Geodesic Disks and Clustering in a Simple Polygon

Magdalene G. Borgelt

Marc van Kreveld

Jun Luo

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2007-043

www.cs.uu.nl

ISSN: 0924-3275

Geodesic Disks and Clustering in a Simple Polygon *

Magdalene G. Borgelt
European Centre for Soft Computing
Mieres, Asturias, Spain
Email: magdalene@borgelt.net

Marc van Kreveld
Dept. of Computer Science,
Utrecht University
Email: marc@cs.uu.nl

Jun Luo
Dept. of Computer Science,
Utrecht University
Email: ljroger@cs.uu.nl

Abstract

Let P be a simple polygon of n vertices and let S be a set of N points lying in the interior of P. A *geodesic disk* $GD(p, r)$ with center p and radius r is the set of points in P that have a geodesic distance $\leq r$ from p (where the geodesic distance is the length of the shortest polygonal path connection that lies in P). In this paper we present an output sensitive algorithm for finding all N geodesic disks centered at the points of S, for a given value of r. Our algorithm runs in $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c and output size k. It is the basis of a cluster reporting algorithm where geodesic distances are used.

1 Introduction

Motivation. Clustering is the determination of relatively large subsets of a set that are in each other's proximity [15, 17, 18]. It is one of the most important and generally applicable techniques in data analysis. Often, the original data is a set of objects with various attribute values which are used as coordinates in a two- or higher-dimensional space. A cluster is a subset of the objects with similar attribute values, and a clustering of the objects is a partitioning into subsets so that objects in the same subset are similar, whereas objects in different subsets are not similar. To determine similarity, or distance, between two objects, a distance measure is needed, for which any L_p-metric (like the Euclidean metric) can be used.

In geographic situations, an important type of clustering is for sets of points in the real world [22, 24]. The coordinates of the points do not stand for attribute values but for a specific location. Depending on the origin of the points, the real world may also include other objects like obstacles that influence the distance between points, and therefore the clusters. Obstacles can be bodies of water that are not crossed by certain land animals, or large open areas that are not used by forest animals like squirrels.

Suppose that we are given a set S of N points in a geographic situation like an island, and we wish to find clusters. Clusters are large enough subsets of S that lie within a region of a maximum radius. The value m that represents the minimum size of a cluster, and the radius r that represents the maximum extent of a cluster region, are fixed and specified by domain experts (biologists). Distances need to be measured by paths that go over the island only, which means that the length of a geodesic shortest path determines the distance. The geometric problem that arises is: Given a set S of N points inside a simple polygon P with n vertices, determine all subsets of S of size

*This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503.

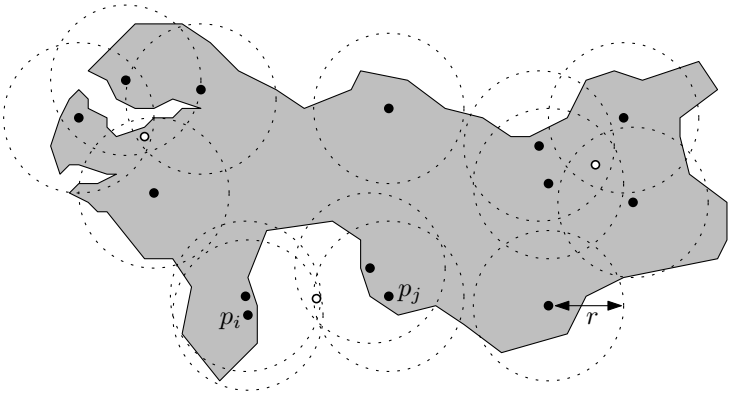

Figure 1: Polygonal island with points, each with a radius-r Euclidean circle shown. The three open markers are centers of circles that contain four points. Only the rightmost one gives a cluster center if geodesic distances within the polygon are used.

at least m for which a center point q exists that has geodesic distance at most r to all points in the subset (see Figure 1).

Related research. There is a large body of literature on clustering. Several papers discuss clustering in the presence of obstacles. COD-CLARANS [27] handles obstacles by replacing the Euclidean distance between two points by their geodesic distances. AUTOCLUST+ [10] uses the Delaunay triangulation for clustering, and uses the distance in the subgraph of the Delaunay triangulation where edges that intersect obstacles are removed. In DBCluC [26], two points may only be in the same cluster if the connecting line segment does not intersect any obstacle. For more on clustering with respect to obstacles, see [6] for a recent survey.

There are also several papers that compute clusters in a point set instead of a clustering of a point set. Laube et al. [19] compute all subsets of size at least m for which a circle of size at most r can be placed around the subset, using higher-order Voronoi diagrams [5]. Approximation algorithms were given in [12]. The problem of finding the cluster with m points and smallest radius has also been addressed extensively. Such problems have been given for squares and for circles, see for instance [8, 9, 16].

A recent approach to geographic clustering is given by Gudmundsson *et al.* [11]. The basic assumption is that in certain situations, there are regions where points can occur and regions where they cannot occur, and therefore a subset of points is a cluster if (i) the subset is large enough, (ii) the region has small enough radius, and (iii) the area where points can occur is small enough. The situation corresponds to nesting locations of sea birds on a group of islands, since nests cannot be in the water. Note that the Euclidean distance is still valid in this situation.

Problems concerning the geodesic distance with respect to a simple polygon have been studied mostly in the past. Among them are the computation of the geodesic center of a given simple polygon, and its geodesic diameter [4, 23]. Given a simple polygon P with n edges in the plane and a set of point sites in its interior or on its perimeter, Aronov [3] studied computing the Voronoi diagram of the set of sites with respect to geodesic distance.

Results of this paper. We solve our cluster reporting problem using geodesic distances inside a polygon by inverting the problem: we generate boundaries of radius-r geodesic disks centered at the points of S, and find points that lie inside at least m geodesic disks. A *geodesic disk* $GD(p, r)$ with radius r and center p is the set of points in P that have a geodesic distance $\leq r$ from p. It is easy to see that a geodesic disk is a shape that is bounded by circular arcs (not necessarily of the same radius) and pieces of the perimeter of P (see Figure 2). The main problem that arises is the computation of the N geodesic disks of the points of S inside polygon P. We present an *output-sensitive* algorithm for this problem that runs in $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time for some constant c and output size k. Note that $k = \Omega(N)$ and $O(n \cdot N)$. To appreciate this result, note that a direct approach to computing a geodesic disk would treat each point $p \in S$ separately

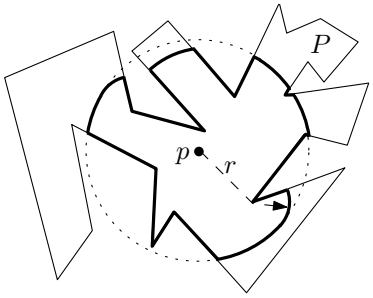

Figure 2: Boundary of a geodesic disk in a simple polygon. The dotted circle is a normal radius-r circle.

by computing the shortest path tree of p inside P, and then determining the circular arcs and boundary parts inside each funnel [13]. This procedure takes $O(n)$ time per geodesic disk, and $O(n \cdot N)$ time in total. In our application we expect k to be significantly smaller than $\Theta(n \cdot N)$, hence the objective to design an output-sensitive algorithm.

Overview of this paper. The remainder of this paper is organized as follows. Section 2 gives the general algorithm to compute a geodesic disk in an output-sensitive manner. It is based on two data structures, which are presented in Section 3. In Section 4 we show how to produce the geodesic disk boundaries from the output of the queries. The analysis of the algorithm is given in Section 5, and Section 6 shows how the geodesic clusters are determined.

2 A query algorithm to compute a geodesic disk boundary

Let $S = \{p_1, p_2, \cdots, p_N\}$ be a set of N points inside or on the perimeter of a simple polygon $P = \{v_1, v_2, \cdots v_n\}$ whose n vertices are given in clockwise order. For a fixed real number r, we present an algorithm that computes all geodesic disks $GD(p_i, r)$, for $i = 1, \ldots, N$.

The boundary of P is denoted as ∂P. For two vertices $v_i, v_j \in \partial P$, let $\partial P[v_i, v_j]$ be the part of boundary of P in clockwise order from v_i to v_j. A ray $\overrightarrow{pv_i}$ is a half line which starts at p and goes through v_i. For a point p in P that is visible from v_i, v_j, we have two rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$. We use $wedge(p, v_i, v_j)$ to denote the wedge which starts from ray $\overrightarrow{pv_i}$ and rotates around p clockwise until it reaches ray $\overrightarrow{pv_j}$. Before illustrating our algorithm to compute a geodesic disk $GD(p, r)$ in P, we assume there exist two query algorithms that use preprocessed data structures. We will discuss the details of the data structures and query algorithms in Section 3.

1. $CLSF(p, v_i, v_j)$: the inputs are a point p in P and two vertices v_i, v_j of P, where p can see v_i, v_j. The query reports the closest line segment or vertex from p among the line segments and vertices of $\partial P[v_i, v_j]$ that are visible from p. If the output is a vertex, then we can use either line segment which is on $\partial P[v_i, v_j]$ and is incident to that vertex as the output.

2. $FVSP(p, v_i)$: the inputs are a point p in P and a vertex v_i of P, the query reports the first vertex of the shortest path from p to v_i.

The inputs of the algorithm $\boldsymbol{GD}(p, v_i, v_j, r')$ are a point p inside P, two vertices v_i, v_j of P such that p is visible to v_i, v_j, and the radius r'. The output is the set of line segments of $\partial P[v_i, v_j]$ such that the shortest path distances from p to those line segments are $\leq r'$. To compute the geodesic disk itself, some straightforward extra work is needed; this is deferred to section 4. At the beginning, p is some point of S, $r' = r$, and $v_i = v_j$, where v_i is a vertex of P that is visible to p, which means the query range is the whole boundary of P. The algorithm runs as follows: using $CLSF(p, v_i, v_j)$ we find the closest line segment from p among all line segments of $\partial P[v_i, v_j]$. Let that closest line segment be $v_q v_{q+1}$. If the distance from p to $v_q v_{q+1}$ is larger than r, then we are done. Otherwise $v_q v_{q+1}$ is reported and there exists a closest point $a \in v_q v_{q+1}$.

Lemma 1 *The closest point a to p (by geodesic distance in P) of the line segment reported by $CLSF(p, v_i, v_j)$ is visible from p.*

Proof. Suppose the line segment $[v_q, v_{q+1}]$ is reported by $CLSF(p, v_i, v_j)$ and the closest point of $[v_q, v_{q+1}]$ to p is a. If a is not visible from p, then the first vertex v_b of the shortest path from p to a is inside or on the boundary of $wedge(p, v_i, v_j)$ and $v_b \in \partial P[v_i, v_j]$. So there is at least one line segment $[v_b, v_{b+1}]$ or $[v_{b-1}, v_b]$ which is $\in \partial P[v_i, v_j]$ that is closer to p than $[v_q, v_{q+1}]$. ∎

We assume without loss of generality that a is vertically above p (see Figure 3). The shortest path from p to v_q is a convex chain and its vertices are vertices of P. Let this convex chain be $pv_{l_1}v_{l_2}v_{l_3}\ldots v_{l_k}v_q$. We know that $pv_{l_1}v_{l_2}v_{l_3}\ldots\ v_{l_k}v_q$ is on the left side of the line though p, a. Similarly, we have the shortest path from p to v_{q+1} which is a convex chain $pv_{h_1}v_{h_2}v_{h_3}\ldots v_{h_{k'}}v_{q+1}$ on the right side of the line through p, a.

$\partial P[v_i, v_j]$ is partitioned into several parts: $\partial P[v_i, v_{l_1}]$, $\partial P[v_{l_1}, v_{l_2}]$, $\partial P[v_{l_2}, v_{l_3}]$, $\ldots$, $\partial P[v_{l_k}, v_q]$, $\partial P[v_{q+1}, v_{h_{k'}}]$, $\ldots, \partial P[v_{h_2}, v_{h_1}], \partial P[v_{h_1}, v_j]$; see Figure 3. They give rise to subproblems that we solve sequentially and recursively. Since the subproblems on the left are the same as those on the right, we discuss the situation on the left. First, we solve subproblems $\boldsymbol{GD}(p, v_i, v_{l_1}, r)$ and $\boldsymbol{GD}(p, v_{h_1}, v_j, r)$ recursively. For all other parts $\partial P[v_{l_1}, v_{l_2}], \partial P[v_{l_2}, v_{l_3}], \ldots, \partial P[v_{l_k}, v_q]$, as long as the shortest path distance from p to v_{l_t} $(1 \le t \le k)$ is $< r$, we solve the subproblem $\boldsymbol{GD}(v_{l_t}, v_{l_t+1}, v_{l_{t+1}}, r - (|pv_{l_1}| + \cdots + |v_{l_{t-1}}v_{l_t}|))$ recursively. We don't need to compute the whole shortest path from p to v_q. We only need to find pv_{l_1} by $FVSP(p, v_q)$. If the distance from p to v_{l_1} is $< r$, then we find $v_{l_1}v_{l_2}$ by $FVSP(v_{l_1}, v_q)$, and so on.

There is one special case. If the output of $CLSF(p, v_i, v_j)$ is v_i or v_j, then there are two cases: (assume that v_i is the one reported by $CLSF(p, v_i, v_j)$)

1. If v_{i+1} is inside $wedge(p, v_i, v_j)$, then the algorithm continues normally.

2. If v_{i+1} is outside $wedge(p, v_i, v_j)$, then we need to do a ray shooting query with $\overrightarrow{pv_i}$. Suppose $\overrightarrow{pv_i}$ first hits v_kv_{k+1}, which is a line segment of $\partial P[v_i, v_j]$. Then the shortest paths from p to v_k and v_{k+1} separate $\partial P[v_i, v_j]$ into several subparts and the problem becomes several subproblems. We can solve those subproblems sequentially and recursively as above.

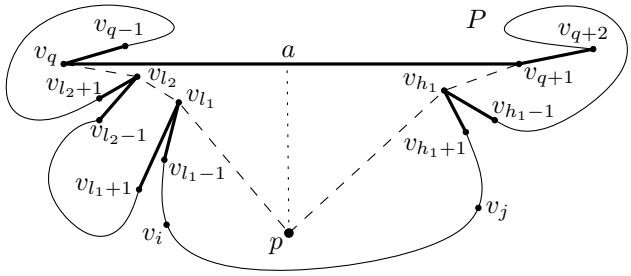

Figure 3: Illustration of the algorithm.

Lemma 2 *The algorithm $\boldsymbol{GD}(p, v_i, v_j, r')$ reports all line segments of $\partial P[v_i, v_j]$ that have geodesic distance at most r' from p at most once, and vertices at most twice.*

Proof. All recursive problems use a partition of the boundary into disjoint parts, except at the vertices $v_{l_1}v_{l_2}v_{l_3}...v_{l_k}$ and $v_{h_1}v_{h_2}v_{h_3}...v_{h_{k'}}v_{q+1}$ themselves. In all recursive problems of the form $\boldsymbol{GD}(v_{l_t}, v_{l_t+1}, v_{l_{t+1}}, r' - (|pv_{l_1}| + \cdots + |v_{l_{t-1}}v_{l_t}|))$, all line segments of $\partial P[v_{l_t+1}, v_{l_{t+1}}]$ will have $v_{l_1}, \ldots, v_{l_t}$ as the last vertices on the geodesic shortest path to p. Hence the recursive problem statements are valid. ∎

3 Data structures for *CLSF* and *FVSP*

In this section we describe the two data structures and query algorithms needed in the algorithm for geodesic disks. We also used a ray shooting data structure; this is standard with $O(\log n)$ query time in preprocessed simple polygons [7].

3.1 Closest boundary point in subpolygon queries

In this section we discuss how to find, for a given query point p and two vertices v_i and v_j of a simple polygon P, the point of the boundary of P between v_i and v_j that is closest to p. Vertices v_i and v_j can also be the answer to the query. We assume that pv_i and pv_j lie completely inside P (in other words: p sees v_i and v_j). To compute geodesic disks, we only need to solve the query problem with this restriction. The closest point q that is found must also be such that segment pq lies inside P.

Assume that some point q on the boundary of P between v_i and v_j is the point closest to p. Because pq lies inside P and p sees v_i and v_j, the angle of $\overrightarrow{pq}$ must be between the angles of $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$.

Without the restriction that pq must be inside P, we could have built a binary search tree T on $v_1, \ldots, v_n$, and construct a Voronoi diagram preprocessed for planar point location as associated structure with every internal node of T. A query would be answered by determining the search paths in T to v_i and v_j, and for all maximal subtrees strictly between these search paths, query the associated structure. But then a point may be found that does not see p inside the whole polygon P (see Figure 4).

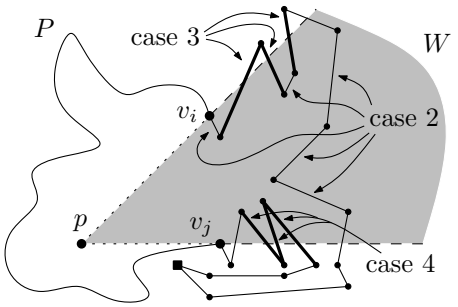

Figure 4: Edges of P of cases 2, 3, and 4 (cases 3 and 4 are shown thicker). Note that the vertex of $\partial P[v_i, v_j]$ closest to p, the square, is not in W.

The solution is to adapt the data structure so that we only query inside the wedge W bounded by the rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$. We have to take care to treat edges that lie partially inside this wedge correctly. We use five different data structures to handle all cases. In each case the main tree T is a binary search tree on $v_1, \ldots, v_n$, and the final associated structure is a planar point location structure on some Voronoi diagram. The first few associated structures (levels between the main tree and the point location structure) allow us to select the vertices or edges to which the case applies [1, 2]. The five cases are the following.

- Case 1: vertices inside W.
- Case 2: edges of which both endpoints lie inside W.
- Case 3: edges that intersect the ray $\overrightarrow{pv_i}$, have one endpoint inside W, and whose angle with $\overrightarrow{pv_i}$ is less than $\pi/2$, measured inside the wedge and closer to p (see Figure 4).
- Case 4: edges that intersect the ray $\overrightarrow{pv_j}$, have one endpoint inside W, and whose angle with $\overrightarrow{pv_j}$ is less than $\pi/2$, measured inside the wedge and closer to p (see Figure 4).

- Case 5: edges that intersect both rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$, and both angles are less than $\pi/2$.

For the first case we use a partition tree as the main tree. For the second case we use two levels of partition trees. We treat the third case in more detail, the fourth case is the same and the fifth case can be treated in the same manner.

For the third case, let T be the main tree with $v_1, \ldots, v_n$ in the leaves. An internal node μ corresponds to a subchain $\partial P[v_s, v_t]$. Let $E_\mu = \{v_s v_{s+1}, \ldots, v_{t-1} v_t\}$ be the edges in this subchain. To be able to select all edges that have one endpoint in the wedge W we take one endpoint of each edge and use a partition tree as the second level structure. To select the edges that intersect $\overrightarrow{pv_i}$ among these, we store the other endpoints in a partition tree as well as the third level structure, and the points dual to the supporting lines of the edges as the fourth level structure. In the fifth level structure we select further on the angle condition. This can be done using a binary search tree on the orientations of the edges. The sixth and last level structure is the point location structure on the Voronoi diagram of the edges. The fourth and fifth cases use similar multi-level trees.

For any query wedge, we can use the levels of the tree to select the edges for which each of the cases apply, and query in the Voronoi diagram to find the closest one. Each of the five cases may give an answer, and we can simply take the closest one as the actual closest vertex or edge. All structures use storage $O(n \log^c n)$ and query time $O(\sqrt{n} \log^c n)$ for some constant c. Combinations of cutting trees and partition trees allow us to get faster query times at the expense of storage and preprocessing [1, 2]. For any $n \leq m \leq n^2$, we can get storage and preprocessing time of $O(m \log^c n)$ and query time $O((n/\sqrt{m}) \log^c n)$.

3.2 First vertex of the shortest path queries

We partition P into $O(n)$ geodesic triangles in $O(n)$ time [7]. The three vertices of a geodesic triangle are vertices of P. The three edge chains are three shortest concave paths inside P. In [7] Chazelle *et al.* show that any line segment interior to P crosses at most $O(\log n)$ geodesic triangles. So the line segment pa in Figure 3 crosses at most $O(\log n)$ geodesic triangles. We observe two properties about the intersections of pa with those $O(\log n)$ geodesic triangle edge chains.

Lemma 3 *pa only intersects each geodesic triangle edge chain at most once.*

Proof. Since each geodesic triangle edge chain is a shortest path inside P and pa is a line segment inside P, a shortest path inside P intersects any line segment inside P at most once. Therefore pa only intersects each geodesic triangle edge chain at most once. ∎

Lemma 4 *pa intersects at most two edge chains of one geodesic triangle.*

Proof. Suppose pa intersects all three edge chains of a geodesic triangle. Since all three geodesic triangle edge chains are concave paths inside P, pa must intersect one of edge chains twice, which contradicts Lemma 3. ∎

Suppose pa crosses a geodesic triangle bcd. According to Lemmas 3 and 4, pa intersects one (if p is inside bcd) or two edge chains of bcd and pa only intersects those intersected edge chains once. For an intersected edge chain, suppose the intersected line segment is xx' and x is on the left side of pa and x' is on the right side of pa. Then x is a candidate for the first vertex of the shortest path from p to v_q, and x' is a candidate for first vertex of the shortest path from p to v_{q+1}. For a non-intersected edge chain, the vertex on a tangent line through p to this edge chain is also a candidate for the first vertex of the shortest path from p to v_q, provided the edge chain is on the left side of pa, and symmetrically, a non-intersected edge chain right of pa may provide a candidate for the first vertex of the shortest path from p to v_{q+1}. We explain those two cases by focusing on one edge chain b–c:

1. pa intersects the edge chain b–c, see Figure 5(a). Suppose the line segment of the edge chain from b to c intersecting pa is $b'b''$, and b' is on the same side of the line through p and a as v_q. Then b' is the only possible vertex from b–c that can be the first vertex of the shortest path from p to v_q. Given the edge chain b–c, we can find b' in $O(\log n)$ time.

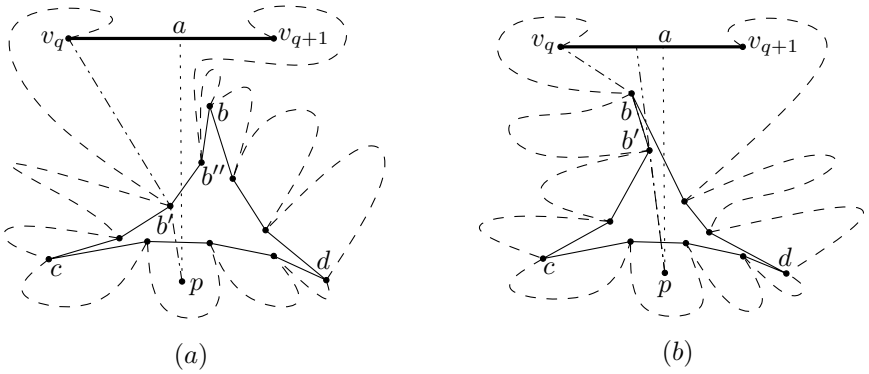

Figure 5: (a) pa intersects the edge chain b–c. b' is the candidate for $FVSP(p, v_q)$. (b) pa does not intersect the edge chain b–c. The tangent point b' is the candidate for $FVSP(p, v_q)$.

2. pa does not intersect the edge chain b–c and b–c is on the same side of the line through p and a as v_q, see Figure 5(b). The only candidate for the first vertex from p to v_q on the edge chain b–c is the vertex on a line through p that is tangent to the edge chain b–c. Given the edge chain b–c, we can find that tangent vertex in $O(\log n)$ time.

Any geodesic triangle intersecting pa gives at most three candidate vertices. To decide whether a candidate vertex y is the first vertex of the shortest path from p to v_q, we do a ray shooting query with $\overrightarrow{py}$. The first vertex of the shortest path from p to v_q is y if and only if py does not intersect any other line segment of ∂P and the first intersection point of $\overrightarrow{py}$ after y with ∂P is on $v_q v_{q+1}$. This can also be tested in $O(\log n)$ time. Since there are $O(\log n)$ geodesic triangles we need to check, the total running time of $FVSP(p, v_q)$ is $O(\log^2 n)$.

4 Computing the geodesic disk boundary

The algorithm as presented so far finds the set of edges of P that have some point at geodesic distance at most r from p, and a set of circular arcs that contain points at geodesic distance exactly r from p. To determine the boundary of the geodesic disk itself, we must combine this information into a simple polygon that has straight edges and circular arcs. Whenever an edge $v_q v_{q+1}$ is detected to be part of the boundary of the geodesic disk, we have the point p (or a vertex of P) which is the query center, and we have a wedge W in which the circular arc of radius r is valid. The following cases can be distinguished (we only treat the case of the left of the closest point a, like in Section 2):

- If $v_q v_{q+1}$ intersects the circular arc inside W, then we have found the (left) intersection point on $v_q v_{q+1}$ that gives a vertex of the geodesic disk.
- If $v_q v_{q+1}$ does not intersect the circular arc inside W, but W contains all of $v_q v_{q+1}$ left of point a, then all of $v_q a$ is part of the boundary of the geodesic disk.
- Otherwise, we repeat the above tests iteratively with v_{l_1}, v_{l_2}, ..., until we either find an intersection point left of a, or discover that all of $v_q a$ is part of the boundary of the geodesic disk.

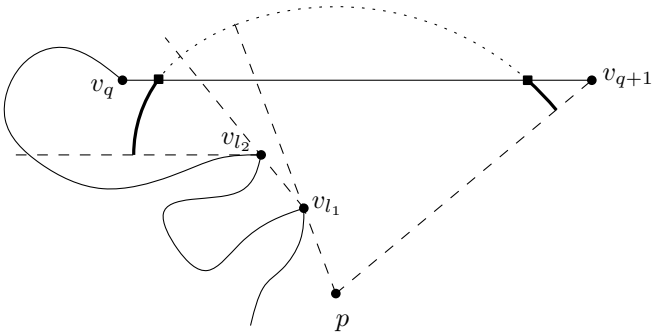

Figure 6: Finding the intersection points of the edges of P with the circular arcs that form the geodesic disk boundary.

In Figure 6, the edge $v_q v_{q+1}$ is found when p was the query center. The dashed lines show the wedge in which p is valid as an arc center, and we test whether the edge $v_q v_{q+1}$ intersects the circular arc inside the wedge. In the figure we only find the intersection point (square) when we test with v_{l_2}.

The tests can easily be integrated into the algorithm that finds the edges of P within geodesic distance r. Hence, the algorithm can also determine the boundary of the geodesic disk centered at p.

5 Complexity analysis

If the output size of N geodesic disks is $O(k)$, then the algorithm will perform $O(k)$ *FVSP* and *CLSF* queries. The preprocessing is $O(n)$ time, plus the time needed to build the multi-level trees of Subsection 3.1. We observed that a preprocessing time/query time trade-off exists: $O(m \log^c n)$ preprocessing time leads to $O((n/\sqrt{m}) \log^c n)$ query time. Assume we know k in advance. Then we can choose m to be such that the total query time and preprocessing time are of the same order: $k \cdot (n/\sqrt{m}) \log^c n = m \log^c n$, giving $m = (kn)^{\frac{2}{3}}$ (provided $n \leq m \leq n^2$).

Unfortunately, the output size k is not known, so we can not balance query time and preprocessing time easily. To overcome this problem, we will guess k, run the algorithm, and if it turns out that the guess was too low, we double our guess of k and start again: We build a data structure with slightly higher preprocessing time and slightly faster queries. Our initial guess is $k' = \max(n^{\frac{1}{3}}, 2N)$, since we know that $k \geq N$. Since we also know that $k \leq n \cdot N$, we will restart the algorithm at most $\log_2 n$ times. The running time is $O((n + (k'n)^{\frac{2}{3}} + k') \log^c n)$ in each round. Summation over the rounds yields $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c.

The adaptations made to find the boundaries of the geodesic disks themselves do not influence the asymptotic running time.

Theorem 1 *Given a simple polygon P with n vertices, a set S of N points inside P, and a positive real r, all N geodesic disks centered at the points of S can be computed in $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c, where k is the total boundary complexity of the geodesic disks.*

6 Geodesic clustering in a simple polygon

In this section, we will show how to solve the geodesic clustering problem: given a simple polygon P with n edges, a set S of N points inside P, a radius r, and a subset size m, find all geodesic disks with radius r which contain at least m points of S. We define a *geodesic cluster center* as a point p in P such that $GD(p, r)$ contains at least m points of S. We define two cluster centers to be *distinct* if they contain different subsets of S, otherwise they are *equivalent*.

We compute N geodesic disks with the algorithm described before. Suppose the complexity of the N geodesic disks is $O(k)$; recall that $k = \Omega(N)$ and $k = O(n \cdot N)$. We can compute the arrangement of the geodesic disks in $O(k \log k + K)$ time and $O(k + K)$ space, where K is the number of intersection points in the arrangement [14]. Since any two geodesic disk boundaries can have at most two proper intersections, $K = O(k + N^2)$. One can expect that for the cluster reporting application, K is considerably smaller than quadratic in N.

After building the arrangement, we determine for each cell by how many geodesic disks it is covered. For one cell we do this brute-force in $O(k)$ time. After that, we do an arrangement traversal (e.g., depth-first) to visit all cells. If we cross a cell boundary that adds a geodesic disk to the cover, we put a one higher value in the cell, and otherwise we put a one lower value in the cell. Hence, entering an adjacent cell and determining the value takes only $O(1)$ time. Therefore, the whole traversal takes time linear in the number of cells.

Reporting all distinct cluster centers comes down to identifying the cells with value at least m. All points inside such a cell are equivalent cluster centers.

Theorem 2 *Given a simple polygon P with n vertices, a set S of N points inside P, a positive real r, and a positive integer m, all distinct cluster centers can be reported in $O((n + (kn)^{\frac{2}{3}} + k) \log^c n + k \log k + K)$ time, for some constant c, where k is the total boundary complexity of the geodesic disks, and K is the number of intersection points in the arrangement of N geodesic disks.*

7 Conclusions

The main contribution of this paper is a new algorithm to determine N geodesic disks inside a simple polygon with n vertices. Instead of treating every point separately, we use preprocessed data structures, which made it possible to develop an output-sensitive algorithm for computing the geodesic disks. If the output size is k, then the running time is $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c. We used this algorithm to solve a cluster reporting problem: find all subsets of the points of at least some size that lie inside a geodesic disk with radius at most a specified value.

Although the geodesic disk algorithm is output-sensitive, the cluster reporting algorithm is not in the true sense. It is an open problem to determine clusters with a truely output-sensitive algorithm. Also, the output-sensitive algorithm for geodesic disks does not have the desired running time of the form $O(f(n) + k)$ or $O(f(n) + k \log n)$, where k is the output size and $f(n) = o(n \cdot N)$. It is also an open problem to find such an algorithm. An important extension of our research is to deal with polygons that have holes, or, a set of polygonal obstacles. It is unclear how to design any output-sensitive algorithm for this case.

References

[1] P.K. Agarwal. Range searching. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 36, pages 809–838. Chapman & Hall/CRC, Boca Raton, 2nd ed., 2004.

[2] P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, B. Chazelle, J.E. Goodman, and R. Pollack (eds), 1–56. AMS, Providence, RI, 1999.

[3] B. Aronov. On the Geodesic Voronoi Diagram of Point Sites in a Simple Polygon. *Algorithmica* 4:109–140, 1989.

[4] T. Asano, and G. Toussaint. Computing the Geodesic Center of a Simple Polygon. In *Perspectives in Computing: Discrete Algorithms and Complexity,* D.S. Johnson, A. Nozaki, T. Nishizeki, and H. Willis (eds.), pages 65–79, 1987.

[5] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surveys*, 23:345–405, 1991.

[6] P. Berkhin. A Survey of Clustering Data Mining Techniques. In *Grouping Multidimensional Data: Recent Advances in Clustering*, K. Jacob, N. Charles, T. Marc (eds.), 25–71, 2006.

[7] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray Shooting in Polygons Using Geodesic Triangulations. *Algorithmica* 12:54–68, 1994.

[8] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. *J. Algorithms*, 19:474–503, 1995.

[9] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discr. & Comput. Geom.*, 11:321–350, 1994.

[10] V. Estivill-Castro and I. Lee. Autoclust+: Automatic clustering of point-data sets in the presence of obstacles. *TSDM '00: Proc. of the First Int. Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining-Revised Papers*, pages 133–146, 2001.

[11] J. Gudmundsson, M. van Kreveld, and G. Narasimhan. Region-restricted clustering for geographic data mining. *Proc. 14th Europ. Sympos. Algorithms*, pages 399–410, 2006.

[12] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. In *GIS 2004: Proc. of the 12th ACM Sympos. on Advances in GIS*, pages 250–257, 2004.

[13] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear-time Algorithms for Visibility and Shortest Path Problems inside Triangulated Simple Polygons. *Algorithmica* 2:209–233, 1987.

[14] D. Halperin. Arrangements. In *J. E. Goodman and J. ORourke, editors, Handbook of Discrete and Computational Geometry,* chapter 24, pages 529–562. Chapman & Hall/CRC, Boca Raton, 2nd ed., 2004.

[15] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, San Diego, 2001.

[16] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41:147–157, 2005.

[17] J. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.

[18] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[19] P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In P.F. Fisher, editor, *Developments in Spatial Data Handling: proc. 11th Int. Sympos.*, pages 201–215, 2004.

[20] D. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, C-31:478–487, 1982.

[21] J. Matoušek. On enclosing k points by a circle. *Inform. Process. Lett.*, 53:217–221, 1995.

[22] H.J. Miller and J. Han. *Geographic Data Mining and Knowledge Discovery.* Taylor & Francis, London, 2001.

[23] R. Pollack, M. Sharir, and G. Rote. Computing the Geodesic Center of a Simple Polygon. *Discr. & Comput. Geometry* 4:611–626, 1989.

[24] D. O'Sullivan and D. Unwin. *Geographic Information Analysis*. John Wiley & Sons, Hoboken, NJ, 2003.

[25] G. Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle,* 3(2):9–42, 1989.

[26] O. R. Zaïane and C. Lee. Clustering spatial data in the presence of obstacles: a density-based approach. *Proc. of the 2002 Int. Symposium on Database Engineering & Applications*, pages 214–223, 2002.

[27] A.K.H. Tung, J. Hou, and J. Han. Spatial clustering in the presence of obstacles. *Proc. of the 17th Int. Conference on Data Engineering*, pages 359–367, 2001.