A Compact Piecewise-Linear Voronoi Diagram for Convex Sites in the Plane*

Michael McAllister David Kirkpatrick Jack Snoeyink
Department of Computer Science
University of British Columbia

Abstract

In the plane, the post-office problem, which asks for the closest site to a query site, and retraction motion planning, which asks for a one-dimensional retract of the free space of a robot, are both classically solved by computing a Voronoi diagram. When the sites are k disjoint convex sets, we give a compact representation of the Voronoi diagram, using $O(k)$ line segments, that is sufficient for logarithmic time post-office location queries and motion planning. If these sets are polygons with n total vertices given in standard representations, we compute this diagram optimally in $O(k \log n)$ deterministic time for the Euclidean metric and in $O(k \log n \log m)$ deterministic time for the convex distance function defined by a convex m-gon.

1 Introduction

One of the earliest successes of computational geometry is the $O(n \log n)$ time computation of the Voronoi diagram of n point sites in the plane, which is the partition of the plane into maximally connected regions that have the same set of closest sites [29, 33]. Aurenhammer [2] has surveyed the many applications and generalizations of the Voronoi diagram. In this paper we concentrate on two classical applications—the post-office problem and the "retraction" method for planning translational motion—when the sites are k disjoint convex polygons with a total of n vertices.

1.1 The post-office problem and retraction motion planning

The post-office problem [33] asks how to preprocess a set of k sites in the plane to be able to determine the closest site to a query point. When the sites are disjoint convex polygons with n total sides, the Voronoi diagram has k faces bounded by $O(n)$ segments of lines and parabolas [21, 24, 35]. Combined with a data structure for point location [14, 18, 31], it gives an $O(n)$ space data structure that answers queries in $O(\log n)$ time after $O(n \log n)$ preprocessing.

The retraction method for motion planning [1, 28, 32] uses the Voronoi diagram of the k sites to determine if there exists a motion of the centre of a disk from an initial position p to a final position q that does not cause the disk to intersect any site. Because edges of the Voronoi diagram are equidistant from their two closest sites, *maximum-clearance paths*, which maximize the minimum distance to an obstacle, can follow Voronoi edges. After $O(n \log n)$ preprocessing one can obtain

*This work has been supported by NSERC in the form of a Graduate Scholarship and two Research Grants.

an $O(n)$ space data structure that can be used to determine, in $O(\log n)$ time, if motion from p to q is possible and to construct a maximum-clearance path in time proportional to the path complexity [30]. Furthermore, by generalizing the distance measure from the Euclidean metric to a convex distance function, one can compute a retraction diagram for translating a convex object. (We elaborate in section 2.3.)

Although the Voronoi diagram is optimal for both the post-office problem and for retraction motion planning for point sites, it can be excessively elaborate when the sites are polygons. If we have k polygonal sites with a total of n vertices then the edges of the Voronoi diagram consist of $O(n)$ segments of lines and parabolas. The post-office problem then needs to search through parabolic and straight-line segments to find the Voronoi cell of the query point. Meanwhile, retraction motion planning on the Voronoi diagram dictates paths of line segments and parabolas to go between two sites when, intuitively, we can pass between two polygons along some separating line. We would like a simpler version of the Voronoi diagram that lets us solve the post-office problem and retraction motion planning for polygonal sites while avoiding the $O(n)$ complexity of the Voronoi diagram.

In this paper we describe a compact approximation of the Voronoi diagram when the k sites are disjoint convex polygons with n total vertices. Our compact diagram is composed of $O(k)$ line segments and is sufficient to solve the post-office in $O(\log n)$ time and retraction motion planning problems in $O(\log k)$ time. If the vertices of each site polygon are represented as ordered lists in arrays or balanced search trees, we can compute the diagram deterministically in $\Theta(k \log n)$ time by a sweep algorithm, as shown in section 3.

Our compact diagram can also represent the generalized Voronoi diagram defined by a convex distance function [9]. For k disjoint convex polygons with total complexity n and the distance function induced by a convex m-gon, the Voronoi diagram can have $\Theta(n + km)$ complexity. Our diagram with $O(k)$ line segments can be computed in $O(k \log n \log m)$ time and can answer post-office queries in $O(\log n + \log m)$ time and motion planning queries on $O(\log k)$ time.

This diagram has several advantages besides its efficient deterministic construction. First, given the compact diagram, the true Voronoi diagram can be computed in time proportional to its complexity ($\Theta(n)$ for the Euclidean and $\Theta(n + km)$ for convex distance functions). Second, for applications where knowing two candidates for the closest site (and not their distances) is sufficient, one can discard the original sites (and distance function) and store only the $O(k)$ segments of the compact diagram. Retraction motion planning is one such application. Third, because the compact diagram is entirely composed of line segments, computing, displaying, and traversing are easier operations than on the Voronoi diagram. This is an advantage even if the k sites are line segments.

In the remainder of this section we compare our diagram to related work on the definition and construction of (compact and generalized and abstract) Voronoi diagrams. For more detail, see Aurenhammer's survey [2]. Section 2 describes the diagram and its application to the post-office problem and to retraction motion planning. Section 3 gives a deterministic construction based on Fortune's sweep algorithm [15] as well as a randomized incremental construction with the same expected time.

1.2 Related work on compact diagrams

There has been considerable recent interest in simplified or compact representations of the Voronoi and other retraction diagrams. Canny and Donald [7] define a simplified Voronoi diagram in d dimensions for retraction motion planning that has a lower algebraic complexity than the Euclidean diagram. In the plane, they obtain a diagram with line segments, but the number of segments depends on the complexity of the obstacles—on n rather than k—and the dependence may be superlinear. Kao and Mount [17] consider the generalized Voronoi diagram, using a distance function defined by a convex polygon with m sides. Even though the diagram may have $\Theta(n+km)$ complexity, Kao and Mount show that a compact representation with space $O(m+n)$ can be computed in $O(n \log n \log^2 m)$ time such that post-office queries take $O(\log n + \log m)$ time. If used for motion planning, their approach would still generate a path of $\Theta(n + km)$ complexity.

Sifrony [34] considers the motion of a fixed m-gon in the plane and computes an $O(n)$-sized *skeletonized* retraction diagram for motion planning using approximately $O(n \log n \log^2 m)$ time. de Berg, Matoušek, and Schwarzkopf [11], in independent work, have generalized these results to higher dimensions and improved them to depend on k instead of n. In the plane, they compute an $O(k)$-size skeletonized diagram in $O(k \log^2(n+m))$ time for moving a fixed m-gon in the plane. Because these approaches depend on a fixed m-gon, they do not solve the post-office problem for convex distance functions.

1.3 The relation to the computation of abstract Voronoi diagrams

For those who are familiar with Klein's monograph [22], our compact diagram can be seen as an instance of an abstract Voronoi diagram. Abstract Voronoi diagrams are defined only in terms of *bisectors* of pairs of sites and are computed using primitives such as determining the ordering of two points along a bisector and the ordering of three bisectors that pass through a common point. Recent work [20] gives $O(\log n)$-time subroutines for these primitives under the Euclidean metric. In section 3.4, we obtain $O(\log n \log m)$ subroutines under a convex distance function defined by an m-gon.

There are three algorithmic paradigms that give optimal $\Theta(k \log k)$ algorithms for the Voronoi diagram of k point sites: divide and conquer [33], randomized incremental construction [16, 27], and sweepline [15]; the first two have been adapted to compute abstract Voronoi diagrams [22, 23, 26], but they do not directly give an optimal construction for our compact diagram. It is instructive to investigate why not.

A divide and conquer algorithm merges pairs of Voronoi diagrams in linear time. In the process, it computes as many as $\Theta(k \log k)$ points equidistant from three sites; in the abstract setting, each such point requires a subroutine call. For our compact representation of the Euclidean Voronoi diagram, this would result in an $\Theta(k \log k \log n)$-time algorithm. The randomized incremental constructions [23, 26] compute an expected $O(k)$ points equidistant from three sites, but evaluate an expected $\Theta(k \log k)$ "conflicts." In our case, a conflict involves a "spoke region," which is a hexagonal region defined by two sites, and a new site. The conflict occurs when at least one point of the spoke region is closer to the new site than to either of the two sites that define the spoke region. Thus, the direct implementation takes $\Theta(k \log k \log n)$ expected time. Section 3.3 improves the expected time by evaluating conflicts for the leftmost point of the new site and updating the diagram from this conflict using a constant number of subroutine calls for each Voronoi vertex created. Fortune's [15] sweep algorithm has not been adapted to the abstract setting. This is not a surprise when one considers that abstract bisectors need not be monotone or have other properties that permit a sweep. For convex distance functions, however, Fortune's sweep can be seen as the computation of a dynamic Voronoi diagram whose sites are the sweep line and the swept portions of objects. (His "parabolic front" is simply the boundary of the Voronoi cell of the sweep line.) The Voronoi cells have a certain "star-shaped" property that allows the sweep algorithm to use only $O(k)$ subroutine calls in total to handle events.

2 Definition of the compact diagram

We are able to define our compact diagram for any convex distance function. In the next section, we construct it by a general algorithm—only the subroutine for computing a point equidistant from three or more polygons depends on the distance function. This generality necessitates some care in the basic definitions to handle degenerate cases.

2.1 Geometric preliminaries

We begin by defining convex distance functions, spokes, bisectors, Voronoi cells, Voronoi vertices, and Voronoi edges.

Minkowski showed that any convex set M whose interior contains the origin defines a *convex distance function* $d_M(p,q)$. The distance from point p to q with respect to M is the amount that M must be scaled to include $q-p$; the distance function d_M has a natural extension to sets A and B.

$$\begin{aligned} d_M(p,q) &= \inf\{\lambda \geq 0 : q-p \in \lambda M\} \\ d_M(A,B) &= \inf\{d_M(a,b) : a \in A, b \in B\} \end{aligned}$$

In this paper, the sets M, A and B are always closed subsets of the Euclidean plane E^2 so the infimum operations could be replaced by minimum operations.

Distance function d_M need not induce a metric because it need not be symmetric: $d_M(p,q)$ need not equal $d_M(q,p)$ if M is not centrally symmetric. It does, however, satisfy the triangle inequality for points [8]: $d_M(p,q) + d_M(q,r) \geq d_M(p,r)$.

The points of the boundary of M are precisely those at unit distance from the origin. Choosing M to be the unit circle gives the Euclidean metric; choosing M to be the diamond defined by four unit vectors in the axial directions gives the L_1 or Manhattan metric. Thus, we can give a geometric interpretation of the distance from a point p to a set A. Let M_p^A denote the convex set M scaled by $d_M(p,A)$ and translated to p. (See figure 1.) That is, $M_p^A = d_M(p,A)M + p$.

Lemma 2.1 *If M and A are closed convex sets and $p \notin A$ then the boundaries of M_p^A and A intersect while their interiors are separated by a tangent line.*

Proof: Suppose that the interiors of A and M_p^A were not disjoint. Then we could find a point $a' \in \mathrm{int}(A) \cap \mathrm{int}(M_p^A)$. Then the distance $d_M(p,a') < d_M(p,A)$, contradicting the definition of $d_M(p,A)$.

On the other hand, there does exist an $a' \in A \cap M_p^A$ because M and A are closed—the boundaries $\partial(A)$ and $\partial(M_p^A)$ intersect.

Since the interiors of A and M_p^A are disjoint, they can be separated by a line ℓ. Line ℓ must pass through a', making it tangent to A and M_p^A. ■

Given a closed convex set $A \subset E^2$ and two points $p \in E^2$ and $a \in A$, we say that $\overline{pa}$ is a *finite spoke* and a is the *attachment point* if $d_M(p, A) = d_M(p, a)$. If $p \in A$ then the degenerate segment $\overline{pp}$ is a spoke. Geometrically, $\overline{pa}$ is a spoke with $p \notin A$ if M_p^A and A intersect at a as in figure 1. The pair p and A define a unique spoke except in the degenerate situation where A and M_p^A share a common line segment on their boundaries. We define $\texttt{spoke}(p, A)$ to be the unique Euclidean shortest spoke defined by p and A. We can also define *infinite spokes*, which are the infinite rays composed of all points p for which $\texttt{spoke}(p, A)$ has the same attachment point and direction.

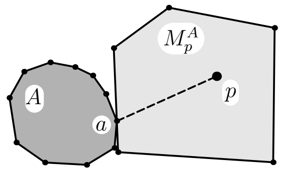

Figure 1: M_p^A and $\texttt{spoke}(p, A)$

We say that a set $X \in E^2$ is *star-shaped with respect to* A if $A \subseteq X$ and every spoke $\texttt{spoke}(p, A)$, with $p \in X$, is contained in X.

The *bisector* of two closed convex sets A and B under the distance function d_M is usually defined as $\{p : d_M(p, A) = d_M(p, B)\}$. The shaded region in figure 2 illustrates that the bisector is not always a curve under this definition. When a boundary segment of M_p^A is an outer common tangent of A and B, then all points in the wedge defined by rays from p directly away from the attachment points of $\texttt{spoke}(p, A)$ and $\texttt{spoke}(p, B)$ are equidistant from A and B with respect to $d_M()$.

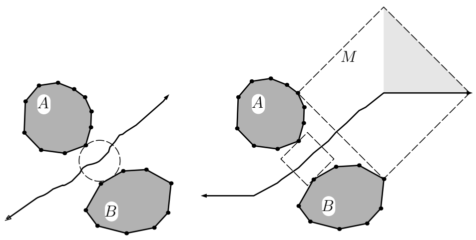

Figure 2: Euclidean and Manhattan bisectors

We therefore base the definition of a bisector on an oriented version of the set M that induces the convex distance function. Orient the boundary of M counterclockwise so that each line segment on the boundary of M becomes a directed edge. The head of an edge is associated with the edge itself while the tail of an edge does not belong to the edge. Consequently, every point on the boundary of M belongs to exactly one directed edge. Under this orientation, we define the AB-bisector to be the set of points where $d_M(p, A) = d_M(p, B)$ and where $\texttt{spoke}(p, A)$ and $\texttt{spoke}(p, B)$ cross the boundary of M_p^A along different directed edges. This definition is depicted in figure 2. Note that this definition is consistent with perturbing M slightly counter-clockwise.

Using the oriented interpretation for the boundary of M, lemma 2.2 relates homothets of M whose centres lie on a common spoke. This relationship is then used to show that our definition of an AB-bisector is a curve that separates the plane into two star-shaped regions in lemmas 2.3 and 2.4.

Lemma 2.2 *If A is a closed convex set, $p \notin A$ is a point in the plane, and $q \neq p$ is a point along* $\texttt{spoke}(p, A)$ *then $M_q^A \subset M_p^A$. Furthermore, if* $\texttt{spoke}(p, A)$ *does not exit M_p^A at a vertex then the boundaries of M_q^A and M_p^A intersect along a single directed edge of M_p^A.*

Proof: The point q lies on $\mathtt{spoke}(p, A)$ so $\mathtt{spoke}(q, A)$ shares the same attachment point to A as $\mathtt{spoke}(p, A)$ and M_q^A has the same point of tangency with A as M_p^A. Since M is convex and $d_M(q, A) < d_M(p, A)$, q is inside M_p^A and $M_q^A \subset M_p^A$.

Next, we show the boundary intersection property. The homothets M_p^A and M_q^A are replicas of M, so their boundaries either intersect in a connected set or they intersect at two or fewer points. Since both M_p^A and M_q^A are tangent to A at a common point ($\mathtt{spoke}(p, A)$ and $\mathtt{spoke}(q, A)$ share an attachment point to A) and neither M_p^A nor M_q^A intersect the interior of A, their boundaries cannot intersect at two distinct unconnected points.

Let b be a vertex of M_p^A that is shared with M_q^A. The segment from b to p goes through the centre of M_p^A. Since M_q^A shares the vertex b with M_p^A, this segment must also go through the centre of M_q^A. Therefore, $\mathtt{spoke}(p, A)$ exits M_p^A at a vertex. As a consequence, if $\mathtt{spoke}(p, A)$ does not exit M_p^A at a vertex then the boundaries of M_p^A and M_q^A can only intersect along a single directed edge of M_p^A since the intersection must remain connected and free of vertices. ∎

Lemma 2.3 *The AB-bisector is a continuous curve.*

Proof: To see that the AB-bisector is a continuous curve, we note that each point p on the bisector can be parameterized by the attachment point of $\mathtt{spoke}(p, A)$, breaking ties with the angle of $\mathtt{spoke}(p, A)$, and the length of $\mathtt{spoke}(p, A)$. As we advance along the AB-bisector, the attachment point either moves along the boundary of A in one direction (spokes to A cannot cross other spokes to A since d_M satisfies the triangle inequality) or the attachment point remains the same. In the latter case, the angle of $\mathtt{spoke}(p, A)$ either changes monotonically and continuously or remains fixed (when the bisector moves in the direction of $\mathtt{spoke}(p, A)$) in which case the length of $\mathtt{spoke}(p, A)$ changes monotonically. ∎

Lemma 2.4 *The AB-bisector bounds two sets—one star-shaped with respect to A and one star-shaped with respect to B.*

Proof: We prove that the region that contains A in the complement of the AB-bisector is star-shaped with respect to A.

Let p be a point on the A side of the AB-bisector. Either M_p^A does not touch B or $M_p^A = M_p^B$ where the attachment point of $\mathtt{spoke}(p, A)$ is the interior of a directed edge (not the head of the edge) and M_p^A is tangent to B along that same directed edge (p is not on the AB-bisector).

Let $q \neq p$ be a point on $\mathtt{spoke}(p, A)$. According to lemma 2.2 we have $M_q^A \subset M_p^A$ and the boundaries of M_q^A and M_p^A intersect along a single directed edge (since p is not on the AB-bisector). If M_q^A does not touch B then $d_M(q, A) < d_M(q, B)$ and q is on the A side of the AB-bisector. Otherwise, M_q^A is tangent to both A and B along a common directed edge, the common intersection of the boundaries for M_p^A and M_q^A. Consequently, q does not lie on the AB-bisector. Since the AB-bisector is a continuous curve (lemma 2.3) and $\mathtt{spoke}(p, A)$ does not cross the bisector, the entire spoke lies on the A side of the AB-bisector and that side of the AB-bisector is star-shaped with respect to A. ∎

Let $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ be a collection of *sites*—convex sets in the plane with disjoint interiors. The *Voronoi cell* of A_i is $\bigcap_{i \neq j, 1 \leq j \leq k}\{\text{the } A_i \text{ side of the } A_iA_j\text{-bisector}\}$. In the Euclidean metric, this definition is equivalent to the definition $\{p : d_M(p, A_i) < d_M(p, A_j) \text{ for all } j \neq i\}$; that is, all points for which A_i is the unique closest site. We denote the Voronoi cell of A_i by $C_{\mathcal{A}}(A_i)$.

Corollary 2.5 *The Voronoi cell $C_{\mathcal{A}}(A_i)$ is star-shaped with respect to A_i.*

Proof: The Voronoi cell of A_i is the intersection of the star-shaped sets containing A_i that are defined by the A_iA_j-bisectors for all $j \neq i$. ∎

The boundary of the cell of A_i is composed of portions of bisectors with other sites. Where two adjacent bisectors intersect we have a finite *Voronoi vertex*, which is equidistant from A_i and the other two sites defining the bisectors. Two adjacent bisectors may go off to infinity rather than intersecting—we can consider them to intersect at a Voronoi vertex at infinity. Later in the paper, we refer to a spoke from the infinite Voronoi vertex to A_i; the spoke is an infinite ray from A_i to the point at infinity whose direction keeps the ray between the two bisectors. The portion of one A_iA_j-bisector that appears between two Voronoi vertices is a *Voronoi edge*.

Corollary 2.6 *By introducing spokes from the (finite and infinite) Voronoi vertices around the boundary of the cell $C_{\mathcal{A}}(A_i)$, one decomposes the cell into regions bounded by portions of a single A_iA_j-bisector.*

Proof: Because bisectors are bi-infinite curves, any region of cell $C_{\mathcal{A}}(A_i)$ that is bounded by A_iA_j- and A_iA_k-bisectors either has a finite Voronoi vertex where these bisectors cross or an infinite Voronoi vertex in the direction that they go off to infinity. The spokes from these vertices to A_i are contained in the star-shaped cell $C_{\mathcal{A}}(A_i)$. ∎

2.2 A compact diagram for the post-office problem

With the notation developed above it is easy to define a diagram of $O(k)$ line segments in which one can determine two candidates for the closest neighbour of a query point. Draw spokes from the (finite and infinite) Voronoi vertices around the cell of A_i, as described in corollary 2.6 and shown in figure 3. Also, replace each A_i by its *core*, which is the convex hull of the spoke attachment points around A_i. The complement of the cores and spokes for all sites is a set of connected *spoke regions* bounded by at most six segments: two core segments and four spokes.

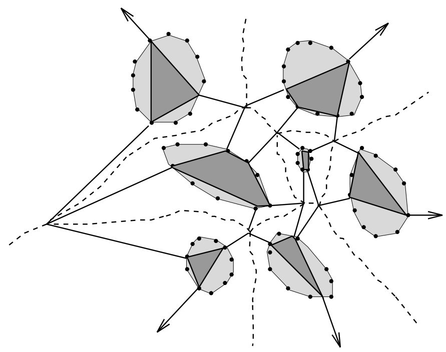

Figure 3: A compact diagram for point location

Theorem 2.7 *By introducing $O(k)$ segments, we partition the plane into cores and spoke regions for which the closest site is known to be among two candidates.*

Proof: Points in the core of A_i are in A_i. The spokes incident on A_i partition the remainder of the cell of A_i into regions bounded by a portion of the bisector of A_i and one other site. The union of the regions that border the same portion of the A_iA_j-bisector form a hexagonal spoke region that is contained in the union of the closures of the cells for A_i and A_j. Therefore A_i or A_j is the closest neighbour for the points in this spoke region.

To establish the size, it is sufficient to prove that the number of spokes is $O(k)$ because the number of core polygon edges is equal to the number of spokes. We can form a planar graph on the nodes $\{A_1, A_2, \ldots, A_k\}$ by connecting A_i with A_j by an edge that crosses the A_iA_j-bisector and stays between the spokes. Because all faces of this graph (except perhaps the outermost) have at least three vertices, Euler's relation implies that the graph has $O(k)$ edges and faces. ∎

If we process this diagram using any optimal point location structure [14, 18, 31] we can determine the two candidates for the closest neighbour to a query point q in $O(\log k)$ time. We can compute the distances to these two candidates by computing spokes from q to each of these candidates. Section 3.4 shows that computing spokes takes $O(\log n)$ time when the distance function is the Euclidean metric and $O(\log n + \log m)$ time when it is specified by a convex m-gon.

2.3 A compact diagram for retraction motion planning

The next lemma is the key to modifying the post-office diagram for retraction motion planning:

Lemma 2.8 *Distance to the convex sets A and B under a convex distance function d_M is does not have any local maxima along the AB-bisector.*

Proof:
We show that the AB-bisector does not contain any local maxima.

Let r be a point on the AB-bisector, let σ_A be a separating tangent of A and M_r^A, and let σ_B be a separating tangent of B and M_r^B. Let a be the attachment point of $\texttt{spoke}(r, A)$ and let b be the attachment point of $\texttt{spoke}(r, B)$.

Suppose that σ_A and σ_B intersect (if at all) in a region where the points a, r, and b appear in counterclockwise order around the boundary (see figure 4). Then the region R between σ_A and σ_B with b, r, a in counterclockwise order is unbounded. Moreover, the boundary of R divides the AB-bisector into two connected parts: one part within R and the other outside R.

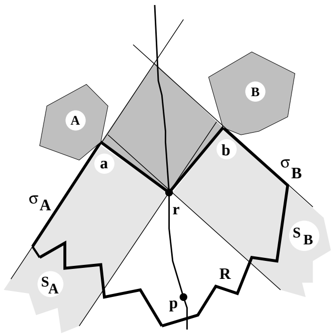

Figure 4: The AB-bisector has no local maxima.

Let p be a point on the AB-bisector in R. Since d_M satisfies the triangle inequality and, within R, tangent σ_A separates the AB-bisector from A, we have $d_M(p, A) \geq d_M(p, \sigma_A)$. Similarly, $d_M(p, B) \geq d_M(p, \sigma_B)$. The point p is on the AB-bisector, so we get $d_M(p, A) \geq \max\{d_M(p, \sigma_A), d_M(p, \sigma_B)\}$. Without loss of generality, assume that $d_M(p, \sigma_A) \geq d_M(p, \sigma_B)$.

Let S_A be the points with distance $d_M(r, A)$ or less from σ_A, and let S_B be the points with distance $d_M(r, B)$ or less from σ_B. Since σ_A and σ_B are infinite lines and M is convex, S_A and S_B are strips parallel to σ_A and σ_B respectively. Since each side of the AB-bisector is star-shaped with respect to A or B, the AB-bisector leaving r in R lies in the wedge formed by extending $\texttt{spoke}(r, A)$ and $\texttt{spoke}(r, B)$ through r, and leaves either S_A or S_B unless σ_A and σ_B are parallel. If σ_A and σ_B are not parallel then the point p on the AB-bisector exits S_A, since we assumed that $d_M(p, \sigma_A) \geq d_M(p, \sigma_B)$. If σ_A and σ_B are parallel then $d_M(p, A) = d_M(r, A)$. In either case, $d_M(p, A) \geq d_M(r, A)$ and the point r cannot be a local maximum point with respect to distance to A along the AB-bisector. ∎

The minimum distance along a Voronoi edge between two Voronoi vertices determines the maximum size of a homothet of M that can translate along the Voronoi edge. Due to weak unimodality, the minimum distance is attained either at a Voronoi vertex of the spoke region R or at a point $p \in R$ on the AB-bisector for which the tangent between A and M_p^A can be chosen to be parallel to a tangent between B and M_p^B $(= M_p^A)$. Let p be such a *minimum point* for the region R. If p is a minimum point on the whole AB-bisector, let τ be a tangent between A and M_p^A that has a parallel tangent between B and M_p^B. Otherwise, let τ be any common tangent between A and M_p^A. We define a *bottleneck segment* for R to be a segment through p parallel to τ. Under the Euclidean metric, the bottleneck segment can be chosen as the perpendicular bisector of the shortest segment joining A and B provided this shortest segment lies within the spoke region R.

In section 3.4 we describe the `bottleneck`(R) routine, which computes a bottleneck segment in $O(\log n)$ time in the Euclidean case and $O(\log n \log m)$ in the convex distance function case.

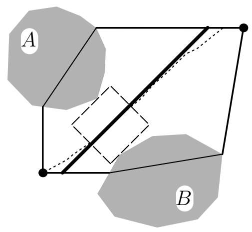

Figure 5: A bottleneck segment in a spoke region

Lemma 2.9 *A homothet of M can traverse a spoke region from Voronoi vertex to Voronoi vertex along a Voronoi edge if and only if it can traverse the spoke region via the bottleneck segment and incident spokes.*

Proof: Let R be a spoke region formed by A and B. The Voronoi edge that traverses R is a portion of the AB-bisector, which is equidistant to A and B. If a homothet of M can traverse R along spokes, and across the bottleneck edge of R, then it can traverse R using the Voronoi edge that avoids A and B equally. Our main task is to show that a path along a Voronoi edge has an equivalent path along spokes and a bottleneck segment without compromising the clearance between the path and A or the path and B.

Suppose that we can traverse R along the Voronoi edge. The homothet of M must not touch either of sets A or B at the minimum points along the AB-bisector within R. We have two cases, depending on whether this minimum point occurs inside spoke region R or at a Voronoi vertex on the boundary of R.

If the minimum point occurs at p and p is a minimum point for the entire AB-bisector, then τ_A is a common tangent between A and M_p^A with a parallel common tangent τ_B between B and M_p^B $(= M_p^A)$ and the bottleneck segment passes through p and is parallel to τ_A The homothet of M can traverse the bottleneck segment because it is separated from A and B by τ_A and τ_B respectively. When the homothet reaches a spoke, it moves to the Voronoi vertex by moving away from A or B, whichever is closer in d_M distance and the distance to A exceeds the distance to B only after we cross the Voronoi vertex (corollary 2.5). Thus it can traverse the spoke region.

If the minimum point p for R is not a minimum point for the AB-bisector, then p is a Voronoi vertex for the spoke region R and the bottleneck segment for R leaves p parallel to a tangent τ_A between A and M_p^A. A tangents τ_B between B and M_p^B diverge from τ_A within R since A, B, and M are convex. Therefore, moving a homothet of M along the bottleneck segment increases the distance to τ_B and therefore to B, and the homothet remains separated from A by τ_A. The homothet of M continues along the bottleneck segment until it reaches a spoke of R and moves to the Voronoi vertex along the spoke as before. ■

If we store the minimum point and its spokes along with the bottleneck segment for each spoke region, then we can determine how to move onto the retraction diagram without using of the original polygon information.

Lemma 2.10 *A homothet of M can move onto the retraction diagram from a free placement by locating its initial spoke region and moving parallel to its bottleneck segment until it encounters the spoke region boundary.*

Proof: For a spoke region R adjacent to polygons A and B, suppose that we have stored a minimum point p and the spokes $\texttt{spoke}(p, A)$ and $\texttt{spoke}(p, B)$. Given an initial free placement of a homothet of M with origin at q, we move the homothet parallel to the bottleneck segment and away from the minimum spokes $\texttt{spoke}(p, A)$ and $\texttt{spoke}(p, B)$. We will show that this movement avoids collision with A and B, so the origin can reach a spoke that forms part of the boundary of R. The origin can follow this spoke away from its closest site up to a Voronoi vertex, which puts the homothet onto the retraction diagram.

Recall that there is a line τ that is tangent to A at the attachment point of $\texttt{spoke}(p, A)$ and is parallel to the bottleneck segment. If the homothet M and A are separated by τ, then M cannot hit A. If M intersects τ then a line separating M and A must cross τ and allow M to move away from $\texttt{spoke}(p, A)$. If M and A are on the same side of τ and the angle from $\texttt{spoke}(p, A)$ to the next spoke on the boundary of R is at most π, then the same argument applies. (The angle condition is automatic for finite spokes bounding R and can be obtained for infinite spokes by choosing an infinite spoke with attachment point opposite that of $\texttt{spoke}(p, A)$.) ■

Finally, by weighting each bottleneck segment by its minimum distance to a site, we compute a maximum weight spanning tree of the compact diagram. We can process this tree [30] to answer retraction motion planning queries and to compute paths that maximize the minimum clearance to sites, where clearance is measured by the distance function d_M.

Theorem 2.11 *Using an $O(k)$-size data structure, one can determine in $O(\log k)$ time if there is a translational motion that gets M from p to q avoiding the k convex obstacles. One can compute a motion in time proportional to its complexity, which is $O(k)$.*

Proof: We can think of the retraction diagram as a weighted planar graph whose $O(k)$ vertices are the Voronoi vertices and whose $O(k)$ edges represent paths that traverse a bottleneck segment. The weight of an edge is the largest scale for a homothet of M that can use the corresponding bottleneck segment. Rohnert [30] has shown that we can compute a maximal spanning tree in this graph and process it so that we can determine in logarithmic time the largest scale homothet that can traverse the diagram from a given initial point to a given final point. We use lemma 2.10 to determine the initial and final nodes of the tree in $O(\log k)$ time. ■

3 Computing the compact Voronoi diagram

In section 3.1 we describe an algorithm based on Fortune's sweep [15] that computes the compact Voronoi diagram under general position assumptions. Section 3.2 modifies this algorithm to the degenerate cases that the general position assumptions avoid. We describe a randomized incremental construction algorithm whose expected execution time matches our worst-case deterministic time bound in section 3.3. Both the sweep and randomised algorithms work for any convex distance function, when given a subroutine to compute Voronoi vertices and their spokes. Section 3.4 describes the subroutines for the Euclidean metric and for the distance function defined by a convex polygon M; section 3.5 discusses lower bounds.

3.1 A deterministic sweep algorithm

The key information for the compact compact Voronoi diagram is the Voronoi vertices for the set of polygons. Given the Voronoi vertices and the polygons that generate each vertex, we can find the spokes to the nearest polygons, identify the spoke regions (by sorting the Voronoi vertices around each polygon), and thus solve the post-office problem. Once we know the spoke regions, we can also find the bottleneck segments for solving retraction motion planning. Our algorithm finds the Voronoi vertices of the polygons and finds the spokes from each Voronoi vertex to the nearest polygons. For simplicity, we assume that the polygons are in general position. In particular, we require that no polygon have a vertical edge, that no vertical line be tangent to two polygons, and that no four polygons be tangent to one homothet of the convex distance function. Section 3.2 describes changes to the algorithm that handle degeneracies and eliminate the general position assumptions.

Our algorithm sweeps a vertical line, called the *sweepline*, from left to right across the Voronoi diagram of the polygons and finds all Voronoi vertices, much the same as Fortune's sweep algorithm [15]. As the sweepline travels across the plane, we examine the Voronoi diagram defined by the polygons (or parts of polygons) to the left of (or on) the sweepline, together with the sweepline itself. We detect all Voronoi vertices by observing the changes in the boundary of the Voronoi cell for the sweepline as the sweepline moves. When the sweepline nears $x = +\infty$, all polygons—as well as all Voronoi vertices of the the Voronoi diagram for the polygons—lie to the left of the sweepline, so our sweep detects all Voronoi vertices. A typical picture of the algorithm in mid-sweep is shown in figure 6.

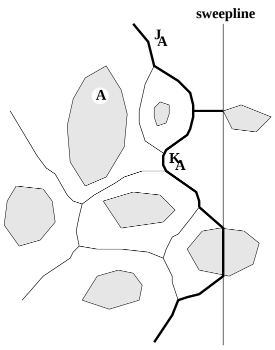

Figure 6: The sweep front and Voronoi diagram left of the sweepline

The boundary of the sweepline's Voronoi cell is called the *sweep front* and consists of Voronoi edges between the sweepline and some polygons. We denote a section J of the sweep front between the sweepline and a single polygon A by J_A and call J_A a *front arc*. Figure 6 illustrates that one polygon (polygon A) may have many front arcs associated with it at any one time (arcs J_A and K_A).

Two data structures underlie the sweep algorithm: a priority queue $\mathcal{Q}$ that schedules changes in the sweep front, called *events*, and a threaded, balanced, binary tree $\mathcal{T}$ that stores the sweep front.

The priority queue $\mathcal{Q}$ schedules events in the order that the sweepline will encounter them in its sweep across the plane. The events correspond to points in the plane and are sorted in the schedule by ascending x-coordinate. Events are added to the queue only if they appear to the right of the sweepline's position at the time of insertion. Each scheduled event stores pointers to the polygon(s) that generate it, and, as applicable, a Voronoi vertex of three polygons as well as the attachment points of the spokes from this vertex to each of the generating polygons.

The threaded, balanced, binary tree $\mathcal{T}$ stores the sequence of front arcs along the sweep front. The threading locates adjacent front arcs in logarithmic time while the balanced binary tree supports a logarithmic time binary search through the sweep front. Each tree node for a front arc J stores a pointer to the polygon that generates J, pointers to any events in the schedule that J has generated, and a spoke to the polygon that defines J that crosses J. Unlike previous sweep algorithms for Voronoi diagrams, we neither store nor compute the curves that make up the front arc J. Theorem 3.6 provides a means to search across the sweep front based on a spoke for each front arc rather than on the intersection points of two front arcs; lemma 3.5 proves that such a spoke exists in the queue $\mathcal{Q}$ for every front arc.

While the sweepline moves from left to right across the plane, the sweep front also moves across the plane (lemma 3.1.) Lemma 3.2 shows that every Voronoi point for the Voronoi diagram of the polygons appears along the sweep front, so our algorithm for the compact Voronoi can restrict itself to watching the sweep front.

Lemma 3.1 *Given a set of polygons $\mathcal{P}$ and two vertical sweeplines ℓ_1 and ℓ_2 at $x = x_1$ and $x = x_2$ respectively with $x_1 < x_2$, every point on the sweep front for ℓ_1 lies closer to some site P than to $\ell 2$.*

Proof: We assume that some polygon exists to the left of sweepline ℓ_1. Otherwise, the sweep front for both ℓ_1 and ℓ_2 does not exist.

Let r be a point on the sweep front for ℓ_1 in $V(\mathcal{P} \cup \{\ell_1\})$ and let σ be the spoke from r to ℓ_1. Since our distance function is convex, and since $x_1 < x_2$, the spoke σ' from r to ℓ_2 in $V(\mathcal{P} \cup \{\ell_2\})$ is an extension of σ.

Let P be the polygon that has r on the boundary of its Voronoi cell in $V(\mathcal{P} \cup \{\ell_1\})$. Then the distance from r to ℓ_2 is strictly greater than from r to P since σ' extends σ. Consequently, the sweep front for ℓ_2 must intersect σ' to the right of r, and hence to the right of the sweep front of ℓ_1 in $V(\mathcal{P} \cup \{\ell_1\})$. ∎

Lemma 3.2 *Every Voronoi vertex of $V(\mathcal{P})$ eventually appears along the sweep front.*

Proof: Let v be a Voronoi vertex of $V(\mathcal{P})$ for the polygons P, Q, and R, and let ℓ be the vertical sweepline. Since M_v^P is convex, it has a rightmost point at $x = x_0$. When ℓ is at $x = x_0$, it is tangent to M_v^P, so v must lie on the Voronoi edge of ℓ in $V(\mathcal{P} \cup \{\ell\})$. Since ℓ is tangent to M_v^P at its rightmost point, v lies left of ℓ and is therefore on the sweep front. ∎

Our sweep algorithm detects Voronoi vertices by observing changes in the order of front arcs on the sweep front. Front arcs can change their order in two ways: a new front arc can appear along the sweep front or some existing front arc can be removed from the sweep front. Notice that two front arcs cannot exchange places along the sweep front since all polygons in the underlying Voronoi diagram are disjoint. Lemma 3.3 shows that front arcs appear along the sweep front only when a leftmost point of a polygon crosses the sweepline. We call such occurrences *site events.* Lemma 3.4 shows that front arcs are removed from the sweep front precisely when a Voronoi vertex crosses the sweep front. We denote these latter events as *circle events.*

Lemma 3.3 *A front arc is created on the sweep front if and only if a site event occurs.*

Proof: We start by showing that a site event creates a front arc.

When the leftmost point p of a polygon P is on the sweepline ℓ, that point has a zero distance to both ℓ and P and lies on the $P\ell$-bisector and on the boundary of ℓ's Voronoi cell. Since point p is not on the right side of ℓ, it belongs to the sweep front and a front arc generated by P. Before P crosses ℓ, polygon P cannot have a front arc, so p must belong to a new front arc.

Next, we show that every new front arc is caused by a site event.

Let J_P be a new front arc generated by polygon P. If no front arcs existed prior to J_P then there are no polygons left of (or on) the sweepline ℓ before the arrival of J_P. Since J_P is a portion of the $P\ell$-bisector, arc J_P can only be caused when polygon P first pierces ℓ.

Otherwise, J_P breaks an existing front arc K_Q in the sweep front at the points p (upper) and q (lower). Arc K_Q is generated by polygon Q. The area between J_P and the part of K_Q that was broken when J_P appeared belongs to the Voronoi cell $C_{\mathcal{P}}(P)$ for P in $V(\mathcal{P})$ where $\mathcal{P}$ is the set of all sites. Since $C_{\mathcal{P}}(P)$ is star-shaped (corollary 2.5), some portion of P must appear within the area A bounded by $\texttt{spoke}(p, Q)$, $\texttt{spoke}(p, \ell)$, $\texttt{spoke}(q, Q)$, $\texttt{spoke}(q, \ell)$, Q, and ℓ. Let r be any point within $C_{\mathcal{P}}(P)$ that also lies inside the area A. Before J_P appeared, point r belonged to the Voronoi cell of the sweepline in $V(\mathcal{P} \cup \{\ell\})$ since the point r is in a spoke region bounded by Q and the sweepline and the front arc K_Q cannot leave $V(\mathcal{P})$Q. Since ℓ is an infinite line, if the attachment point of $\texttt{spoke}(r, P)$ lies left of or on ℓ then r is either on the $P\ell$-bisector or on the P-side of the $P\ell$-bisector. In either of these cases, some portion of the $P\ell$-bisector appears along the sweep front. So, before J_P appeared, all the attachment points to P for the points in A were to the right of the sweepline.

Let a be the attachment point to P of $\texttt{spoke}(r, P)$. Consider what happens as the sweepline crosses a if some portion of P already lies to the left of the sweepline. Once ℓ crosses a, the point r becomes part of the sweep front or part of the interior of A; both cases imply that arc J_P then exists. Since P is convex, immediately prior to ℓ crossing a, ℓ cuts through P in a neighbourhood of a and $P \cap \ell$ is part of the sweep front. So, when ℓ crosses a, some portion of the $P\ell$-bisector at a is on the sweep front and J_P is an extension of that portion. This contradicts the fact that K_Q appears both above and below J_P on the sweep front, so no portion of P lies to the left of the sweepline. Consequently, arc J_P only appears when the sweepline first encounters P. ■

Lemma 3.4 *A front arc is removed from the sweep front if and only if a Voronoi vertex equidistant to three or more polygons crosses the sweep front.*

Proof: We begin by showing that a front arc is removed from the sweep front when a Voronoi vertex crosses the sweep front.

Let v be a Voronoi vertex between the polygons P, Q, and R that lies on the sweep front. We assume that $\texttt{spoke}(v, P)$ lies immediately counterclockwise from $\texttt{spoke}(v, \ell)$ and $\texttt{spoke}(v, R)$ lies immediately clockwise from $\texttt{spoke}(v, \ell)$ (see figure 7).

The Voronoi cell for Q does not extend beyond $\texttt{spoke}(v, P)$ and $\texttt{spoke}(v, R)$ since $\texttt{spoke}(v, P)$ and $\texttt{spoke}(v, R)$ lie completely in the Voronoi cells for P and R, respectively. Any front arc generated by Q that extends between the PQ-bisector and the QR-bisector cannot pass the point v; the front arc must stay in Q's Voronoi cell. Lemma 3.1 shows that the sweep front leaves v when ℓ moves right of its current position, so the front arc generated by Q between the PQ- and QR-bisectors disappears when v lies on the sweep front.

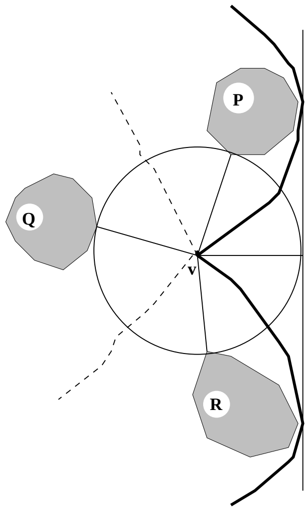

Figure 7: The front arc for Q is removed at v.

Next, we show that a largest homothet of the convex distance function is tangent to three polygons and the sweepline when a front arc is removed from the sweep front. The origin of the homothet is a Voronoi vertex of the three polygons.

Let K_Q be a front arc, generated by polygon Q, that is being removed from the sweep front. The front arcs at either end of the sweep front are infinite and cannot be removed from the sweep front, so arc K_Q must have a front arc J_P above and a front arc L_R below it on the sweep front. If K_Q is a single point s, then point s belongs to K_Q as well as to J_P and L_R as endpoints. The polygons P and R must be distinct since the Voronoi cell for P would otherwise surround the cell for Q, so the point s is a Voronoi vertex for P, Q, and R.

Otherwise, the front arc K_Q is a curve rather than a single point that is removed from the sweep front at one instant. When the sweepline moves right, the sweep front K_Q also moves (lemma 3.1), though it must stay within the Voronoi cell for Q $C_{\mathcal{P}}(Q)$ in $V(\mathcal{P})$. Since K_Q is removed from the sweep front whenever the sweepline moves to the right, the cell $C_{\mathcal{P}}(Q)$ does not extend between K_Q and the sweepline. The arc K_Q must therefore be a Voronoi edge in $V(\mathcal{P})$ between Q and some polygonal site P, i.e. part of the PQ-bisector. However, K_Q is also part of the bisector between Q and the vertical sweepline. The site P must have a vertical line segment on its boundary to match the $Q\ell$-bisector, contradicting the general position assumptions; the front arc K_Q can only be removed from the sweep front if it degenerates to a point first. ∎

A summary of the sweep algorithm appears in figure 8. After we specify the $\texttt{vertex}(ABC)$ subroutine, we present invariants for the data structures $\mathcal{Q}$ and $\mathcal{T}$ and show how the algorithm treats each type of event to preserve these invariants. As a consequence of lemma 3.4, we conclude that all Voronoi vertices are reported at the end of the sweep.

The subroutine $\texttt{vertex}(ABC)$ accepts three sites and computes a finite or infinite Voronoi vertex v where the Voronoi cells for sites A, B and C occur in counterclockwise order around v. In the finite case, this corresponds to the point p where $M_p^A = M_p^B = M_p^C$. If there is no such largest homothet of the distance function M then the point at infinity is returned as the Voronoi vertex of A, B, and C. The subroutine returns the vertex v, the rightmost point of M_v^A, and a spoke from v to each of A, B, and C (in the case where v is an infinite vertex, the spokes to A, B, and C extend to infinity in a direction that keeps the spoke inside the Voronoi cells of A, B, and C respectively). Section 3.4 discusses two implementation of this subroutine: one in $O(\log n)$

```
Schedule a leftmost point of each polygon as site events
While the schedule is not empty do
  Remove the next event p
  if p is a new site event
    Binary search the sweep front to find the arc J nearest p
    Unschedule all circle events for the arc J
    Split J into two front arcs
    Create a new front arc for p
    Schedule circle events for the new front arcs
  otherwise, p is a circle event
    Record the centre of p as a Voronoi vertex
    Delete a front arc
    Unschedule any circle events associated with the front arc
    Schedule circle events coming from the disappearance of a front arc
  endif
endwhile
```

Figure 8: Outline of the sweepline algorithm.

time under the Euclidean metric and the other in $O(\log n \log m)$ time under the distance function induced by an m-gon.

The priority queue $\mathcal{Q}$ changes the conceptually continuous sweep into a set of discrete steps; we must ensure that all changes to the sweep front generate an event in $\mathcal{Q}$. To identify all new front arcs, all site events must appear in $\mathcal{Q}$ until the sweepline encounters them (lemma 3.3). Also, every Voronoi vertex for three consecutive front arcs along the sweep front must appear in $\mathcal{Q}$ (lemma 3.4). This latter condition implies that the tree $\mathcal{T}$ must always contain the correct order of the front arcs along the sweep front. These restrictions dictate the invariants

- The tree $\mathcal{T}$ always contains the ordered list of sites that generate the front arcs on the sweep front for the current position of the sweepline.
- The queue $\mathcal{Q}$ contains precisely one site event for each polygon that the sweepline has not encountered and one circle event for each set of three consecutive front arcs. For the circle event, if M' is the homothet of M that is tangent to the three polygons that define the consecutive front arcs, then the rightmost point of M' lies right of the sweepline.

The sweep algorithm begins by satisfying the invariants in the initial position of the sweepline. One site event is added to $\mathcal{Q}$ for each polygon. Next, we initialise the tree $\mathcal{T}$ with the sites whose leftmost x-coordinate equals the leftmost x-coordinate of the entire set of sites. The order of the sites in $\mathcal{T}$ is that of their y-coordinates. We also remove the site events in $\mathcal{Q}$ for each of the sites already in $\mathcal{T}$. We then remove the event from the head of $\mathcal{Q}$ and process events until $\mathcal{Q}$ is empty.

When the sweepline encounters a site event at a point p, a new front arc J appears along the sweep front (lemma 3.3). We begin by re-establishing our invariant on the sweep front stored in the tree $\mathcal{T}$. Theorem 3.6 describes a search to locate K the the front arc nearest the point p. We add J to the sweep front, split K into arcs K' (above J) and K'' (below J), and add the site that generates J to $\mathcal{T}$ to satisfy the invariant on $\mathcal{T}$. The changes to $\mathcal{T}$ create a reference to the site that generated K both before and after the site that generates J within the site order of $\mathcal{T}$.

Next, we update the schedule $\mathcal{Q}$ to reflect the change in the sweep front. The site event for p was removed from the schedule when we decided that we had a site event. The sweepline can encounter the leftmost point of only one polygon at a time (since the polygons are in general position) so $\mathcal{Q}$ contains precisely the required set of site events. Therefore, we only concern ourselves with the circle events. The single circle event defined when front arc K was the middle of three consecutive arcs must be removed from the schedule since adding arc J split arc K. The circle events where K was the first or third arc in the sequence remain in the schedule since arcs K' and K'' fulfill the role of K in those instances. Finally, if the sequence of front arcs was IKL before arc J divided arc K, then we have three new sets of consecutive arcs to be added to the schedule as circle events: $\texttt{vertex}(IK'J)$, $\texttt{vertex}(JK''L)$, and $\texttt{vertex}(K'JK'')$, the last of which is guaranteed to be a vertex at infinity.

In the search for the nearest front arc to a site event, we obtain information about each front arc through a spoke that crosses it. Lemma 3.5 shows that such a spoke is always available either in the priority queue $\mathcal{Q}$ or as a degenerate spoke.

Lemma 3.5 *For each front arc K_Q in the sweep front there exists a spoke to the polygon Q defining K_Q that intersects K_Q.*

Proof: Let K_Q be a front arc generated by site Q, let J_P be the front arc above K_Q on the sweep front, and let L_R be the front arc below K_Q on the sweep front. Arc J_P is generated by polygon P, and arc L_R is generated by polygon R. Neither J_P nor L_R necessarily exists. There are three configurations for K_Q that we must consider:

- the sweepline has not crossed the rightmost point of Q
- K_Q is the highest or lowest front arc on the sweep front
- a circle event for $\texttt{vertex}(PQR)$ is in the queue $\mathcal{Q}$

If the sweepline has not crossed the rightmost point of Q then the sweepline cuts through Q and both the sweep front and K_Q follow this cut inside Q. We have a degenerate spoke σ at any of the intersection points between the interior of Q and the sweepline.

If K_Q is the highest or lowest front arc on the sweep front then the Voronoi cell for Q is unbounded. If K_Q is not the only front arc then an infinite spoke perpendicular to the outer tangent of Q and the generating site of K_Q's sole neighbouring front arc remains within the Voronoi cell for Q and crosses K_Q. Otherwise, any infinite spoke that attaches itself to Q and goes to the right of Q crosses K_Q.

If both front arcs J_P and L_R exist then there is a circle event for $\texttt{vertex}(PQR)$ at the vertex v is in the schedule $\mathcal{Q}$. The vertex v may be at infinity. The $\texttt{spoke}(v, Q)$ lies between the PQ and QR bisectors as does the front arc K_Q. This spoke is computed at the same time as $\texttt{vertex}(PQR)$ and is stored in $\mathcal{Q}$ so it is available to the algorithm. Since the sweepline has not encountered the circle event yet, vertex v lies to the right of the sweep front, and $\texttt{spoke}(v, Q)$ crosses K_Q. ∎

If we have a spoke that crosses a front arc, we are indirectly given a point along the front arc. Theorem 3.6 uses this information to find the front arc nearest a site event.

Theorem 3.6 *Given a point p on the sweepline, we can locate the nearest front arc to p under a convex distance function by a binary search and one call to* `vertex()`.

Proof: The Voronoi cell for the sweepline is star-shaped (corollary 2.5), so the spokes from the front arcs to the sweepline partition the sweepline into disjoint intervals. Finding the nearest front arc to the point p is equivalent to finding in which interval the point p lies.

We could perform a binary search for p based on the endpoints of the intervals; however, locating each interval endpoints requires a call to `vertex()`. (The interval endpoint is the attachment point for the spoke from the ends of two front arcs, which is the Voronoi vertex between the sweepline and the polygons generating the front arcs.) Instead, we perform a binary search based on a representative point for each interval. The resulting space between two representative points spans exactly two intervals and contains one interval endpoint. We compute that endpoint with one call to the `vertex()` subroutine and determine which of the two candidate intervals contains the point p.

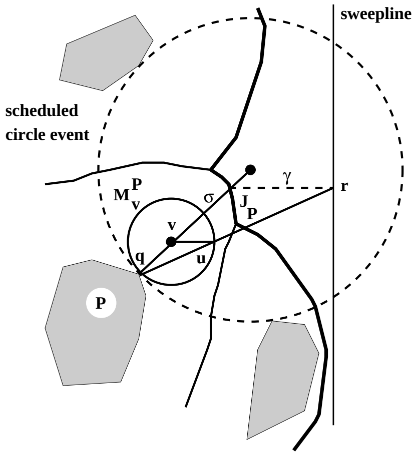

Figure 9: Spoke σ crosses J_P.

What is the representative point in the interval for front arc J_P of site P? We derive the representative point from a spoke that crosses J_P. Let σ be the spoke to P from lemma 3.5 that crosses J_P and let q be the attachment point of σ to P (see figure 9). If v is a point on σ other than q, then let u be the rightmost point of M_v^P. Extend the line from q to u so that it crosses the sweepline; the intersection point r is our representative point.

How do we know that the point r lies in the interval for J_P on the sweepline? Point r is the attachment point for the spoke from the intersection of σ and J_P to the sweepline. The direction of a spoke to a vertical line is always the same; let γ be the line/spoke through r in that direction, extending to the left of the sweepline. The triangle formed by r, q, and the intersection point of σ and γ is mathematically similar to, and shares a corner at q with, the triangle formed by u, q, and v, so the former triangle is inscribed in a homothet M' of the convex distance function that is tangent to both P and the sweepline. Being equidistant to P and the sweepline, the centre of M' is on the front arc J_P, and therefore on the intersection of σ and J_P. ∎

A circle event signals the removal of a front arc J from the sweep front and the existence of a Voronoi vertex among polygons (lemma 3.4). As with the site event, we must alter the sweep front $\mathcal{T}$ and the schedule $\mathcal{Q}$ to maintain our invariants. The change to the sweep front $\mathcal{T}$ is simple: the circle event stores a pointer to the front arc J to be removed from the sweep front. The threaded nature of $\mathcal{T}$ allows an efficient removal operation.

Once arc J is removed from the sweep front, it can no longer play a role as one of three consecutive front arcs along the sweep front for circle events. The circle events in $\mathcal{Q}$ where J appears as either the first or last of three consecutive front arcs must be removed from $\mathcal{Q}$. Also, the neighbouring front arcs of J take J's place as a first or last arc in sequences of three consecutive front arcs. Suppose the sequence of front arcs around J, from top to bottom, is $HIJKL$, then we remove the circle events corresponding to `vertex`(HIJ) and `vertex`(JKL) and replace them with the circle events for `vertex`(HIK) and `vertex`(IKL) respectively. The sweep front did not change anywhere else, so all other scheduled circle events remain valid.

When the sweepline reaches $x = +\infty$, all polygons are to its left. The Voronoi vertex topology obtained for polygons left of the sweepline is precisely the topology for the Voronoi diagram of all polygons. Our treatment of site events and circle events, along with the characterisations of lemma 3.3 and lemma 3.4, culminate in the proof of theorem 3.7.

Theorem 3.7 *The sweepline algorithm (outlined in figure 8) correctly finds all Voronoi vertices and their order about each polygon for a convex distance function.*

Before analysing the time complexity of our algorithm to find the compact Voronoi diagram, we must establish the time required to perform operations on the schedule $\mathcal{Q}$ and the sweep front $\mathcal{T}$. Each data structure performs its operations in logarithmic-time of the structure size. Lemma 3.8 proves the sweep front maintains $O(k)$ arcs and lemma 3.9 proves the schedule always has $O(k)$ events scheduled, so both data structures perform insertions and deletions in $O(\log k)$-time.

Lemma 3.8 *At most $2k - 1$ arcs appear along the sweep front.*

Proof: A new arc appears on the sweep front only when we encounter a site event (lemma 3.3). While processing a site event, one new arc is added and an existing arc is split into two smaller arcs if some arc already exists along the sweep front. Consequently, the first new site event adds a single arc to the sweep front and every subsequent new site event adds at most two arcs to the front. There are k new site events, one for each polygon, so the sweep front has size at most $2k - 1$. ∎

Lemma 3.9 *The schedule for the sweep algorithm contains at most $2k$ events at any moment.*

Proof: The schedule contains two types of events: site events and circle events. If there are s site events in the schedule, then $k - s$ site events have been processed, producing at most $2(k - s) - 1$ front arcs. Each consecutive triple of front arcs produces at most one circle event in the schedule. Moreover, when one front arc J_P is removed from the sweep front, all circle events in the schedule that depend on J_P are removed from the schedule. Consequently, the schedule contains at most $s + 2(k - s) - 1 = 2k - s - 1$ events. Since $0 \leq s \leq k$, no more than $2k$ events are in the schedule at any one time. ∎

In conclusion, we show that our algorithm for finding the compact Voronoi diagram requires $O(k(T_v + \log k))$ time for k polygons having a total of n vertices, where T_v denotes an upper bound on the time required to complete one call to the `vertex()` subroutine. The algorithm handles exactly k site events, one per polygon, and $O(k)$ circle events since each circle event discovers one Voronoi vertex and there are $O(k)$ Voronoi vertices. We examine the processing time for each event type separately.

When handling a site event, we perform a binary search through the $O(k)$ arcs of the sweep front and require $O(\log k)$ search steps to narrow our search to two candidate front arcs. Theorem 3.6 shows how each step of the binary search requires constant time to complete. Once we have two candidate front arcs, we need $O(T_v)$ operations to find the front arc being split and an additional $O(\log k)$ time to insert the new front arc into the sweep front.We then calculate three Voronoi vertices for the new arc in $O(T_v)$ time and add them to the schedule in $O(\log k)$ time; we also remove circle events that are no longer valid, in $O(\log k)$ time apiece. Each site event therefore requires $O(T_v + \log k)$ time to complete.

The circle events are simpler to process than site events. We remove a front arc from the sweep front in $O(\log k)$ time and remove at most two circle events from the schedule, again in $O(\log k)$ time. Next, we compute two Voronoi vertices and insert them into the schedule as future circle events in $O(T_v + \log k)$ time.

We sum the time required for each type of event to obtain the complexity of the algorithm. We spend $O(k(T_v + \log k))$ time processing site events, and we spend $O(k(T_v + \log k))$ time on circle events, yielding a total complexity of $O(k(T_v + \log k))$.

Theorem 3.10 *The compact Voronoi diagram for a set of k polygons with a total of n corners can be computed in time $O(k(T_v + \log k))$.*

3.2 Degeneracies

The sweep algorithm of section 3.1 for the compact Voronoi diagram assumes that all the sites are in general position to avoid degenerate cases. This section presents enhancements to the sweep algorithm that makes the general position assumptions unnecessary. We describe the set of changes to the algorithm before addressing each position assumption, the degeneracies that the assumption avoided, and the manner in which the modified algorithm handles the degeneracies.

Three changes to the sweep algorithm are sufficient to remove the general position assumptions: one change to the priority queue data structure $\mathcal{Q}$, one change to our handling of circle events, and one clarification for scheduling site events.

First, to the priority queue $\mathcal{Q}$ we add keys that are secondary to the x-coordinate of the event. The keys for $\mathcal{Q}$, in order of precedence, are

1. x-coordinate of the event (in ascending order)
2. type of event (site events before circle events)
3. y-coordinate of the event (in descending order)
4. for circle events, the angle α as measured ccw from the sweepline's spoke to the second spoke encountered ccw, $0 \leq \alpha < 2\pi$ (in ascending order)

Second, we process multiple circle events together under special conditions. This change allows us to recognize that a Voronoi vertex may be defined by more than three sites.

Third, we resolve an ambiguity when scheduling site events. If a site does not have a unique leftmost vertex then we create a site event based on the highest of its leftmost vertices.

These three changes, along with some careful implementation in the rest of the sweep algorithm, are sufficient to correctly handle all conditions that our general position assumption of section 3.1 precluded.

3.2.1 Assumptions related to site events

The only assumption that relates directly to individual sites, and consequently site events, is that no site has any vertical edges. Vertical edges are problematic when they are the leftmost edge of a polygonal site. When the sweepline crosses a vertical leftmost edge, the entire portion of the sweep front that is closest to the vertical edge is part of the Voronoi diagram for the polygons. We must recognize all Voronoi vertices along this portion of the sweep front, even as the sweep front jumps to the sweepline at this same instant.

The degeneracy caused by vertical leftmost edges is solved by two of the changes to the sweep algorithm. First, the additional keys for the priority queue $\mathcal{Q}$ enforce a nested-sweep tactic in the algorithm. While the sweepline moves from left-to-right in the plane, whenever multiple events occur at a common x-coordinate, the tertiary y-coordinate key processes events along the sweepline from top to bottom, a vertical sweep. The intermediate key on event types forces this vertical sweep to happen twice: once for site events and a second time for circle events. We can imagine the first of the vertical sweeps as discovering new circle events and the second sweep as handling the circle events and advancing the sweep front.

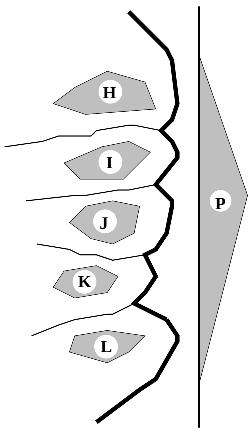

Figure 10: A vertical edge for P can produce many Voronoi vertices.

The second change to the algorithm is to eliminate an ambiguity for scheduling a site event. The additional rule ensures that the site event triggers as high along the sweepline as possible by scheduling the highest leftmost point for each site.

These two changes work together to locate all the Voronoi vertices associated with the vertical edge. Let the vertical edge for site P be from point u to point w where u is above w. Let the sequence of front arcs nearest edge (u, w) be generated by polygons $HIJKL$ (in order along the sweep front) as in figure 10. When we encounter the site event for P, a single front arc is inserted for point u as if point w had been perturbed to the right. Suppose the new front arc divides the arc for site H. If there is a Voronoi vertex associated with edge (u, w) then the front arc for u generates a circle event from `vertex`(PHI) to occur at the sweepline's current position. When we process the circle event for `vertex`(PHI) after all other site events are done, front arc for H is removed from the sweep front and we schedule a new circle event for `vertex`(PIJ) since their three front arcs become consecutive. This circle event goes immediately to the top of $\mathcal{Q}$. Processing this new circle event removes the front arc for I from the sweep front and schedules yet another circle event for `vertex`(PJK). This ripple of removing front arcs continues until the undiscovered Voronoi vertices are no longer associated with the edge (u, w).

The general position assumption that no two sites are tangent to a common vertical line indirectly concerns site events. The degeneracy occurs only when these site events are the first of all site events that occur—each event creates a single front arc and does not split any existing front arc. This degeneracy is resolved when we initialise the sweep front with one front arc for each site event at the same x-coordinate as the first site event. The front arcs are ordered consistently with the sites along the sweepline.

3.2.2 Assumptions related to circle events

When more than three sites are tangent to a largest homothet of the convex distance function, the origin of the homothet is a Voronoi vertex with degree greater than three. Without any changes to the sweep algorithm, a Voronoi vertex of degree four would be split into two vertices of degree three, where both vertices have two sites in common, are adjacent across a spoke region, and occur at the same position in the plane. Post-processing of the Voronoi vertices could then merge the points into one. However, the sweep algorithm itself can recognize Voronoi vertices of degree greater than three.

Suppose that polygons P, Q, R, and S are tangent to a common homothet of the distance function, and suppose that the ccw order of these polygons around the homothet are $PQRS$, starting from the attachment point for the spoke to the sweepline from `vertex`(PQR). Then the sweep front has the matching order of front arcs (in order from top to bottom) and circle events `vertex`(PQR) and `vertex`(QRS) appear in the queue $\mathcal{Q}$.

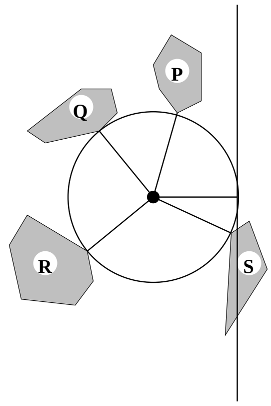

Figure 11: A Voronoi vertex of degree four

Again, two modifications to the sweep algorithm cooperate to solve this problem. The events `vertex`(PQR) and `vertex`(QRS) have queue keys identical in the first three components. Consequently, the two events are grouped together in $\mathcal{Q}$. The last key places `vertex`(PQR) ahead of `vertex`(QRS) in the queue since Q precedes R ccw around the homothet.

Since we know that these two circle events are grouped together, we can change our handling of circle events to remove both of them from the queue $\mathcal{Q}$ at the same time. If we are handling a circle event for the Voronoi vertex v, we can remove all the circle events that relate to v from the top of $\mathcal{Q}$. All these events appear at the top of the queue when the first event for v is removed. For instance, in figure 11 the circle event for `vertex`(PQR) will be immediately followed by the circle event for `vertex`(QRS). We can remove all circle events from $\mathcal{Q}$ that relate to v, remove the front arcs between the topmost front arc J and the lowest front arc K (the arcs generated by both Q and R in the figure) from the sweep front, and schedule new circle events as if there had been a single front arc to remove from between J and K. The change recognises all the sites that generate v at one instant rather than watch for coincident Voronoi vertices or edges of zero length between Voronoi vertices.

The changes to the priority queue $\mathcal{Q}$ also resolves the degeneracy in which a circle event and a site event occur at the same position. The priority queue schedules the site event first and an additional circle event at the same position as the site event will be added to $\mathcal{Q}$. All the circle events at the same position as the site event are grouped within $\mathcal{Q}$ and are handled as one group by our changes to the circle event processing.

3.3 A randomized incremental construction algorithm

In the introduction, we noted three techniques commonly used to compute Voronoi diagrams: divide-and-conquer algorithms, sweep algorithms, and randomized incremental constructions. Section 3 elaborated on a sweep algorithm to compute our compact Voronoi diagram; a divide-and-conquer approach for this same diagram computes $O(k \log k)$ Voronoi vertices and therefore cannot match the time complexity of the sweep algorithm. In this section, we detail a change to the randomized incremental construction (RIC) of Boissonnat et al. [3] or the abstract Voronoi RIC of Klein et al. [23] to compute our compact Voronoi diagram in an expected time that matches the deterministic time of section 3.1's sweep algorithm. Throughout this section we assume that all polygons are in general position, just as we did for the deterministic algorithm.

The RIC technique common to both Klein et al. and Boissonnat et al. hinges on the concept of a *conflict* between a polygon and a region. In the earlier RICs, conflicts are often considered between a polygon and a Voronoi cell. For the compact Voronoi, we consider conflicts between a polygon and spoke regions.

If $\mathcal{P}$ is a set of convex and disjoint polygons, recall that $V(\mathcal{P})$ is the Voronoi diagram of the polygons in $\mathcal{P}$, and $C_{\mathcal{P}}(Q)$ is the Voronoi cell for $Q \in \mathcal{P}$ in $V(\mathcal{P})$. A polygon $R \notin \mathcal{P}$ is said to be in conflict with a point p in the plane if $p \in C_{\mathcal{P} \cup \{R\}}(R)$. By extension, we say that polygon R conflicts with a spoke region A of $V(\mathcal{P})$ if R conflicts with some point of A.

The idea of the RIC is to introduce sites one at a time into the Voronoi diagram. The compact Voronoi diagram is maintained for those sites already inserted. When we add a new site, we try to modify the existing diagram rather than re-compute the diagram from scratch.

Two data structure have been used to support the insertion of a new site. The first data structure is a conflict graph [10, 25] that stores the conflict relation between those sites not yet in the incremental diagram and the regions of the incremental diagram. The second data structure is a conflict history DAG (directed acyclic graph) [5, 6, 4, 12]. We use the history DAG since it is considered the more dynamic of the two approaches.

Each node of the DAG represents a spoke region that has been computed by the algorithm. If a node for spoke region X has children nodes for spoke regions $Y_1, \ldots, Y_j$, then X was divided among regions $Y_1, \ldots, Y_j$ at some point in the algorithm's progress, i.e. $X \subset \bigcup_{1 \le i \le j} Y_i$ and $X \cap Y_i \neq \phi$ for all i, $1 \le i \le j$. Leaf nodes in the DAG represent the regions in the current state of the algorithm.

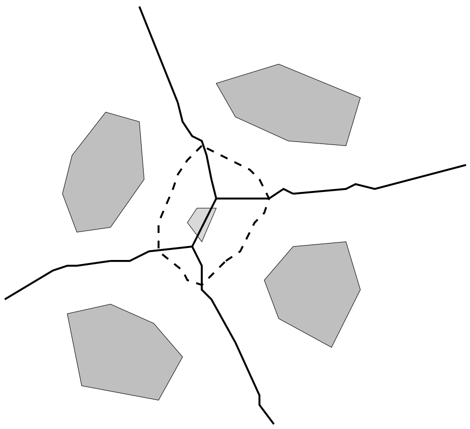

Figure 12: A polygon inserted into a set of Voronoi cells.

The standard RIC [3] uses Voronoi cells rather than spoke regions as the nodes in the conflict history DAG. The algorithm begins with the Voronoi diagram for a fixed number of sites: for instance, it can begin with a single site where the DAG consists of a single node representing the entire plane. We call this first node of the DAG the root of the DAG. To add a site P to the Voronoi diagram of a set of sites $\mathcal{P}$, the algorithm must identify the Voronoi cells in $V(\mathcal{P})$ with which P conflicts and partition these cells to make up $C_{\mathcal{P} \cup \{P\}}(P)$. It identifies the Voronoi cells with which P conflicts by traversing the conflict history DAG from the root. Polygon P conflicts with a region R only if P conflicts with at least one parent of R in the DAG. Consequently, we can trace a set of conflicts through the DAG from the root to the current Voronoi cells. The process ends by dividing the

Voronoi cells with which P conflicts, thus creating children in the DAG.

The expected running time for this RIC is $O(k \log k)$ for k point sites [3, 23]. Briefly, the algorithm is expected to compute $O(k)$ Voronoi vertices as it adds the k sites. However, we expect to encounter $O(\log i)$ conflicts traverse the conflict history DAG when i sites have been added. Consequently, we expect to compute $O(k \log k)$ conflicts.

When the sites for the Voronoi diagram are polygons rather than points, the expected number of Voronoi vertices and conflicts calculated by the RIC remains the same, within the big-O notation, but the cost of computing both Voronoi vertices and conflicts themselves increases from constant time to $O(T_v)$-time, yielding an expected total complexity of $O(kT_v \log k)$ as noted in [3], which is worse than our $O(k(T_v + \log k))$-time deterministic algorithm in the case of polygons.

Two modifications to the RIC allow us to improve its expected time complexity for polygonal sites when computing the compact Voronoi. First, we use the spoke regions in the conflict history DAG rather than the Voronoi cells themselves. We also approximate the spoke regions by joining the attachment points within each polygon, as we did for point location in the post-office problem of section 2.2; each region of the DAG becomes a hexagon. These regions are much less complex than the Voronoi cells of polygons, yet we still have only $O(k)$ of them.

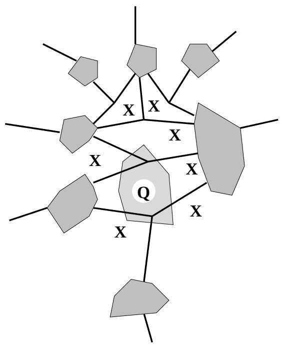

Figure 13: Polygon Q conflicts with the spoke regions marked **X**.

Second, we observe that a polygon conflicts with a set of spoke regions whose underlying Voronoi edges form a tree. Consequently, a polygon Q conflicts with a set of spoke regions whose union is a connected set. Provided that we can locate one spoke region in conflict with Q efficiently, we can find all other conflicting spoke regions with traversal techniques across this connected region. A first spoke region with which Q conflicts can be found by tracing a single point of Q through the conflict history DAG.

We now apply these two changes to the standard RIC [3]. We begin with the spoke regions for a fixed number of polygons in our conflict history DAG. To add the polygon Q to a set of polygons $\mathcal{P}$ we find those spoke regions of $V(\mathcal{P})$ with which Q conflicts. Since each vertex of Q conflicts with some spoke region of $V(\mathcal{P})$, we trace the leftmost vertex of Q as a single point through the conflict history DAG and obtain one spoke region that conflicts with Q and place this spoke region in a set T. If the Voronoi cell for Q in $V(\mathcal{P} \cup \{Q\})$ extents across one of the bounding spokes for the spoke region in T then Q conflicts with the Voronoi vertex v that defines the bounding spoke. The point v cannot remain a Voronoi vertex since the spoke from v to one of its defining sites crosses the Voronoi cell of Q and contradicts the star-shaped property of Voronoi cells (corollary 2.5). The algorithm extends the set T by repeatedly applying this fact: if A is a spoke region already in T with a Voronoi vertex v then if Q conflicts with v we add the spoke regions incident with v to T.

How can we be sure that this traversal finds all the spoke regions that Q conflicts with? Let S be the set of spoke regions with which Q conflicts. Since the skeleton of T must form a connected graph [23], If Q conflicts with two spoke regions in S then the Voronoi cell for Q in $V(\mathcal{P} \cup \{Q\})$

crosses a spoke boundary of each spoke region and, as previously discussed, Q must conflict with a Voronoi vertex for each region. Exploring the Voronoi vertices is therefore sufficient to find all the spoke regions with which Q conflicts.

The final task of the RIC is to update the conflict history DAG by partitioning each spoke region with which Q conflicts into new spoke regions. Let Q conflict with spoke region A, let u and v be the Voronoi vertices that bound A, and let R and S be the polygons that bound A. If Q conflicts with both u and v (figure 14a) then $C_{\mathcal{P}\cup\{Q\}}(Q)$ cuts completely through A, so A will be replaced in the DAG by two spoke regions that include portions of the neighbouring spoke regions of A (not necessarily just immediate neighbours of A). If Q conflicts with neither u nor v (figure 14b) then $C_{\mathcal{P}\cup\{Q\}}(Q)$ is contained entirely within A. We compute the two Voronoi vertices w and x between Q, R, and S and partition A into four new spoke regions. Finally, assume Q conflicts with v but not with vertex u (figure 14c). Then we compute the Voronoi vertex of Q, R, and S that lies inside A and split A into three spoke regions, two of which merge with the spoke regions that neighbour A at the vertex v (and possibly more spoke regions beyond those immediate neighbours).

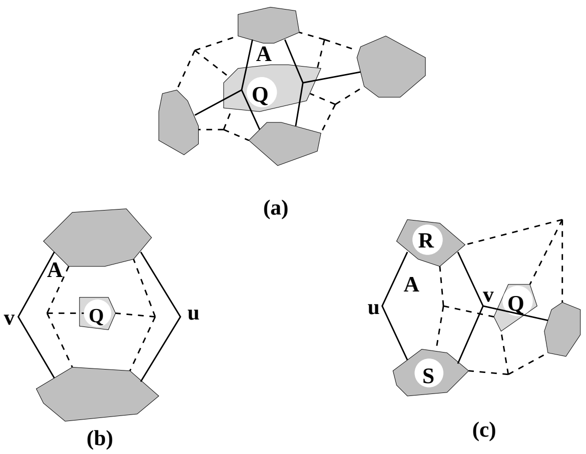

Figure 14: The three possible ways for polygon Q to conflict with spoke region A. Dotted lines indicate the new spoke regions after Q has been inserted.

Theorem 3.11 *The expected running time of the RIC on a set of polygons is* $O(k(\log k + T_v + T_s))$ *where* T_v *is the time required to compute a Voronoi vertex and* T_s *is the time required to compute a spoke from a point to a polygon.*

Proof: The expected-time analysis of the RIC attributes the cost of the algorithm to two sources: polygons and Voronoi vertices. The analysis attributes a cost to a polygon at the time it is inserted into the diagram and attributes a cost to Voronoi vertices when the vertex is created and when it is destroyed. Each of these events occurs only once per site or Voronoi vertex so the sum of the allocated costs is the expected time of the algorithm.

We start with the costs that are attributed to the polygon. The first stage of the RIC algorithm finds one spoke region that conflicts with the new polygon Q. The algorithm traces a single point through the conflict history DAG, so conflict calculations with a spoke region take constant time. Klein, Mehlhorn, and Meiser [23] show that a single point is expected to encounter $O(\log i)$ conflicts for the i^{th} site, so we obtain our first conflicting spoke region A in expected $O(\log k)$-time and add A to a list T of spoke regions with which Q conflicts. Next, we test whether Q conflicts with the bounding Voronoi vertices of A by calculating the distance from each vertex to Q (with a call to the `spoke()` subroutine) in $O(T_s)$ time. For those vertices where Q conflicts, we add the incident spoke regions to T. A total cost of $O(\log k + T_s)$ is attributed to the polygon Q.

The remaining costs of the algorithm, expanding the set T of spoke regions with which Q conflicts and updating the DAG, are attributed to the Voronoi vertices. Let v be a Voronoi vertex of a spoke region in T where v has not been tested for a conflict with Q yet. The cost of extending T (or not extending) through the spoke regions incident to v comes from calculating the spoke from v to Q. This $O(T_s)$ cost is attributed to the other vertex of the spoke region. A spoke region is added to T only when we have tested one of its Voronoi vertices and that vertex vertex is to be removed. This completes the costs of finding the tree T for one polygon Q.

When we partition the conflicting spoke regions to update the history DAG, the $O(T_v)$ cost of computing new Voronoi vertices is associated with the vertex created, and the cost of adding a spoke region to the conflict history DAG is allocated to the Voronoi vertex of the new spoke region that was created with the insertion of Q.

Since each Voronoi vertex with which Q conflicts is deleted when we compute new spoke regions, each Voronoi vertex has degree three, and each Voronoi vertex is "new" only once, each Voronoi vertex receives a total cost of $O(T_s + T_v)$.

In summary, as with Klein, Mehlhorn, and Meiser [23], the algorithm expects to compute $O(k)$ Voronoi vertices. These vertices receive a cost of $O(T_s + T_v)$ each while each of the k polygons receive an expected cost of $O(\log k + T_s)$. This gives a total expected time for the RIC algorithm of $O(k(\log k + T_v + T_s))$. ■

3.4 Implementing the subroutines

What remains is to implement the subroutines used in previous sections. We assume that the vertices of the convex polygons involved are given in an array or balanced binary trees in the order that they appear around the polygons. We also assume that each vertex knows (or can compute in constant time) a line tangent to the polygon at that vertex. Since such representations allow access to a "midpoint" of a chain in constant time, it is common and useful to view them as giving hierarchical decompositions of the polygons into triangles [13, 20].

We consider the subroutines in order of difficulty for the Euclidean metric and for the distance function defined by a convex m-gon M.

`spoke`(p, A) Given a point p and a convex polygon A, compute the spoke, which is the shortest segment joining p to A. This is used to answer a post-office query in section 2.2—it takes $O(\log n)$ time under the Euclidean metric and $O(\log n + \log m)$ time under the distance function d_M.

`bottleneck`(R) Given a spoke region R, incident to two convex polygons A and B, compute the bottleneck segment used by the retraction motion planning diagram in section 2.3. This takes $O(\log n)$ time under the Euclidean metric and $O(\log n \log m)$ time under d_M.

`vertex`(ABC) Given three sites, compute the Voronoi vertex where the cells of A, B and C occur in counterclockwise (ccw) order. Return the spokes to the three sites and the location of the circle event—the rightmost point of the homothet of M that is tangent to A, B and C in ccw order. This Voronoi vertex may be at infinity, in which case infinite spokes and an infinite circle event are returned. This subroutine is heavily used in the construction algorithms of section 3.1. It also takes $O(\log n)$ time under the Euclidean metric and $O(\log n \log m)$ time under d_M.

Since each of these subroutines depends only on the constant number of sites that it receives as arguments, the variable n in this section could be replaced by $n_{\max}$, the maximum number of vertices on a polygon.

Some of our routines make use of Kirkpatrick and Snoeyink's tentative prune-and-search technique [20].

Theorem 3.12 (From [20]) *Given f, g and h, which are continuous, monotone decreasing real functions whose domains are partitions of the real line into k intervals, we can determine an interval containing the fixed-point of the composition $h \circ g \circ f$ using $\Theta(\log k)$ tests of the form "is $f(a) < b$?"*

This theorem is proved by inspecting a candidate triple of reals, one from each domain, and using local information to discard half of one of the domains. For cases in which local information is insufficient, portions of the domain can be "tentatively" discarded with the assurance that the algorithm does some correct work on every third step. An easy proof by potential function is in [19] and the full version of [20].

We can use standard prune-and-search to compute spokes.

Lemma 3.13 `spoke`(p, A) *can be computed in $O(\log n)$ time under the Euclidean metric and $O(\log n + \log m)$ time under the distance function d_M.*

Proof: Under the Euclidean metric, the spoke is a normal to A that passes through p. This can easily be found by binary search among the slopes of tangents to A.

Under the convex distance function d_M, the attachment point is a point where M_p^A contacts A. (Recall that M_p^A denotes a homothet of M that has been scaled by $d_M(p, A)$ and translated to p.) The set A and M_p^A share a common tangent at their point of tangency, so we find the attachment point of `spoke`(p, A) by locating points q and a on M_p^A and A respectively that share parallel tangents, and the ray from p through the point q intersects A at the point a. By precomputing outer tangents between M_p^A and A, we restrict our search to two polygonal chains that share a single pair of parallel tangents that also satisfy the ray intersection property.

We select point a as the middle of the chain A and point q as the middle of the chain M_p^A. We assume that the tangent to A at a and the tangent to M at q intersect on the right side of the line from a to q; the opposite configuration is handled with a symmetric argument. Let a' be the intersection point of the ray from p to q with A. The tangent to A at a' and the tangent to M at q also intersect (otherwise, we are done). We want to eliminate part of A whose tangents always intersect the tangent at q or eliminate part of M_p^A whose tangents always intersect the tangent at a. If a' is clockwise (cw) of a (figure 15a), then the ray from the centre of M to any point counterclockwise (ccw) of q will touch A cw of a' and the tangent to M parallel to the tangent to A this new contact point lies above q. Consequently, we can discard the half of M_p^A that lies ccw of q.

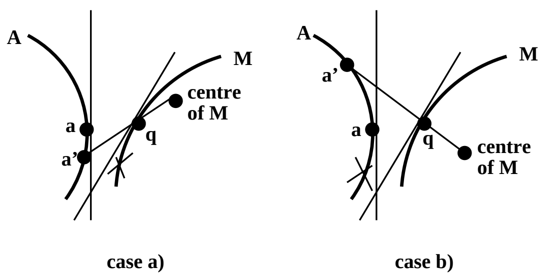

Figure 15: Half of either chain A or chain M can be discarded.

Otherwise, point a' is ccw of a (figure 15b). For any point a'' cw of a on A, the point on M whose tangent is parallel to the tangent at a'' lies above the point q and the ray from the centre of M through this point intersects A ccw of the point a'. We can discard the points of A that are cw of a. We can make at most $O(\log n + \log m)$ discards.

To test if a is cw or ccw of the point a', we test if a is left (cw) or right (ccw) of the line from p to q, without computing the point a. All the tests for the discards can therefore be done in constant time, so the search terminates after $O(\log n + \log m)$ steps. ∎

The bottleneck segment of a spoke region R bounded by portions of convex polygons A and B can be determined once a point on the AB-bisector with minimum distance to A and B is known. (Refer back to figures 2 and 5.)

Lemma 3.14 *The smallest homothet of M that touches convex polygons A and B can be computed in $O(\log n)$ time when M is a circle and in $O(\log n \log m)$ time when M is an convex m-gon.*

Proof: Edelsbrunner [13] has shown how to compute the Euclidean shortest segment joining A and B, which solves the problem when M is a circle.

When M is a convex polygon, the smallest homothet of M touches A and B at points with parallel tangents. We restrict the search on A and B to the portions between the outer common tangents. Inspect vertices with median indices, $a \in A$ and $b \in B$, and their tangents τ_a and τ_b. Place a homothet of M tangent to τ_a at a and tangent to τ_b by locating these tangencies in $O(\log m)$ time. If M touches τ_b between $\tau_a \cap \tau_b$ and b, discard the half of B away from $\tau_a \cap \tau_b$; otherwise discard the half of A. ∎

If the $O(m)$ preprocessing on M is allowed, then we can compute the desired homothets in $O(\log n + \log m)$ additional time by tentative prune-and-search. In preprocessing, we merge the slopes of tangents of M so that knowing one tangent of a given slope we can obtain the other in constant time. Then we parameterize A and B counterclockwise and M clockwise. Then we define functions $f\colon A \to B$ and $g\colon B \to M$ so that the slope of tangents at a and $f(a)$ and at b and $g(b)$ are the same, and we define $h\colon M \to A$ to send $m \in M$ to the point on A that intersects the line through m and the point $m' \in M$ with parallel tangent. A fixed point of the composition $h \circ g \circ f$ gives the smallest homothet. Note that we can determine in constant time, for given points $a \in A$, $b \in B$, and $m \in M$, whether $f(a) < b$, $g(b) < m$ and $h(m) < a$ by geometric tests. With $O(m)$ preprocessing of M alone, we can compute a fixed point in $O(\log n + \log m)$ time by theorem 3.12.

Computation of the Voronoi vertices is the most involved because one must deal with finite and infinite vertices and degenerate cases.

Lemma 3.15 `vertex`(ABC) *can be computed in $O(\log n)$ time under the Euclidean metric and $O(\log n \log m)$ time under the distance function d_M.*

Proof: We begin with the Euclidean case. Compute outer common tangents from A clockwise (cw) to B and from B cw to C. If a portion of B appears cw between these tangents, then the desired Voronoi vertex is infinite and any normal to this exposed portion of B can be returned as a spoke. Otherwise, the vertex is finite. We clip A, B and C at their points of tangency and consider only the polygonal chains that can be attachment points for spokes to `vertex`(ABC). We parameterize these chains ccw and define functions of the form $f\colon A \to B$ maps $a \in A$ to the intersection of the normal to A at a with B. Again, theorem 3.12 allows us to compute a fixed point and Voronoi vertex in $O(\log n)$ time. See Kirkpatrick and Snoeyink [20] for more detail.

For the convex distance function d_M, we start with the same tangents, for example the tangent from A cw to B. We compute where M can contact this tangent when M is separated from A and B. Typically, this contact will be a vertex v of M; then we compute the tangents on A and B that are parallel to the edges cw and ccw of v, respectively. These tangencies determine the attachment points of the infinite spokes for the infinite endpoint of the AB-bisector. The direction of the endpoint and the spokes can be determined by placing a homothet of M so these two segments touch A and B and drawing the ray from v through the translated origin of the homothet.

Again, if a portion of B appears cw between the attachment points on B, then the Voronoi vertex is infinite. Otherwise, we use the algorithm for the Euclidean case, with the modification that instead of defining $f(a)$ in terms of the normal at a, we determine a placement of M that is tangent to A at a and using the line through a and the reference point of M in place of the normal. This line can be computed in $O(\log m)$ time, so theorem 3.12 gives us an $O(\log n \log m)$ time computation of the Voronoi vertex. ∎

3.5 Precise upper and lower bounds

The $O(k \log n)$ and $O(k \log n \log m)$ time bounds on the computation of the compact diagrams under the Euclidean and d_M distance functions, respectively, can be more accurately stated as $O(k(\log k + \log n_{\max}))$ and $O(k(\log k + \log n_{\max} \log m))$, where $n_{\max}$ denotes the maximum number of vertices on a convex polygonal site. We can prove a matching lower bound to compute and verify the Voronoi diagram under the Euclidean metric; for d_M we can prove a lower bound of the form $\Omega(k(\log k + \log n_{\max} + \log m))$.

Theorem 3.16 *To compute the compact diagram under the Euclidean metric requires $\Omega(k(\log k + \log n_{\max}))$ operations; under d_M it requires $\Omega(k(\log k + \log n_{\max} + \log m))$ operations.*

Proof: Because one can recover the true Voronoi diagram from the compact diagram, the $\Omega(k \log k)$ lower bound for the Voronoi diagram of points applies when n and m are small.

If the Voronoi vertices are given with the names of the objects that their spokes attach to, then the actual attachment points must be found by binary search. Since the average number of spokes incident to a site is small, the cost of locating the attachment points on the half of the polygons that have at most six attachment points gives the rest of the lower bound. ∎

It would be interesting to close the gap between the upper and lower complexity bounds for the convex distance function d_M.

4 Conclusions and Open Problems

We have given a piecewise-linear representation of the generalized Voronoi diagram of convex sites in the plane that depends on the number of sites k and not on their complexity or on the complexity of the distance function. We also compute the diagram by a general algorithm, where the dependence on site and distance function complexity is restricted to subroutines that are called $O(k)$ times. We have begun implementation for the Euclidean metric and polygonal distance functions. Extensions to sites and distance functions bounded by arcs or splines could be achieved by implementing subroutines that compute tangents and Voronoi vertices.

Of greatest interest is the extension of these diagrams to higher dimensions, analogous to the work of de Berg et al. [11]. The efficient subroutines for computing Voronoi vertices and bottleneck segments will not extend because there may be many local minima and maxima along a curve equidistant from three objects. However, we believe that a piecewise-linear retraction diagram can be identified that does not depend on a polyhedron of a fixed scale.

Acknowledgments

We thank Stephan Meiser for discussions on the randomized incremental construction of our compact diagrams.

References

[1] H. Alt and C. K. Yap. Algorithmic aspects of motion planning: a tutorial, part 2. *Alg. Rev.*, 1(2):61–77, 1990.

[2] F. Aurenhammer. Voronoi diagrams—A survey of a fundamental geometric data structure. *ACM Comp. Surveys*, 23(3):345–405, 1991.

[3] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8:51–71, 1992.

[4] J.-D. Boissonnat, O. Devillers, and M. Teillaud. A dynamic construction of higher-order Voronoi diagrams and its randomized analysis. Report 1207, INRIA Sophia-Antipolis, Valbonne, France, 1990.

[5] J.-D. Boissonnat and M. Teillaud. A hierarchical representation of objects: the Delaunay tree. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 260–268, 1986.

[6] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. Report 1140, INRIA Sophia-Antipolis, Valbonne, France, 1989.

[7] J. Canny and B. Donald. Simplified Voronoi diagrams. *Disc. & Comp. Geom.*, 3:219–236, 1988.

[8] J. W. S. Cassels. *An Introduction to the Geometry of Numbers.* Springer-Verlag, Berlin, 1959.

[9] L. P. Chew and R. L. Drysdale. Voronoi diagrams based on convex distance functions. In *Proc. ACM Symp. Comp. Geom.*, pages 235–244, 1985.

[10] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[11] M. de Berg, J. Matoušek, and O. Schwarzkopf. Piecewise linear paths among convex obstacles. In *Proc. 25th Ann. ACM STOC*, pages 505–514, 1993.

[12] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Comput. Geom. Theory Appl.*, 2(2):55–80, 1992.

[13] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *J. Alg.*, 6:213–224, 1985.

[14] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comp.*, 15:317–340, 1986.

[15] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[16] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[17] T. C. Kao and D. M. Mount. An algorithm for computing compacted Voronoi diagrams defined by convex distance functions. In *Proc. Third Can. Conf. Comp. Geom.*, pages 104–109, Simon Fraser University, Vancouver, 1991.

[18] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comp.*, 12:28–35, 1983.

[19] D. Kirkpatrick and J. Snoeyink. Computing constrained segments: Butterfly wingspans in logarithmic time. In *Proc. Fifth Can. Conf. Comp. Geom.*, pages 163–168, Waterloo, Canada, 1993.

[20] D. Kirkpatrick and J. Snoeyink. Tentative prune-and-search for computing Voronoi vertices. In *Proc. 9th Ann. ACM Symp. Comp. Geom.*, pages 133–142, 1993.

[21] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proc. 18th FOCS*, pages 162–170, 1977.

[22] R. Klein. *Concrete and Abstract Voronoi Diagrams.* Number 400 in LNCS. Springer-Verlag, 1989.

[23] R. Klein, K. Mehlhorn, and S. Meiser. On the construction of abstract Voronoi diagrams, II. In *Algorithms: International Symposium Sigal 90*, number 450 in LNCS, pages 138–151, 1990.

[24] D. Leven and M. Sharir. Planning a purely translational motion for a convex polygonal object in two dimensional space using generalized Voronoi diagrams. *Disc. & Comp. Geom.*, 2:9–31, 1987.

[25] K. Mehlhorn, S. Meiser, and C. Ó'Dúnlaing. On the construction of abstract Voronoi diagrams. Report A01/89, Fachber. Inform., Univ. Saarlandes, Saarbrücken, West Germany, 1989.

[26] K. Mehlhorn, C. O'Dunlaing, and S. Meiser. Minimum vertex hulls for polyhedral domains. In *STACS 90: 7th Annual Symposium on Theoretical Aspects of Computer Science*, number 415 in LNCS, pages 126–137. Springer-Verlag, 1990.

[27] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, N.J., 1993.

[28] C. O'Dunlaing and C. K. Yap. A "retraction" method for planning the motion of a disc. *J. Alg.*, 6:104–111, 1985.

[29] F. P. Preparata and M. I. Shamos. *Computational Geometry—An Introduction.* Springer-Verlag, New York, 1985.

[30] H. Rohnert. Moving a disc between polygons. *Algorithmica*, 6:182–191, 1991.

[31] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *CACM*, 29:669–679, 1986.

[32] J. T. Schwartz, M. Sharir, and J. Hopcroft, editors. *Planning, Geometry, and Complexity of Robot Motion.* Ablex Series in Artificial Intelligence. Ablex, Norwood, New Jersey, 1987.

[33] M. I. Shamos and D. Hoey. Closest point problems. In *Proc. 16th FOCS*, pages 151–162, 1975.

[34] S. Sifrony. A real nearly linear algorithm for translating a convex polygon. Technical Report 479, NYU Courant Inst. of Math. Sci., 1989.

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