Smallest Intersecting Circle for a Set of Polygons

Peter Otfried Joachim Christian Marc Esther
René Michiel Antoine Alexander

31st August 2005

1 Introduction

Motivated by automated label placement of groups of islands, we consider the following problem: given a set $S = \{P_1, \ldots, P_m\}$ of m polygons, with n vertices in total, find the circle c of smallest radius such that c intersects P_i for $1 \leq i \leq m$.

2 The Furthest Polygon Voronoi Diagram

The center of the optimum circle lies on an edge or vertex of the furthest polygon Voronoi diagram (FPVD). The furthest polygon Voronoi diagram is the subdivision of the plane such that within each cell, one polygon of the set is the furthest polygon (where the distance of a point to a polygon is the shortest distance); see Figure 1 for an illustration.

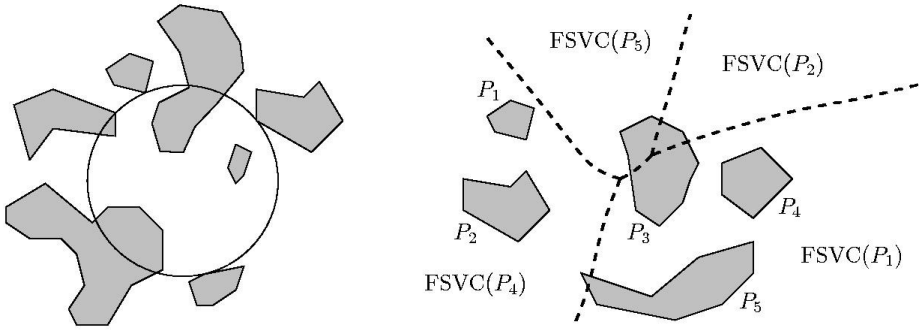

Figure 1: The furthest polygon Voronoi diagram.

The FPVD of a set of m polygons with n vertices in total is an abstract furthest site Voronoi diagram [6]. Algorithms to compute abstract furthest site Voronoi diagrams require $O(n \log n)$ time, assuming that sites and bisectors have constant complexity. In our situation, we cannot use these results directly, because bisectors can have complexity $O(n)$. However, the fact that the diagram has $O(m)$ cells does hold. By Euler's formula, this implies that the number of vertices of degree three is $O(m)$ as well. Although the bisectors can have linear complexity in n, we show that the number of vertices of degree two is $O(n)$.

Lemma 1 *The FPVD has $O(n)$ vertices of degree two.*

PROOF.Just a sketch.. An FPVD-vertex v of degree two lies on a bisector between two polygons, and is defined by two features of one of the polygons, and one of the features of the other. Imagine a circle c that touches two features f_1 and f_2 of the polygon P_i and whose center lies on $b(f_1, f_2)$. If we move the center of c over $b(f_1, f_2)$, we grow the radius r_c. At a certain position t_{start}, we start intersecting all polygons P_j, for $1 \leq j \leq m$, and at another position t_{end}, we (might) stop intersecting all polygons. In any case, the portion of $b(f_1, f_2)$ on which the circle c_t intersects all polygons is a single connected component. Exactly the two end points of this connected component yield a degree-two vertex in the FPVD. So, all bisectors in the Voronoi diagrams of individual polygons produce at most two vertices in the FPVD. Therefore, we have that the number of degree-two vertices of the FPVD is $\sum_{1 \leq i \leq m} O(n_i) = O(n)$. □

Since the FPVD has total complexity $O(n)$, we can construct it in an incremental way in $O(mn \log n)$ time. The algorithm is described below and is taken from [7]. We are given a set $\{P_1, P_2, \ldots, P_m\}$ of polygons, with $\{n_1, n_2, \ldots, n_m\}$ vertices, respectively, and we perform the following steps:

1. Compute a rectangle R that encloses all m polygons.
2. Compute the bisector of P_1 and P_2 and overlay this with the boundary of R; we denote this subdivision of R by S_2.
3. For $i = 3..m$ do:
   (a) Compute the (closest) Voronoi diagram of P_i and overlay it with R.
   (b) Traverse the two overlays of R, one with S_{i-1} and one with $\text{VD}(P_i)$ simultaneously; update the overlay with S_{i-1} such that it includes the parts where P_i is further than all polygons P_j, for $1 \leq j \leq i$.
   (c) For all points on the boundary where there is a P_j that is equally far as P_i, overlay the bisector of P_i and P_j with the cell of P_j in S_{i-1}. This gives us all boundaries between the cells of P_i and P_j in S_i. The boundary intersections give cells of of other polygons with which P_i also has a boundary, and these are treated in the same way. When we have found all bisectors for P_i starting at the boundary of R, we add the relevant bisector pieces, and we have constructed S_i from S_{i-1}.

When we add polygon P_i with n_i vertices, we construct its Voronoi diagram in $O(n_i \log n_i)$ time [1, 8]. By Lemma 1, simultaneous traversal of the boundary of R takes at most $O(n)$ time. The bisector with P_j can be computed in $O((n_i + n_j) \log(n_i + n_j))$ time [8, 1], and the intersection with the cell of P_j can be done in the same amount of time, asymptotically [2, 3]. Since $\sum_{1 \leq i \leq m} n_i = n$, the time needed to add P_i is $O(n \log n)$, from which the $O(mn \log n)$ time bound for the construction of S_m follows.

We have a linear number of candidates for the location of the center of the smallest intersecting circle, namely all vertices and edges of S_m. For each edge or vertex of the FPVD, we know which (two or three) polygons are furthest away, and thus we can compute in $O(1)$ time what the radius of the smallest intersecting circle with its center located at the candidate location is. Finally, we choose the candidate circle with the smallest radius.

3 Special Cases

3.1 Convex Polygons

In this section, we look at the special case in which all polygons in S are convex, and develop an $O(n \log n)$ time decision problem. We then apply parametric search [5] to obtain an $O(n \log^2 n)$ time algorithm for the optimization problem.

In the decision problem, we are given a radius r, and we want to determine whether there exists a circle c_r that intersects P_i for $1 \leq i \leq m$. The first step is to compute the Minkovski sum of all polygons with a disk of radius r. If the answer to the decision problem is yes, then these blown-up polygons have a nonempty intersection. Because all polygons are convex, their Minkovski sums are convex as well, and thus we can compute their common intersection in $O(n \log n)$ time.

3.2 Rectilinear Polygons

Now we look at the rectilinear version of the problem, i.e., all edges of the polygons are axis-parallel, and we use the L_∞ metric. Like in the previous section, we first develop an algorithm for the decision problem for a given radius r. Again, we blow up the polygons by the given radius, but of course now it is too expensive to compute the common intersection. Therefore, we compute the maximum depth in the arrangement of the blown-up polygons, which can be done in $O(n \log n)$ time by a sweepline algorithm where the status structure is a segment tree for stabbing counting queries. Finally, we apply parametric search [5] to obtain an $O(n \log^2 n)$ time algorithm for the optimization problem.

4 Approximation Algorithm

In this section, we present an $O(n\sqrt{n})$ time 2-approximation algorithm for the problem.

Consider we are given an optimum solution c_{opt} with radius r_{opt}. By definition, c_{opt} intersects all polygons in S, and thus all P_i's have at least one boundary point that lies on c_{opt}. Now take any polygon P_i and a point p that lies in $opt \cap \partial P_i$. If we place a circle c centered at p, with radius $2r_{opt}$, then we have that $c \supset c_{opt}$. Therefore, if we would find the optimum circle that intersects all P_j, $j \neq i$, and which has its center on the boundary of P_i, we find a circle with radius at most two times r_{opt} and thus we have a 2-approximation.

Since the above argument holds for any polygon in S, we choose the smallest one, say P_k. The number of edges of P_k is at most n/m. Finding the optimum over the boundary of P_k requires that we compute the optimum over each of the n/m edges. To compute the optimum on an edge e, we look at the distance functions of the other edges, restricted to the supporting line of e. For each polygon P_j, $j \neq i$, we compute the lower envelope of the (bivariate) distance functions of its edges on e, and then we compute the upper envelope of the m lower envelopes. This upper envelope has linear complexity, and can be computed in $O(n \log n)$ time. Therefore, we can find a 2-approximation in $O(\frac{n^2}{m} \log n)$ time. @Christian: Can you fill in the details of the last two claims?

Recall from Section 2 that we can compute the exact optimum solution in $O(mn \log n)$ time. If we run the exact and approximation algorithms in parallel, with running times $O(\frac{n^2}{m} \log n)$ and $O(mn \log n)$ time, and stop whenever the first algorithm terminates, we are guaranteed to have a 2-approximation. The minimum of the two running times is $O(n\sqrt{n})$.

5 3SUM-hardness

We show that the problem SMALLEST-INTERSECTING-CIRCLE of finding the smallest intersecting circle for a given set of m *non-disjoint* simple polygons is 3SUM-hard, by showing that deciding whether the m polygons have a point in common is 3SUM-hard (EMPTY-POLYGON-INTERSECTION). We give a reduction from the problem STRIPS-COVER-BOX in which we want to decide whether the union of a set of n strips covers a given axis-parallel rectangle R completely. This problem is known to be 3SUM-hard [4].

Let $S = \{S_1, \ldots, S_n\}$ be the set of strips, and B be an axis-parallel rectangle, that are the input of STRIPS-COVER-BOX; we transform this into an instance of EMPTY-POLYGON-INTERSECTION in the following way:

- We create a set $\mathcal{P}$ of polygons. For every strip S_i in S, we add a polygon P_i to $\mathcal{P}$. First, we compute the complement of S_i within B. If $B \backslash S_i$ is connected, P_i simply is this connected component, see Figure 2(a). If $B \backslash S_i$ is disconnected, we construct the simple polygon P_i by connecting the two components by a 'handle' outside of B, as illustrated in Figure 2(b). Finally, we add $P_{n+1} = B$ to $\mathcal{P}$. Clearly, the sum of the number of vertices of the polygons in $\mathcal{P}$ is linear in n. We take $\mathcal{P}$ as the input to our problem EMPTY-POLYGON-INTERSECTION.
- Now it is easy to verify that the rectangle B is covered by the strips in S, that is, $B \subseteq \bigcup_{i=1}^{n} S_i$, if and only the polygons in the set $\mathcal{P}$ have an empty intersection, that is, $\bigcap_{P_i \in \mathcal{P}} P_i = \emptyset$.

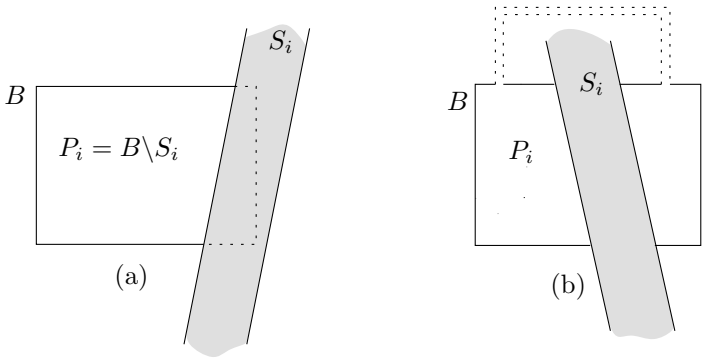

Figure 2: (a) The complement of S_i within B is connected. (b) The complement of S_i within B is disconnected.

The reduction trivially takes $O(n)$ time, and thus we conclude that SMALLEST-INTERSECTING-CIRCLE is 3SUM-hard.

References

[1] F. Aurenhammer and R. Klein. Voronoi diagrams. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, Sept. 1979.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[4] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

[5] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[6] K. Mehlhorn, S. Meiser, and R. Rasch. Furthest site abstract Voronoi diagrams. *Int. J. Comput. Geom. & Appl.*, 11:583–616, 2001.

[7] M. van Kreveld and T. Schlechter. Automated label placement for groups of islands. *The 22nd International Cartographic Conference*, A Coruña, Spain, 2005.

[8] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete Comput. Geom.*, 2:365–393, 1987.