Delineating Boundaries for Imprecise Regions*

Iris Reinbacher† Marc Benkert‡ Marc van Kreveld† Alexander Wolff‡

Abstract

In geographic information retrieval users use names of geographic regions that do not have a well-defined boundary, like Southern France. We present two approaches to compute reasonable boundaries of such regions, based on evidence of points that are likely to lie inside or outside this region.

1 Introduction

Geographic information retrieval is concerned with information retrieval for spatially related data, including Web searching. Certain specialized search engines allow queries that ask for things (hotels, museums) in the (geographic) neighborhood of some named location. Another typical query could specify locations such as Central Mexico or the British Midlands. The latter is an example of a named region for which no exact boundaries exist. The extent of such a region is in a sense in the minds of the people.

Since geographic queries may ask for Web pages on castles in the British Midlands, it is still useful to have a reasonable boundary for this imprecise region. We would then be able to find Web pages for locations in the British Midlands that mention castles, even if they do not contain the words British Midlands. We would need to store a reasonable boundary for the British Midlands in the ontology (that stores all geographic information, including coordinates and geographic concepts) during preprocessing, and use query time for searching in the spatial part of the combined spatial and term index.

To determine a reasonable boundary for an imprecise region we can use the Web once again. The enormous amount of text on all Web pages can be used as a source of data; the idea of using the Web as a geo-spatial database appeared before in [9, 11]. A possible 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...". Such fragments give a location that is most likely in the British Midlands, while other cities like London or Cardiff, that do not appear in similar sentence fragments, give locations that are not in the British Midlands. Details of using trigger phrases to determine locations inside or outside a region to be delineated can be found in [2]. Obviously, the process is not too reliable and false positives and false negatives are likely to occur. In the reliable case, Alani et al. [1] give a Voronoi Diagram based method to delineate regions.

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 well. Possible criteria for a reasonable polygon are that it is simply-connected and has small perimeter (but as it must still contain most red points, it is fat and almost convex).

In computational geometry, red-blue separation algorithms exist of various sorts. Red-blue separation by a line is simply linear programming and takes $O(n)$ 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 [4]. Other fixed separation shapes like strips, wedges, and sectors can also be considered [3]. When polygons are the separator, then the most natural problem is the minimum perimeter polygon that separates the bichromatic point set. Euclidean travelling salesperson can be reduced to this problem, which makes it intractable [5]. Gudmundsson and Levcopoulos [7] give an $O(n \log n)$-time algorithm that finds a polygon whose perimeter is at most $O(\log n)$ times as long as the minimum perimeter one. Separation by minimum link shapes received attention as well (e.g., [10]).

In this paper we present two approaches to determine a reasonable polygon for a set of red and blue points. The first approach, described in Section 2, starts with a red polygon with blue points inside, and we try to change the shape of the polygon to get more blue points outside. The second approach, which changes the color of points to obtain a better shape of the polygon is presented in Section 3.

2 Adaptation Method

Let R be the set of red points, B the set of blue points, and let n be the total number of points. In the adap-

*This research is supported by the EU-IST Project No. IST-2001-35047 (SPIRIT) and by the grant WO 758/4-1 of the German Science Foundation (DFG).

†Institute of Information and Computing Sciences, Utrecht University, { iris, marc }@cs.uu.nl

‡Fakultät für Informatik, Universität Karlsruhe, http://i11www.ira.uka.de/people.

tation method, we start with some simply-connected polygon P and adapt it until all blue points inside P are no longer inside, or the shape has to be changed too dramatically.

For an appropriate value of α, we choose our initial polygon P to be the largest simply-connected component of the α-hull of the red points. This way, we can determine a suitable initial shape and possibly remove red outliers (the red points outside P) in the same step. Once we have computed P, the problem that remains is changing P so that the blue points are no longer inside. The resulting polygon should be contained in P and its perimeter should be as small as possible. In this section we discuss the problem of making sure that no blue point remains in the interior, although in practice it may be better for the final shape to allow some blue points to stay inside. They would be considered misclassified.

2.1 One blue point inside P

First, we assume that there is only one blue point b inside the polygon P. We want to determine a polygon P' so that b is not inside, P' is contained in P, all vertices of P are not outside P', and the perimeter of P' is minimal. It is clear that P' cannot have edges that intersect the exterior of P. We consider two cases: the special case where only point b is inside P, and the more general case where P contains b and a number of red points.

Lemma 1 *The optimal polygon P' is a possibly degenerate simple polygon (i.e. vertices and edges may be repeated) (i) with b on the boundary, and (ii) which includes all edges of P, with the exception of one edge.*

2.1.1 One blue and no red points inside P

Let P be a red polygon with only one blue point b inside. Let $e = \overline{p_1p_2}$ be the edge of P that does not appear in P'. The endpoints p_1 and p_2 are connected by a path π via b in P'. We denote the path that leads to the smallest perimeter of P' by $\pi_{\min}$; it consists of a shortest geodesic path between b and p_1, and between b and p_2. The optimal polygon P' has the same boundary as P, except that edge e is replaced by $\pi_{\min}$. In the optimal solution P', the edge e and the shortest path $\pi_{\min}$ have the following properties:

Lemma 2 *1. The path $\pi_{\min}$ is a simple path.*

2. A funnel π with root b and base e can only be minimal if e is partially visible from b.

For every two adjacent vertices p_i and p_{i+1} of the polygon, we compute the shortest paths connecting them to b. The algorithm of Guibas et al. [8] can find them in $O(n)$ time. For each possible base $e = \overline{p_ip_{i+1}}$ and corresponding funnel, we add the length of the two paths and subtract the length of the edge e to get the additional length of this choice.

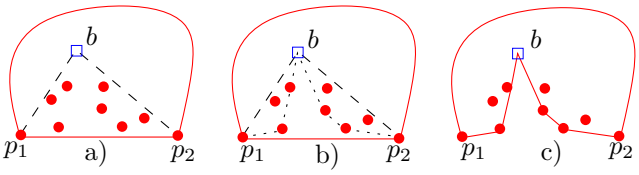

Figure 1: Removing one blue point from the red α-shape. a) The old polygon b) Computing the shortest path c) The new polygon

Theorem 3 *For a simple polygon P with n vertices and with a single point b inside, we can compute the shortest perimeter polygon P' that is contained in P, that contains all vertices of P, and that does not have b in its interior, in $O(n)$ time.*

2.1.2 One blue and several red points inside 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 these red points into account. We at first ignore all red points. We again compute all funnels from b to every edge e of P. We get a partitioning of P into $O(n)$ funnels with disjoint interiors. In every funnel F we do the following: If there are no red points inside F, we store the length of the funnel without its base edge e. Otherwise, we need to find a shortest path $\pi_{\min}$ from one endpoint of e to b and back to the other endpoint of e, such that all red points in R still lie inside the resulting polygon P'.

The shortest path $\pi_{\min}$ 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 forms a funnel F'. This funnel is not allowed to contain points of $R \cap F$. We need to consider all possible ways of making such funnels. Using dynamic convex hulls, we can obtain:

Theorem 4 *For a simple polygon P with n vertices, a single point b inside and set R of $O(n)$ red points inside, we can compute the shortest perimeter polygon P' that is contained in P, that contains all vertices of P and all red points of R, and that does not have b in its interior, in $O(n \log n)$ time.*

2.2 Several blue points inside P

2.2.1 Several blue and no red points inside P

For now, we assume that P is convex and has n vertices. With $m \ll n$ blue points inside the red polygon

P, we could try to adapt the first algorithm to compute a polygon that has all blue points no longer inside. However, as an iterative application of this algorithm may not lead to an optimal, smallest perimeter solution, we need to find a different approach. First we will prove some properties of an optimal solution.

Lemma 5 *Let P be a convex polygon with a set B with m blue points inside. In an optimal solution, let B be partitioned into k subsets. Then we have:*

1. *Each optimal path π of a subset B' consists of a part of the convex hull of B', and the two outer tangents of B' and some edge e of P, such that e and π form a convex polygon.*
2. *No two optimal paths intersect.*

It follows directly from this lemma that we need only consider partitionings of the blue points into subsets with disjoint convex hulls.

Lemma 6 *For a set of m points in the plane, there are $O(C^m)$ different partitionings into subsets with disjoint convex hulls, for some constant C.*

It is easy to see that every subset B_i chooses its optimal edge independently of all other sets. We give the following algorithm.

Let $\mathcal{P} = \{B_1, \ldots, B_k\}$ be a partitioning of the m blue points inside the polygon P into k subsets with disjoint convex hulls. For each B_i and every edge e of P we determine their outer tangents and compute the length of the shortest path. We store the optimal configuration with the shortest added length for $\mathcal{P}$. We do this for every possible partitioning $\mathcal{P}$ of the blue points inside P into subsets with disjoint convex hulls. Finally, we generate the polygon P' by replacing the appropriate edges of P with the optimal paths for the groups in the optimal partitioning.

Theorem 7 *For a convex polygon P with n vertices and m blue points inside, we can compute a minimum perimeter polygon P' that has no blue points in the interior and no vertices of P in the exterior, in $O(C^m \cdot n)$ time, for some constant C.*

In the case of a simple polygon P, we generate all possible $O(C^{m \log m})$ partitionings of the m blue points inside P. However, the properties of an optimal solution as well as the algorithm to find it, remain essentially the same. We can state:

Theorem 8 *For a simple polygon P with n vertices and m blue points inside, we can compute a minimum perimeter polygon P' that is contained in P, has no blue points in the interior and no vertices of P in the exterior, in $O(C^{m \log m} \cdot n)$ time, for some constant C.*

3 Recoloring methods

In this section we present the recoloring approach. 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. We will recolor a point if it is surrounded by points of the other color. We define for each point its green angle:

Definition 1 *Let $p \in P$ and let the edges of DT(P) be colored as above. Then the* green angle ϕ *of p is*

- $360°$*, if p is only incident to green edges,*
- *the maximum turning angle between two or more radially consecutive incident green edges,*
- $0°$*, if p has at most one radially consecutive incident green edge (or no incident green edges).*

We recolor points only if their green angle ϕ is at least some threshold value $\Phi \geq 180°$; a suitable value 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 or two points of M. For each triangle that contains two points of M, we connect them by a straight line segment. They define the boundary between the red and the blue region. Before we present specific recoloring schemes, we make the following basic observation.

Observation 1 *If we can recolor a blue point to be red, then we do not destroy this option if we first recolor other blue points to be red. If we can recolor a red point to be blue, then we do not destroy this option if we first recolor other red points to be blue.*

In the preferential recoloring scheme, we first recolor all blue points with green angle $\phi \geq \Phi$ red, and then all red points with green angle $\phi \geq \Phi$ blue. This scheme has linear running time, however, it is not fair and therefore not satisfactory.

3.1 The Angle-and-Perimeter Scheme

In the angle-and-perimeter scheme, we require that every recoloring decreases the perimeter of the separating polygon(s). Also, only points with green angle $\geq \Phi$ ($\Phi \geq 180°$) will be recolored. When there are several choices of recoloring a point, we select the one that has the largest green angle.

Theorem 9 *The number of recolorings in the angle-and-perimeter recoloring algorithm is at least $\Omega(n^2)$ and at most $2^n - 1$ in the worst case.*

The condition that recoloring is only allowed if it decreases the separation perimeter is needed, otherwise the separation perimeter may increase. Every recoloring implies the recoloring of all edges incident to the recolored point and updating its green angle as well as the green angle of all its neighbors and the perimeter of the polygons. We summarize:

Theorem 10 *The running time for the angle-and-perimeter recoloring algorithm is $O(n \cdot Z)$, where $Z \leq 2^n - 1$ denotes the maximum number of recolorings.*

3.2 The Angle-and-Degree Scheme

In the angle-and-degree scheme we require that a point p may only be recolored if the number of green edges incident to it goes down. This requirement is used in conjuction to the green angle $\geq \Phi$ condition for p. The degree condition gives a higher importance to the number of witnesses for the recoloring of a point. We need the following definition.

Definition 2 *A point p is a* good *neighbor of p' if p and p' are connected and have the same color, otherwise p is a* bad *neighbor of p'. Let $\delta(p)$, the δ-*value *of p, be the difference between the numbers of bad and good neighbors of p.*

We recolor a point p if its green angle ϕ is at least some threshold Φ and its δ-value is larger than some threshold $\delta_0 \geq 1$. This implies the recoloring of all edges incident to p and updating its green angle as well as the δ-value of p and all its neighbors. We can state the following two theorems:

Theorem 11 *The number of recolorings done in the angle-and-degree recoloring algorithm is $\Theta(n)$.*

Theorem 12 *The running time for the angle-and-degree recoloring algorithm is $O(\Delta \cdot Z)$, where $Z = \Theta(n)$ denotes the number of recolorings and Δ is the maximum degree in the Delaunay triangulation.*

4 Conclusions

This paper discussed algorithmic problems related to determining a reasonable boundary of a polygon, based on a set of points assumed to be inside (red), and a set of points assumed to be outside (blue). We presented two basic approaches. The first was to formulate the problem as a minimum perimeter polygon computation, based on an initial red polygon and one or more blue points that should not be inside. For the case of one point in the polygon we presented a linear time algorithm, and also $O(n \log n)$ time algorithm if there are also points that must stay inside. For the case of m points inside the polygon, we presented fixed-parameter tractable algorithms running in $O(C^m \cdot n)$ and $O(C^{m \log m} \cdot n)$ time, for convex and simple polygons, respectively.

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 criteria of recoloring. A remaining open problem is whether the angle-only version of this recoloring method terminates or not.

Another open problem is computing a minimum perimeter polygon for m blue points inside and n red points that must stay inside. Can a fixed-parameter tractable algorithm be given in this case as well?

Acknowledgements: Iris Reinbacher was also supported by a travel grant of the Netherlands Organization for Scientific Research (NWO). We thank Subodh Vaid, Hui Ma, and Markus Völker for implementing the algorithms.

References

[1] 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.

[2] A. Arampatzis, M. van Kreveld, I. Reinbacher, C. Jones, S. Vaid, P. Clough, H. Joho, and M. Sanderson. Web-based delineation of imprecise regions. manuscript, 2004.

[3] E. M. Arkin, F. Hurtado, J. S. B. Mitchell, C. Seara, and S. S. Skiena. Some separability problems in the plane. In *Abstracts EWCG 2000*, pages 51–54.

[4] T. M. Chan. Low-dimensional linear programming with violations. In *Proc. FOCS 2002*, pages 570–579.

[5] P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recogn. Lett.*, 14:715–718, 1993.

[6] 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.

[7] J. Gudmundsson and C. Levcopoulos. A fast approximation algorithm for TSP with neighborhoods and red-blue separation. *LNCS*, 1627:473–482, 1999.

[8] 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.

[9] A. Markowetz, T. Brinkhoff, and B. Seeger. Exploiting the internet as a geospatial database. In *Workshop on Next Generation Geospatial Information*, 2003.

[10] C. S. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems (extended abstract). In *Proc. SoCG 1995*, pages 360–369.

[11] Y. Morimoto, M. Aono, M. Houle, and K. McCurley. Extracting spatial knowledge from the web. In *Proc. SAINT'03*. 2003.