take as local gradient at p the value represented by the resulting vector, which gives as local gradient the value zero in this example.

When treating gradient as vector, and the neighborhood cuts off the terrain at the same height everywhere at its boundary, the resulting gradient should indicate gradient zero. In case of a piecewise linear terrain in two dimensions, we can achieve

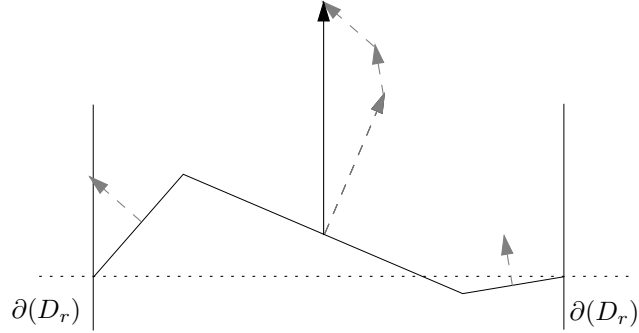


Figure 4.2: When the terrain is cut off at the same height at the boundary of D_r (∂D_r), the resulting gradient vector should point vertically upwards, indicating a value zero.

this by normalizing the y -component of the normal vectors to 1, before weighing it with the projected length of the terrain. The local gradient at p is derived from the weighted vector sum of all normal vectors. It is easy to show that in case of a piecewise linear terrain in two dimensions, the local gradient is a vector that points vertically upwards (see also Figure 4.2). Therefore, when treating the gradient as a vector of the terrain, we normalize the z -component of the vector at every point before integration.

In both cases, whether we treat the given gradient value as a scalar or as a vector, it is easy to see that the uniform weighing leads to a continuous function F_g for local gradient. Hence, isogradients will generally be closed loops or end at the boundary of T .

Aspect

As we want the local aspect to correspond to the local gradient at each point p , we will use the same uniform weighing and z -normalization as for the gradient given above. When treating gradient as a vector, we get the same result by simply computing the resulting vector gradient at point p and deriving the aspect vector directly from it. Note that we still need to project the result into the xy -plane and normalize it onto the unit circle centered at p .

It is easy to see that for example on a TIN, the standard aspect values given for each point may jump from one of the possible values to any other one as a point passes an edge of the terrain. However, with our method of uniform weighing, i.e., averaging over the neighborhood D_r , it is clear that the local aspect function F_a becomes continuous, which means that it can only change to adjacent values (or Flat) with respect to the circular scale of cardinal directions.

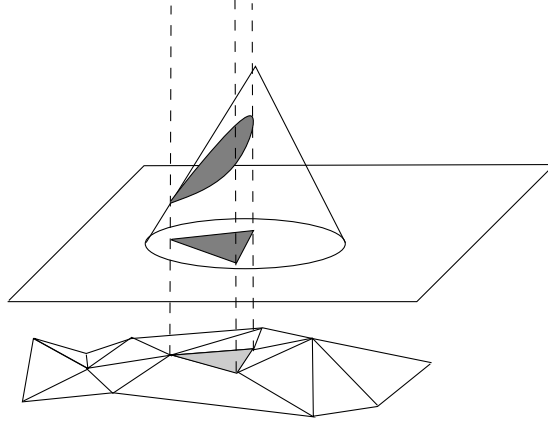


Figure 4.3: Computing the non-uniform weight for one triangle of a TIN is equivalent to computing the intersection of the weight cone with the prism erected on the triangle.

4.1.2 Non-uniform weighing over a neighborhood

Gradient

In the non-uniform weighing we give a higher importance to points that are closer to p and a lower importance to points that are closer to the rim of the disk D_r . We do this by a weight that decreases linearly with the distance to p . The weight values themselves form a cone with its tip at p . On the rim of D_r and outside D_r , the weight is zero.

Computing the weighted average over all points p' inside the disk D_r is equivalent to computing the height of the point on the cone C directly above p' times the standard gradient value at the point p' . See Figure 4.3 for an illustration of this definition on a TIN.

Stated more formally, we get the following integral representing the local gradient at point p :

$$F_g(p) = \frac{1}{\text{vol}(C)} \int_{p' \in D_r} \left(h - \frac{h}{r} \sqrt{x^2 + y^2} \right) \cdot \hat{F}_g(p') \quad dx dy \quad (4.2)$$

Here, h denotes the height of the cone, r the radius of the disk, x and y are the coordinates of a point p' with respect to $p = (0, 0)$, the center of D_r , and $\hat{F}_g(p')$ denotes the gradient at p' according to the standard definition.

Again, we can treat the gradient value $\hat{F}_g(p')$ as either a scalar or a vector. In the first case we compute the local gradient value by integrating over all weighted values and dividing by the total volume of the cone. In the second case, when treating the gradient value as vector, we again need to normalize the z -component before the weighing, and the local gradient value is a weighted vector sum as before. The non-uniform weighing method also gives a continuous function F_g for local gradient.

Aspect

For the local aspect vector at a point p , weighted non-uniformly over the disk, we can use the same model with the linearly decreasing weight that corresponds to the local gradient. The local aspect is the weighted vector sum with the resulting vector at p , projected into the xy -plane and normalized onto the unit disk centered at p .

The model of non-uniform weighing also gives a local aspect function F_a that is continuous, which means that its values can only change to adjacent values in the classified situation.

4.1.3 Maximum value in a neighborhood**Gradient**

In the maximum value method for a neighborhood D_r , we set the local gradient at a point p to be the absolute maximum gradient value from p to any other point p' inside D_r . The gradient between two points p and p' in space is defined as their z -distance divided by their Euclidean distance in the xy -plane, which leads to the following definition for the local gradient at a point p :

$$F_g(p) = \sup_{p' \in D_r \setminus \{p\}} \frac{dist_z(p, p')}{\sqrt{dist_x(p, p')^2 + dist_y(p, p')^2}} \quad (4.3)$$

Note that this definition gives local gradient values that are at least as large as the standard gradient definition. As the local gradient is not averaged, but depends only on one single value, there is no distinction between a scalar or vector version. Furthermore, note that the point p' that gives the maximum gradient value may not be unique. This model also leads to a continuous function F_g for local gradient.

Aspect

As local aspect at p we choose the vector between p and the point that gives the maximum gradient and project it into the xy -plane and onto the unit circle centered at p . When there is more than one point p' giving the maximum gradient, the aspect is generally not well-defined. A possible solution could be to choose the point p' that has the largest z -distance to p , but this is an arbitrary choice.

Note that this definition does not lead to a continuous local aspect function F_a , as whenever the point of maximum gradient changes, the local aspect vector jumps directly to the point of the new maximum. This means that the aspect value derived from this vector can change abruptly, e.g., from the class North directly to the class South, without passing through any intermediate classes or Flat first.

4.1.4 Uniform weighing over a diameter of the neighborhood**Gradient**

The uniform weighing over a diameter method is in some way a combination of the first and the third method presented in the previous subsections. As the standard slope value is realized in one direction, it is natural to take points in only one direction into account for the averaging in a local slope definition.

We define as local gradient the maximum average gradient over all line segments ℓ that are diameters of D_r . Any such line segment ℓ has length $2r$ in case of a circular neighborhood. The local gradient at a point p is defined by the following integral:

$$F_g(p) = \max_{\phi} \frac{1}{2r} \int_{p' \in \ell(p, \phi)} \hat{F}_g(p') dx dy \quad (4.4)$$

Here, ϕ denotes the angle of the line segment ℓ and the x -axis, and $\hat{F}_g(p')$ denotes the gradient at p' according to the standard definition.

As we are again taking an average value for local gradient, we can treat the gradient value $\hat{F}_g(p')$ as either a scalar or a vector for the averaging. In the first case we compute the local gradient value by integrating over all values and dividing by the length of ℓ inside the neighborhood. In the second case, when treating the gradient value as vector, we again need to normalize the z -component before integration, and the local gradient value is a vector sum as before. The uniform weighing over a diameter method also yields a continuous function F_g for local gradient.

Aspect

For local aspect, we use the corresponding definition as for gradient, i.e., we take the line ℓ that leads to the maximum average gradient and compute the average weighted aspect using the vectors from all points that lie on the line and inside D_r . We need to project the resulting vector into the xy -plane and normalize it onto the unit disk centered at p .

This definition of local aspect does not lead to a continuous function F_a . Although the local gradient is a continuous function, the diameter leading to the maximum average gradient may change abruptly.

4.2 Algorithms for Local Slope and Isolines

In this section we will describe efficient algorithms to compute an explicit representation of the local gradient and local aspect on the whole terrain. This will allow us to determine isogradients and isoaspects in a simple way. We will focus on presenting the algorithms for a TIN. For the first two methods, where we compute the weighted average, we need to know the area of projection of each triangle that is intersected by the neighborhood. This area of intersection is given by a function in the coordinates of p . In case of a circular neighborhood, this function may consist of up to a linear number of terms, all involving square roots. Such functions generally cannot be simplified, and hence, the equations describing isogradients are too complex to be used.

Therefore, we restrict ourselves to the case of a square neighborhood with side length $2r$, denoted by D_r . In this case, the area of intersection is given by a quadratic function, hence the problems mentioned above do not occur. We observe that all algorithms can easily be adapted from square neighborhoods to regular polygons, if a better approximation to a circular neighborhood is desired. Furthermore, our focus will lie on computing the local scalar gradient for each method; however, computing the local vector gradient and local aspect is very similar for all four methods.

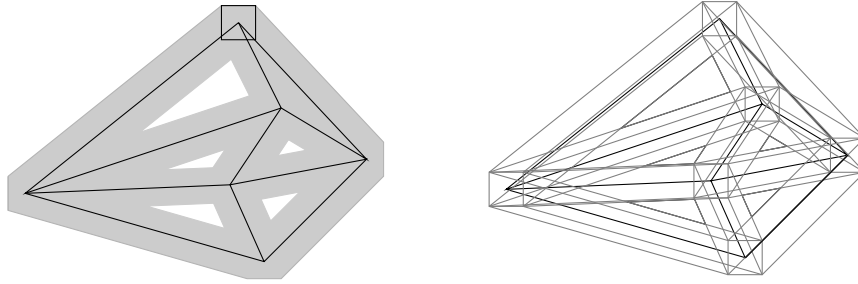


Figure 4.4: Left: The Minkowski sum (gray) of a triangulation and a square D_r with fixed orientation. Right: The subdivision S , such that for any placement of a reference point of D_r inside a cell of S the edges of D_r intersect the same features of T' .

Note that on the edges and vertices of a TIN, the standard gradient and aspect are not defined. However, in our definitions of local gradient and aspect we assume that every point inside the neighborhood has a value for gradient and aspect. We can overcome this problem by either excluding these points from the neighborhood and hence from the computation, or by assigning the value from any neighboring triangle.

As mentioned in the last paragraph, we want the local gradient at each point p to be determined by a function $F_g(x, y) \rightarrow \mathbb{R}$ expressed in the coordinates of p . It is therefore natural to subdivide each triangle of the TIN into cells such that for each point inside a cell, the function $F_g(x, y)$ is determined by the same features, i.e., the same edges or vertices of the TIN. Whenever the boundary of such a cell is crossed, the list of features influencing $F_g(x, y)$ changes. When this function is given in each cell, it can be evaluated in constant time to compute the value for local gradient at each point p . Also, for every chosen gradient value we can compute the isogradient in one cell in constant time by setting the function of that cell equal to the chosen value. The resulting equation gives the isogradient in that cell. We use $f_g(x, y)$ to denote the function $F_g(x, y)$ inside one cell only. All functions $f_g(x, y)$ for all cells together define $F_g(x, y)$.

Van Kreveld et al. show in [168] how to compute the *placement space* for a general subdivision and a square, such that for each cell of the placement space a reference point of the square can be placed such that the sides of the corresponding square intersect only fixed sets of edges of the subdivision. They prove bounds on the number of cells and on the running time for computing the placement space.

However, in our case we have a triangulation instead of a general subdivision, and hence, we can achieve refined bounds. Let T be a TIN, which we project into the xy -plane to get a planar triangulation T' with n edges. Let m be the maximum number of edges in S that are intersected by any placement of a square D_r . This number can be as large as $O(n)$, but typically it is much smaller.

Analyzing the number of distinct placements of a square D_r with fixed size and orientation in a triangulation T' can be achieved by considering the Minkowski sum

of D_r and T' . It can be visualized by placing the center of D_r (which is chosen to be the reference point) at a vertex of T' , sliding D_r along each edge of T' , and taking the union of all generated polygons. This is shown in Figure 4.4, left. Note that the white triangles are not part of the Minkowski sum; they depict placements of the square such that its sides do not intersect any of the edges of the triangulation. Any point in T' can be covered by at most m Minkowski sums of D_r and an edge. This is easy to see, as otherwise there must be at least one square intersecting more than m edges of T' . Furthermore, the boundaries of every pair of Minkowski sums can have only two proper intersection points [96]. With these observations, we can use a result of Sharir [153], which gives us an upper bound of $\Theta(mn)$ on the number of combinatorially distinct placements of D_r on T' .

We can improve the running time to compute the placement space of D_r on T' . First we compute the Minkowski sums of the triangulation T' with each corner of a square D_r with side length $2r$. This gives four equivalent triangulations, each translated by $+r$ or $-r$ in x and y direction. As these triangulations are simply connected and planar, we can directly use an algorithm of Finke and Hinrichs [64] three times to compute their overlay in $O(n + mn) = O(mn)$ time. To get the final subdivision S (see Figure 4.4, right), we also need to insert the square centered at each original vertex of the triangulation. We separately insert the vertical and horizontal sides of D_r , which each connect two of the translated vertices. We do so by traversing the cells of the subdivision S between the two vertices that are connected, and adding a new edge and a vertex for every intersection point of a side of a square with an existing edge of the subdivision. Observe that every cell of S has constant complexity because T' is a triangulation. This property remains valid when horizontal and vertical sides of the squares are inserted. Hence, any side of a square can be inserted in time linear in the added complexity of the subdivision, that is, in the number of intersection points of the new edge. There can only be $O(m)$ intersected edges per inserted side, therefore all edge insertions take $O(nm)$ time altogether. We summarize this in the following lemma:

Lemma 13 *Let E be the set of n edges of a triangulation T , and let D_r be a square of fixed size and orientation. Let m be the maximum number of edges intersected by any placement of D_r . There are $O(mn)$ distinct subsets of edges of E intersected by different placements of D_r , and the subdivision representing all distinct placements of D_r with respect to T can be constructed in $O(nm)$ time.*

The combinatorially distinct placements of a square D_r on the terrain give a subdivision S of the projected terrain T' into smaller cells (see Figure 4.4, right). When placing the reference point (i.e., the center of D_r) anywhere inside such a cell, the sides of D_r will intersect the same sets of at most m features (i.e., edges or vertices) of T' .

4.2.1 Uniform weighing method

In each cell of the subdivision S the local gradient is given by a function $f_g(x, y)$, in the coordinates of the center of D_r . The gradient function also depends on the $O(m)$ terrain features the square D_r will intersect. We need to determine the local gradient function for each cell of the subdivision S . A straightforward way to do

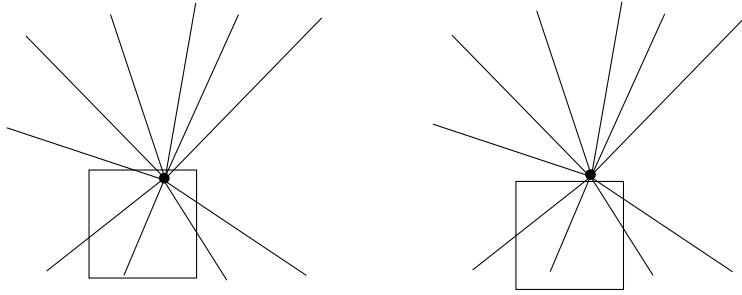


Figure 4.5: The set of features changes when an edge of D_r moves over a vertex of the terrain.

this takes $O(nm^2)$ time. However, by not explicitly computing the $O(m)$ features for each cell of the subdivision, we can improve this. The idea is to precompute the change of $f_g(x, y)$ across all cell boundaries. Since $f_g(x, y)$ is a quadratic function in x and y , we can store the change with each edge of S in $O(1)$ space.

We do this as follows (see Figure 4.5): We place D_r over a vertex of the triangulation such that the upper side lies directly above the vertex, and we compute the $O(m)$ features and the gradient function for this placement of D_r . Then we move D_r downwards until the upper side lies directly below the vertex. We compute the change of the set of intersected features and of the gradient function between these two placements of D_r . We store the change of the gradient function with the corresponding $O(m)$ horizontal edges of the subdivision S . Note that for any horizontal positioning of D_r with respect to the vertex, the same set of intersected features is changing when the upper side of D_r crosses the vertex. Therefore, we need to compute the change of the gradient function only once. We do this for all four sides of D_r and every vertex of the triangulation. Furthermore, for all other edges of S , the change of the gradient function is easy to determine in $O(1)$ time, as only one feature of T' changes when these edges are crossed. The whole precomputation takes $O(nm)$ time.

Afterwards, we traverse the subdivision S to determine the gradient function $f_g(x, y)$ of each cell. We choose a starting cell of S , in which we have to compute the $O(m)$ features and the gradient function. From there, we traverse S to an adjacent cell. Whenever the reference point crosses a boundary of a cell, there are two possible, corresponding events: Either a corner of the square D_r crosses an edge of the triangulation, or a side of D_r crosses a vertex of the triangulation. As the change of the gradient function is stored with each cell boundary, both types of events can be handled in constant time. The traversal of S to determine all gradient functions $f_g(x, y)$ in all cells of S takes $O(nm)$ time overall. We summarize:

Lemma 14 *Given a subdivision S representing the combinatorially distinct placements of a square D_r with fixed orientation and side length on a triangulation, we can determine the gradient functions $f_g(x, y)$ for all cells of S in $O(nm)$ time.*

It is easy to see that the same statements hold for the aspect function. We would not compute $f_a(x, y)$ in each cell, but the three-dimensional vector that represents

slope. Its three components are quadratic functions in (x, y) . From this vector, the aspect function can be determined in constant time per cell. Hence, the aspect function for the whole terrain can be computed in $O(nm)$ time as described above.

The gradient function $F_g(x, y)$ for the uniform weighing method is continuous over the whole terrain, but not differentiable at the boundaries of the cells. To determine the isogradients for any given value, we set the gradient function $f_g(x, y)$ in each cell to this value and the equation gives the curve in one cell. We can do this for each cell of S in constant time. As the local gradient function $F_g(x, y)$ is continuous, the isogradients are curves on the terrain, which are either closed loops or end at the boundaries of the terrain.

In a similar fashion, we can compute the local aspect values for each point p . It is important to note that the cells of the subdivision S , where we get a function $f_a(x, y)$ for the local aspect, do not correspond to the isoaspect areas. The boundaries of the isoaspect areas depend on the classification and can be determined by setting $f_a(x, y)$ to the value in degrees of the boundary between adjacent aspect values in the compass rose (e.g., 22.5° for N-NE). Wherever $f_a(x, y)$ is undefined, we assign the value Flat. We can summarize:

Theorem 13 *Let T be a TIN terrain with n triangles, let D_r be a square neighborhood with side length $2r$, and let m be the maximum number of edges of T intersected by any placement of D_r . We can compute in $O(nm)$ time a subdivision S of T that has $O(nm)$ cells, and for each cell the local gradient $f_g(x, y)$. The uniformly weighted local gradient $f_g(x, y)$ is a quadratic function. We can compute the corresponding subdivision for the uniformly weighted local aspect in the same time.*

4.2.2 Non-uniform weighing method

In the non-uniform weighing method, we give a higher importance to points that lie close to p and a lower importance to points that lie at the boundary of the neighborhood. We can do this by applying a weight that decreases linearly with the distance to p . As our neighborhood D_r is now a square, it is natural to use the L_∞ distance instead of the Euclidean distance. This way, the weight values form a pyramid with its tip at p . Computing the weight for each point inside a triangle is equivalent to computing the intersection volume of the pyramid P centered at p with a prism A , which has a triangular base and edges parallel to the z -axis. The volume of one such prism can be computed as

$$V_A = A_{xy} \cdot \frac{a + b + c}{3}, \quad (4.5)$$

where A_{xy} denotes the area of the projection of the terrain triangle onto the xy -plane and a, b, c depend on the coordinates of p and denote the side lengths of the three sides of the prism.

The algorithm to compute local gradient on the whole terrain is as follows: We construct the subdivision S of the projected terrain with D_r as before, where D_r is partitioned into four triangles by its diagonals. This increases the number of cells only by a constant factor. The gradient is represented by a cubic function $f_g(x, y)$ in

each cell of S . The gradient function $F_g(x, y)$ is continuous over the whole terrain, but not differentiable at the boundaries of the cells.

To determine the isogradients for any given value, we set the gradient function in each cell to this value and the resulting equation gives the curve in the cell. We can do this for each cell of S in constant time.

In a similar fashion, we can compute the local aspect function $F_a(x, y)$. The boundaries of the isoaspect areas can be determined in each cell by setting $f_a(x, y)$ to the value of the boundary between adjacent aspect values in the compass rose. Wherever the aspect function is undefined, we assign the value Flat. We conclude by the following theorem.

Theorem 14 *Let T be a TIN terrain with n triangles, let D_r be a square neighborhood with side length $2r$, and let m be the maximum number of edges of T intersected by any placement of D_r . We can compute in $O(nm)$ time a subdivision S of T that has $O(nm)$ cells, and for each cell the local gradient $f_g(x, y)$. The non-uniformly weighted local gradient $f_g(x, y)$ is a cubic function. We can compute the corresponding subdivision for the non-uniformly weighted local aspect in the same time.*

4.2.3 Maximum value method

In the maximum value method, the local gradient at p is defined by the maximum absolute gradient from p to any other point p' in D_r . We observe that on a TIN, the maximum gradient will occur between p and a vertex or an edge of the terrain or the boundary of D_r .

Computing the gradient from p to any other point p' inside D_r is straightforward. We can show that for an arbitrary line ℓ and any point p that is not on ℓ , the gradient between p and ℓ can have only one maximum. That means that the maximum gradient between a point p and an edge e of the terrain T either lies in the interior of e or at one of its endpoints. Note that when only part of an edge e lies inside D_r , we only consider this part for the computation of the maximum gradient. The point of maximum gradient on ℓ can be determined by applying analytical methods using Equation (4.3), the equation for ℓ and the coordinates of p . This takes constant time for each line and thus for each edge e of the terrain.

The algorithm to compute local gradient on the whole terrain is as follows: We generate the subdivision S from the projected terrain and D_r as described in the beginning of this section, and overlay it with the original triangulation T' to assure that p is inside a single triangle if p is in a cell of this new subdivision S' . We can further subdivide each of the $O(mn)$ cells such that in each cell of the refined subdivision S'' , there is exactly one vertex or edge of the TIN or part of the boundary of D_r that defines the maximum gradient. This means that the boundary of each cell of S'' is determined by exactly two features, and for every point on the boundary, the gradient to both of the features is the same. There are only three possible pairs of features to define such a boundary: two vertices, a vertex and an edge, or two edges of the TIN or on D_r . We can determine the boundaries between the cells of S' , where the maximum gradient is determined by the same feature, as follows: We compute the gradient function from every point p to each of the $O(m)$ features

inside each cell in $O(nm^2)$ time for all cells of S' . This will yield for each cell of the subdivision $O(m)$ surface patches in three dimensional space given by the fraction of a quadratic function and the square root of a quadratic function in x and y , derived from Equation (4.3). To find the one feature that determines the maximum for a given point, we need to find the pointwise maximum of all surface patches inside the cell. The pointwise maximum of m surfaces is called the upper envelope. It has complexity $O(m^{2+\epsilon})$ and can be computed in $O(m^{2+\epsilon})$ time [3], where $\epsilon > 0$ is an arbitrarily small constant.

The boundaries of the cells of the refined subdivision S'' are defined by the intersections of two surface patches on the upper envelope, and we project them onto the terrain. In each cell of S'' , there is only one feature such that the gradient from each point in the cell to this feature is maximal, and the gradient function $f_g(x, y)$ itself is the fraction of a quadratic function and the square root of a quadratic function.

It is easy to see that the upper envelope of all surface patches over the whole terrain is the representation of the gradient of every point of the terrain. The gradient function $F_g(x, y)$ is continuous, but not differentiable at the boundaries of the cells. For each cell, we can determine the isogradients as before in constant time.

For the local aspect function, we determine the vector from p to the point with maximal gradient and convert it to the aspect value. It is easy to see that the subdivision S'' is the same as the subdivision for local gradient. Note that the local aspect function $F_a(x, y)$ is not continuous.

Theorem 15 *Let T be a TIN terrain with n triangles, let D_r be a square neighborhood with side length $2r$, let m be the maximum number of edges of T intersected by any placement of D_r , and let $\epsilon > 0$ be an arbitrarily small constant. We can compute in $O(n \cdot m^{3+\epsilon})$ time a subdivision S of T that has $O(n \cdot m^{3+\epsilon})$ cells, and for each cell the local gradient $f_g(x, y)$. The maximum value local gradient $f_g(x, y)$ is a fraction of a quadratic function and the square root of a quadratic function. We can compute the corresponding subdivision for the maximum value local aspect in the same time.*

4.2.4 Uniform weighing over diameter method

In the uniform weighing over diameter method, the local gradient is defined to be the maximum average gradient over all diameters of D_r . We observe that for a square neighborhood D_r the diameter length lies in the interval $[2r, 2\sqrt{2}r]$, depending on the angle of the diameter with the x -axis, and is not a fixed value. For each point p on the terrain, the local gradient is determined by a function in (x, y, ρ) , where (x, y) are the coordinates of p and ρ denotes the angle of the diameter with the x -axis. To compute the maximum average gradient at any point p , we have to determine the value of ρ that maximizes the gradient function. This cannot be done analytically, not even for a given point p , because it requires solving a polynomial equation in ρ that can be of maximum degree m . Therefore, for this method, we only give a heuristic to approximate the local gradient at each point. We do this by computing the value of the gradient function for a constant number k predefined values of ρ ,

and then taking the maximum of these values.

The heuristic to compute an approximate local gradient on the whole terrain is as follows: We compute the subdivision S of the projected terrain with D_r , where D_r is partitioned into $2k$ pieces by k diameters. We assume that k is a constant, and each neighboring pair of the k diameters encloses an angle $\phi = 360/k$ degrees. This increases the number of cells only by a constant factor k . We can further subdivide each of the $O(nm)$ cells, such that in each cell of the refined subdivision S' the local gradient for each point is defined by the same value of ϕ . Inside each cell of S' , the local gradient is defined by a linear function $f_g(x, y)$.

We can determine the boundaries between the cells of S' , where the local gradient is defined by the same value of ϕ as follows. For each point p in a cell of the subdivision S , we compute the average gradient over a diameter through p for a set of k predefined values for ϕ . In each point and for a fixed value of ϕ , we get k linear functions $f_g(x, y)$ defining the average gradient over the diameters. We can find the maximum average gradient in each cell of S by computing the upper envelope of all k linear functions in $O(k \log k)$ time, which is constant for constant k . The intersection of two functions of the upper envelope gives the boundaries of the refined subdivision S' , which can have $O(k)$ cells for each cell of S .

As before, the upper envelope of all linear functions over the whole terrain is the representation of the gradient of every point of the terrain. The resulting gradient function $F_g(x, y)$ is continuous, but not differentiable at the boundaries of the cells. For each cell, we can determine the isogradients in constant time as before.

The local aspect value is computed in a similar fashion. It is easy to see that the subdivision S is the same as the subdivision for local gradient. The representation of the local aspect of every point is not continuous, as the diameter leading to the maximum average gradient may change abruptly whenever a cell boundary is crossed.

Theorem 16 *Let T be a TIN terrain with n triangles, let D_r be a square neighborhood with side length $2r$, and let m be the maximum number of edges of T intersected by any placement of D_r . We can compute in $O(nm)$ time a subdivision S of T that has $O(nm)$ cells, and for each cell the local gradient $f_g(x, y)$. The approximation of the local gradient $f_g(x, y)$, uniformly weighted over a diameter, is a linear function. We can compute the corresponding subdivision for the local aspect, uniformly weighted over a diameter, in the same time.*

We observe that, as we only average over a number of diameters of the neighborhood D_r , the shape of D_r does not directly influence the computations. Therefore, in our approximation of the uniform weighing over a diameter a circular neighborhood can be used as well.

4.3 Experimental Results for Grid Data

We have implemented our methods for DEM data in Java and compared them for data of different types of terrain. We downloaded the data set presented here

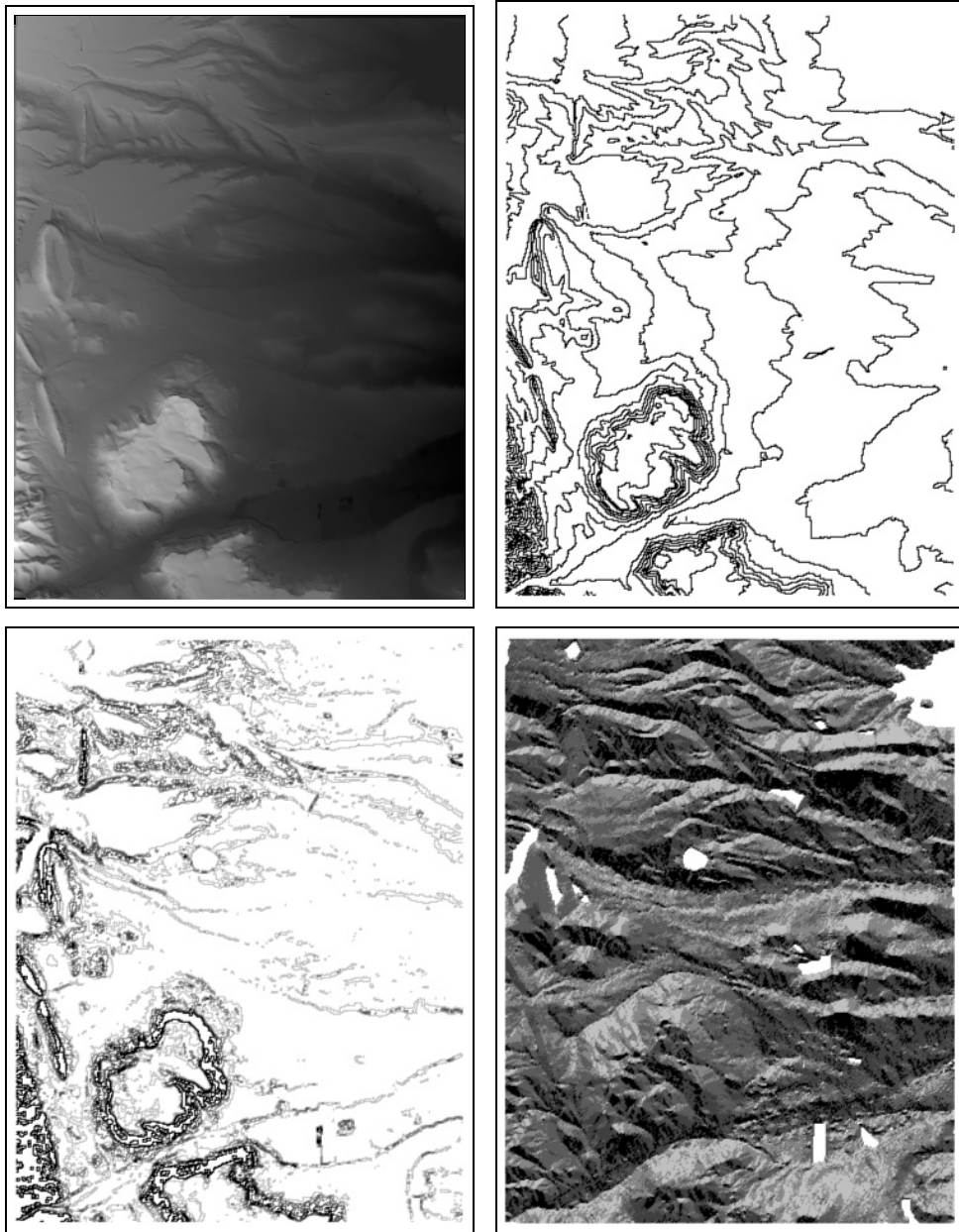


Figure 4.6: From top to bottom and left to right: The original terrain as a DEM, the 100 m contour map of the original terrain, standard gradient, and standard aspect.

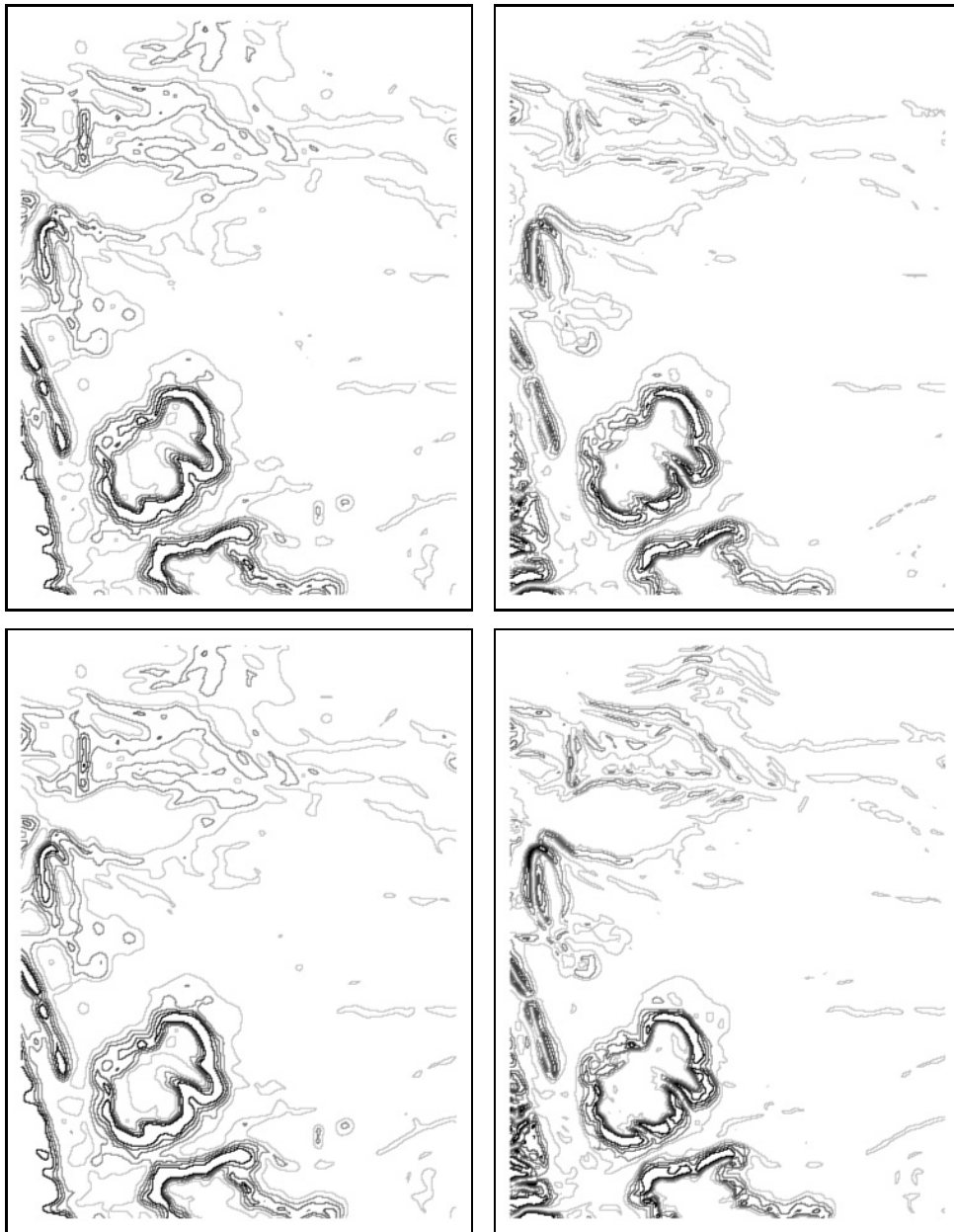


Figure 4.7: Comparison of different gradient methods for radius $r = 5$. From top to bottom and left to right: uniform weighing (scalar method), uniform weighing (vector), non-uniform weighing (scalar), non-uniform weighing (vector).

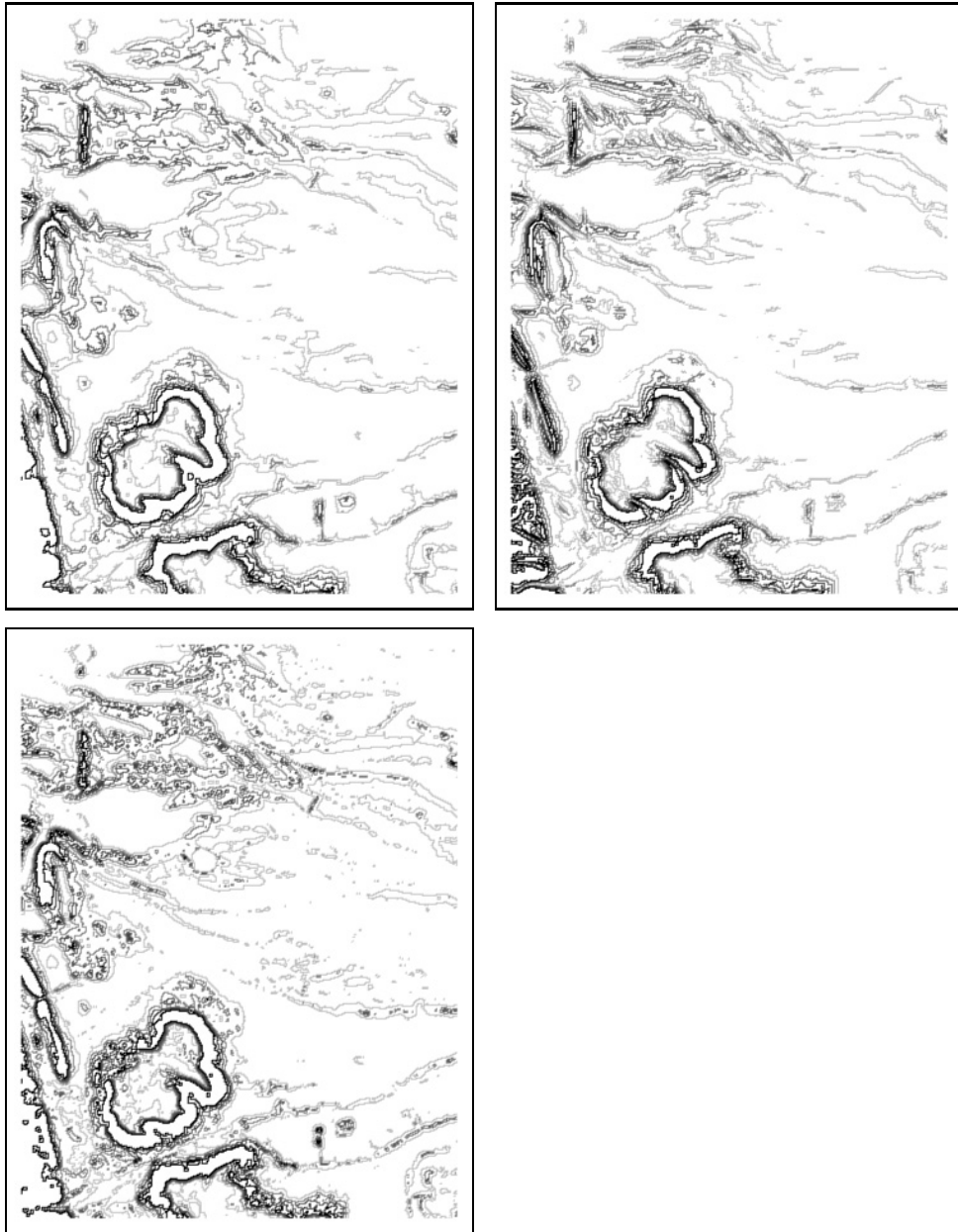


Figure 4.7 (cont.): Comparison of different gradient methods for radius $r = 5$. From top to bottom and left to right: uniform weighing over diameter (scalar method), uniform weighing over diameter (vector), maximum value.

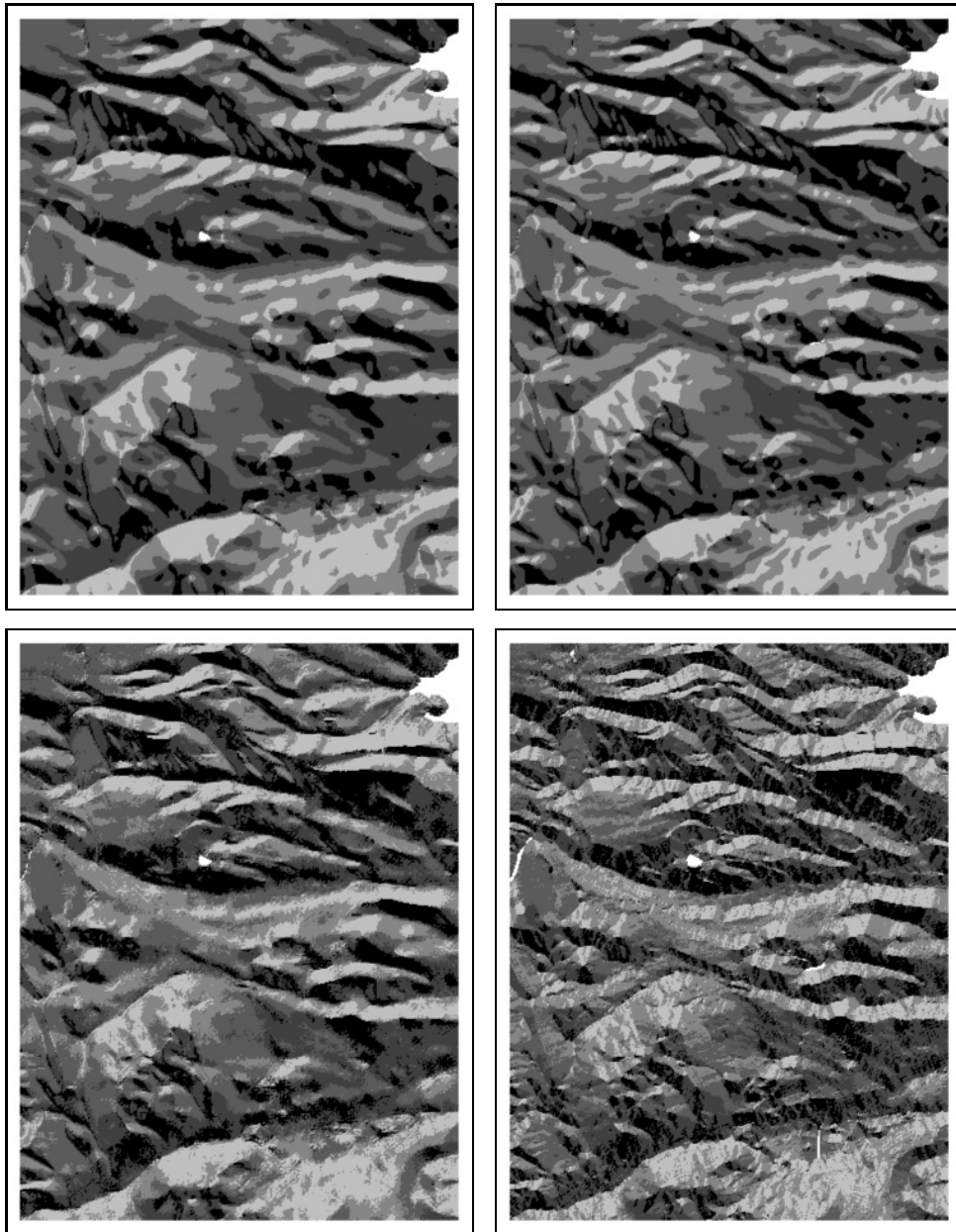


Figure 4.8: Comparison of four different aspect methods for radius $r = 5$. From top to bottom and left to right: uniform weighing, non-uniform weighing, uniform weighing over diameter, maximum value.

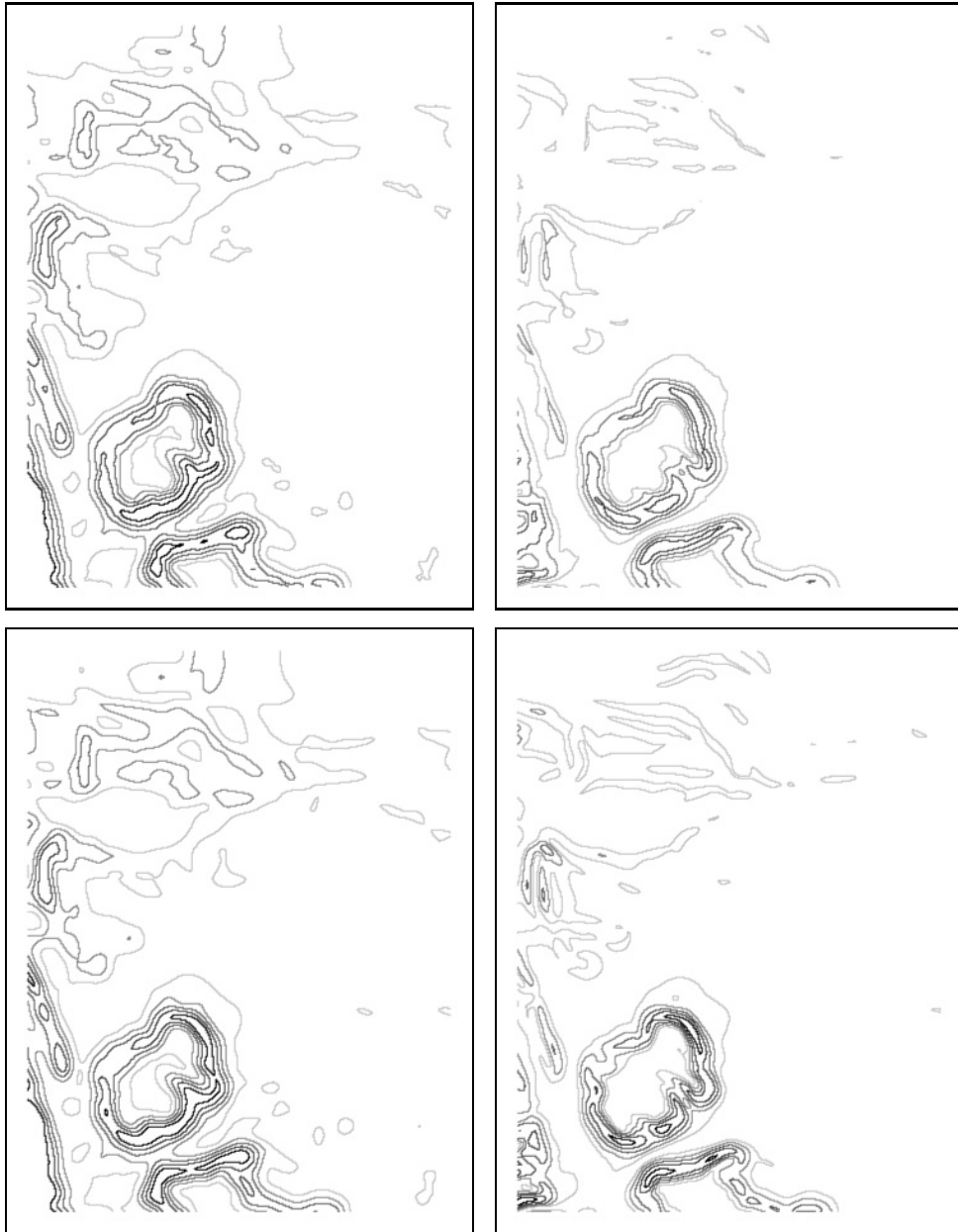


Figure 4.9: Comparison of different gradient methods for radius $r = 10$. From top to bottom and left to right: uniform weighing (scalar method), uniform weighing (vector), non-uniform weighing (scalar), non-uniform weighing (vector).

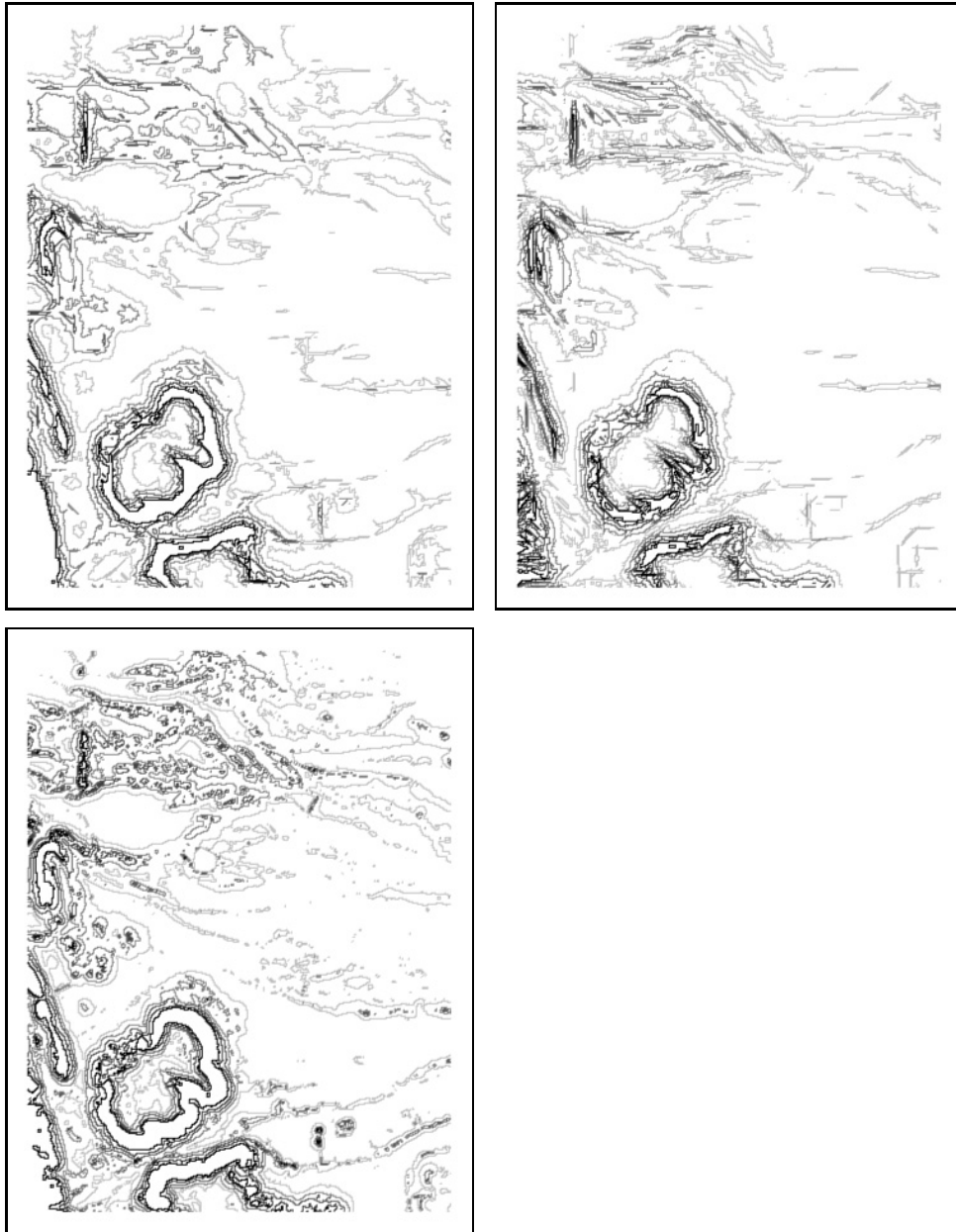


Figure 4.9 (cont.): Comparison of different gradient methods for radius $r = 10$. From top to bottom and left to right: uniform weighing over diameter (scalar method), uniform weighing over diameter (vector), maximum value.

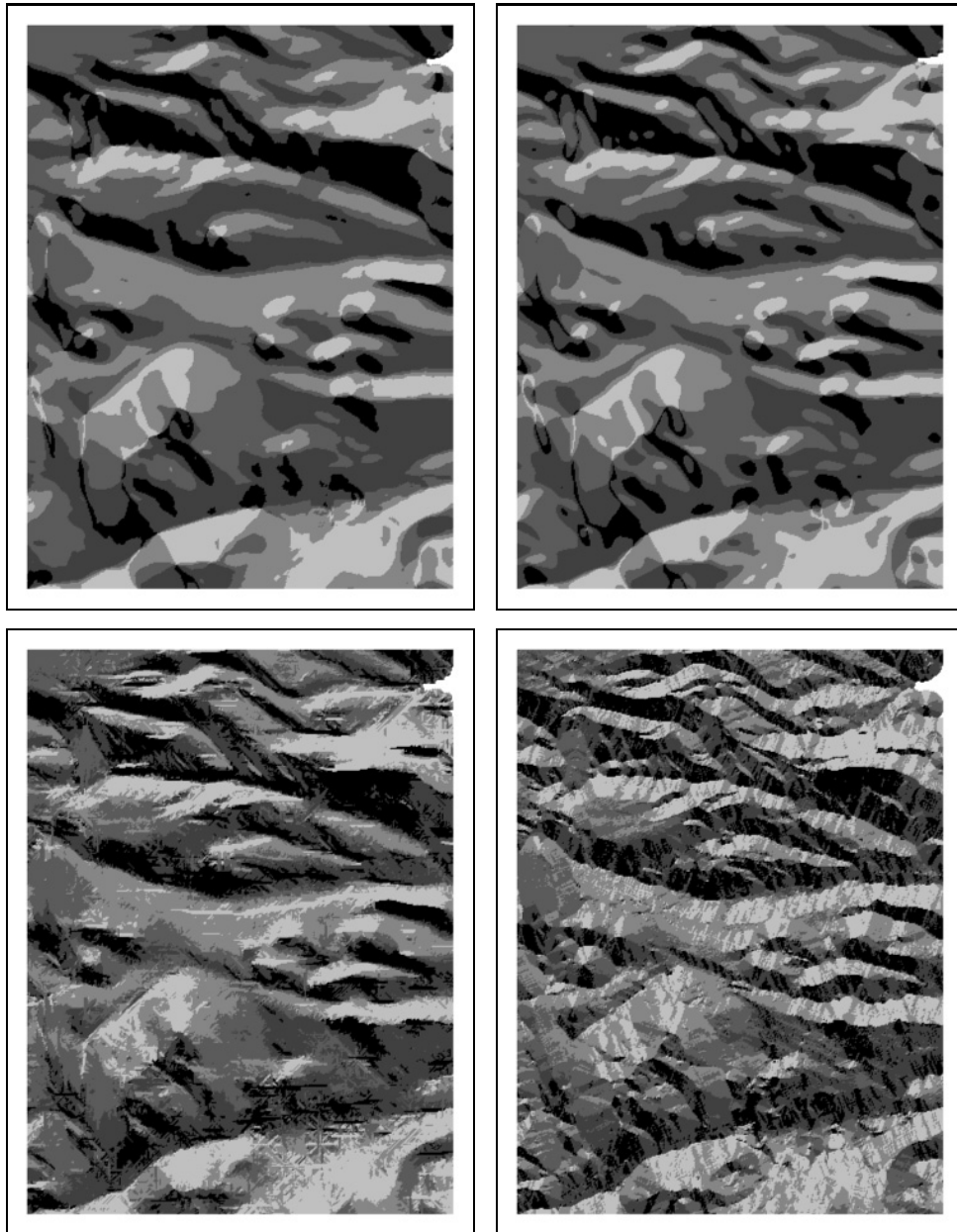


Figure 4.10: Comparison of four different aspect methods for radius $r = 10$. From top to bottom and left to right: uniform weighing, non-uniform weighing, uniform weighing over diameter, maximum value.

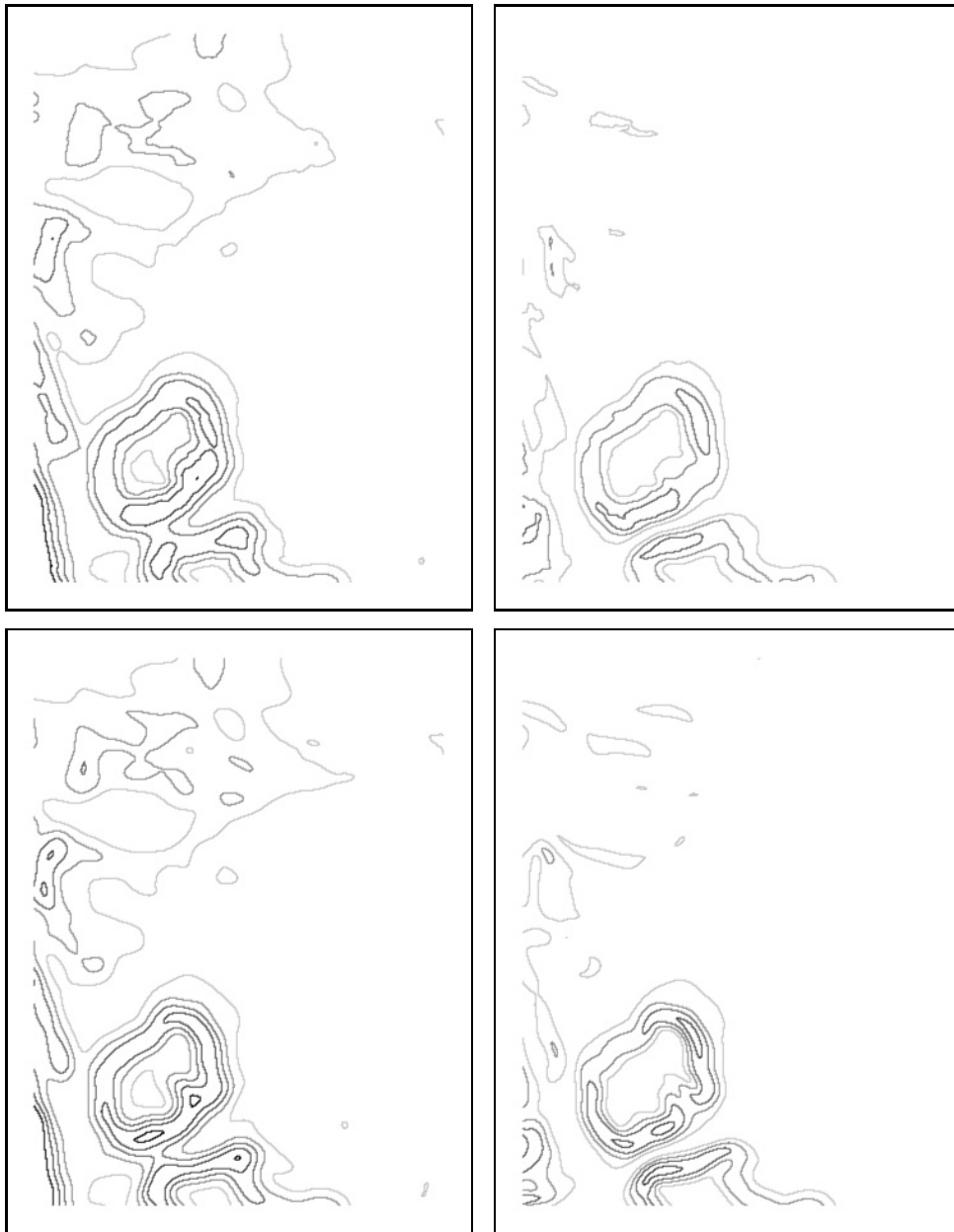


Figure 4.11: Comparison of four different gradient methods for radius $r = 15$. From top to bottom and left to right: uniform weighing (scalar method), uniform weighing (vector), non-uniform weighing (scalar), non-uniform weighing (vector).

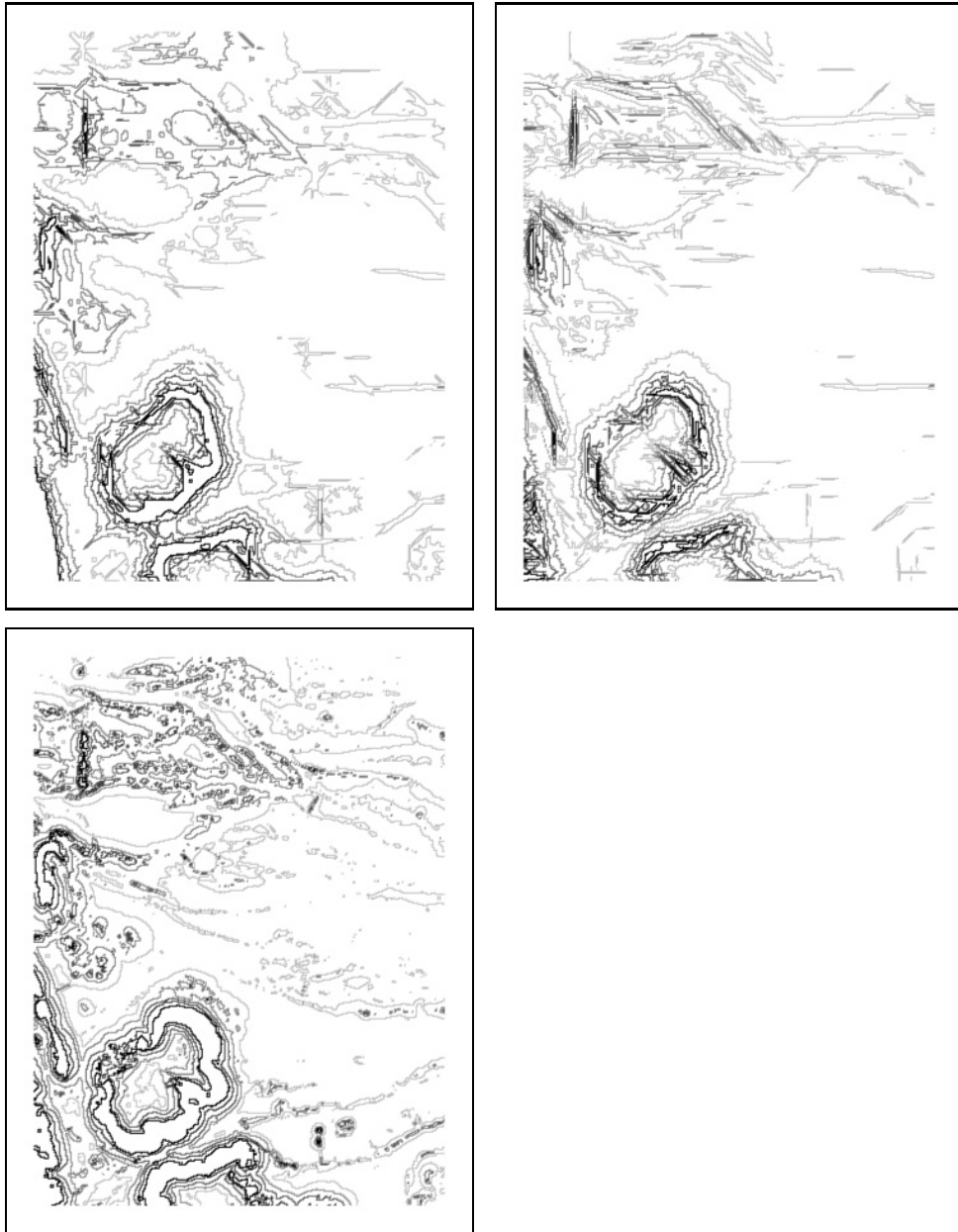


Figure 4.11 (cont.): Comparison of four different gradient methods for radius $r = 15$. From top to bottom and left to right: uniform weighing over diameter (scalar method), uniform weighing over diameter (vector), maximum value.

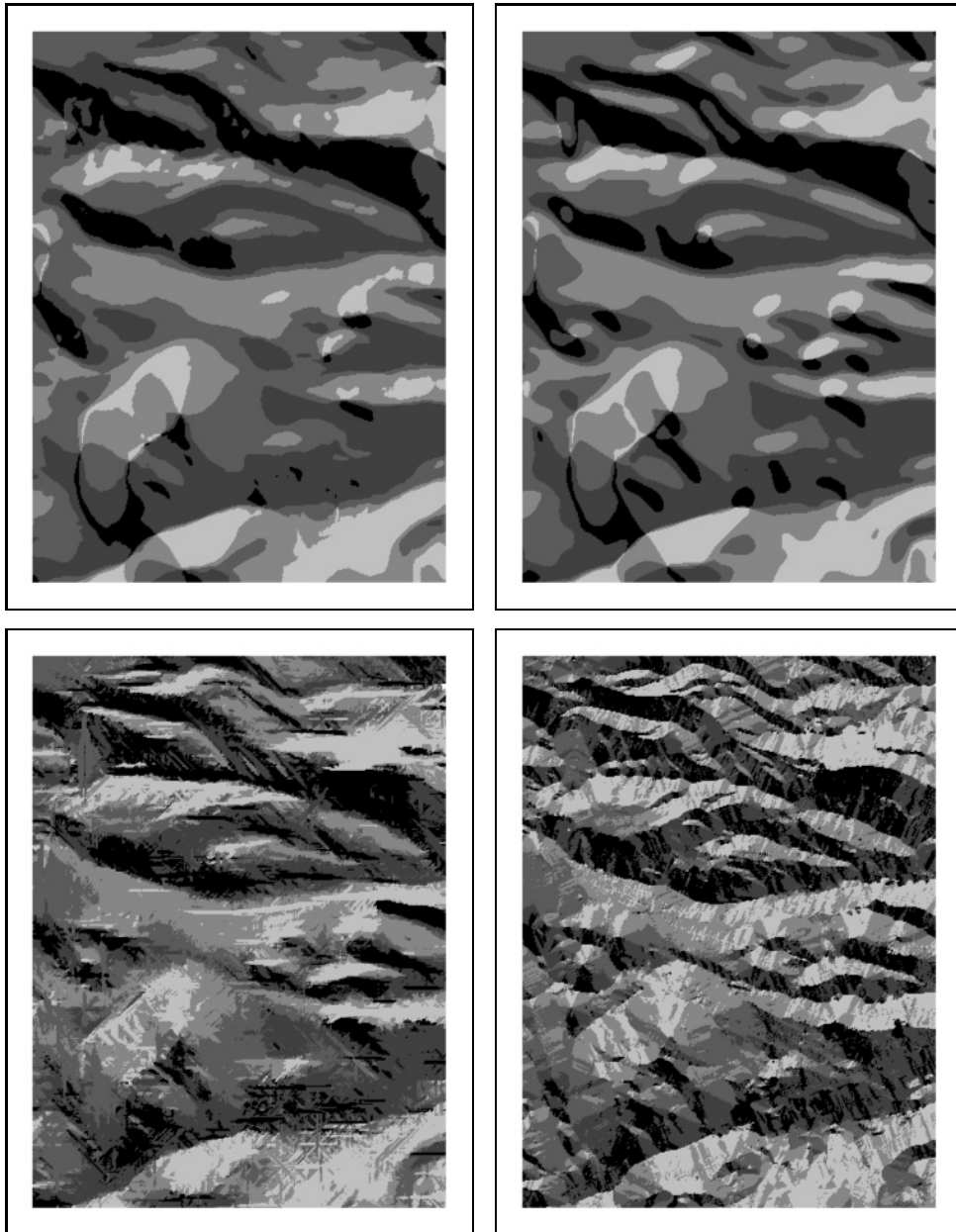


Figure 4.12: Comparison of four different aspect methods for radius $r = 10$. From top to bottom and left to right: uniform weighing, non-uniform weighing, uniform weighing over diameter, maximum value.

from [107], it is a 358 by 468 pixel grid representing an area of 11.3 by 14.5 kilometers northwest of Denver, USA, with a grid spacing of approximately 30 meters. In Figure 4.6, we show the original elevation data (black is low and white is high elevation) and its 100 m contour map at the top. For the presented data set and a radius of $r = 5$ pixels, our experiments on an iBook G4 with 1.2 GHz CPU and 384 MB memory took only a few seconds for all methods.

In all figures that we will discuss, we have used a circular neighborhood of given radius r around p . We approximate the disk D_r on the grid as follows: Every grid cell whose center is closer to p than r is part of D_r , all other grid cells are not. The chosen values for the isogradient lines were 0.3 (light gray), 0.6, 0.9, 1.2, and 1.5 (black). The aspect maps are computed with the same settings as the gradient maps. In the aspect maps, white represents a flat area, black has aspect facing South and light gray is aspect North. To increase readability of the aspect maps, no distinction was made between the coloring of East and West. The bottom half of Figure 4.6 shows the standard gradient map and the standard aspect map of the terrain. The white, flat areas of the standard aspect map depict lakes in the chosen area.

Note that when the chosen neighborhood did not fully cover the terrain, i.e., at the edges and corners of the data set, there is no proper neighborhood to define local gradient and aspect, so we set the value for local gradient and aspect to zero. This is the reason for having a frame of width r around each of the maps.

4.3.1 Comparison of different methods

Here we compare the outcome of the different methods for local gradient and aspect. The following observations are valid for all three investigated values of radius $r = 5, 10,$ and 15 pixels, which corresponds to approximately 150, 300, and 450 meters.

For the local gradient, we see in Figures 4.7, 4.9, and 4.11, that the maximum value method results in the most detailed map. This is as expected, as this method does not perform any averaging. As expected, the level of detail of the uniform weighing over a diameter method lies in between the maximum value method and the averaging methods, which give the smoothest maps. Furthermore we see that the scalar methods provide more detail in the flat parts of the terrain (upper half), whereas the vector methods give more detail in the rugged parts of the terrain (lower half). This is consistent with the other terrains we have investigated, and can be explained with the additional smoothing of the vector method, which is more effective in flat terrains. There appears to be more detail in the non-uniform weighing methods than in the uniform weighing methods.

In general, it seems that the uniform weighing method, either scalar or vector, provides the best output when looking for a generalization of the isogradients. When high detail is desired, the maximum value method is preferred. This is also the only method that preserves the highest isogradient class well. It gives a smoothing of a different character than the other methods.

The situation for the three (vector) methods for local aspect is similar, see Figures 4.8, 4.10, and 4.12. Again, the maximum value method shows the most detailed map of all methods, followed by the uniform weighing over a diameter method. Both the uniform and the non-uniform weighing method give a similar result for this ter-

rain; the non-uniform method produces a slightly more detailed aspect map.

4.3.2 Comparison of different radii

Here we discuss the influence of different radii on the outcome of isogradients and isoaspects.

Comparing Figures 4.7, 4.9, and 4.11, we can see that the smaller the radius and therefore the area of influence, the more detailed the gradient map becomes. We get many isolines with highly detailed boundaries, many of which are relatively short. Also the spacing between isolines with different values is small. When gradually increasing the radius, we get less detailed maps, the isolines show less detail and become more smooth, and also the distance between isolines with different values gets larger.

The situation for aspect is similar, see Figures 4.8, 4.10, and 4.12. Again, the aspect map with smallest radius of the neighborhood shows the highest detail, and the map with largest radius has the largest and smoothest areas. Note especially the gradual decrease in size of the flat white lake area in the upper right corner of the aspect map. As an area is flat only if the aspect points exactly in z -direction, even one cell inside the neighborhood that is outside the flat area will change the outcome of the averaging and cause the local aspect to be non-flat.

This effect of decreasing detail with increasing radius is smallest in case of the maximum value method and larger, but similar, in all other methods.

4.4 Concluding Remarks

In this chapter we have introduced the notion of scale dependent local slope, i.e., local gradient and local aspect, for each point of a given terrain. We suggested four different definitions of local slope inside a neighborhood with radius r around a point and presented efficient algorithms to compute it on a TIN. Once we have derived local slope at each point of the terrain it is straightforward to compute maps with lines of constant gradient or areas of constant aspect.

The results of the implementation on a gridded DEM show the expected smoothing behavior compared to the slope values computed by the standard method. The maximum value method gives the most detailed maps for all investigated radii, followed by the uniform weighing over a diameter. The output of the unweighted and weighted methods (vector and scalar based) are similar to each other. It is unclear to us whether a geomorphologist can see significant differences among the methods, and would prefer any of the methods. The uniform weighing scalar method is the easiest one to implement, it may be the method of choice. However, which method to prefer for a certain application may also depend on the desired level of detail of the result.

Future work includes more extended experiments, for example to compare the length of isogradients and the areas of isoaspects for different methods and radii. Furthermore, it is interesting to develop similar methods as investigated in this

chapter to compute scale dependent maps that show plan and profile curvature, and other measures used in geomorphometry.

5 SCATTERED RELEVANCE RANKING FOR GEOGRAPHIC IR

In the last three chapters we have given methods to delineate boundaries of imprecise regions. These can be used for geographic information retrieval to aid in queries like ‘hotels in northern Spain’ or ‘Aikido dojos in the Dutch Randstad’. Once all relevant documents are retrieved, the other main issue of geographic information retrieval needs to be resolved: The ranking of the documents according to their textual and spatial relevance, represented by a term and a spatial score.

For example, the Web search could be for campgrounds in the neighborhood of Neuschwanstein, and the documents returned ideally have a score for the query term ‘campground’ and a score for the proximity to Neuschwanstein. This implies that a Web document resulting from this query can be mapped to a point in the 2-dimensional plane, where both axes represent a score. In Figure 5.1, the map

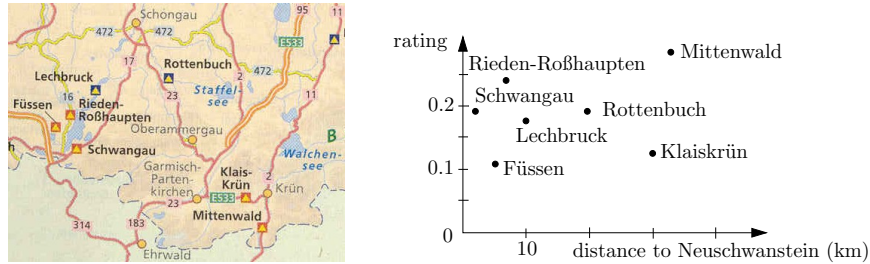


Figure 5.1: Campgrounds near Neuschwanstein mapped to points in the plane.

(taken from [7]) indicates campgrounds near the castle Neuschwanstein, which is situated close to Schwangau, with the distance to the castle on the x -axis and the rating given by the ANWB¹ on the y -axis.

Another query is for example ‘castles near Koblenz’. When mapping to the plane, a cluster of points could be several documents about the same castle. If this castle is in the immediate vicinity of Koblenz, all of these documents would be ranked high, provided that they also have a high score on the term ‘castle’. However, the user probably also wants documents about other castles that may be a bit further away, especially when these documents are more relevant for the term ‘castle’.

¹Algemene Nederlandse Wielrijders Bond (Dutch automobile association)

To incorporate this idea in the ranking, we introduce *multi-dimensional scattered ranking* in this chapter. We present various models that generate ranked lists where closely ranked documents are dissimilar. We also present efficient algorithms that compute scattered rankings, which is important to keep server load low.

Scattered ranking requires two scores, a spatial score and a term score. Methods as how these are defined for a document have been presented in Chapter 1.3. In this chapter, we will assume that for each document a spatial as well as a term score are given.

We can also use scattered ranking to obtain *geographically scattered* ranked lists of documents. In this case we use the geographical locations associated with Web documents explicitly in two coordinates. If we also have a term score, each document has three values (scores) that can be used for ranking the relevant documents. Using our example, two documents referring to two castles at the same distance from Koblenz, but in opposite directions, may now be ranked consecutively. The geographically scattered ranking problem is related to the so-called *settlement selection problem*, which appears in map generalization [99, 169].

Also in other applications it can be useful to rank documents according to more than one score. For example we could distinguish between the scores of two textual terms, or a textual term and a spatial term, or a textual term and metadata information, and so on. A common example of metadata for a web document is the number of hyperlinks that link to it; a document is probably more relevant when more links point to it. In all of these cases we get two or more scores which need to be combined for the ranking.

In traditional information retrieval, the separate scores of each document would be combined into a single score (e.g., by a weighted sum or product) which produces the ranked list by sorting. Besides the problem that it is unclear how the scores should be combined, it also makes a scattered ranking impossible. Two documents with the same combined score could be similar documents or quite different. If two documents have two or more scores that are the same, one has more reason to suspect that the documents themselves are similar than when two documents have one (combined) score that is the same.

Related research has been conducted in [18], where each document or information object is assigned a spatial, temporal and thematic rank. Using the spatial relationships inside, overlap, and containment between a document and a query, the spatial and temporal rank depend on the area of overlap of two spatial regions or two time frames. One or more thematic ranks of a document can be assigned by textual methods. The three ranks for a document are combined into a three part glyph, where different colors and hues indicate the relevance of the document. This has the advantage of clearly showing the relevance of the document with respect to each rank separately. However, the disadvantage of this presentation is that it does not lead to a list of documents ranked from most relevant to least relevant.

Rauch et al. focus in [142] on disambiguating geographical terms of a user query. The disambiguation of the geographical location is done by combining textual information, spatial patterns of other geographical references, relative geographical references from the document itself, and population heuristics from a gazetteer. This gives the final value for the geoconfidence. The georelevance is composed of the geoconfidence and the emphasis of the place name in the document. The textual

relevance of a document is computed as usual in information retrieval. Once both textual and geographical relevance are computed, they are combined by a weighted sum.

Finding relevant information and at the same time trying to avoid redundancy has so far mainly been addressed in producing summaries of one or more documents. Carbonell and Goldstein use in [30] the maximal marginal relevance (MMR), which is a linear combination of the relevance of the document to the user query and its independence of already selected documents. MMR is used for the reordering of documents. A user study has been performed, showing that users preferred MMR ranking to the usual ranking of documents. The paper [30] contains no algorithm how to (efficiently) compute the MMR. Following up on this, a Novelty Track of TREC [85] discusses experimenting with rankings of textual documents such that every next document has as much additional information as possible.

Goldstein et al. propose in [76] another scoring function for summarizing text documents. Every sentence is assigned a score combined of the occurrence of statistical and linguistic features. They are combined linearly with a weighting function. In [77], MMR is refined and used to summarize multiple documents. Instead of full documents, different passages or sentences are assigned a score.

The remainder of this chapter is organized as follows. In Section 5.1 we introduce the two basic ranking methods angle-to-ranked and distance-to-ranked. Documents can be efficiently ranked by these two methods in $O(n^2)$ time worst case. In Section 5.2 we introduce two more basic scattered ranking methods, two addition methods that run in worst case $O(n^2)$ time and a wavefront approach. For the wavefront method we can give an algorithm that computes a ranking according to it in $O(n \log^2 n)$ time. We give three extensions to the basic methods. First we restrict the number of eligible points to be next in the ranking by the staircase enforcement method. Only points that lie on the lower left staircase of the point set may be ranked next. Second, we restrict the influence of the already ranked points to the not yet ranked ones in the limited windows method. Finally, we investigate the possibility of extending all presented basic algorithms to higher dimensions. We show how to adapt all four algorithms and give the new running times in Section 5.3. In Section 5.4 we show how the different ranking methods behave on real-world data.

5.1 Basic Scattered Ranking Methods and Algorithms

In this section we present two basic scattered ranking methods. Like in traditional information retrieval, we want the most relevant documents to appear high in the ranking, while at the same time avoiding that documents with similar information appear close to documents already ranked. We will focus on the two-dimensional case only, although in principle the ideas and formulas apply in higher dimensions too. We will discuss extensions to higher dimensions explicitly in Subsection 5.3.3.

We assume that a Web query has been conducted and a number of relevant documents were found. Each document is associated with two scores, for example a textual and a spatial score (which is the case in the SPIRIT search engine). The relevant documents and the query are mapped to points in the plane. We perform

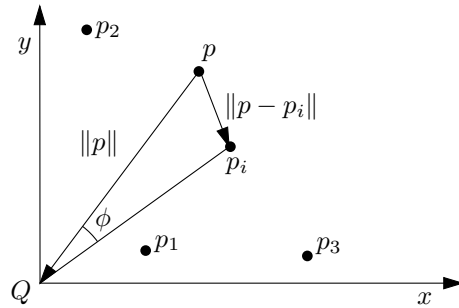


Figure 5.2: An unranked point p amidst ranked points p_1, p_2, p_3, p_i , where p is closest to p_i by distance and by angle.

the mapping in such a way that the query is represented by a point Q at the origin, and the documents are mapped to a set of points $P = \{p_1, \dots, p_n\}$ in the upper right quadrant, such that the documents with high scores are the points close to Q . We can now formulate the two main objectives for our ranking procedure:

1. **Proximity to query:** Points close to the query Q are favored.
2. **High spreading:** Points farther away from already ranked points are favored.

A ranking that simply sorts all points in the representation plane by distance to Q is optimal with respect to the first objective. However, it can perform badly with respect to the second. Selecting a highly scattered subset of points is good with respect to the second objective, but the ranked list would contain too many documents with little relevance early in the list. We therefore seek a compromise where both criteria are considered simultaneously. Note that the use of a weighted sum to combine the two scores into one as in [142] makes it impossible to obtain a scattered ranking.

The point with the smallest Euclidean distance to the query is considered the most relevant and is always first in any ranking. The remaining points are ranked with respect to already ranked points. At any moment during the ranking, we have a subset $R \subset P$ of points that have already been ranked, and a subset $U \subset P$ of points that are not ranked yet. We choose from U the ‘best’ point to rank next, where ‘best’ is determined by a *scoring function* that depends on both the distance to the query Q and the set R of ranked points. Intuitively, an unranked point has a higher added value or relevance if it is not close to any ranked points.

For every unranked point p , we consider only the closest point $p_i \in R$, where closeness is measured either in the Euclidean sense, or by angle with respect to the query point Q . This is illustrated by $\|p - p_i\|$ and ϕ , respectively, in Figure 5.2. Using the angle to evaluate the similarity of p and p_i seems less precise than using the Euclidean distance, but it allows more efficient algorithms, and certain extensions of angle-based ranking methods give nicely scattered results. See the experiments in Section 5.4 for this. We will first present the basic models, and then give the algorithms in Subsection 5.1.3.

5.1.1 Distance to query and angle to ranked

Our first ranking method uses the angle measure to obtain the similarity between an unranked and a ranked point. In the triangle $\triangle pQp_i$ (see Figure 5.2) consider the angle $\phi = \phi(p, p_i)$ and rank according to the score $S(p, R) \in [0, 1]$, which can be derived from the following normalized equation:

$$S(p, R) = \min_{p_i \in R} \left(\frac{2(\phi(p, p_i) + c)}{\pi + 2c} \cdot \left(\frac{1}{1 + \|p\|} \right)^k \right) \quad (5.1)$$

Here, $k > 0$ denotes a constant. If k is small, the emphasis lies on the spreading, and if k is large, we give a large importance to the proximity to the query. The additive constant $c > 0$ ensures that all unranked points $p \in U$ are assigned an angle dependent factor greater than 0. This is necessary if several points lie on the same halfline originating in Q . The score $S(p, R)$ necessarily lies between 0 and 1, and is appropriate if we do not have a natural upper bound on the maximum distance of unranked points to the query. If such an upper bound were available, there are other formulas that give normalized scores.

5.1.2 Distance to query and distance to ranked

In the previous section we ranked by angle to the closest ranked point. It may be more natural to consider the Euclidean distance to the closest ranked point instead. In Figure 5.2, consider the distance $\|p - p_i\|$ from p to the closest ranked point p_i and rank according to the outcome of the following equation:

$$S(p, R) = \min_{p_i \in R} \left(\frac{\|p - p_i\|}{\|p\|^2} \right) \quad (5.2)$$

The denominator needs a squaring of $\|p\|$ (or another power > 1) to assure that documents far from Q do not appear too early in the ranking, which would conflict with the proximity to query requirement. A normalized equation such that $S(p, R) \in [0, 1]$ is the following:

$$S(p, R) = \min_{p_i \in R} \left((1 - e^{-\lambda \cdot \|p - p_i\|}) \cdot \frac{1}{1 + \|p\|} \right) \quad (5.3)$$

Here, $\lambda > 0$ is a constant that defines the base e^λ of the exponential function.

5.1.3 Basic algorithms

For both methods described above, during the ranking algorithms we always choose the one unranked point p that has the highest score $S(p, R)$ and rank it next. This implies an addition to the set R and hence, recomputation of the scores of the other unranked points may be necessary.

In this section we describe two simple algorithms that can be applied for the two basic methods given above as well as for the other methods we present in the next section. We assume that relevant documents have already been retrieved and mapped to points in the plane. We only describe the scattered ranking algorithms themselves.

A straightforward implementation of the ranking algorithms takes $O(n^3)$ time, where n is the number of points to be ranked. However, by keeping the closest ranked point for each unranked point in evidence, we can create a very simple, generic algorithm which has a running time of $O(n^2)$. The distance referred to in Steps 2(a) and 4 can be interpreted either as angle or as Euclidean distance in the plane.

Algorithm 1: Given: A set P with n points in the plane.

1. Rank the point r closest to the query Q first. Add it to R and delete it from P .
2. For every unranked point $p \in P$ do
 - (a) Store with p the point $r \in R$ as the closest point.
 - (b) Compute the score $S(p, R) = S(p, r)$ and store it with p .
3. Determine and choose the point p with the highest score $S(p, R)$ to be next in the ranking; add it to R and delete it from P .
4. Compute for every point $p' \in P$ the distance to the last ranked point p . If it is smaller than the distance to the point stored with p' , then store p with p' and update the score $S(p', R)$.
5. Continue at Step 3 if there are still unranked points.

The first four steps of this algorithm all take linear time. As we need to repeat Steps 3 and 4 until all points are ranked, the overall running time of this algorithm is $O(n^2)$. If we are only interested in the top 10 documents of the ranking, we only need linear time for the computation. More generally, the top t documents are determined in $O(tn)$ time.

The next algorithm is a variation of the first, and computes the same ranking. It uses a Voronoi diagram to find the closest unranked points to a ranked point p . Its worst case running time is the same as of Algorithm 1, namely $O(n^2)$; however, a typical case analysis shows that it generally runs in $O(n \log n)$ time in practice.

Algorithm 2: Given: A set P with n points in the plane.

1. Rank the point r closest to the query Q first. Add it to R and delete it from P . Initialize a list with all unranked points and store it with r . Determine the point p in this list with the highest score and insert it in an initially empty priority queue H .
2. Choose the point p with the best overall score from the priority queue H as next in the ranking; add it to R , delete it from P , and perform the following updates:
 - (a) Delete p from the priority queue H .
 - (b) Insert p as a new site in the Voronoi diagram of R .
 - (c) Create for the newly created Voronoi cell $V(p)$ a list of unranked points that lie in $V(p)$ by traversing the list of each ranked point p' whose Voronoi cell $V(p')$ neighbors $V(p)$, removing the points from it that are closer to p than to p' , and adding these points to the list of p .

- (d) Compute the point with the best score for the newly created Voronoi cell $V(p)$ and insert it in a priority queue H . For all Voronoi cells whose lists changed, recompute the unranked point with the best score and update the priority queue H accordingly.
3. Continue at Step 2 if the priority queue H is non-empty.

For the efficiency analysis, assume first that we used the distance-to-ranked version for assigning scores. Then the Voronoi diagram is the usual planar subdivision, and the average degree of a Voronoi cell is almost six. One can expect that a typical addition of a point p to the ranked points involves a set of neighbors $R' \subseteq R$ with not many more than six ranked points. If we also assume that, typically, a point in R' loses a constant fraction of the unranked points in its list, we can prove an $O(n \log n)$ time bound for the whole ranking algorithm. The analysis is the same as in [86, 169]. In the angle-to-ranked version of assigning scores, the set of neighbors R' will have only two ranked points. We conclude with the following theorem:

Theorem 17 *A set of n points in the plane can be ranked according to the basic scattered models in $O(n^2)$ time in the worst case. By maintaining a Voronoi diagram of the closest points, the points can be ranked in $O(n \log n)$ time under certain natural assumptions.*

5.2 Other Scattered Ranking Methods and Algorithms

In this section we present more basic ranking methods, namely addition methods and a wavefront approach. The addition methods are simple variations of the ideas of the methods from Section 5.1. Instead of dividing the angle (or distance) of a point p to the closest ranked point by the distance of p to the query, we add those two values (one of them inverted). We can adapt both algorithms given above. However, as we will see in Subsection 5.2.1, for one of the methods we can give an algorithm with better running time. The wavefront method is in a way an inversion of the ideas of the basic methods. In the last section, we ranked the point with the highest score according to some scoring function next. For a fixed set of ranked points, the subset of the plane with the same score is a set consisting of curved pieces. When decreasing the score, the curved pieces move and change shape. The next point to be ranked is the one encountered first by these curves when decreasing the score. In Subsection 5.2.2 we will use this process instead of a scoring function to define a ranking: we predefine the shape of a wavefront and move it until it hits a point, which will be ranked next. As before, we first introduce the methods, and then present the algorithms in Subsection 5.2.3.

5.2.1 Addition methods

So far, both our scattered ranking methods were based on a scoring function that essentially is a division of the angle or distance to the closest ranked point by the distance to the query. In this way, points closer to the query get a higher relevance. We can obtain a similar effect, but a different ranking, by adding up two terms,

obtained from the angle or distance to the closest ranked point, and the distance to the query. The term depending on the distance to the query should be such that a larger distance gives a lower score. Although it is unusual to add angles and distances, it is not clear beforehand which method will be more satisfactory for users, so we analyse these methods as well. If the results are satisfactory, this method may be the one of choice, since there is an efficient algorithm for it.

$$S(p, R) = \min_{p_i \in R} \left(\alpha \cdot (1 - e^{-\lambda \cdot (\|p\|/\|p_{max}\|)}) + (1 - \alpha) \cdot \phi(p, p_i) \cdot \frac{2}{\pi} \right) \quad (5.4)$$

In this equation, p_{max} is the point with maximum distance to the query, $\alpha \in [0, 1]$ denotes a variable which is used to put an emphasis on either distance or angle, and λ is a constant that defines the base of the exponential function.

Another, similar, addition method adds the distance to the query and the distance to the closest ranked point:

$$S(p, R) = \min_{p_i \in R} \left(\alpha \cdot (1 - e^{-\lambda_1 \cdot (\|p\|/\|p_{max}\|)}) + (1 - \alpha)(1 - e^{-\lambda_2 \cdot \|p - p_i\|}) \right) \quad (5.5)$$

Again, p_{max} is the point with maximum distance to the query, $\alpha \in [0, 1]$ is a variable used to influence the weight given to the distance to the query (proximity to query) or to the distance to the closest point in the ranking (high spreading), and λ_1 and λ_2 are constants that define the base of the exponential functions.

5.2.2 The wavefront approach

The ranking methods presented so far can also be described visually. For example, consider the angle-to-ranked method of Subsection 5.1.1. At any moment in the algorithm, some selection has been made, and the next point for the ranking must be found. This point is the first point hit by a *wavefront* of a particular shape, as shown in Figure 5.3. From the shape of the wavefront it is clear that points that are not closest to the query could easily be chosen next, if the angle of the vector p (with the positive x -axis) is not close to the angle of p_i for any previously ranked point p_i . This illustrates the desired behavior of trying to choose a point next that is different from already ranked points. We can derive new ranking methods by specifying the shape of the wavefront instead of a scoring function. At the same time, the idea gives rise to other ways to compute the ranking.

Figure 5.3 depicts the positions in space where the score values are the same, namely 0.05, for the three ranked reference points $p_1 = (1, 1)$, $p_2 = (2, 3)$, and $p_3 = (7, 2)$. These functions are of the following form (constants omitted): $\arccos(\text{dist to closest}/\text{dist to query})/\text{dist to query}$. They can easily be derived from Equation 5.1. Note that the curves are symmetric with respect to the line starting at the origin and passing through the ranked point they correspond to, and that the intersection with this line lies on a circle with the same radius r for all three curves. Furthermore, two curves of neighboring sectors meet exactly at the bisecting lines between the ranked points.

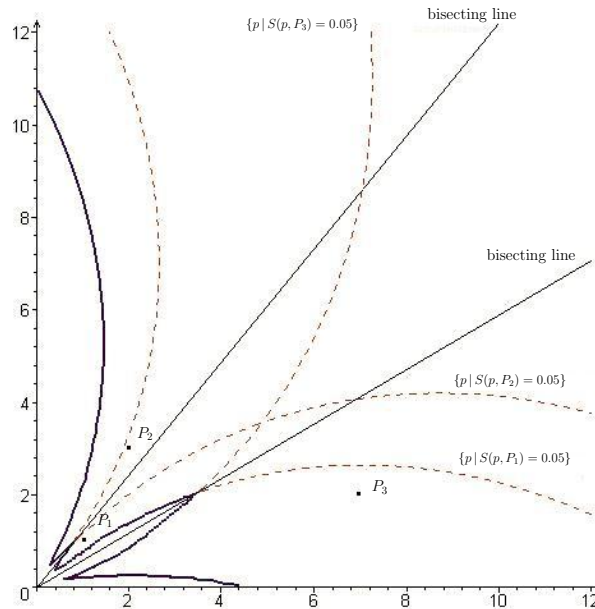


Figure 5.3: Wavefront of the angle method of Subsection 5.1.1. The solid line is the locus of all points p , such that the score $S(p, \{p_1, p_2, p_3\}) = a$, for $a = 0.05$. The wavefront moves when a is increased.

5.2.3 Efficient algorithms for addition and wavefront model

For both methods described above, Algorithms 1 and 2 can be applied, which gives a worst case running time of $O(n^2)$ in both cases. But in fact, we can seriously improve the worst case running time with a different algorithm. As the angle $\phi(p, p_i)$ is an additive and not a multiplicative part of the score equation, we can give an algorithm with a worst case running time of $O(n \log n)$ for the angle-distance addition method. We will also present an algorithm for the piecewise linear wavefront method which computes such a ranking in $O(n \log^2 n)$ time in the worst case.

The angle-distance addition algorithm

To initialize for the ranking algorithm, we select the point r from P that is closest to the query and rank it as the first point. Then we build two augmented binary search trees, one for the subset of points $P_0 \subseteq P \setminus \{r\}$ that are below the line through Q and r and one for the subset $P_1 = P \setminus \{r\} - P_0$ of points above or on this line.

The point set P_0 is stored in the leaves of a binary tree T_0 , sorted by counter-clockwise (ccw) angle to the y -axis. In every leaf of the tree we also store: (i) ccw and clockwise (cw) angle to r and to the x -axis respectively; (ii) the distance to the query; (iii) ccw and cw score, where the angles used are taken from (i). We augment T_0 as follows (see e.g., [42] for augmenting data structures): In every internal node we store the best cw and the best ccw score per subtree. We additionally store for

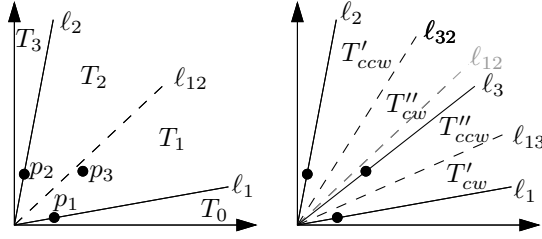


Figure 5.4: The split and concatenate of trees in Algorithm 3.

the whole tree T_0 that its closest ranked point, r , is counterclockwise, and correction values, one cw and one ccw, to be used later. They are additive corrections for all cw and ccw scores of points in the whole tree. Note that the augmentation allows us to descend in a tree towards the point with the best score along a single path; at any node we simply proceed in the subtree that contains the point with the higher score (cw or ccw score, depending on the information stored with the tree).

The point set P_1 is stored in the same way in a tree T_1 , with the following differences. The cw angle used in the leaves is to r and the ccw angle is taken to the y -axis, and we store for the whole tree that its closest ranked point, r , is clockwise. Finally, we initialize a priority queue with the point from T_0 with the best score and the point from T_1 with the best score.

During the algorithm, for every ranked point two trees are present. It is easy to see that all trees T_{2i+1} , $i = 0, \dots, m-1$ have their closest ranked point in cw direction, whereas all trees T_{2i} , $i = 0, \dots, m-1$, have their closest ranked point in ccw direction. As shown left in Figure 5.4, between two already ranked points p_1 and p_2 , indicated by ℓ_1 and ℓ_2 , there are two binary trees, T_1 cw and T_2 ccw of the bisecting line ℓ_{12} . All the points in T_1 are closer in angle to p_1 and all the points in T_2 are closer in angle to p_2 . If we insert a new point p_3 to the ranking, this means we insert a new imaginary line ℓ_3 through p_3 and we need to perform the following operations on the trees:

1. Split T_1 and T_2 at the bisecting lines ℓ_{32} and ℓ_{13} , creating the new trees T'_{cw} and T'_{ccw} and two intermediate trees \bar{T}_{cw} and \bar{T}_{ccw} .
2. Concatenate the intermediate trees from (1), creating one tree \bar{T} .
3. Split \bar{T} at the newly ranked point p_3 , creating T''_{cw} and T''_{ccw} .

Figure 5.4, right, shows the outcome of these operations. Whenever we split or concatenate the binary trees we need to make sure that the augmentation remains correct. In our case, this is no problem, as we only store the best initial scores in the inner leaves. However, we need to update the information in the root of each tree about the closest cw and ccw ranked point and the correction values. The former is easy to update. For the correction values, note that all scores for points in the same tree are calculated with respect to the same ranked point and they change with the same additive amount in the addition model. Therefore, we should simply subtract $(1 - \alpha) \cdot \phi' \cdot 2/\pi$ from the scores, where ϕ' denotes the angle between the previously closest ranked point cw (ccw) and the newly closest ranked point cw (ccw),

respectively). For efficiency reasons, we do not do this explicitly but at once using the cw (or ccw) correction value. So we subtract $(1 - \alpha) \cdot \phi' \cdot 2/\pi$ from the appropriate correction value. If there is no previously closest ranked point cw or ccw (first and last trees), then we take the angle between the newly ranked point and the x -axis or y -axis instead.

Furthermore, we need to update the score information in the priority queue. This involves deleting scores that are no longer valid, and inserting scores that are new best scores in a tree. Now we can formulate an algorithm for the addition method that runs in optimal $O(n \log n)$ time.

Algorithm 3: Given: A set P with n points in the plane.

1. Determine the closest point r and remove it from P , split $P \setminus \{r\}$ into sets P_0 and P_1 as described, and initialize the augmented trees T_0 and T_1 . Determine the points in T_0 and T_1 with the best score and store them in a priority queue H .
2. Choose the point p with the highest score as next in the ranking by deleting the best one from the priority queue H .
3. For every last ranked point p do:
 - (a) Split and concatenate the binary trees as described above and update the information in their roots.
 - (b) Update the best score information in the priority queue H :
 - i. Delete the best score of the old tree T_1 or T_2 that did not contain p .
 - ii. Find the four best scores of the new trees T'_{cw} , T'_{ccw} , T''_{cw} , and T''_{ccw} and insert them in the priority queue H .
4. Continue at Step 2 if the priority queue H is non-empty.

The initialization (Step 1) takes $O(n \log n)$ time. Step 2 takes $O(\log n)$ time for each execution. In Step 3, the split and concatenate of at most four binary trees, takes $O(\log n)$ time for each tree. Updating the best score information in the priority queue also takes $O(\log n)$ time. So, overall, the algorithm takes $O(n \log n)$ time in the worst case.

Note that this algorithm is not applicable for the second addition method, where we add up the distance to closest and the distance to the query. This is easy to see, since the distance to the closest ranked point does not change by the same amount for a group of points. This implies that the score for every unranked point needs to be adjusted individually when adding a point to R , which is done by the two basic algorithms.

Theorem 18 *A set of n points in the plane can be ranked according to the angle-distance addition method in $O(n \log n)$ time.*

The wavefront algorithm

The linear wavefront method generates a ranking as follows (see Figure 5.5). Assume a set P of n points in the plane is given, and also an angle ρ which, intuitively, captures how much preference should be given to scattering. It is the angle between

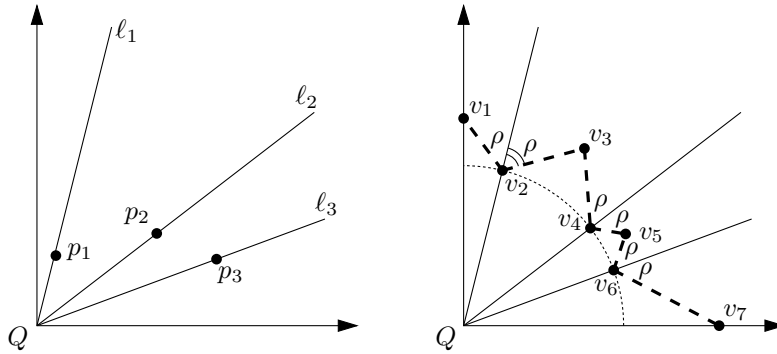


Figure 5.5: Illustration of the linear wavefront method.

a segment of the wavefront and the nearest (by angle) line through Q and a ranked point. Let p_1, \dots, p_i be the points ranked so far. Draw lines ℓ_1, \dots, ℓ_i where ℓ_j passes through Q and p_j , where $1 \leq j \leq i$ and remove duplicates, if any. Assume without loss of generality that the lines ℓ_1, \dots, ℓ_i are sorted by slope in non-increasing order (steepest first). Consider a circle C_s with radius s centered at Q . The wavefront for p_1, \dots, p_i and s (Figure 5.5) is defined as the polygonal line $v_1, v_2, \dots, v_{2i+1}$, where v_{2j} is the intersection point of ℓ_j and C_s for $1 \leq j \leq i$, and the edges $v_{2j-1}v_{2j}$ and $v_{2j}v_{2j+1}$ include an angle ρ with the line ℓ_j . This defines the position of the vertices $v_3, v_5, \dots, v_{2j-1}$. Vertex v_1 is on the y -axis such that v_1v_2 includes an angle ρ with ℓ_1 , and vertex v_{2i+1} is on the x -axis such that $v_{2i}v_{2i+1}$ includes an angle ρ with ℓ_i . When s is increased, the segments of the wavefront move away from Q and they get longer, but they keep their orientation.

Algorithm 4: The first point in the ranking is the point of P that is closest to the query Q . The next point p in the ranking is determined as follows. Start with $s = 0$ and increase s continuously until some segment of the linear wavefront hits the first non-ranked point. This point p is ranked next. Then restart (or continue) the growing of s with the new wavefront, which has two more vertices and two more segments.

To implement this ranking scheme efficiently, we use a geometric data structure that allows us to find the next point p in $O(\log^2 n)$ time. Since we will perform $n - 1$ queries, the total time to rank will be $O(n \log^2 n)$. The data structure to be designed stores all unranked points and must be able to answer queries $\text{SEGMENT-DRAG}(\phi, \psi, \rho)$, where the first point hit by a moving and growing line segment must be found. The moving and growing line segment is specified by the three angles ϕ , ψ , and ρ , and is defined by two consecutive vertices of the wavefront. Let ℓ_ϕ and ℓ_ψ be the lines through Q that have angle ϕ , respectively ψ with the positive x -axis. The line segment that is moved and grows has its endpoints on ℓ_ϕ and ℓ_ψ , and includes an angle ρ with ℓ_ϕ (if the endpoint on ℓ_ϕ is closer to Q than the endpoint on ℓ_ψ , or with ℓ_ψ otherwise).

Assume that all points of P lie between the lines ℓ_ϕ and ℓ_ψ . Then the query problem reduces to querying with a full line that lies outside the convex hull of P

and is translated towards P . The data structures of Hershberger and Suri [88] and Brodal and Jacob [25] have linear size, $O(\log n)$ deletion time, and answer queries in $O(\log n)$ time amortized. These data structures are dynamic convex hull query structures.

For a data structure that can answer $\text{SEGMENTDRAG}(\phi, \psi, \rho)$, we store the set P sorted by angle in the leaves of a binary search tree T . For an internal node ν , denote by T_ν the subtree of T rooted at ν , and by $P_\nu \subseteq P$, the subset of points that are stored in the leaves of T_ν . For any internal node ν , we store a pointer to an associated structure T'_ν which stores the points P_ν in a data structure for dynamic convex hull queries [25, 88]. The technique of extending a data structure by adding a one-dimensional range restriction (here on angle) is quite standard [131, 173] and used for instance for orthogonal range queries [46]. The storage requirements, preprocessing time, query time, and update time all increase by a factor of $O(\log n)$. This implies that a SEGMENTDRAG query can be answered in $O(\log^2 n)$ time with a data structure that has size $O(n \log n)$, construction time $O(n \log^2 n)$, and deletion time $O(\log^2 n)$.

To obtain an $O(n \log^2 n)$ time algorithm for ranking in the linear wavefront method, we maintain a set of candidate points to be ranked next in a priority queue. We store one point for each sector defined by lines ℓ_j and ℓ_{j+1} , where ℓ_j (and ℓ_{j+1}) is the line through Q and the point v_j (resp. v_{j+1}) on the linear wavefront, where $1 \leq j \leq 2i$. The point we store in a sector is the one that is hit for the lowest value of s , the radius of the circle that defines the wavefront. The overall best candidate, and hence the point to be ranked next, is the one with the lowest value of s among all candidates. Since the candidates are stored in a priority queue H , the best one can be found and extracted in $O(\log n)$ time. After selecting and ranking a point p we must update the priority queue H and the data structure for SEGMENTDRAG queries. The first update is easy; it is the deletion of point p . To update H further, we must first delete the best candidate in every sector that has changed due to the ranking of p . There can be at most two such sectors, so we delete two points from the priority queue. Then we must find the new best candidate in every new sector arising due to the ranking of p . There are at most four new sectors, and we find the new candidates by four SEGMENTDRAG queries. These new candidates are then inserted in the priority queue H , which prepares it for ranking the next point. We conclude with the following theorem:

Theorem 19 *A set of n points in the plane can be ranked according to the linear wavefront method in $O(n \log^2 n)$ time.*

5.3 Extensions of the Basic Ranking Methods

In this section we present extensions of the basic ranking methods introduced in Sections 5.1 and 5.2. We divided the given point set P into the set of already ranked points R and the set of unranked points U during the algorithms. We chose the candidates to be ranked next from all points in U and computed the score functions with respect to all ranked points in R . In this section we present two extensions where this is not the case anymore. The staircase enforcement extension limits the set of candidates to be ranked next to the points in U that lie on the lower left staircase of the point set P . The limited windows method limits the set of ranked

reference points to be used for the score computation to only the last k ranked points. Furthermore, we will describe the necessary adaptations of the algorithms in higher dimensions.

5.3.1 Staircase enforcement

In the basic methods, every unranked point was eligible to be next in the ranking, if it had the highest score. This can lead to a ranking where a point is ranked before other points that are better in both aspects. Often this is an undesirable outcome. As an alternative, we choose the candidates to be ranked next only from the points p that lie on the lower left staircase of the point set. A point p is on the lower left staircase of the point set P if and only if for all $p' \in P \setminus \{p\}$, we have $p_x < p'_x$, or $p_y < p'_y$, or $p_x = p'_x$ and $p_y = p'_y$. If we always select from the staircase of unranked points, we automatically obtain the property that any ranked document is more relevant in at least one aspect than all documents that are ranked later.

We can easily adapt the basic ranking algorithms for staircase enforcement. We consider as eligible candidates for next in the ranking only the points on the staircase of unranked points, which we maintain throughout the algorithm. When a point is ranked, the staircase has to be updated, and points that are new on it become eligible too. Computing the staircase of a point set with n points takes $O(n \log n)$ time, insertion or deletion of one point takes $O(\log n)$ time per point. In every update of the staircase, we delete one point from it. There can be as many as a linear number of points that need to be inserted in one update, so a single update can have a running time of $O(n \log n)$. However, every point of P is inserted to the staircase and deleted from it exactly once. This means that maintaining the staircase takes only $O(n \log n)$ time during the entire execution of the algorithm.

In the following, a brief description of the necessary changes to the basic algorithms is given.

In the generic Algorithm 1, which can be used for all models, we compute the staircase at the beginning, and then we only need to change Step 3 as follows:

3. Update the staircase of P and choose from the points on the staircase the one with highest score $S(p, R)$ as next in the ranking; add it to R and delete it from P and from the staircase.

Computing the staircase in the beginning takes $O(n \log n)$ time. We only need to do this once. Updating the staircase in Step 3 takes $O(\log n)$ time per insertion and deletion, and the other computations can be performed in constant time. As this step needs to be repeated n times, it takes $O(n \log n)$ time overall, and hence, staircase enforcement does not influence the total running time of $O(n^2)$.

In Algorithm 2 for the angle and distance models, we maintain the Voronoi diagram of all ranked points, and for each Voronoi cell, a list of all unranked points in that cell. The main adaptation is that only points on the staircase of unranked points are stored in the priority queue. That is, for every list of a Voronoi cell the point with highest score and that lies on the staircase will be in the priority queue. We must also maintain the staircase of unranked points throughout the algorithm to

have this information. Neither the worst case nor the typical time analysis change in this case, so the worst case time bound is still $O(n^2)$, and the typical running time remains $O(n \log n)$.

For Algorithm 3 we change the algorithm as follows: We start by computing the staircase of the point set $P \setminus \{r\}$ and create the initial, augmented trees T_0 and T_1 only for the points from the staircase. The priority queue is initialized with the best points from T_0 and T_1 . We rank the next point p , update the corresponding trees and the priority queue as before, and delete p from the staircase. Step 4 needs to be modified as follows:

4. Update the staircase to incorporate the deletion of p . Insert the new points on the staircase into the binary trees while keeping the augmentation correct. Update the best-score information in the priority queue. Continue with Step 2 if the priority queue is non-empty.

The new version of Step 4 takes $O(n \log n)$ time overall, so the total running time of $O(n \log n)$ is not affected.

Algorithm 4 for the wavefront model is also easy to adapt. In the original model, we move a line segment towards the convex hull of the whole point set P and rank the first point that is hit next. Here, the point set is restricted to the points of P that lie on the staircase of P . We do not need to change any of the data structures described in Section 5.2, but we need to precompute the staircase once ($O(n \log n)$ time) and maintain it through the algorithm ($O(\log n)$ time per insertion and deletion). This leaves the asymptotic running time of the wavefront algorithm unchanged.

5.3.2 Limited windows

In all previously presented basic ranking methods as well as in the staircase enforced extensions, the closest point from the set R of ranked points is used to determine the score of an unranked point. However, if the second best point lies very close to the best point, it should not be late in the ranking just because it is very similar to the best point. We would like the influence of a ranked point to stop after, say, another ten points have been ranked after it. This is captured in the limited windows extension, where only the k latest ranked points are kept in evidence for the future ranking. The value of k is fixed throughout the algorithm.

The general idea is as follows. We store the set of ranked points that we consider in a queue W , which we will call the window. The basic algorithms remain exactly the same as long as the size of W does not exceed k . When the $(k + 1)$ -th point is ranked, we add it at the tail of the queue and delete the point p at the head of it. For all unranked points that had the deleted point p as their closest, we need to find the new closest point among the k ranked points in W and recompute their score.

We give a brief description of the necessary adaptations of the basic algorithms in the next paragraphs. An adaptation of the staircase enforced methods to get a combination of the two extensions is straightforward and therefore not given here. We assume that k is fixed.

The adaptation of Algorithm 1 is straightforward. We now determine in Step 4 for all unranked points p' the smallest distance to the k last ranked points in R

to determine the score of each unranked point. This clearly takes $O(kn)$ time for each next ranked point, so the overall running time of the algorithm is $O(kn^2)$. If we allow $O(kn)$ extra storage, we can store with every unranked point all k last ranked points sorted by distance. We can keep the sequence of all k points sorted in $O(n^2 \log k)$ time in total.

Algorithm 2 for the angle or distance model is adapted as follows. We keep all ranked points in a queue W . When W contains more than k points, we delete the first point p from the queue and from the Voronoi diagram of R . All unranked points that lie in the Voronoi cell of p need to be redistributed to the at most k neighboring cells, which means that their lists of unranked points need to be updated. Furthermore, we need to recompute the highest score value from these k lists and update the priority queue H accordingly.

The worst case time analysis for deleting one point from the window is as follows. The operations on W only take constant time each. Deleting the point p from the Voronoi diagram can be done in $O(k)$ time. There can be a linear number of unranked points that need to be redistributed, which takes $O(kn)$ time in the worst case. Updating the k lists can be done in $O(n)$ time in total and updating the priority queue takes $O(k \log k)$ time. This leads to an overall worst case running time of $O(kn^2)$. This can be improved to $O(n^2 \log k)$ time by doing the redistribution more cleverly.

In a typical time analysis we can delete a point p from the queue and the Voronoi diagram in $O(1)$ time. The list of p contains—on the average and once k points are ranked— $O(n/k)$ points, and redistributing them to the $O(1)$ neighboring cells takes $O(n/k)$ time. Updating the priority queue H takes $O(\log k)$ time typically. Overall, we therefore get a typical running time of $O(n^2/k + n \log k)$.

Algorithm 3 for the addition model can be easily adapted. As before, every ranked point is added to a queue W . When it contains more than k points, the first element p is deleted from the head of the queue, which means that we delete the imaginary line ℓ that passes through p . We have to concatenate the four binary trees that lie between p and its closest clockwise neighbor p' and counterclockwise neighbor p'' , thus creating one tree \bar{T} . We then split \bar{T} at the angle of the bisecting barrier line $\ell_{p'p''}$. During the concatenate and split operations we need to keep the augmentation correct. Finally, we need to update the best score information in the priority queue H , which means that we delete the best scores of the old trees and insert the best scores of the new trees. The analysis of the running time is similar as before, and the asymptotical running time does not change.

Algorithm 4 for the wavefront method can be adapted in a similar manner. We add every ranked point to a queue W , and if it contains more than k points, the first element p is deleted from the head of the queue, which means that we delete the line ℓ_p that passes through p . Thus, we create one new sector out of two old ones. We delete the three vertices v_{p-1} , v_p , and v_{p+1} of the wavefront and add one new vertex, namely the intersection of the two edges that make an angle ρ with the bounding lines of the new sector. Finally, we need to delete the two old candidate points of the old sectors from the priority queue and insert the new candidate point

of the new sector into it. The overall running time remains $O(n \log^2 n)$ in the worst case.

5.3.3 Higher dimensions

So far, we have considered only two possible scores that are combined to give a scattered ranking. The applications from the introduction show that dealing with more than two scores for each point of the point set P can also be useful. For example, for the query ‘castles near Koblenz’, there could be two castles with the same distance to Koblenz, one north and one south of it. If the documents about them have a similar textual score, they will be mapped to points in the plane which are very close to each other, and therefore they will be ranked apart. However, in this example, this outcome is unsatisfactory, and it would be better if we could distinguish the relative position of the castles to Koblenz by using two spatial dimensions (scores) and one textual score.

The four presented methods can be used for any number of scores. Computing the ranking requires extending the basic algorithms to higher dimensions. However, not all of the algorithms presented in Sections 5.1 and 5.2 can be extended to work as efficiently in higher dimensions. In this section we will briefly discuss the extensions of the presented algorithms to higher dimensions, where possible, or otherwise, state why this cannot be done.

The generic Algorithm 1, which can be applied to all four presented models, works as presented in any dimension d . Therefore, we conclude that the worst case running time for the generic algorithm is $O(n^2)$ in any dimension.

Algorithm 2 for the basic models can be extended to higher dimensions $d \geq 3$, but the worst case running time will go up with the dimension. We maintain the d -dimensional Voronoi diagram for the ranked points, and for each cell we maintain the unranked points in that cell in a list. The point with the highest score value per list is stored in a priority queue, where the next point to be ranked is chosen from.

For $d \geq 3$, the Voronoi diagram has complexity $O(n^{\lceil d/2 \rceil})$ in the worst case, and it can be constructed in $O(n^{\lceil d/2 \rceil})$ time [69]. It can also be constructed in $O(N \log n)$ time, where N is the actual complexity of the Voronoi diagram [151]. This leads to $O(n^{\lceil d/2 \rceil})$ time to rank a set of n points in d dimensions, $d \geq 3$, in the distance model. The Voronoi diagram for the angle model in three dimensions can be determined as follows: we project the ranked points to the surface of the unit sphere and compute the Voronoi diagram on this surface, which essentially is a 2-dimensional Voronoi diagram. In general, the Voronoi diagram needed for the angle model for point sets in d -dimensional space has dimension $d - 1$. Consequently, in the angle model, ranking can be done in $O(n^{\lceil (d-1)/2 \rceil}) = O(n^{\lfloor d/2 \rfloor})$ time for $d \geq 4$, and in $O(n^2)$ time for $d = 3$.

To analyze the typical running time for the algorithm in higher dimensions, we will make some assumptions and show time bounds under these assumptions. Whether the assumptions hold in practice cannot be verified without implementation. We will assume that any newly ranked point only has a constant number of neighbors in the Voronoi diagram, and every list has length $O(n/r)$, where r is the

number of ranked points so far. Note that the former assumption is, in a sense, stronger than in the planar case, because there are point sets where the average number of neighbors of a cell in the Voronoi diagram is linear, when $d \geq 3$. However, for uniformly distributed point sets, a higher-dimensional Voronoi diagram has linear complexity [69], and the average number of neighbors of a cell is indeed constant. Under these two assumptions, we can obtain a running time of $O(n \log n)$, as in the planar case.

Algorithm 3 for the addition model cannot be extended to higher dimensions $d \geq 3$. In the plane we have a linear ordering by angle on the point set P . This ordering is crucial for the functioning of the algorithm, because the split and concatenate operations are with respect to this ordering. In three or more dimensions the approach does not apply anymore.

It is possible to extend the wavefront model to higher dimensions, but the higher-dimensional version of Algorithm 4 would be complex and considerably less efficient. It would require linearization and higher-dimensional partition trees. Because of the highly increased problem complexity in higher dimensions we will not discuss the wavefront model any further.

5.4 Experiments

We implemented the generic ranking Algorithm 1 for the basic scattered ranking methods described in Subsections 5.1.1, 5.1.2, and 5.2.1. Furthermore we implemented the extensions for staircase enforcement (Subsection 5.3.1) and limited windows (Subsection 5.3.2). We compare the outcomes of these algorithms for the two different point sets shown in Figure 5.6. The point set on the left consists of 20 uniformly distributed points, and the point set on the right shows the 15 highest ranked documents for the query ‘safari africa’ which was performed on a data set consisting of 6,500 Lonely Planet web pages. The small size of the point sets was chosen out of readability considerations.

5.4.1 Basic ranking algorithms

Figure 5.6 shows the output of a ranking by distance to query only. It is useful as a reference when comparing with the other rankings.

In the other basic ranking methods, shown in Figure 5.7, points close together in the plane are not necessarily close in the ranking sequences. This is visible in the ‘breaking up’ of the cluster of four points in the upper left corner of the Lonely Planet point set rankings. Note also that the points ranked last by the simple distance ranking are always ranked earlier by the other methods. This is because we enforced higher spreading over proximity to the query by the choice of parameters. The rankings are severely influenced by this choice. In our choice of parameters we did not attempt to obtain a ‘best ranking’ for the particular methods. We used the same parameters in all three ranking methods presented here, to simplify qualitative comparison.

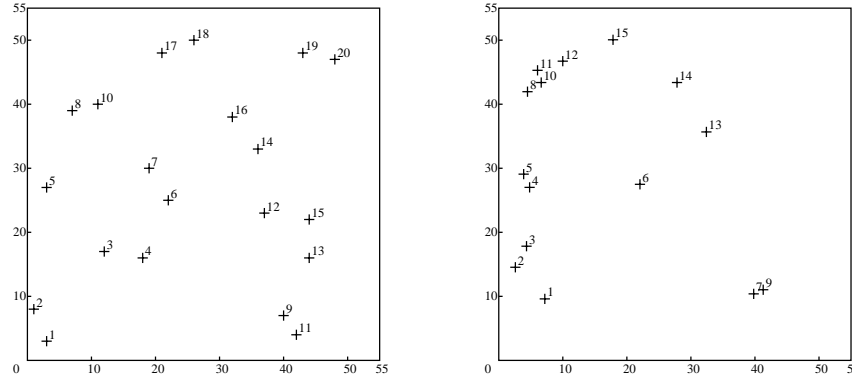


Figure 5.6: Ranking by distance to origin only.

5.4.2 Staircase enforced ranking algorithms

In the staircase enforced methods, shown in Figure 5.8, the candidates to be ranked next are only those points that lie on the (lower left) staircase of the unranked points. The scoring functions and parameters are as before. With this adaptation, proximity to the query gets a higher importance than before. This is clearly visible in the figures, as the points farthest away from the query are almost always ranked last. Still, spreading in the rankings is present. This can for instance be seen from the two points in the lower right corner of both point sets: they are the closest pair of points in space, but their ranking differs significantly. In fact, this is just the behavior we expected from our scattered ranking methods.

5.4.3 Ranking algorithms with limited windows

In the final experiments, shown in Figure 5.9, the already ranked reference point we need to compute angle or distance to, is not the closest of all ranked, but the closest among the five last ranked points. This way we want to avoid that the angle or distance to the closest becomes too small to ensure a good scattering. The scoring functions and parameters are as before. With a window size of five, the first six ranked points are the same as with the basic models. It can be seen that points that lie very close to early ranked points are now ranked higher than with the basic methods, although the overall ranking remains well spread.

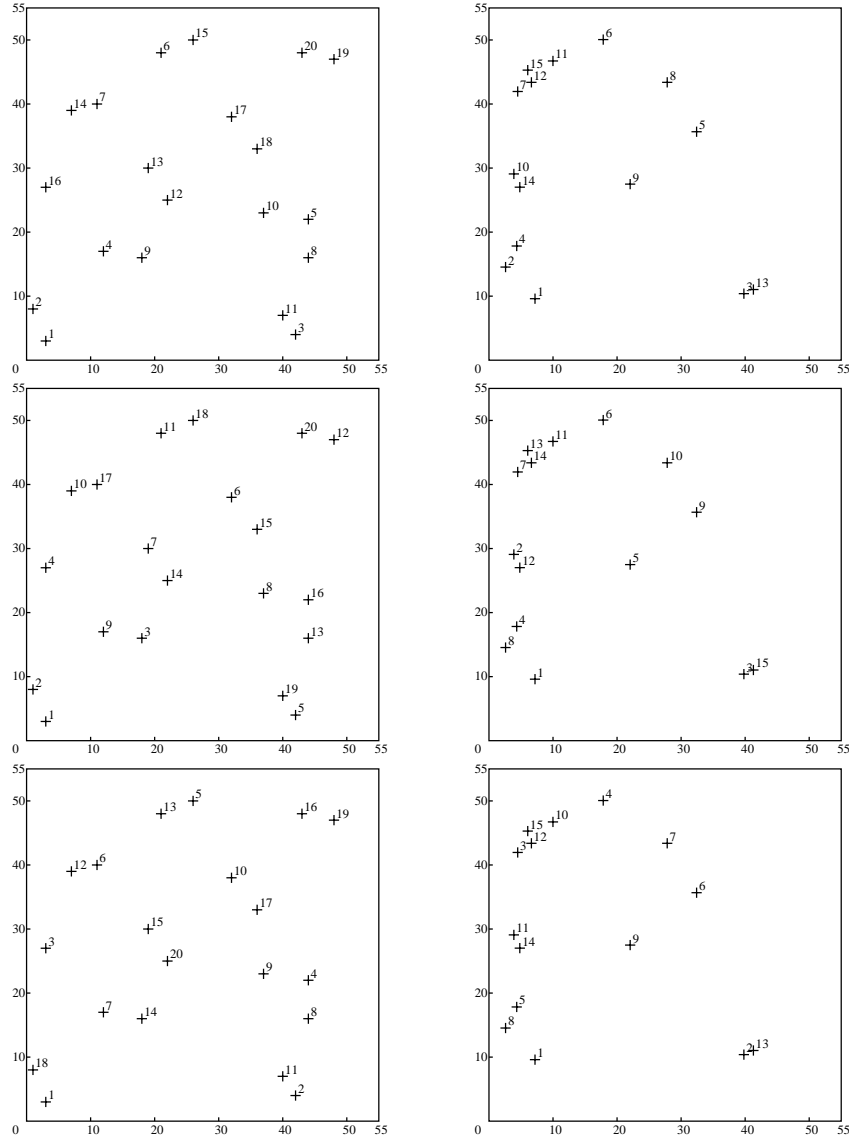


Figure 5.7: A comparison of the basic ranking methods. Top: Ranking by distance to origin and angle to closest ($k = 1, c = 0.1$). Middle: Ranking by distance to origin and distance to closest (Equation 5.3, $\lambda = 0.05$). Bottom: Ranking by additive distance to origin and angle to closest ($\alpha = 0.4, \lambda = 0.05$).

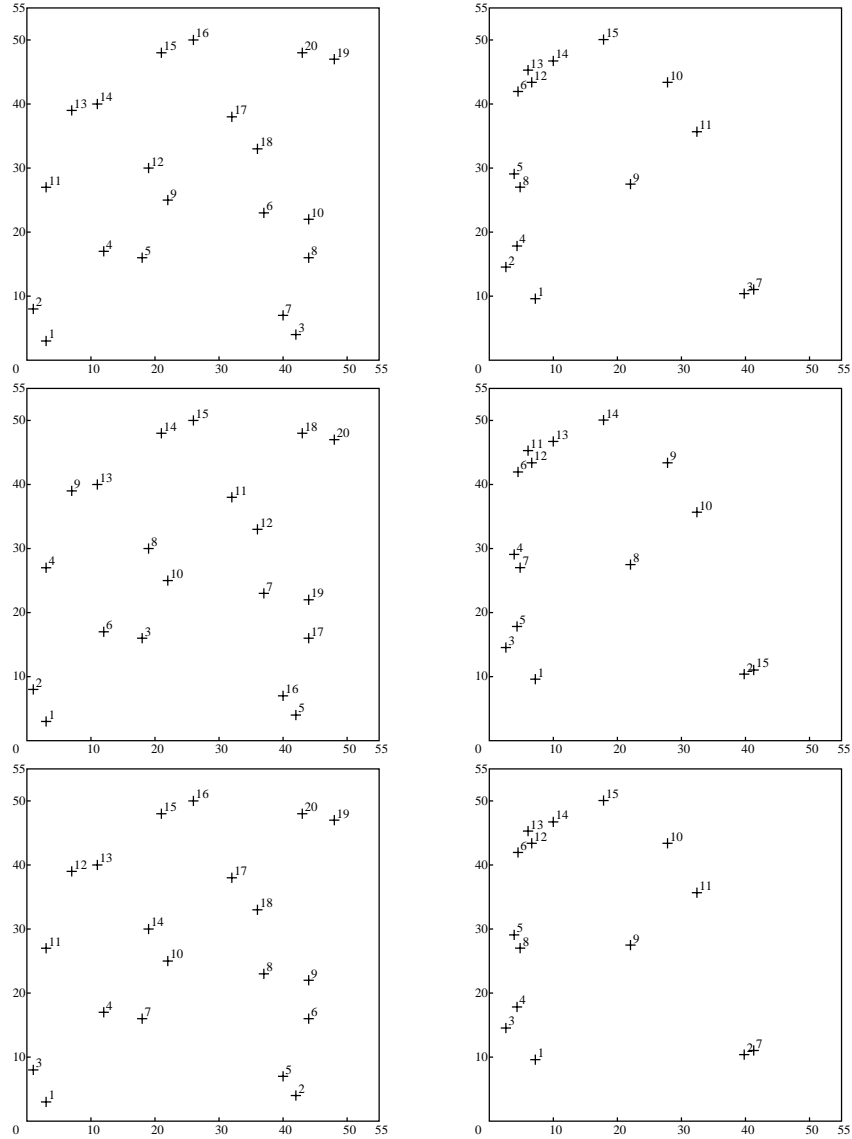


Figure 5.8: A comparison of the staircase enforced methods. Top: Ranking by distance to origin and angle to closest ($k = 1, c = 0.1$). Middle: Ranking by distance to origin and distance to closest (Equation 5.3, $\lambda = 0.05$). Bottom: Ranking by additive distance to origin and angle to closest ($\alpha = 0.4, \lambda = 0.05$).

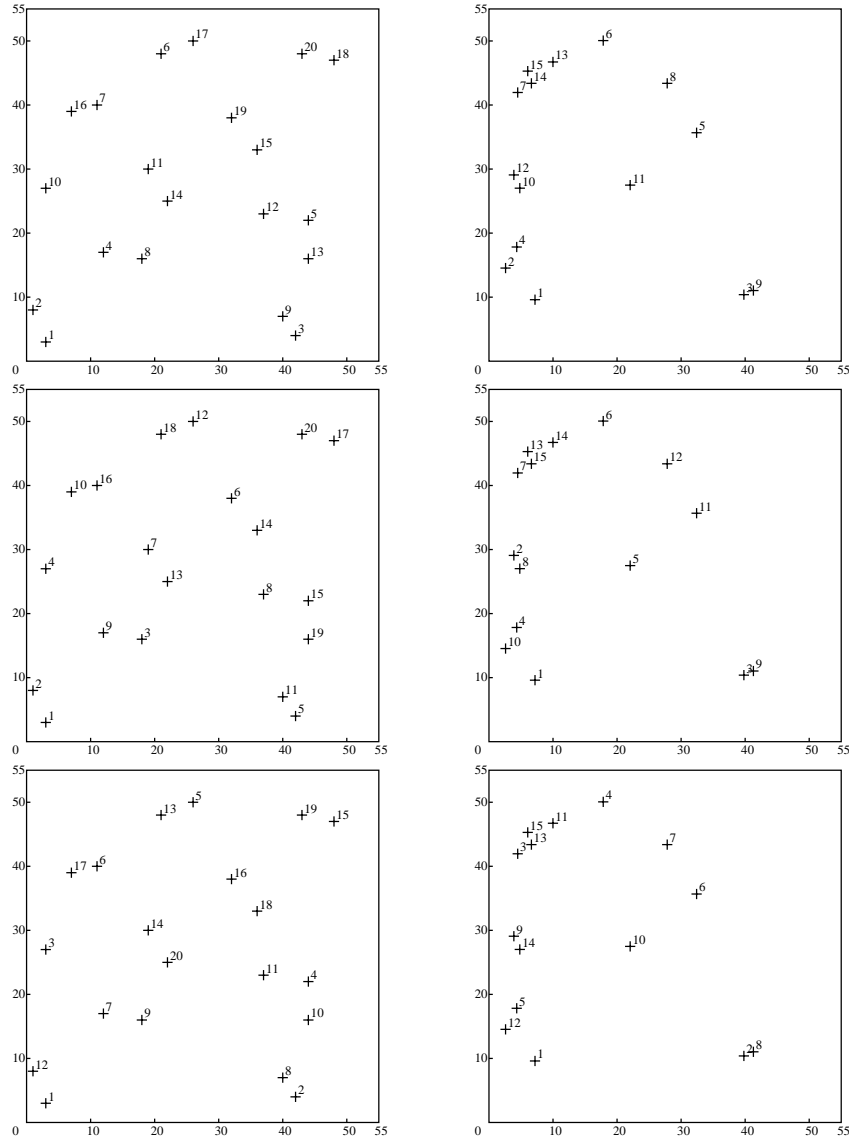


Figure 5.9: A comparison of the basic methods with a limited window of size 5. Top: Ranking by distance to origin and angle to closest ($k = 1, c = 0.1$). Middle: Ranking by distance to origin and distance to closest (Equation 5.3, $\lambda = 0.05$). Bottom: Ranking by additive distance to origin and angle to closest ($\alpha = 0.4, \lambda = 0.05$).

5.5 Concluding Remarks

This chapter introduced multi-dimensional scattered relevance ranking for documents that have two or more scores. It is particularly useful for geographic information retrieval, where documents have both a textual and a spatial score. Scattered ranking combines high relevance of the query while avoiding redundancy. To our knowledge, it is the first time that these issues have been addressed with a purely geometric approach.

We have presented several methods for scattered ranking that are mainly based on a combination of the distance to the query and the distance to the closest already ranked point. Furthermore, we presented two extensions of the given basic methods, by staircase enforcement and limited windows. We first presented a generic algorithm with quadratic running time, which can be used for any of the presented methods and in any dimension. For some of the methods we can give efficient algorithms with better worst case running time.

The visual inspection of the outcome of the conducted experiments show that both requirements for a good ranking, small distance to query and high spreading, can be obtained simultaneously. Especially the staircase enforced methods and the methods with limited windows seem to perform well.

Initial experiments conducted using the SPIRIT prototype indicate that the basic scattered ranking methods based on a spatiotextual index perform better than a ranking based on a textual index only [139]. However, these experiments are time consuming and need a large number of volunteers to be conclusive. At this moment, there is not enough data available to claim any stronger statement. Furthermore, a user evaluation is still needed to discover which of the described scattered ranking methods is preferred, and which specific parameters should be used.

6

CONCLUSIONS AND FUTURE WORK

How to model and determine geographical regions with imprecise boundaries, like the south of Austria or the German Schwarzwald, has attracted the interest of researchers of various disciplines. Research in linguistics and cognitive science aims to give definitions for geographical features like lakes, hills, and forests, or investigates the reason why many people can easily place the town they live in inside a certain part of their country, but are surprised by the fact that North and South America are not aligned in north-south direction (South America is more to the east).

Defining boundaries for imprecise geographical regions has been a research issue in geographic information systems for a number of years [26]. Giving crisp boundaries for regions that have none, is important in GIS and GIR to be able to answer queries, for example, for hotels in northern Spain, or for Aikido dojos in the German Ruhrgebiet.

This thesis introduced a number of geometric, algorithmic approaches to determine reasonable, crisp boundaries for various types of imprecise geographical regions. In Chapters 2 to 4, we presented a number of different geometric methods to delineate various types of imprecise geographical regions. The presented methods are based on simple geometric specifications, like the equal area share of the good NEWS regions in Chapter 2, or the requirement of finding the shortest boundary in the adaptation method of Chapter 3. Our methods do not take any kind of cognition, perception, or personal knowledge into account. This has the advantage that any kind of region can be delineated without being biased, prejudiced, or just plain wrong because of what people think they know. However, this is also a drawback, as including this knowledge into our approaches may lead to better boundaries. Furthermore, our methods do not take any existing natural boundaries like rivers, mountain ranges, or change in vegetation into account. By including them into our approaches, the derived boundaries may be better and appear more natural.

We presented in Chapter 3 several methods to derive boundaries of imprecise regions from data points that are classified as inside or outside. This is the most versatile way examined in this thesis to address the delineation problem. First, because any kind of point data can be used, for example census data on the average household income in a town, the portion of people speaking a certain dialect, or belonging to an ethnic minority. Also point data that is sampled in the field, like soil classes and vegetation types at a certain location, can be used as input to these models. Second, because these methods can most easily include non-point data rep-

representing natural boundaries like mountain ranges or coast lines. In this context, assigning weights to the data representing the confidence in the inside/outside classification is a useful extension. This way it is possible to derive boundaries with different values of confidence, which is similar to a fuzzy set approach.

So far, we have only implemented the basic methods to delineate imprecise regions, and they do well for most of the instances we have tested. However, for all presented methods it will be necessary to do more extensive testing on various input data to verify whether they perform well in most cases, or whether there are classes of instances where they fail completely. It is also desirable for the applications to find fixed parameters that lead to a good result in most cases, rather than manually tuning them. It is also possible that an incorporation of the suggestions given above may lead to even better results of our methods.

The basic methods for the ranking of spatio-textually relevant documents, presented in the previous chapter, have been implemented and used in the SPIRIT Web search engine. So far, with evaluation still underway, we can only state that our methods give a ranking different than the classical text-only based ranking. What remains to be done is a user evaluation, where the users are asked which ranking best fulfills their needs and expectations. Also, finding good standard values for the parameters that influence the spreading of our methods is an open problem.

Other research questions in GIS and GIR involving imprecise regions include the following. Imprecise regions can also be delineated in three dimensions. An example is to compute the volume of high concentration of a certain natural resource, like coal or ore, to determine if mining is economically justifiable, or the extent of soil contamination around a factory. In these applications, we can assume that the data is sparse and mainly classified correctly. Therefore, the adaptation method may be the method of choice here, as it does not change the classification of the data, except for obvious outliers. Allowing a certain amount of clearance around the data points and thus taking the sparseness of the data into account, may improve the results of this method. How to efficiently implement the adaptation method in three dimensions is unclear; the resulting region will consist of one or more polyhedra. An extension of the good NEWS delineation method to three dimensions would yield a partition of a three-dimensional object into parts that resemble the classification top/bottom, left/right, and front/back, possibly with an additional center part. This can be used in a second step of the application mentioned above, to give a sequence of parts in which the minings should take place, e.g., starting with the top and center parts, and eventually ending with the bottom part.

How to give spatial scores to a document is another main issue of GIR. A spatial score for point data is often based on Euclidean distance. For two-dimensional regions, the Euclidean distance as well as other measures, like the area of overlap, could be included in assigning a spatial score. Different ways of determining a spatial score are possible. First of all, it is unclear which reference points to choose to determine the Euclidean distance between two regions. For example, we can choose the reference point to be the center of gravity of the region, but in this case we may have the problem that the center lies outside. A way to overcome this problem is to compute the largest enclosed circle of the region and choose its center to be the reference point. Second, we should take the topological relations between

two regions into account when assigning a spatial score. If one region lies fully inside another, the assigned score should be higher than if they only partially overlap. We could build on the system of topological relations of Egenhofer and Herring [60], and give different basic scores to different types of relations.

Related to assigning spatial scores is to give a town inside a region a weight describing its degree of 'north', 'central', etc. For example, both Utrecht and Amsterdam lie in the central area of the Netherlands, but Utrecht should get a higher weight for 'central' than Amsterdam. In this context, defining a center which is assigned the highest weight is interesting. The center could be a point of the region, the weight linearly decreasing with increasing distance from it. This point could be defined by using the medial axis of the boundary, and then retracting it to one center point. Or the center could be defined as a central region by defining buffer zones around the boundary of the whole region, where each buffer zone is assigned a different weight, increasing as it becomes more centered.

Overall, applying geometric, algorithmic methods to address problems in GIS and GIR may provide new impulses for these areas. Any problem that can be abstracted and modeled in a strictly geometric manner and is solved in an efficient algorithmic way, will give useful results. These results, which depend on the way the problem is modeled, will lead to new insight into GIS and GIR.

BIBLIOGRAPHY

- [1] M. Abellanas, F. Hurtado, and P. Ramos. Structural tolerance and delaunay triangulation. *Inf. Proc. Lett.*, 71:221–227, 1999.
- [2] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Computational Geometry: Theory and Applications*, 21:39–61, 2002.
- [3] P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.
- [4] P. K. Agarwal and S. Suri. Surface approximation and geometric partitions. *SIAM J. Comput.*, 27:1016–1035, 1998.
- [5] H. Alani, C. Jones, and D. Tudhope. Voronoi-based region approximation for geographical information retrieval with gazetteers. *Int. J. Geographical Information Science*, 15(4):287–306, 2001.
- [6] E. Amitay, N. Har’El, R. Sivan, and A. Soffer. Web-a-where: Geotagging web content. In *Proc. 27th Annu. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 273–280, 2004.
- [7] ANWB. Campinggids 2. 2002.
- [8] A. Arampatzis, M. van Kreveld, I. Reinbacher, C. B. Jones, S. Vaid, P. Clough, H. Joho, and M. Sanderson. Web-based delineation of imprecise regions. *Computers, Environment and Urban Systems (CEUS)*, in press; available online: doi:10.1016/j.compenvurbsys.2005.08.001.
- [9] E. M. Arkin, F. Hurtado, J. S. B. Mitchell, C. Seara, and S. S. Skiena. Some separability problems in the plane. In *Abstracts 16th European Workshop Comput. Geom.*, pages 51–54. Ben-Gurion University of the Negev, 2000.
- [10] E. M. Arkin, S. Khuller, and J. S. B. Mitchell. Geometric knapsack problems. *Algorithmica*, 10:399–427, 1993.
- [11] E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 153–156, 1991.
- [12] E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Minimum-link watchman tours. *Inform. Process. Lett.*, 86(4):203–207, 2003.
- [13] S. Arora and K. Chang. Approximation schemes for degree-restricted MST and red-blue separation problem. *Algorithmica*, 40(3):189–210, 2004.

- [14] T. Asano, S. K. Ghosh, and T. C. Shermer. Visibility in the plane. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 829–876. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
- [15] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.
- [16] D. Bandyopadhyay and J. Snoeyink. Almost-delaunay simplices: nearest neighbor relations for imprecise points. In *SODA '04: Proc. 15th Annu. ACM-SIAM Symp. Disc. Alg.*, pages 410–419, 2004.
- [17] G. Bashein and P. R. Detmer. Centroid of a polygon. In P. Heckbert, editor, *Graphics Gems IV*, pages 3–6. Academic Press, Boston, MA, 1994.
- [18] K. Beard and V. Sharma. Multidimensional ranking for data in digital spatial libraries. In *Int. J. of Digital Libraries*, pages 153–160, 1997.
- [19] B. Ben-Moshe, M. Katz, and J. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Proc. 16th ACM-SIAM Symp. Discrete Algorithms*, pages 515–524, 2005.
- [20] B. Bennett. What is a forest? On the vagueness of certain geographic concepts. *Topoi*, 20:189–201, 2001.
- [21] K.-F. Böhlinger, B. Donald, and D. Halperin. On the area bisectors of a polygon. *Discrete and Computational Geometry*, 22(2):269–285, 1999.
- [22] J.-D. Boissonnat and S. Lazard. Convex hulls of bounded curvature. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 14–19, 1996.
- [23] P. Bose, J. Czyzowicz, E. Kranakis, D. Krizanc, and D. Lessard. Near-optimal partitioning of rectangles and prisms. In *Proc. 11th Canad. Conf. Comput. Geom.*, pages 162–165, 1999.
- [24] P. Bose, J. Czyzowicz, E. Kranakis, D. Krizanc, and A. Maheswari. A note on cutting circles and squares in equal area pieces. In *Proc. FUN with Algorithms '98*, 1998.
- [25] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd IEEE Sympos. Found. Comput. Sci.*, pages 617–626, 2002.
- [26] P. Burrough and A. Frank, editors. *Geographic Objects with Indeterminate Boundaries*, volume II of *GISDATA*. Taylor & Francis, London, 1996.
- [27] P. A. Burrough and R. A. McDonnell. *Principles of Geographical Information Systems*. Oxford University Press, 1998.
- [28] G. Cai. GeoVSM: An integrated retrieval model for geographic information. In *Proc. 2nd GIScience*, number 2478 in *Lect. Notes in Computer Science*, pages 65–79, 2002.
- [29] L. Cai and J. M. Keil. Computing visibility information in an inaccurate simple polygon. *Internat. J. Comput. Geom. Appl.*, 7:515–538, 1997.
- [30] J. G. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Research and Development in Information Retrieval*, pages 335–336, 1998.
- [31] A. Carrara, G. Bitelli, and R. Carla. Comparison of techniques for generating digital terrain models from contour lines. *Int. J. Geogr. Inf. Sys.*, 11(5):451–473, 1997.
- [32] W. Caspary and R. Scheuring. Positional accuracy in spatial databases. *Computers, Environment and Urban Systems*, 17(2):103–110, 1993.
- [33] The CGAL homepage <http://www.cgal.org/>.

-
- [34] T. M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005.
- [35] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.
- [36] Z. T. Chen and W. R. Tobler. Quadtree representation of digital terrain. In *Proc. Autocarto*, pages 475–484, 1986.
- [37] T. Cheng, P. Fisher, and Z. Li. Double vagueness: Effect of scale on the modelling of fuzzy spatial objects. In *Developments in Spatial Data Handling, Proc. 11th Int. Symp. on Spatial Data Handling*, pages 299–314, 2004.
- [38] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph.*, 14(3):203–232, 1995.
- [39] E. Clementini and P. D. Felice. An algebraic model for spatial objects with indeterminate boundaries. In P. Burrough and A. Frank, editors, *Geographic Objects with Indeterminate Boundaries*, volume II of *GISDATA*, pages 155–169. Taylor & Francis, London, 1996.
- [40] P. D. Clough, H. Joho, C. B. Jones, and R. Purves. Modelling vague places with knowledge from the Web. Manuscript, 2005.
- [41] A. Cohn and N. Gotts. The ‘egg-yolk’ representation of regions with indeterminate boundaries. In P. Burrough and A. Frank, editors, *Geographic Objects with Indeterminate Boundaries*, volume II of *GISDATA*, pages 171–187. Taylor & Francis, London, 1996.
- [42] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.
- [43] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
- [44] M. Dakowicz and C. M. Gold. Extracting meaningful slopes from terrain contours. *Int. J. Comput. Geometry Appl.*, 13(4):339–357, 2003.
- [45] M. de Berg, H. Meijer, M. Overmars, and G. Wilfong. Computing the angularity tolerance. *Internat. J. Comput. Geom. Appl.*, 8:467, 1998.
- [46] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.
- [47] L. de Floriani, P. Magillo, and E. Puppo. Applications of computational geometry to geographic information systems. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 333–388. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
- [48] L. De Floriani and E. Puppo. A survey of constrained Delaunay triangulation algorithms for surface representaion. In G. G. Pieroni, editor, *Issues on Machine Vision*, pages 95–104. Springer-Verlag, New York, NY, 1989.
- [49] M. N. DeMers. *Fundamentals of Geographic Information Systems*. John Wiley & Sons, 3rd edition, 2002.
- [50] B. D. Dent. *Cartography*. WCB/McGraw-Hill, Boston, 5th edition, 1999.
- [51] M. Díaz and J. O’Rourke. Computing the center of area of a polygon. In *Proc. 1st Workshop Algorithms Data Struct.*, volume 382 of *Lecture Notes Comput. Sci.*, pages 171–182. Springer-Verlag, 1989.

- [52] M. Díaz and J. O'Rourke. Ham-sandwich sectioning of polygons. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 282–286, 1990.
- [53] M. Díaz and J. O'Rourke. Algorithms for computing the center of area of a convex polygon. *Visual Comput.*, 10:432–442, 1994.
- [54] J. S. Doerschler and H. Freeman. A rule-based system for dense-map name placement. *Commun. ACM*, 35(1):68–79, 1992.
- [55] J. A. Dougenik, N. R. Chrisman, and D. R. Niemeyer. An algorithm to construct continuous area cartograms. *Professional Geographer*, 37:75–81, 1985.
- [56] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *STOC '03: Proc. 35th Annu. ACM Symp. Th. Comp.*, pages 473–482, 2003.
- [57] P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recogn. Lett.*, 14:715–718, 1993.
- [58] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.
- [59] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Trans. Inform. Theory*, IT-29:551–559, 1983.
- [60] M. Egenhofer and J. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. Technical report, Department of Surveying Engineering, University of Maine, 1990.
- [61] I. S. Evans. The morphometry of specific landforms. *International Geomorphology*, Part II:105–124, 1986.
- [62] B. Falcidieno and M. Spagnuolo. A new method for the characterization of topographic surfaces. *Int. J. Geo. Inf. Sys.*, 5(4):397–412, 1991.
- [63] G. Ferrari. Boundaries, concepts, language. In P. Burrough and A. Frank, editors, *Geographic Objects with Indeterminate Boundaries*, volume II of *GISDATA*, pages 99–108. Taylor & Francis, London, 1996.
- [64] U. Finke and K. Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 119–126, 1995.
- [65] P. Fischer, J. Wood, and T. Cheng. Where is Helvellyn? Fuzziness of multi-scale landscape morphometry. In *Transactions of the Institute of British Geographers*, 29(1), pages 106–128, 2004.
- [66] P. Fisher. Models of uncertainty in spatial data. In P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind, editors, *Geographical Information Systems*. John Wiley & Sons, 1999.
- [67] C. C. Fonte and W. A. Lodwick. Area, perimeter and shape of fuzzy geographical entities. In P. Fisher, editor, *Developments in Spatial Data Handling (11th International Symposium on Spatial Data Handling)*, pages 315–326. Springer, Berlin, 2004.
- [68] C. C. Fonte and W. A. Lodwick. Modelling the fuzzy spatial extent of geographical entities. In F. E. Petry, V. B. Robinson, and M. A. Cobb, editors, *Fuzzy Modeling with Spatial Information for Geographic Problems*, pages 121–142. 2005.
- [69] S. Fortune. Voronoi diagrams and Delaunay triangulations. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 225–265. World Scientific, Singapore, 2nd edition, 1995.
- [70] A. Frank. Qualitative spatial reasoning about distances and directions in geographic space. *Journal of Visual Languages and Computing*, 3:343–371, 1992.

- [71] A. Frank. Qualitative spatial reasoning: Cardinal directions as an example. *IJGIS*, 10(3):269–290, 1996.
- [72] G. Fu, C. Jones, and A. Abdelmoty. Building a geographical ontology for intelligent spatial search on the web. In *Proc. IASTED Int. Conf. Databases Appl.*, 2005.
- [73] G. D. Garson and R. S. Biggs. *Analytic Mapping and Geographic Databases*. Sage Publications, Newbury Park, 1992.
- [74] M. T. Gastner, C. R. Shalizi, and M. E. J. Newman. Maps and cartograms of the 2004 us presidential election results, <http://www-personal.umich.edu/mejn/election/>.
- [75] L. Gewali, A. Meng, J. S. B. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *ORSA J. Comput.*, 2(3):253–272, 1990.
- [76] J. Goldstein, M. Kantrowitz, V. O. Mittal, and J. G. Carbonell. Summarizing text documents: Sentence selection and evaluation metrics. In *Research and Development in Information Retrieval*, pages 121–128, 1999.
- [77] J. Goldstein, V. O. Mittal, J. G. Carbonell, and J. P. Callan. Creating and evaluating multi-document sentence extract summaries. In *Conference on Information and Knowledge Management*, pages 165–172, 2000.
- [78] M. Goodchild, K. Kemp, and T. Poiker. *NCGIA Core Curriculum*. National Center for Geographic Information Analysis, University of California, Santa Barbara, USA, 1989.
- [79] J. Goodman and J. O’Rourke, editors. *Handbook of Discrete and Computational Geometry*. Chapman & Hall/CRC, 2004.
- [80] M. T. Goodrich and J. Snoeyink. Stabbing parallel segments with a convex polygon. *Comput. Vision Graph. Image Process.*, 49:152–170, 1990.
- [81] H. W. Guesgen. Fuzzy reasoning about geographic regions. In F. E. Petry, V. B. Robinson, and M. A. Cobb, editors, *Fuzzy Modeling with Spatial Information for Geographic Problems*. Springer, 2004.
- [82] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.
- [83] L. J. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 9:534–560, 1993.
- [84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf. Principles Database Systems*, pages 47–57, 1984.
- [85] D. Harman. Overview of the TREC 2002 novelty track. In *NISI Special Publication 500-251: Proc. 11th Text Retrieval Conference (TREC 2002)*, 2002.
- [86] P. S. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Report CMU-CS-95-181, Carnegie Mellon University, 1995.
- [87] F. Heinzle and M. Sester. Derivation of implicit information from spatial data sets with data mining. In *XXth Congress of the International Society for Photogrammetry and Remote Sensing (ISPRS)*, volume 35, 2004.
- [88] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.
- [89] S. Hert. Connected area partitioning. In *Abstracts 17th European Workshop Comput. Geom.*, pages 35–38. Freie Universität Berlin, 2001.

- [90] G. Heuvelink. Propagation of error in spatial modelling with GIS. In P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind, editors, *Geographical Information Systems*, pages 207–217. John Wiley & Sons, 1999.
- [91] R. R. Hoffman. What is a hill? Computing the meanings of topographic and physiographic terms. In U. Schmitz, R. Schütz, and A. Kunz, editors, *Linguistic Approaches to Artificial Intelligence*, pages 97–128. Lang, Frankfurt/Main, 1990.
- [92] K. Hoffmann, K. Mehlhorn, P. Rosenstiehl, and R. E. Tarjan. Sorting Jordan sequences in linear time using level-linked search trees. *Inform. Control*, 68:170–184, 1986.
- [93] Y. Inoue, R. Lee, H. Takakura, and Y. Kambayashi. Web locality based ranking utilizing location names and link structure. In *WISEW '02: Proceedings of the Third International Conference on Web Information Systems Engineering (Workshops)*, pages 56–63, 2002.
- [94] C. Jones, A. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The SPIRIT spatial search engine: Architecture, ontologies and spatial indexing. In *Geographic Information Science: Third International Conference, GIScience*, pages 125–139, 2004.
- [95] C. Jones, R. Purves, A. Ruas, M. Sanderson, M. Sester, M. van Kreveld, and R. Weibel. Spatial information retrieval and geographical ontologies – an overview of the SPIRIT project. In *Proc. 25th Annu. Int. Conf. on Research and Development in Information Retrieval (SIGIR 2002)*, pages 387–388, 2002.
- [96] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.
- [97] A. A. Khanban and A. Edalat. Computing Delaunay triangulation with imprecise input data. In *Proc. 15th Can. Conf. Comp. Geom.*, pages 94–97, 2003.
- [98] M. Kraak and F. Ormeling. *Cartography*. Longman, 1996.
- [99] G. Langran and T. Poiker. Integration of name selection and name placement. In *Proc. 2nd Int. Symp. on Spatial Data Handling*, pages 50–64, 1986.
- [100] R. Larson and P. Frontiera. Spatial ranking methods for geographic information retrieval (GIR) in digital libraries. In *Europ. Conf. on Digital Libraries*, 2004.
- [101] LEDA <http://www.mpi-sb.mpg.de/leda/leda.html>.
- [102] Y. Leung. On the imprecision of boundaries. *Geographical Analysis*, 19(2):125–151, 1987.
- [103] Y. Liu and J. Snoeyink. Flooding triangulated terrain. In *Proc. 11th Int. Symp. on Spatial Data Handling*, pages 137–148, 2004.
- [104] C. P. Lo and A. K. W. Yeung. *Concepts and Techniques of Geographic Information Systems*. Prentice Hall, 2002.
- [105] P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographical Information Systems*. John Wiley & Sons, 1999.
- [106] P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, 2001.
- [107] Mapmart homepage, <http://www.mapmart.com/samples/samples.htm>.
- [108] D. M. Mark. Toward a theoretical framework for geographic entity types. In A. Frank and I. Campari, editors, *Spatial Information Theory: A Theoretical Basis for GIS*, number 716 in Lect. Notes in Computer Science, pages 270–283. 1993.

- [109] A. Markowetz, T. Brinkhoff, and B. Seeger. Exploiting the internet as a geospatial database. In P. Agouris and A. Croitoru, editors, *Next Generation Geospatial Information*, pages 5–14, 2005.
- [110] B. Martins, M. Chaves, and M. J. Silva. Assigning geographical scopes to web pages. In *Advances in Information Retrieval: 27th European Conference on IR Research, ECIR 2005*, volume 3408 of *LNCS*, pages 564–567, 2005.
- [111] B. Martins, M. J. Silva, and L. Andrade. Indexing and ranking in geo-IR systems. In *Proceedings of the 2005 workshop on Geographic information retrieval*, pages 31–34, 2005.
- [112] C. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. Technical report, Department of Applied Mathematics, SUNY Stony Brook, NY, 1994.
- [113] C. Mata and J. S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 264–273, 1997.
- [114] J. Matousek. *Lectures on Discrete Geometry*. Springer Verlag, Berlin, 2002.
- [115] C. T. Meadow, B. R. Boyce, and D. H. Kraft. *Text Information Retrieval Systems*. Academic Press, 2nd edition, 2000.
- [116] C. W. Mitchell. *Terrain Evaluation*. Longman, London, 2nd edition, 1991.
- [117] J. S. B. Mitchell. Approximation algorithms for geometric separation problems. Technical report, Department of Applied Mathematics, SUNY Stony Brook, NY, July 1993.
- [118] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k -MST, and related problems. *SIAM J. Comput.*, 28:1298–1309, 1999.
- [119] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38:18–73, 1991.
- [120] M. Molenaar. *An Introduction to the Theory of Spatial Object Modelling*. Taylor & Francis, 1998.
- [121] Y. Morimoto, M. Aono, M. Houle, and K. McCurley. Extracting spatial knowledge from the Web. In *Proc. IEEE Sympos. on Applications and the Internet (SAINT'03)*, pages 326–333, 2003.
- [122] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7:217–236, 1978.
- [123] J.-C. Müller, J.-P. Lagrange, and R. Weibel, editors. *GIS and generalization*, volume I of *GISDATA*. Taylor & Francis, London, London, 1995.
- [124] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [125] T. Nagai and N. Tokura. Tight error bounds of geometric problems on convex objects with imprecise coordinates. In *Proc. Jap. Conf. Disc. Comp. Geom.*, pages 252–263, 2000.
- [126] G. Nagy. Terrain visibility. *Computers & Graphics*, 18(6):763–773, 1994.
- [127] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2nd edition, 1998.
- [128] J. O'Rourke. Visibility. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 643–665. 2004.

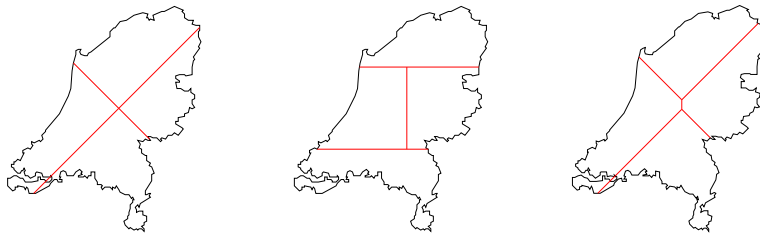
- [129] Y. Ostrovsky-Berman and L. Joskowicz. Uncertainty envelopes. In *Proceedings of 21st European Workshop on Computational Geometry*, pages 175–178, 2005.
- [130] D. O’Sullivan and D. J. Unwin. *Geographic Information Analysis*. John Wiley & Sons, 2003.
- [131] M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983.
- [132] L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graph. Models Image Process.*, 56(4):304–311, 1994.
- [133] D. J. Pennock, B. Zebarth, and E. de Jong. Landform classification and soil distribution in hummocky terrain, Saskatchewan, Canada. *Geoderma*, 40:297–315, 1987.
- [134] F. E. Petry, V. B. Robinson, and M. A. Cobb, editors. *Fuzzy Modeling with Spatial Information for Geographic Problems*. Springer, 2004.
- [135] D. Peuquet and Z. Ci-Xiang. An algorithm to determine the directional relationship between arbitrarily-shaped polygons in the plane. *Patt. Recogn.*, 20(1):65–74, 1987.
- [136] C. D. Piatko. *Geometric Bicriteria Optimal Path Problems*. Ph.D. thesis, Cornell University, 1993.
- [137] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, 1990.
- [138] D. Pullar. Spatial overlay with inexact numerical data. In *Proceedings of AutoCarto 10*, pages 313–329, 1991.
- [139] R. Purves. Personal communication, 2006.
- [140] R. S. Purves, P. Clough, and H. Joho. Identifying imprecise regions for geographic information retrieval using the web. In *Proceedings of GISRUK 2005, Glasgow*, pages 313–318, 2005.
- [141] S. Rana, editor. *Topological Data Structures for Surfaces*. Wiley, 2004.
- [142] E. Rauch, M. Bukatin, and K. Baker. A confidence-based framework for disambiguating geographic terms. In A. Kornai and B. Sundheim, editors, *HLT-NAACL 2003 Workshop: Analysis of Geographic References*, pages 50–54, Edmonton, Alberta, Canada, 2003. Association for Computational Linguistics.
- [143] I. Reinbacher, M. Benkert, M. van Kreveld, J. S. B. Mitchell, J. Snoeyink, and A. Wolff. Delineating boundaries for imprecise regions. *Algorithmica*, accepted for publication.
- [144] I. Reinbacher, M. van Kreveld, and M. Benkert. Scale dependent definitions of gradient and aspect and their computation. In *Proc. 12th Int. Symp. Spatial Data Handling*, to appear.
- [145] J.-R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry*. North-Holland, Amsterdam, 2000.
- [146] C. Schlieder, T. Vögele, and U. Visser. Qualitative spatial reasoning for information retrieval by gazetteers. In *Proc. COSIT 2001*, volume 2205 of *LNCS*, pages 336–351, 2001.
- [147] H. R. Schmidtke. The house is north of the river: Relative localization of extended objects. In *Proceedings of COSIT 2001*, volume 2205 of *LNCS*, pages 415–430, 2001.
- [148] B. Schneider. Geomorphologically sound reconstruction of digital terrain surfaces from contours. In *Proc. 8th Int. Symp. on Spatial Data Handling*, pages 657–667, 1998.

- [149] M. Schneider. Modelling spatial objects with undetermined boundaries using the realm/ROSE approach. In P. Burrough and A. Frank, editors, *Geographic Objects with Indeterminate Boundaries*, volume II of *GISDATA*, pages 141–152. Taylor & Francis, London, 1996.
- [150] C. Seara. *On Geometric Separability*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, 2002.
- [151] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 404–413, 1986.
- [152] J. Sellen, J. Choi, and C.-K. Yap. Precision-sensitive Euclidean shortest path in 3-space. *SIAM J. Comput.*, 29:1577–1595, 2000.
- [153] M. Sharir. On k -sets in arrangements of curves and surfaces. *Discrete Comput. Geom.*, 6:593–613, 1991.
- [154] M. Sharir and E. Welzl. On the number of crossing-free matchings, (cycles, and partitions). In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 860–869, 2006.
- [155] T. C. Shermer. A linear algorithm for bisecting a polygon. *Inform. Process. Lett.*, 41:135–140, 1992.
- [156] R. Sivan and H. Samet. Algorithms for constructing quadtree surface maps. In *Proc. 5th Int. Symp. Spatial Data Handling*, volume 1, pages 361–370, Aug. 1992.
- [157] J. G. Speight. Parametric description of land form. In G. A. Stewart, editor, *Land Evaluation papers of a CSIRO Symposium 1968*, pages 239–250, 1968.
- [158] SPIRIT <http://www.geo-spirit.org>.
- [159] T. Strijk. *Geometric Algorithms for Cartographic Label Placement*. PhD thesis, Universiteit Utrecht, 2001.
- [160] L. Talmy. How spoken language and signed language structure space differently. In *Proc. COSIT 2001*, number 2205 in Lect. Notes in Computer Science, pages 274–262, 2001.
- [161] N. Tate and P. Atkinson, editors. *Modelling Scale in Geographical Information Science*. John Wiley & Sons, Chichester, 2001.
- [162] G. Toussaint. Pattern recognition pages, <http://cgm.cs.mcgill.ca/~godfried/teaching/pr-web.html>.
- [163] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *Proc. 9th Int. Symp. Spatial and Temporal Databases*, pages 218–235, 2005.
- [164] M. van Kreveld. Geographic information systems. In J. E. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1293–1314. 2004.
- [165] M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, editors. *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *LNCS*. 1997.
- [166] M. van Kreveld and I. Reinbacher. Good NEWS: Partitioning a simple polygon by compass directions. *Int. J. Computational Geometry and Applications*, 14(4 & 5):233–259, 2004.
- [167] M. van Kreveld, I. Reinbacher, A. Arampatzis, and R. van Zwol. Multi-dimensional scattered ranking methods for geographic information retrieval. *Geoinformatica*, 9(1):61–84, 2005.
- [168] M. van Kreveld, E. Schramm, and A. Wolff. Algorithms for the placement of diagrams on maps. In *Proc. 12th Int. Symp. ACM GIS (GIS’04)*, pages 222–231, 2004.

-
- [169] M. van Kreveld, R. van Oostrum, and J. Snoeyink. Efficient settlement selection for interactive display. In *Proc. Auto-Carto 13: ACSM/ASPRS Annual Convention Technical Papers*, pages 287–296, 1997.
- [170] U. Visser, T. Vögele, and C. Schlieder. Spatio-terminological information retrieval using the BUSTER system. In *Proc. of the EnviroInfo*, pages 93–100, 2002.
- [171] C. Wang, X. Xie, L. Wang, Y. Lu, and W.-Y. Ma. Detecting geographic locations from web resources. In *GIR '05: Proceedings of the 2005 workshop on Geographic information retrieval*, pages 17–24. ACM Press, 2005.
- [172] R. Weibel and C. Jones, editors. *Special issue on Automated Map Generalization*, volume 2 of *GeoInformatica*. 1998.
- [173] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32:597–617, 1985.
- [174] S. Wise. *GIS Basics*. Taylor & Francis, 2002.
- [175] J. Wood. *The Geomorphological Characterisation of Digital Elevation Models*. PhD thesis, University of Leicester, 1996.
- [176] A. Woodruff and C. Plaunt. GIPSY: Geo-referenced information processing system. *Journal of the American Society for Information Science*, 45(9):645–655, 1994.
- [177] C.-K. Yap. Robust geometric computation. In J. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 927–952. 2004.
- [178] S. Yu, M. van Kreveld, and J. Snoeyink. Drainage queries in TINs: From local to global and back again. In *Proc. 7th Int. Symp. on Spatial Data Handling*, pages 13A.1 – 13A.14, 1996.
- [179] L. Zadeh. Fuzzy sets. *Information and control*, 8:335–353, 1965.
- [180] J. Zhang and M. Goodchild. *Uncertainty in Geographical Information*. Taylor & Francis, 2002.

SAMENVATTING

Ieder van ons is gewend aan benamingen van geografische gebieden zoals ‘het zuiden van Utrecht’, ‘de Hollandse Randstad’ of ‘de bergen van Oostenrijk’. Sommige van deze gebieden hebben precieze, vastomlijnde grenzen zoals de stad Utrecht of het land Oostenrijk. Andere daarentegen, zoals de Randstad en de bergen van Oostenrijk, hebben geen precieze grenzen en worden in het gebruik op een minder vastgelegde manier omschreven. Echter, soms is het noodzakelijk om dergelijke gebieden toch precies te omlijnen teneinde vragen te kunnen beantwoorden zoals ‘Hoeveel mensen onder 25 leven in de Randstad?’ of ‘Zijn er 4-sterren hotels aanwezig in het zuiden van Utrecht?’ Zonder een gespecificeerde, redelijke afbakening van deze gebieden is het onmogelijk om antwoord te geven op dergelijke vragen. Deze vragen zijn standaard in Geografische Informatiesystemen en -retrieval (GIS en GIR), maar dienen ook beantwoord te kunnen worden door zoekmachines op Internet. Het hoofdthema van dit proefschrift is, hoe onduidelijk gedefinieerde gebieden zoals hierboven vermeld af te bakenen met behulp van zuiver geometrisch-algoritmische methoden ten dienste van de zoekanalyse, gebruikt in GIS en GIR, alsmede voor visualisaties.



Figuur 6.1: drie verdelingen van Nederland in noord, oost, west en zuid.

In hoofdstuk 2 worden drie verschillende methoden beschreven om een willekeurig stuk grondgebied te verdelen in noord-, oost-, west- en zuidgedeeltes. In de eerste methode (Figuur 6.1, links) wordt het grondgebied op een dussdanige manier verdeeld door lijnen met een hellingshoek van $+1$ en -1 , dat de intersecties van

deze lijnen precies samenvallen met het zwaartepunt van het gebied. Het is hierbij gemakkelijk voor te stellen dat dit leidt tot vier deelgebieden welke sterk in grootte kunnen variëren. Bij de tweede methode (Figuur 6.1, midden) wordt het grondgebied verdeeld in gebieden van 25% van het totale grondgebied door gebruik te maken van verticale en horizontale lijnen. Bij de derde methode (Figuur 6.1, rechts) wordt het gebied in vier delen van gelijke grootte verdeeld en wel op zo'n manier dat de som der afstanden van de vier zwaartepunten van de deelgebieden maximaal is. Deze verdeling heeft een simpele vorm, ze lijkt op \succ of χ en is altijd uniek.

Hoofdstuk 3 presenteert methoden die gebieden zoals de Randstad kunnen afbakenen door gebruik te maken van een aantal steden die geïnclassificeerd worden als binnen of buiten het gebied. Om deze classificatie te bepalen wordt het Internet gebruikt als een ruimtelijk databestand. Documenten waarin tekstgedeeltes voorkomen zoals 'Utrecht, een stad in de Randstad' leiden tot de classificatie van Utrecht als binnen. Door deze manier van dataverwerking kan de classificatie binnen/buiten foutief toegekend worden. Ons doel is echter om redelijke begrenzingen aan te geven voor onduidelijk gedefinieerde gebieden. Deze grenzen moeten (de meeste) steden omvatten die als binnen geïnclassificeerd werden en tegelijkertijd (de meeste) steden niet omvatten die als buiten te boek staan. Volgens de *aanpassingsmethode* wordt een gebied doorberekend op het aantal binnen steden, en veranderen zijn grenzen zodanig dat de andere steden buiten het omliggende gebied komen te liggen. Volgens de *herkleuringsmethode* nemen we aan, dat bijvoorbeeld Amstelveen geïnclassificeerd dient te worden als binnen, als Amstelveen geïnclassificeerd wordt als buiten de Randstad maar omgeven wordt door een voldoende aantal steden die geïnclassificeerd worden als binnen. De uiteindelijke begrenzing van de Randstad is vervolgens berekend indien alle herclassificaties hebben plaatsgevonden.

In hoofdstuk 4 worden niet precies omschreven gebieden afgebakend in natuurlijke terreinen. Daarbij worden aan ieder punt binnen het terrein hellinggradiënt- en hellingliggingwaarden gegeven op basis waarvan kaarten ontwikkeld worden die gebieden aangeven met dezelfde gradiënt- en liggingwaarden. Gradiënt en ligging behoren tot de belangrijkste waarden in geomorfometrie, en gezamenlijk met hoogte, plan- en profielverloop is het mogelijk om landstreken te omschrijven als heuvels, valleien of berggebieden. We introduceren enkele methoden om schaalafhankelijke lokale gradiënten en lokale liggingen te berekenen, om van daaruit kaarten samen te stellen van lokale gradiënt en ligging. Volgens deze definities van gradiënt en ligging in combinatie met andere geomorfometrische waarden kunnen precieze grenzen verkregen worden voor onduidelijk gedefinieerde gebieden als 'de bergen van Oostenrijk'.

Zoals hierboven beschreven is het aangeven van precieze begrenzingen van onduidelijk gedefinieerde gebieden bruikbaar in GIS en GIR om zoekvragen te beantwoorden als: 'Aikido dojo's in het oosten van Amsterdam', vooropgesteld dat we dit onderzoek laten verrichten door een zoekmachine op Internet en we op basis hiervan verscheidene documenten hebben verkregen met de relevante tekstuele inhoud alsmede ruimtelijke locatie. Vervolgens is er de vraag deze documenten naar tekstuele en ruimtelijke relevantie te ordenen. In hoofdstuk 5 van dit proefschrift stellen we een aantal methoden voor om deze documenten die scoren op tekstuele

alsmede ruimtelijke relevantie te ordenen. Bij al deze methoden nemen we aan dat documenten met een gelijke score dezelfde inhoud hebben, en derhalve niet even hoog in de rangorde moeten staan. We presenteren een aantal methoden die documenten ordenen naar relevantie van de zoekvraag en de mate van ongelijkheid met documenten die al eerder een rangorde hebben gekregen.

De geometrisch-algoritmische methoden die in dit proefschrift werden gepresenteerd om geografische gebieden af te bakenen en documenten te ordenen, zijn een nieuwe manier om deze problemen in GIR te benaderen.

ACKNOWLEDGEMENTS

This thesis bears only a single author name, ignoring the fact that there are many people behind the scenes who helped making it possible. This is their curtain call.

I feel that coming to Utrecht and writing a PhD thesis here was one of the best decisions I have made, and I will always be grateful to Mark Overmars, for offering me the opportunity to work in his group.

Marc van Kreveld has been my supervisor, and he deserves an extra big ‘Hartstikke bedankt!’ for all his guidance, encouragement, insight, and patience during the years. He always made me feel welcome in his office, no matter how busy he was, or how silly my questions.

Also, all the people in Karlsruhe who made my stays there feel like coming home get a big ‘Dankeschön’, Alexander (Sascha) Wolff for inviting me, and Marc Benkert and Etienne Schramm for being such wonderful roommates.

I must not forget to mention the members of the SPIRIT consortium, who helped me see the bigger picture, and a considerable part of Europe.

The papers that comprise this thesis have been coauthored by: Tim Adelaar, Avi Arampatzis, Marc Benkert, Joseph Mitchell, Jack Snoeyink, Alexander Wolff, Roelof van Zwol, and, of course, Marc van Kreveld.

As I am not a good programmer, I admire everybody who believes that programming is fun and makes it seem so easy. I am indebted to Hui Ma, Subhod Vaid, and Markus Völker, as well as Tim Adelaar for implementing the algorithms presented in Chapters 3 and 4, respectively.

Piet van Oostrum was so kind to help me preparing the style for the layout of this thesis.

I would also like to thank everybody involved in reading (parts of) my thesis and improving it by making valuable comments: the reading committee, consisting of Mark de Berg, Sándor Fekete, Chris Jones, Joe Mitchell, and Peter van Oosterom; and the volunteers Natasha Bradley, Peter Lennartz, Roelof van Zwol, Geert-Jan Giezeman, and Wilko Vriesman.

Many of my evenings would have been spent lonely watching TV, if it were not for the people of the Ars Magica rolegame group. Every now and then, Availle has visions involving Lucius (Peter L.), Tristis-Isabel-Torquetus-Natira (Andres), Perplexa (Clara), Coogan (Patrick), Morgen (Peter V.), and Etienne (Karl Trygve).

Other friends who have made my life away from my mountains more pleasant include Mirela, Jaesook, Karina, Inma, Zinnia, Sergio, and Herman.

As mentioned a couple of times in this thesis, Aikido is my latest passion, thanks to my Aikido teachers Wilko Vriesman, Gerrit-Bartus Dielissen, and Frank Groenen. Wilko and Frank will do me the honor of serving as my paranimfen during my defense. I also want to thank my various other training partners, Jeroen, Luc, Tomek, Hans, Else, and Natasha, just to name a few, for being so gentle with me. Most of the times, at least.

And last but always first - my greatest gratitude goes to my grandmother, for all her endless love, support, and understanding. I would like to thank her for always accepting my choices in life, whether she liked them or not.

CURRICULUM VITAE

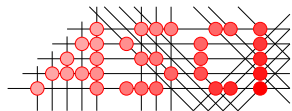
IRIS REINBACHER

28. 02. 1975 born in Knittelfeld, Austria
1989 - 1994 college of economics, Liezen
1994 - 2002 studies of Mathematics
 at Graz University of Technology, Austria
15. 02. 2002 ‘Diplom’ Mathematics
1996 - today studies of Physics
 at Graz University of Technology, Austria
2002 - 2006 PhD student (‘assistent in opleiding’)
 at Utrecht University, The Netherlands
May 2004 guest at Computer Science Department
 at Karlsruhe University, Germany

COLOPHON

This thesis was typeset by the author in L^AT_EX2_ε.

Cover illustration: Wereldkaart in de ‘Nieuwe Zee-Atlas’, Pieter van Alphen, 1660.
Courtesy of Utrecht University Library, FGW VIII.A.a.7.



Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.
ASCI dissertation series number 125.



This thesis has been supported by the EU-IST Project No. IST-2001-35047 (SPIRIT) and partially by a travelgrant of the Netherlands Organization for Scientific Research (Nwo).

Printed by Ridderprint offsetdrukkerij B.V.

© Iris Reinbacher, 2006

ISBN-10: 90-393-4281-4

ISBN-13: 978-90-393-4281-7