Delineating Boundaries for Imprecise Regions*

Iris Reinbacher[1], Marc Benkert[2], Marc van Kreveld[1], Joseph S. B. Mitchell[3], and Alexander Wolff[2]

[1] Institute of Information and Computing Sciences, Utrecht University, {iris,marc}@cs.uu.nl
[2] Dept. of Comp. Science, Karlsruhe University, i11www.ira.uka.de/algo/group
[3] Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, jsbm@ams.sunysb.edu

Abstract. In geographic information retrieval, queries often use names of geographic regions that do not have a well-defined boundary, such as "Southern France." We provide two classes of algorithms for the problem of computing reasonable boundaries of such regions, based on evidence of given data points that are deemed likely to lie either inside or outside the region. Our problem formulation leads to a number of problems related to red-blue point separation and minimum-perimeter polygons, many of which we solve algorithmically. We give experimental results from our implementation and a comparison of the two approaches.

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. These search engines cannot use the standard term matching on a large term index, because the user is not interested in the term "neighborhood" or "near". When a user asks for Web pages on museums near Utrecht, not only Web pages that contain the terms "museum" and "Utrecht" should be found, but also Web pages of museums in Amersfoort, a city about 20 kilometers from Utrecht. Geographic search engines require ontologies—geographic databases—to be able to answer such queries. The ontologies store all geographic information, including coordinates and geographic concepts. Geographic search engines also require a combined spatial and term index to retrieve the relevant Web pages efficiently.

Besides specifying neighborhoods, users of geographic search engines may also use containment and directional concepts in the query. Furthermore, the named location need not be a city name or region with well-specified, administrative boundaries. A typical query could ask for campgrounds in Northern Portugal, or specify locations such as Central Mexico, the Bible Belt, or the British Midlands.

* This research is supported by the EU-IST Project No. IST-2001-35047 (SPIRIT) and by grant WO 758/4-2 of the German Science Foundation (DFG).

The latter two are examples of named regions for which no exact boundaries exist. The extent of such a region is in a sense in the minds of the people. Every country has several such imprecise regions.

Since geographic queries may ask for Web pages about castles in the British Midlands, it is useful to have a reasonable boundary for this imprecise region. This enables us to find Web pages for locations in the British Midlands that mention castles, even if they do not contain the words British Midlands. We need to store a reasonable boundary for the British Midlands in the geographic ontology 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 has appeared before [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 sentence fragments give a location that is most likely in the British Midlands, while other cities like London or Cardiff, which 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 [1]. Obviously the process is not very reliable, and false positives and 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 rather a compact shape. Therefore, possible criteria for such a polygon are: The red points are inside and the blue points are outside the polygon, which is simply connected and has small perimeter. Other shape measures for polygons that are used in geography [12], e.g. the compactness ratio, can also be applied.

In computational geometry, various red-blue separation algorithms exist; see [14] for a survey. Red-blue separation by a line can be solved by two-dimensional linear programming in $O(n)$ time for n points. Red-blue separation by a line with k misclassified points takes $O((n + k^2) \log k)$ expected time [5]. Other fixed separation shapes, e.g. strips, wedges, and sectors, have also been considered [14]. For polygonal separators, a natural choice is the minimum-perimeter polygon that separates the bichromatic point set. This problem is NP-hard (by reduction from Euclidean traveling salesperson [6]); polynomial-time approximation schemes follow from the m-guillotine method of Mitchell [10] and from Arora's method [3]. Minimum-link separation has also received attention [2].

In this paper we present two 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. The first approach takes a red polygon with blue and red points inside, and 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 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. The implementation and test results on several data sets for both approaches are given in Section 4.

2 Adaptation method

In the adaptation method, we start with a polygon P and adapt it until all blue points inside P are no longer inside, or the shape has to be changed too dramatically. By choosing P initially as an α-shape (see [7] for a definition), 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.

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 1. *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.*

We call a simple polygon *degenerate*, if there are vertices that appear more than once on its boundary. We consider two versions of the problem: the special case where P contains only one point (namely b), and the general case where P contains b and a number of red points.

One blue and no red points inside P. Let b be the only point in the interior of P and let $e = \overline{v_1 v_2}$ be the edge of P that is not an edge of P^*. The boundary of the polygon P^* contains a path F^* that connects v_1 and v_2 via b. The path F^* consists of a shortest geodesic path between b and v_1, and between b and v_2. Note that these paths can contain red points other than v_1 and v_2 which are concave vertices of P, see Figure 1 (left). In the optimal solution P^*, the path F^* and the edge e have the following additional properties.

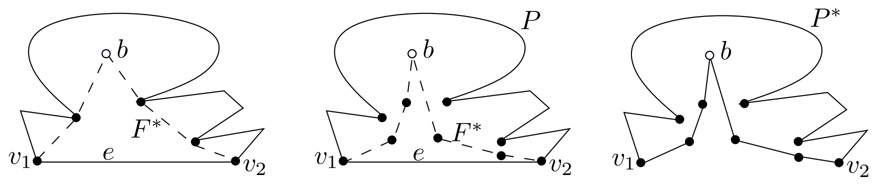

Fig. 1. Left: the path F^* if $R = \emptyset$. Middle: the case $R \neq \emptyset$. Right: P^* for $R \neq \emptyset$.

Lemma 2. *(i) The path F^* is a simple funnel. (ii) The base e of the funnel F^* is partially visible from b.*

We use the algorithm of Guibas et al. [8] 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. [8] 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 value of the perimeter change for this choice. We obtain the following result.

Theorem 1. *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 that has b on its boundary.*

One blue and several red points inside *P*. Let R be the set of the red points in the interior of P. 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 P. 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 just store the length of the funnel without its base edge. Otherwise, we need to find a shortest path $\pi_{\min}$ 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 $\pi_{\min}$ 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^*, see Figure 1 (middle). This funnel is not allowed to contain points of $R \cap F$. We need to consider all possible ways of making such funnels, which involves partitioning the points of $R \cap F$ into two subsets. Fortunately, the red points of $R \cap F$ can only appear as reflex points on the funnel F^*, and therefore we can use an order of 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 2, 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 $r_1, \ldots r_h$, counterclockwise.

Lemma 3. *There is an i such that the points $r_1, \ldots, r_i$ lie between the left chains of F and F^*, and $r_{i+1}, \ldots, r_h$ 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 [4]. 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 2. *For a simple polygon P with n vertices, a point b in P, and a set R of $O(n)$ red points in P, we can compute the minimum-perimeter polygon $P^* \subseteq P$ that contains all vertices of P and all red points of R, and that has b on its boundary, in $O(n \log n)$ time.*

2.2 Several blue points inside P

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

Several blue and no red points inside P. We sketch the dynamic-programming solution for the case that there are only blue points inside P. Details are given in the full version [13]. Let B be the set of blue points and let P^* be the optimal solution with no points of B inside, see Figure 2, left. The difference $P \setminus P^*$ defines a set of simple polygons called *pockets*. Each pocket contains one edge of P, which is called the *lid* of the pocket.

The structure of the solution is determined by a partitioning of the blue points into groups. Blue points are in the same group if they are on the boundary of the same pocket, or inside it. All blue points on the boundary of a pocket are convex for the pocket and concave for P^*.

The dynamic-programming solution is based on the idea that if $\overline{uv}$ is a diagonal of P^* between two red vertices of P, then the solutions in the two subpolygons

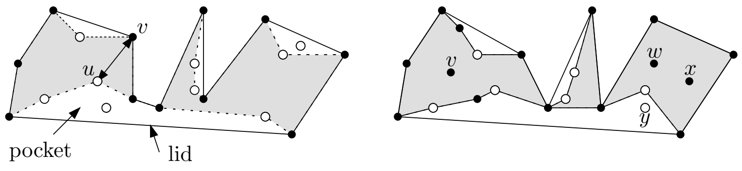

Fig. 2. Structure of an optimal subpolygon if only blue points are inside P (left), and if blue and red points are inside (right). Blue points are shown as white disks, red points as black disks.

to either side of the diagonal are independent. If $\overline{uv}$ is a diagonal of P^* between a red point u of P and a blue point v of B, *and we know the lid of the pocket that v is part of*, then the solutions in the two subpolygons are also independent. Similarly, if u and v are both blue, we need to know both lids of both pockets. Therefore, optimal solutions to subproblems are characterized by a function with four parameters. This function gives the length of the optimal subpolygon up to the diagonal $\overline{uv}$ (if applicable, with the specified pockets).

The recursion needed to set up dynamic programming takes the diagonal $\overline{uv}$ and possibly, the lid specifications, and tries all possibilities for a point w such that triangle $\triangle uvw$ is part of a triangulation that gives an optimal subpolygon up to $\overline{uv}$ for this set of parameters. If $\overline{uw}$ and $\overline{vw}$ are also diagonals, we need the smaller optimal solutions up to these two diagonals and add the lengths. To compute a function value, we must try $O(n)$ choices for w, and if w is a blue point, we also need to choose the lid of the pocket, giving another $O(n)$ choices. Hence, it takes $O(n^2)$ time to determine each of the $O(n^4)$ function values needed to compute the optimal polygon.

Theorem 3. *For a simple polygon P with n vertices and $O(n)$ blue points inside it, we can compute a minimum-perimeter polygon $P^* \subseteq P$ in $O(n^6)$ time. If there are $m = \Omega(n)$ blue points, the running time is $O(m^3 n^3)$.*

Several blue and red points inside *P*. We next give an algorithm that can handle k red and blue points inside a red polygon P with n vertices. The algorithm is fixed-parameter tractable: it takes $O(C^{k \log k} \cdot n)$ time, where C is some constant.

Let R and B be the sets of red and blue points inside P, respectively, and let $k = |R| + |B|$. The structure of the solution is determined by a partitioning of $R \cup B$ into groups, see Figure 2, right. One group contains points of $R \cup B$ that do not appear on the boundary of P^*. In Figure 2, right, 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 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 in 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 replace 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). We repeat this for every (ordered) group in the split permutation.

Theorem 4. *For a simple polygon P with n vertices, and k red and blue points inside it, we can compute a minimum-perimeter polygon $P^* \subseteq P$ in $O(C^{k \log k} \cdot n)$ time, for some constant C.*

3 Recoloring methods

In the adaptation method, 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 methods for recoloring the given points, i.e. assigning a new inside-outside classification. The starting point for our methods 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 1 *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 below $180°$, there is a simple example where there is no termination. So we assume in any case that $\Phi \geq 180°$; 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. We define the perimeter of the separation to be the total length of the boundary cycles and chains.

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. We also cannot create possibilities for coloring points red if we have only colored other points blue.*

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

Another scheme is the true *adversary scheme*, where an adversary may recolor any point with green angle $\phi \geq 180°$. For this scheme we do not have a termination proof, nor do we have an example where termination does not occur.

3.1 The Angle-and-Perimeter Scheme

In the angle-and-perimeter scheme, we require for a point to be recolored, in addition to its green angle being larger than $\Phi \geq 180°$, that this recoloring decreases the perimeter of the separating polygon(s). Since we are interested in a small perimeter polygon, this is a natural choice. When there are several choices of recoloring a point, we select the one that has the largest green angle.

Theorem 5. *The number of recolorings in the angle-and-perimeter recoloring scheme is at least $\Omega(n^2)$ and at most $2^n - 1$.*

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.

Theorem 6. *The running time for the angle-and-perimeter recoloring algorithm is $O(Z \cdot n \log n)$, where Z denotes the actual number of recolorings.*

3.2 The Angle-and-Degree Scheme

In the angle-and-degree scheme we use the same angle condition as before, complemented by requiring 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 p if its green angle ϕ is at least some threshold Φ and its δ-value is larger than some threshold $\delta_0 \geq 1$. We always choose the point with largest δ-value, and among these, the largest green angle. In every recoloring the number of green edges in $DT(P)$ decreases, so we get a linear number of recolorings.

Theorem 7. *The angle-and-degree recoloring algorithm requires* $O(n^2 \log n)$ *time.*

4 Experiments

In cooperation with our partners in the SPIRIT project [9] we got four data sets: Eastanglia (14, 57), Midlands (56, 52), Southeast (51, 49), and Wales (72, 54). The numbers in parentheses refer to the numbers of red and blue points in each data set, respectively. The red points were determined by Web searches using www.google.uk in Sept.'04 and trigger phrases such as "located in the Midlands" and then extracting the names of the corresponding towns and cities in the search results. The coordinates of the towns were looked up in the SPIRIT ontology. Finally the same ontology was queried with an axis-parallel rectangle 20% larger than the bounding box of the red points. The blue points were defined to be those points in the query result that were not red. The exact procedure is described in [1]. More experimental results are described in the full version [13].

We have implemented both the adaptation and the recoloring method, and we show and discuss a few screen shots. Figure 3 features the adaptation method for two different values of α. The corresponding radius-α disk can be found in the lower right corner of each subfigure. Regarding the recoloring method we give an example of the angle scheme and of the angle-and-degree scheme, see Figure 4. In each figure blue points are marked by white disks and red points by black disks. Due to the alpha shape that is used as initial region in the adaptation method, the area of the resulting region increases with increasing α, see Figure 3. We found that good values of α have to be determined by hand. For smaller values of α, the alpha shape is likely to consist of several components, which leads to strange results, see Figure 3 (left). Here, the largest component captures Wales quite well, but the other components seem to make not much 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 (right) Wales has an enormous extent. When the initial value of α was well-chosen, the results matched the region quite well for all data sets.

For the basic recoloring method we found the best results for values of Φ that were slightly larger than 180°, say in the range 185°–210°. Larger values of

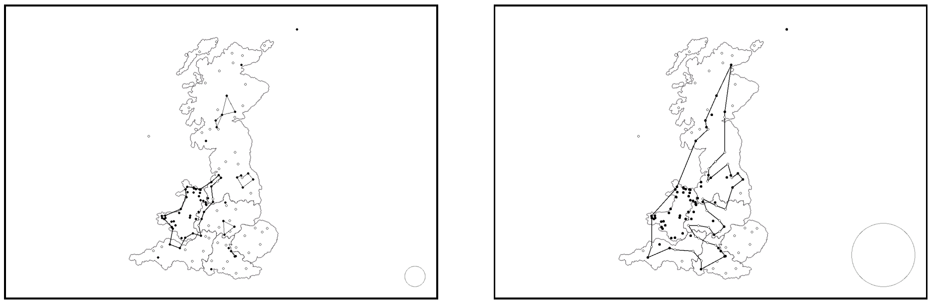

Fig. 3. Regions for *Wales* computed by the adaptation method. The α-values are shown as radius-α disks in the lower right corner.

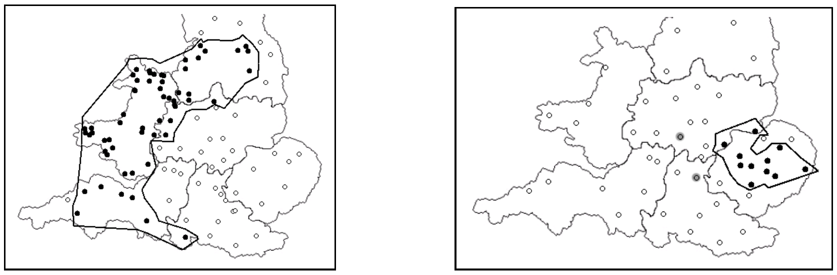

Fig. 4. Region for *Wales* computed by the angle scheme ($\Phi = 185°$, left) and for *Eastanglia* computed by the angle-and-degree scheme ($\Phi = 200°$ and $\delta_0 = 4$, right).

Φ severely restrict color changes. This often results in a long perimeter of the red region, which is not very natural. However, the results strongly depended on the quality of the input data. Figure 4 shows this effect: Although Wales is contained in the resulting region, the region is too large. Too many points were falsely classified positive. The quality of the Eastanglia data was better, thus the resulting region nearly matches the typical extent of Eastanglia.

For a small degree threshold, say $\delta_0 \leq 4$, the angle-and-degree scheme yields nearly the same results as the angle scheme, as points having a 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 reasonably precise, larger values of δ_0 can also yield good results, see Figure 4 (right) where only the two light-shaded points were recolored.

For random data Figure 5 shows how the number of recolorings (y-axis) depends on the angle threshold Φ (left, x-axis) for sets of size $n = 800$ and on the number of points (right, x-axis) for a fixed angle threshold of $\Phi = 210°$. In both graphs, a data point represents the average number of recolorings over 30 instances. The error bars show the minimum and maximum total number of recolorings that occurred among the 30 instances. For $n = 800$ the data sets were generated by picking 400 blue points and 100 red points uniformly distributed

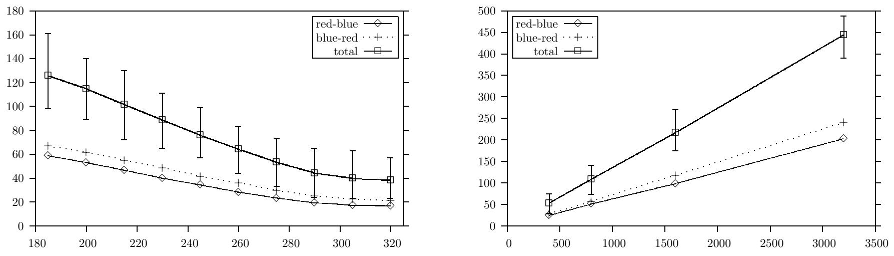

Fig. 5. Number of recolorings as a function of Φ (left) for random data sets of size $n = 800$ and as a function of the number of points (right) for random data sets and fixed angle threshold $\Phi = 210°$.

over the unit square. Then, another 300 red points were uniformly distributed over a radius-1/4 disk centered on the square. It is interesting to see 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.

Comparing the adaptation method and the recoloring scheme shows that the two schemes behave similarly on the inner part, the "core", of a point set. The main differences occur along the boundaries. The adaptation method may produce more fjord-like boundaries than the recoloring scheme. For example, compare Figure 3 (right) and Figure 4 (left).

The given real–world data appeared to be noisy, which influences the outcome, however, we think that the results are promising. Both improved data and more refined variations of our methods (e.g. using confidence values for inside–outside classification) should lead to better boundaries.

5 Conclusions

This paper discussed the problem of computing a reasonable boundary for an imprecise geographic region based on noisy data. 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 was to formulate 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).

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. An interesting open problem is whether the most general version of this recoloring method terminates or not. In tests we always had less than n recolorings. Furthermore, different schemes gave nearly the same result.

We also presented test results of our algorithms, based on real-world data, which 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.

Acknowledgements: Iris Reinbacher was partially supported by a travel grant of the Netherlands Organization for Scientific Research (NWO). J. Mitchell is partially supported by the NSF (CCR-0098172, ACI-0328930, CCF-0431030), NASA (NAG2-1620), Metron Aviation, and the US-Israel Binational Science Foundation (2000160).

We thank Subodh Vaid, Hui Ma, and Markus Völker for implementation, Paul Clough and Hideo Joho for providing the data, and Emo Welzl for useful ideas.

References

1. 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. Manuscript, 2005.
2. E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Minimum-link watchman tours. *Inform. Process. Lett.*, 86(4):203–207, May 2003.
3. S. Arora and K. Chang. Approximation schemes for degree-restricted MST and red-blue separation problem. *Algorithmica*, 40(3):189–210, 2004.
4. G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd IEEE Sympos. Found. Comput. Sci.*, pages 617–626, 2002.
5. T. M. Chan. Low-dimensional linear programming with violations. In *Proc. 43rd IEEE Symp. on Foundations of Comput. Sci. (FOCS'02)*, pages 570–579, 2002.
6. P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recogn. Lett.*, 14:715–718, 1993.
7. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, 1987.
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. 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.
10. 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.
11. 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.
12. D. O'Sullivan and D. J. Unwin. *Geographic Information Analysis.* John Wiley & Sons Ltd, 2003.
13. I. Reinbacher, M. Benkert, M. van Kreveld, J. S. Mitchell, and A. Wolff. Delineating boundaries for imprecise regions. Tech. Rep. UU–CS–2005–026, Utrecht University.
14. C. Seara. *On Geometric Separability.* PhD thesis, UPC Barcelona, 2002.