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 $\Theta(k \log n)$ 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, 34]. Aurenhammer [3] 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.

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

1.1 The post-office problem and retraction motion planning

The post-office problem [34] takes a set of k sites in the plane and asks for a data structure, based on these sites, suitable for efficiently determining the closest site to an arbitrary query point. When the sites are disjoint convex polygons with n total vertices, the Voronoi diagram has k faces bounded by $O(n)$ segments of lines and parabolas [22, 25, 37]. Combined with a data structure for point location [15, 19, 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 [2, 28, 32] uses the Voronoi diagram of 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 each edge of the Voronoi diagram is equidistant from its 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 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. Suppose that we have k polygonal sites with a total of n vertices (we say the sites have total complexity n). The edges of the Voronoi diagram consist of $O(n)$ segments of lines and parabolas. Answering a post-office query then involves searching through parabolic and straight-line segments to find the Voronoi cell that contains the query point. Retraction motion planning paths on the Voronoi diagram ask a robot to traverse pieces of line segments and parabolas. 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.

This paper describes a compact approximation of the Voronoi diagram when the k sites are disjoint convex polygons with n total vertices. The compact diagram

is composed of $O(k)$ line segments and is sufficient to solve the post-office problem 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.

The compact diagram can also represent the generalized Voronoi diagram defined by a convex distance function [11]. 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 derived 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, the compact diagram is easier to compute, display, and traverse than the Voronoi diagram. This is an advantage even if the k sites are line segments.

The remainder of this section compares our diagram to related work on the definition and construction of (compact, generalized, and abstract) Voronoi diagrams. For more detail, see Aurenhammer's survey [3]. 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 [16] 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 [9] 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 [18] consider the generalized Voronoi diagram using a distance function defined by a convex polygon with m sides. Even though the Voronoi 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 paths of $\Theta(n + km)$ complexity.

Sifrony [35] 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 [12], 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

Those who are familiar with Klein's monograph [23] will see our compact diagram 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 [21] gives an $O(\log n)$-time subroutine for the Voronoi vertex problem, which asks to find points equidistant from three sites, under the Euclidean metric. In section 3.5, we obtain $O(\log n \log m)$ subroutines under a convex distance function defined by an m-gon for the Voronoi vertex problem and for finding the nearest point on a polygon to a query point.

There are three algorithmic paradigms that give optimal $\Theta(k \log k)$ algorithms for the Voronoi diagram of k point sites: divide and conquer [34], randomized incremental construction [17, 27], and sweepline [16]; the first two have been adapted to compute abstract Voronoi diagrams [23, 24, 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 solves as many as $\Theta(k \log k)$ instances of the Voronoi vertex problem; in Klein's abstract setting, each such instance requires a call to a primitive. 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 [24, 26] solves an expected $O(k)$ instances of the Voronoi vertex problem, but evaluates an expected $\Theta(k \log k)$ "conflicts." In our case, a conflict involves a "spoke region" (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 3.4 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 calls to primitives for each Voronoi vertex created. Fortune's [16] 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. (As noted by Siedel [33], the "parabolic front" is simply the boundary of the Voronoi cell of the sweep line.)

2 Definition of the compact diagram

We define the 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 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.

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

Distance function d_M is not necessarily a metric: $d_M(p,q)$ need not equal $d_M(q,p)$ if M is not centrally symmetric. It does, however, satisfy the triangle inequality for points [10]: $d_M(p,q) + d_M(q,r) \geq d_M(p,r)$.

In this paper, the sets M, A and B are always closed subsets of the Euclidean plane E^2 and M is bounded so the infimum operations could be replaced by minimum operations. The set M contains a neighbourhood around the origin so that the distance between any two points is finite.

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, $M_p^A = d_M(p,A)M + p$.

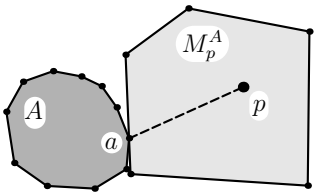

Figure 1: M_p^A and $\texttt{spoke}(p,A)$

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 $a' \in \text{int}(A) \cap \text{int}(M_p^A)$. 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 segment $\overline{pa}$ 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.

Definition 1 *Let* $\texttt{spoke}(p, A)$ *be the unique Euclidean shortest finite spoke defined by* p *and* A.

We can also define *infinite spokes* as the infinite rays composed of all points p for which $\texttt{spoke}(p, A)$ has the same attachment point and direction.

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

When d_M is the Euclidean distance, the *bisector* of two closed convex sets A and B is defined as $\{p : d_M(p, A) = d_M(p, B)\}$ (figure 2a). The shaded region in figure 2b illustrates that the bisector is not always a curve under this definition for arbitrary convex distance functions in the plane. Specifically, when a boundary segment of M_p^A at some point p 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.

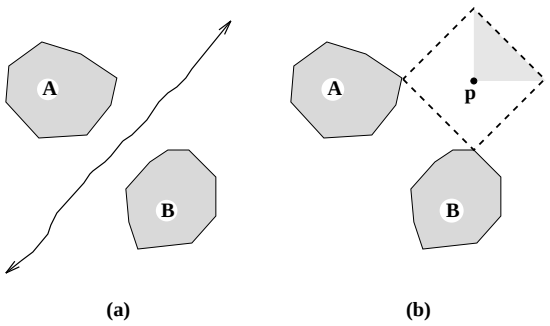

Figure 2: Euclidean bisector (a) (drawn solid) and Manhattan region of bisectors (b) (shaded).

We therefore base the definition of a bisector on an oriented version of the convex 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.

Definition 3 *Under this orientation of* M, *the* AB-bisector *is the set of points* p *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 3. 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, in lemma 2.3 and corollary 2.4, that our definition of an AB-bisector is a curve that separates the plane into two star-shaped regions.

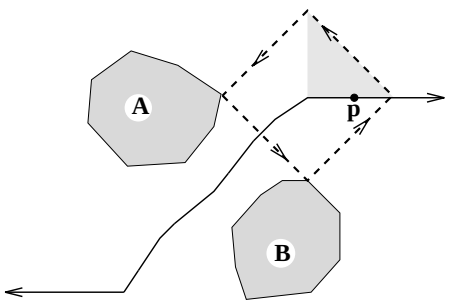

Figure 3: Manhattan bisector, drawn solid.

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: Both $\texttt{spoke}(p, A)$ and $\texttt{spoke}(q, A)$ share a common attachment point on A. Since M_p^A and M_q^A are both homothets of M, it follows that $M_q^A \subset M_p^A$. ∎

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

Proof: Each point p on the bisector can be parameterized by the attachment point of $\texttt{spoke}(p, A)$, breaking ties with the angle and length of $\texttt{spoke}(p, A)$. 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 $\texttt{spoke}(p, A)$ either changes monotonically and continuously or remains fixed (when the bisector moves in the direction of $\texttt{spoke}(p, A)$) in which case the length of $\texttt{spoke}(p, A)$ changes monotonically. ∎

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

Proof: The AB-bisector divides the plane into two sets since it is continuous (lemma 2.3). The star-shaped property of each set follows from lemma 2.2. ∎

Let $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ be a collection of *sites*—convex sets in the plane with disjoint interiors. Let $V(\mathcal{A})$ be their Voronoi diagram. The *Voronoi cell* of A_i is

$\bigcap_{i \neq j, 1 \leq j \leq k}$\{the A_i side of the A_iA_j-bisector\}. In the Euclidean metric where M is a circle, this definition is equivalent to $\{p : d_M(p, A_i) < d_M(p, A_j)$ for all $j \neq i\}$; that is, all points for which A_i is the unique closest site.

Corollary 2.5 *The Voronoi cell of A_i in $V(\mathcal{A})$ 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, all of which are star-shaped with respect to 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 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 Voronoi cell of A_i in $V(\mathcal{A})$, 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 A_i's Voronoi cell 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 Voronoi cell of 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.

Definition 4 *The* core *of polygon A_i is the convex hull (and its interior) of the spoke attachment points around A_i.*

Replace each polygon A_i by its *core*. The union of the spokes and cores forms the compact diagram (see figure 4).

Definition 5 *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.*

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

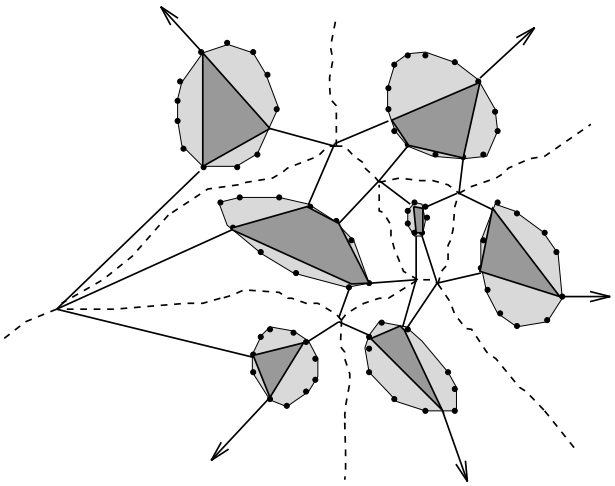

Figure 4: A compact diagram for point location

Proof: Points in the core of A_i are within 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 two 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 Voronoi 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. Let $a_1 \ldots a_k$ be representative points in the interior of $A_1 \ldots A_k$ respectively. We can form a plane graph on $\{a_1, a_2, \ldots, a_k\}$ by connecting a_i with a_j, whenever A_i and A_j have a common bisector, using 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 [15, 19, 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

finding spokes from q to each of the candidates. Section 3.5 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 compact diagram for retraction motion planning:

Lemma 2.8 *For convex sets A and B, the function $d_M(p, A)$, where point p is restricted to the AB-bisector, has no local maxima.*

Proof: Let r and s be distinct points along the AB-bisector and suppose that $d_M(s, A) \leq d_M(r, A)$. Let r_A and r_B be the attachment points of $\texttt{spoke}(r, A)$ and $\texttt{spoke}(r, B)$ respectively; define s_A and s_B similarly (see figure 5).

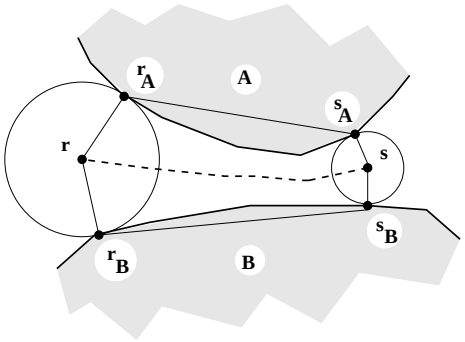

Figure 5: The AB-bisector is limited by the maximum distance of r and s to A and B.

For any point p in the hexagon formed by r_A, s_A, s, s_B, r_B, and r we have the inequality $\min\{d_M(p, \overline{r_A s_A}), d_M(p, \overline{r_B s_B})\} \leq d_M(r, A)$. Since segments $\overline{r_A s_A}$ and $\overline{r_B s_B}$ are inside A and B respectively (by the convexity of A and B) it follows that $\min\{d_M(p, A), d_M(p, B)\} \leq \min\{d_M(p, \overline{r_A s_A}), d_M(p, \overline{r_B s_B})\}$. Hence, for points p on the AB-bisector between r and s we have $d_M(p, A) \leq d_M(r, A)$. ∎

For every contiguous section s of an AB-bisector there is a largest homothet of M that can translate along s without colliding with A or B. The minimum distance of s to either site determines the scale factor for this largest homothet. Since the AB-bisector has a single section of minimum distance, the minimum distance on s is attained either at an endpoint of s or at a point $p \in s$ 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 section s. 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. In the following definition, the entire Voronoi edge that traverses one spoke region is taken as the section s.

Definition 6 *A bottleneck segment for a spoke region R is a segment through a point p of minimum distance along the Voronoi edge through R that is parallel to τ (figure 6).*

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.

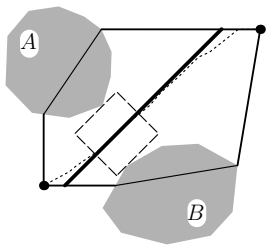

Figure 6: A bottleneck segment in a spoke region

In section 3.5 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.

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

Proof: Let R be a spoke region formed by sites 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 a 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 M can traverse R along the Voronoi edge. Then the homothet of M does not touch either of A or B at a minimum point p along the edge. We have two cases, depending on whether or not p is a minimum for the whole AB-bisector.

If 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 ($= M_p^A$). Choose the bottleneck segment that 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 the homothet can traverse the spoke region.

If p is not a minimum point for the AB-bisector, then we can choose p as a Voronoi vertex of R and choose the bottleneck segment for R that leaves p parallel to a tangent τ_A between A and M_p^A. A tangent τ_B between B and M_p^B diverges 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 to B; 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 for each spoke region a minimum point, its spokes, and a bottleneck segment then we can determine how to move onto the retraction diagram without using 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 to determine in logarithmic time the largest scale homothet that can traverse the tree 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

The key information for computing the compact Voronoi diagram is the set of Voronoi vertices for the sites. Given the Voronoi vertices and the sites that generate each vertex, we can find the spokes to the sites, identify the spoke regions (by sorting the Voronoi vertices around each site), and thus construct the compact diagram. Site adjacencies across spoke regions also define which bottleneck segments we must compute to solve retraction motion planning. Consequently, we concentrate on how to find efficiently the Voronoi vertices of polygonal sites under a convex distance function.

We use a variant of Fortune's sweep algorithm to locate all Voronoi vertices. Section 3.1 reviews Fortune's algorithm and provides the key proofs (lemmas 3.3 and 3.4) that the plane sweep locates all Voronoi vertices under convex distance

functions. The variant on the sweep algorithm appears in section 3.2 with details on handling degeneracies in section 3.3. A randomized incremental construction algorithm for locating Voronoi vertices appears in section 3.4. Finally, sections 3.5 and 3.6 describe low-level subroutines that are used in the previous sections and discuss lower bounds for computing the compact Voronoi diagram.

3.1 Fortune's plane sweep algorithm

Fortune's algorithm views the static task of locating all Voronoi vertices for a set of sites as a dynamic operation. It sweeps a vertical line, called the *sweepline*, from left to right across the Voronoi diagram of the sites and finds all Voronoi vertices. As the sweepline travels across the plane, the algorithm examines the Voronoi diagram defined by the sites (or parts of sites) to the left of (or on) the sweepline, together with the sweepline itself. It detects 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 sites—as well as all Voronoi vertices of the the Voronoi diagram for the sites—lie to the left of the sweepline, so the sweep detects all Voronoi vertices. A typical picture of the algorithm in mid-sweep on polygonal sites is shown in figure 7.

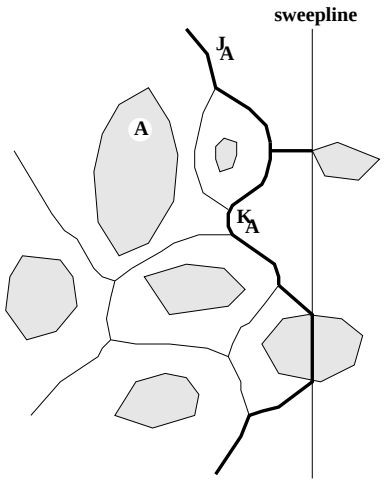

Figure 7: The sweep front (in bold) and Voronoi diagram left of the sweepline

Throughout this section, we assume that the sites are convex polygons 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. These restrictions are handled by a later section.

Definition 7 *The boundary of the sweepline's Voronoi cell is called the* sweep front*; it consists of Voronoi edges between the sweepline and some sites.*

Definition 8 *A maximal connected portion J of the sweep front between the sweepline and a single site A is denoted by J_A and called a* front arc.

Figure 7 illustrates that one polygon (polygon A) may have many front arcs associated with it at any one time in the sweep (arcs J_A and K_A). The sweep algorithm maintains the list of front arcs along the sweep front by handling "events" where front arcs are added or deleted to the sweep front.

Definition 9 *A* site event *occurs where the sweepline reaches a leftmost point of a site.*

Definition 10 *A* circle event *occurs where the sweepline reaches the rightmost point of M' where M' is a homothet of the convex set M that is tangent to three sites of consecutive front arcs on the sweep front.*

The sweep algorithm maintains two data structures: a balanced binary tree $\mathcal{T}$ that stores the sweep front and a priority queue $\mathcal{Q}$ that schedules changes in the sweep front in the form of site and circle events.

The balanced binary tree $\mathcal{T}$ stores the sequence of front arcs along the sweep front. The tree provides a logarithmic-time binary search through the sweep front and logarithmic-time location of adjacent front arcs. Each tree node corresponds to one front arc J_A and stores

- a pointer to the site A that generates J_A
- pointers to any events in the priority queue $\mathcal{Q}$ that J_A has generated
- a spoke that crosses J_A to site A.

Notice that the curves that make up the front arcs are not stored. Theorem 3.7 of section 3.2 will show how to search the sweep front without them.

The priority queue $\mathcal{Q}$ schedules events in the order that the sweepline will encounter them in its sweep across the plane, thus discretizing the continuous sweep. 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 their event point is finite and appears to the right of the sweepline's position at the time of insertion. Information stored in $\mathcal{Q}$ with each event includes

- pointers to the front arcs or sites that generate it
- for circle events, a Voronoi vertex of three sites
- for circle events, the attachment points of the spokes from the Voronoi vertex to each of the generating sites

As the sweep algorithm progresses, the algorithm maintains two invariants on the tree $\mathcal{T}$ and the priority queue $\mathcal{Q}$:

- The tree $\mathcal{T}$ always contains the ordered list of front arcs on the sweep front for the current position of the sweepline.
- The queue $\mathcal{Q}$ contains precisely one site event for each site that the sweepline has not encountered and one circle event for each set of three consecutive front arcs in the current sweep front when the event point is finite.

Before detailing the front arc and schedule changes in Fortune's sweep algorithm as events are handled, we must show that the basic principles of the algorithm hold under convex distance functions.

Lemma 3.1 *For the set of sites $\mathcal{P}$, 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 sites P, Q, and R, and let ℓ be the 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. ■

Lemma 3.2 *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 ℓ_2.*

Proof: We assume that some polygon exists to the left of sweepline ℓ_1. Otherwise, the sweep front for ℓ_1 is empty.

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 3.3 *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 site Q. The area between J_P and the part of K_Q that was broken when J_P appeared belongs to the Voronoi cell for P in $V(\mathcal{P})$ where $\mathcal{P}$ is the set of all sites. Since the Voronoi cell 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 the Voronoi cell of 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 the Voronoi cell of Q in $V(\mathcal{P})$. 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 of three polygonal sites 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 `spoke`(v, P) lies immediately counterclockwise from `spoke`(v, ℓ) and `spoke`(v, R) lies immediately clockwise from `spoke`(v, ℓ) (see figure 8).

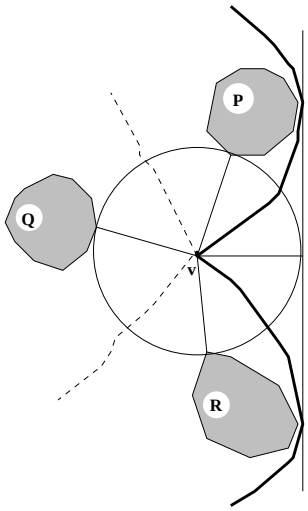

Figure 8: The front arc for Q is removed at v.

The Voronoi cell for Q does not extend beyond `spoke`(v, P) and `spoke`(v, R) since `spoke`(v, P) and `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 3.2 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.

Next, we show that a largest homothet of the convex distance function is tangent to three consecutive sites 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 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 3.2), though it must stay within the Voronoi cell for Q in $V(\mathcal{P})$. Since K_Q is removed from the sweep front whenever the sweepline moves to the right, the Voronoi cell of 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 9. The algorithm relies upon one subroutine, `vertex`(ABC):

Definition 11 *The subroutine* `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.*

For three convex sites, there exists up to two Voronoi vertices; `vertex`() distinguishes these vertices by the order of the incident Voronoi cells around the vertex.

In the case of finite Voronoi vertices, `vertex`(ABC) corresponds to the point p where $M_p^A = M_p^B = M_p^C$ and A, B, C appear in counterclockwise order around p. 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 3.5 discusses two implementation of this subroutine: one in $O(\log n)$ time under the Euclidean metric and the other in $O(\log n \log m)$ time under the distance function induced by an m-gon.

Given the characterizations of lemmas 3.3 and 3.4, Fortune's algorithm preserves the data structure invariants as each event of the priority queue $\mathcal{Q}$ is handled. The result is a complete list of the Voronoi vertices.

```
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 involving the arc J
    Split J into two front arcs
    Create a new front arc for p that appears between the fragments of J
    Schedule circle events for the new front arcs
  otherwise, p is a circle event
    Record the centre of p as a Voronoi vertex
    Delete the front arc K associated with the circle event
    Unschedule any circle events involving the arc K
    Schedule circle events coming from the disappearance of front arc K
  endif
endwhile
```

Figure 9: Outline of the sweepline algorithm.

The sweep algorithm begins by satisfying the data structure invariants at the sweepline's initial position. The site whose leftmost x-coordinate equals the leftmost x-coordinate of the entire set of sites generates one front arc in the sweep front $\mathcal{T}$. The algorithm adds one site event to the schedule for each site not represented in $\mathcal{T}$; no circle events appear in the initial schedule. The sweep now begins with the sweepline at the leftmost point of the set of sites.

When the sweepline encounters a site event at a point p, a new front arc J appears along the sweep front (lemma 3.3).

First, the algorithm re-establishes the invariant on the sweep front in $\mathcal{T}$. It searches the sweep front for the front arc K that is nearest to the point p (under the convex distance function). This is done by a binary search across the sweep front where the position of point p is compared with the endpoints of front arcs. (Section 3.2 provides an improvement on this search.) The new front arc J splits K into arcs K' (above J) and K'' (below J); arc J is added to $\mathcal{T}$ between the split arcs K' and K'' to satisfy the invariant on $\mathcal{T}$.

Second, the algorithm updates the schedule $\mathcal{Q}$ to reflect the changes in the

sweep front. The site event for p was removed from the schedule when the algorithm detected the event. The sweepline encounters 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. Only circle events may now violate the invariants. The single circle event defined when front arc K was the middle of three consecutive arcs must be removed from the schedule since that triple no longer exists in $\mathcal{T}$. If the sequence of front arcs was IKL before arc J divided arc K, three new sets of consecutive arcs replace the triple and must 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. The circle events where K was the first or third arc in the sequence remain unchanged in the schedule $\mathcal{Q}$ since arcs K' and K'' fulfill the role of K in those instances. The invariants are now satisfied and the next event from $\mathcal{Q}$ can be processed.

When the sweepline encounters a circle event, the front arc J corresponding to the event must be removed from the sweep front and a Voronoi vertex is detected (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: arc J is removed from the sweep front.

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. 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 the algorithm removes the circle events corresponding to $\texttt{vertex}(HIJ)$ and $\texttt{vertex}(JKL)$ from $\mathcal{Q}$ and replace them with the circle events for $\texttt{vertex}(HIK)$ and $\texttt{vertex}(IKL)$. The sweep front did not change anywhere else, so all other scheduled circle events remain valid.

When the sweepline reaches $x = +\infty$, all sites are to its left. The Voronoi vertex topology obtained for sites left of the sweepline is precisely the topology for the Voronoi diagram of all polygons. The treatment of site events and circle events by Fortune's algorithm, along with the characterizations of lemma 3.3 and lemma 3.4, culminate in the proof of theorem 3.5.

Theorem 3.5 *The sweepline algorithm (outlined in figure 9) correctly finds all Voronoi vertices and their order about each polygon for a convex distance function.*

3.2 Improving Fortune's algorithm for polygonal sites

When Fortune's algorithm, as described in section 3.1, is applied to k polygonal sites having a total of n vertices under the Euclidean metric, the resulting time complexity is $O(k \log k \log n)$. With a simple change to the algorithm's binary search across the sweep front on site events, this time complexity is reduced to $O(k \log n)$.

For each site event, the algorithm must locate the nearest front arc on the sweep front to the event point. The previous section uses the intersection points of adjacent front arcs to distinguish one arc from another in a binary search; these intersection points are expensive to compute for polygonal sites. In this section, we show that an equivalent binary search can be accomplished using spokes that are available in the priority queue $\mathcal{Q}$ and only one computation of front arc intersections. Lemma 3.6 shows that a spoke for every front arc of the sweep front is always available.

Lemma 3.6 *For each front arc K_Q in the sweep front there exists a spoke σ to the polygon Q defining K_Q that intersects K_Q. Moreover, σ exists in the schedule $\mathcal{Q}$ is a degenerate spoke, or can be computed when a new site breaks an infinite front arc.*

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 `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 σ 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 σ 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 spoke $\sigma = \texttt{spoke}(v, Q)$ 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 σ crosses K_Q. ■

If we have a spoke that crosses a front arc, we are indirectly given a point on the arc. Theorem 3.7 uses these front arc points to partition the sweepline and perform a binary search. In the proof, the actual point of intersection between the front arc and the spoke is never computed.

Theorem 3.7 *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.

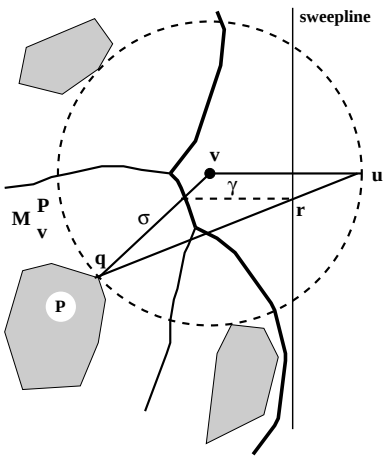

Figure 10: Spoke σ crosses J_P.

We perform a binary search based on representative points for each interval. The space returned by the binary search is bounded by two representative points, spans exactly two intervals, and contains one intersection point of two front arcs. We compute the intersection

point with one call to the `vertex()` subroutine and determine which of the two candidate intervals contains the point 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 3.6 that crosses J_P and let q be the attachment point of σ to P (see figure 10). If v is a point on σ (or its extension) 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. ■

To analyze the time complexity of the sweep algorithm, we must establish time bounds for operations on the schedule $\mathcal{Q}$ and the sweep front $\mathcal{T}$. Each data structure performs its operations in logarithmic-time of its size. Lemma 3.8 proves that 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 ever 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. ∎

We may now conclude that the sweep algorithm finds the compact Voronoi diagram in $O(k(T_v + \log k))$ 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, one per Voronoi vertex. Each site event requires an $O(\log k + T_v)$-time search through the sweep front followed by a constant number of $O(\log k)$-time changes to the sweep front and schedule. Each circle event requires one deletion from the sweep front and at most five additions or deletions to the schedule. The additions to the schedule require $O(T_v)$ time to compute and all the data structure manipulations are completed in $O(\log k)$ time. These values provide the proof of theorem 3.10.

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

3.3 Degeneracies

The sweep algorithm of section 3.2 for the compact Voronoi diagram assumes that all the sites are in general position to avoid degenerate conditions. This section presents refinements to the sweep algorithm and data structures that eliminate these assumptions. Subsections describe how the changes affect each type of event.

Three changes to the sweep algorithm are sufficient to remove the general position assumptions. First, three additional keys, all secondary to the x-coordinate of the event point, order the priority queue $\mathcal{Q}$. 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, the algorithm processes all circle events that have the same event point together. Such a group of circle events arises from Voronoi vertices that are defined by more than three sites. Third, if a site does not have a unique leftmost vertex then its site event is based on the highest of its leftmost vertices.

Although these degeneracies have simple solutions in the context of the algorithm, there remains one significant problem with any implementation: errors arising from floating point arithmetic. Such problems are beyond the scope of this paper.

3.3.1 Assumptions related to site events

The only assumption that relates directly to individual sites, and consequently to site events, is that no site has a vertical edge. Vertical edges are problematic when they are the leftmost edge of a polygonal site; as 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. The algorithm must recognize all Voronoi vertices along this portion of the sweep front, even as the sweep front jumps onto the sweepline at this same instant.

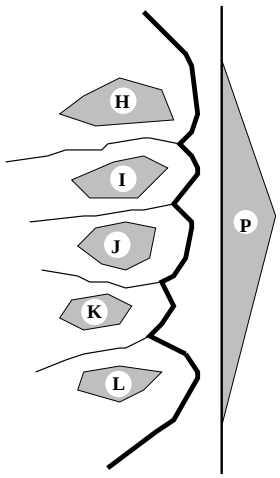

Figure 11: A vertical edge may produce many Voronoi vertices.

A standard solution to vertical edges is to apply a rotation to the input sites or to the sweep direction. From a theoretical viewpoint, the degeneracy disappears; from a practical viewpoint, the rotation must be arbitrary and thus may introduce additional computation errors.

Alternatively, we handle this degeneracy with two changes to the sweep algorithm and require no rotations. First, the additional keys for the priority queue Q enforce a nested-sweep tactic in the algorithm and force the circle events for

such a degenerate site to occur in immediate succession to one another. 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 like 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.

The second change to the algorithm eliminates 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 and is similar to the y-coordinate key of the priority queue $\mathcal{Q}$.

3.3.2 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 by recognizing that the circle events for the two Voronoi vertices coincide with one another (and grouped by the y-coordinate key). The fourth key of the priority queue $\mathcal{Q}$ provides a nice property for this grouping: a set of coincident circle events appear in $\mathcal{Q}$ in the same order as the top-to-bottom sequence of disappearing front arcs along the sweep front.

3.4 A randomized incremental construction algorithm

In this section, we detail a change applicable to both the randomized incremental construction (RIC) of Boissonnat et al. [5] and to the abstract Voronoi RIC of Klein et al. [24] to compute our compact Voronoi diagram in an expected time that matches the deterministic time of section 3.2's sweep algorithm. Throughout this section we assume that all polygons are in general position.

The RIC technique common to both Klein et al. and Boissonnat et al. hinges

on the concept of a *conflict* between a site and a region. In the earlier RICs, these regions are Voronoi cells; for the compact Voronoi diagram, the regions are spoke regions. In each case, the relation between a region and the sites that generate it is always available.

If $\mathcal{P}$ is a set of sites, recall that $V(\mathcal{P})$ is the Voronoi diagram of the sites in $\mathcal{P}$. A site $P \notin \mathcal{P}$ is said to be in conflict with a point p in the plane if p belongs to a region of $V(\mathcal{P} \cup \{P\})$ that is generated by P. By extension, we say that P conflicts with a region A derived from $V(\mathcal{P})$ if P conflicts with some point of A.

The sequence of regions produced as the RIC algorithm inserts sites into a diagram (Voronoi or compact Voronoi) are stored in a conflict history DAG (directed acyclic graph) [7, 8, 6, 13]. If a node for region X has children nodes in the DAG for 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. $X \subset \bigcup_{1 \leq i \leq j} Y_i$ and $X \cap Y_i \neq \phi$ for all i, $1 \leq i \leq j$. Leaf nodes in the DAG represent the regions in the current state of the algorithm. The DAG has the property that a site P conflicts with a region R of the DAG only if P conflicts with at least one parent of R. Consequently, the RIC algorithm can trace a set of conflicts through the DAG from the root to the current regions of the diagram.

The standard RIC [5] represents Voronoi cells with 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. This first node of the DAG is the root. To add a site P to the Voronoi diagram of a set of sites $\mathcal{P}$, the algorithm identifies the Voronoi cells in $V(\mathcal{P})$ with which P conflicts by tracing conflicts from the root of the conflict history DAG down to the leaves. The insertion ends by splitting each of these Voronoi cells into the Voronoi cells of $V(\mathcal{P} \cup \{P\})$, thus creating children in the DAG (figure 12).

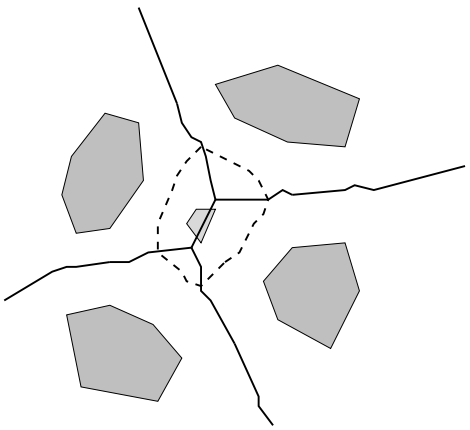

Figure 12: A polygon inserted into a set of Voronoi cells.

The expected running time for this RIC is $O(k \log k)$ for k point sites [5, 24]. Briefly, the algorithm is expected to compute $O(k)$ Voronoi vertices as it adds the k sites. It also expects to encounter $O(\log i)$ conflicts as it traverses the conflict history DAG when i sites have been added. Consequently, the algorithm expects to compute $O(k \log k)$ conflicts.

When the sites for the Voronoi diagram are polygons rather than points, the cost of finding Voronoi vertices and conflicts is no longer constant. Recall that T_v is the time required to complete one call to the `vertex`() subroutine and to find Voronoi vertices. The cost of testing a site against a region for conflict is also $O(T_v)$. Consequently, the expected total complexity for finding all Voronoi vertices is $O(kT_v \log k)$ as noted in [5].

Two modifications to the RIC reduce its expected time complexity for polygonal sites when computing the compact Voronoi. First, we use the spoke regions as nodes in the conflict history DAG rather than the Voronoi cells themselves. Second, we observe that a polygon conflicts with a set of spoke regions whose underlying Voronoi edges form a tree (figure 13). Consequently, a polygon P conflicts with a set of spoke regions whose union is a connected set. As long as we can efficiently locate one spoke region with which P conflicts, traversal techniques across this connected region find all other conflicting spoke regions [36]; tracing any single point of P through the conflict history DAG locates this first spoke region.

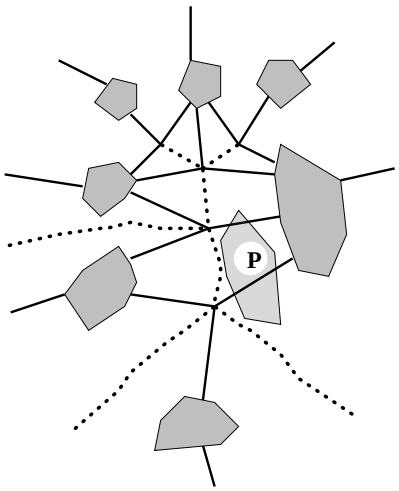

Figure 13: The tree of Voronoi edges (dashed lines) for regions in conflict with P.

The traversal of regions that conflict with a site P only relies on distance calculations to determine whether or not P conflicts with a spoke region. Suppose that P is known to conflict with the spoke regions in the set T. Let v be a Voronoi vertex of spoke region $A \in T$ where v is also adjacent to spoke regions B and C with respect to $V(\mathcal{P})$ (not both in T). If $d_M(v, P) < d_M(v, X)$ where X is a site generating A then the point v cannot remain a Voronoi vertex in $V(\mathcal{P} \cup \{P\})$.

Regions B and C are added to T and the process is iterated until T is maximal.

How can we be sure that this traversal finds all the spoke regions that P conflicts with? Let S be the set of spoke regions with which P conflicts. Since the Voronoi edge skeleton of T must form a connected graph [24], if P conflicts with two spoke regions A and B in S then it must also conflict with a sequence of spoke regions that link the skeleton of A with the skeleton of B. Consequently, P must cross a spoke of A and B and conflicts with the corresponding Voronoi vertices of A and B. Exploring the Voronoi vertices is therefore sufficient to find all the spoke regions with which P conflicts.

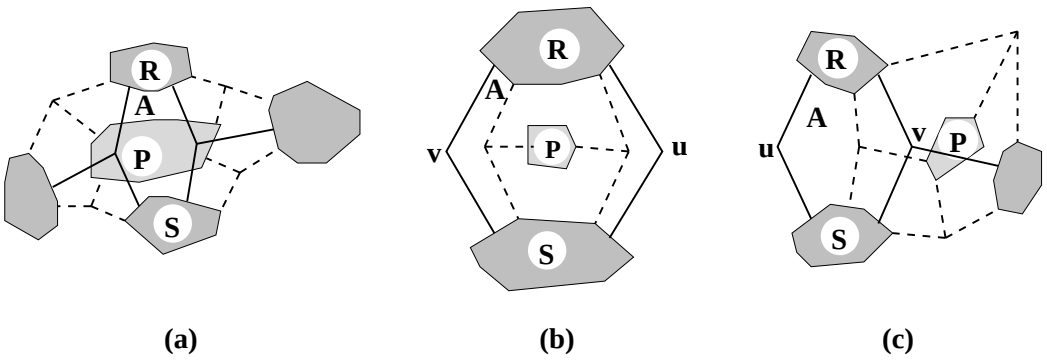

Figure 14: The three possible ways for polygon P to conflict with spoke region A. Dotted lines indicate the new spoke regions after P has been inserted.

The final task of the RIC is to update the conflict history DAG by partitioning each spoke region with which P conflicts. Let P conflict with spoke region A, let u and v be the Voronoi vertices that bound A, and let R and S be the sites that bound A. If P conflicts with both u and v (figure 14a) then A must be replaced in the DAG by two spoke regions: one between P and R, the other between P and S. If P conflicts with neither u nor v (figure 14b) then the algorithm computes the two Voronoi vertices between P, R, and S and partitions A into four new spoke regions. Finally, if P conflicts with v but not with vertex u (figure 14c) then the algorithm computes the Voronoi vertex of P, R, and S that lies inside A and splits A among three spoke regions.

Theorem 3.11 *The expected running time of the RIC for k sites 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 site.*

Proof: The expected-time analysis of the RIC attributes the cost of the algorithm to sites as they are inserted into the diagram and to Voronoi vertices as they are created and destroyed. Each of these events occurs only once per site or Voronoi vertex.

We start with the site costs. The first stage of the RIC algorithm finds one spoke region that conflicts with the new polygon P. The algorithm traces a single point through the conflict history DAG, so conflict calculations are constant-time operations. Klein, Mehlhorn, and Meiser [24] show that a single point is expected to encounter $O(\log i)$ conflicts for the ith site, so we obtain our first conflicting spoke region A in expected $O(\log k)$-time and add A to a set T of spoke regions with which P conflicts. The inserted site also accepts the cost of testing for a conflict between P and the Voronoi vertices of A (possibly adding neighbours of A to T); this requires $O(T_s)$ time—the time for two subroutine calls to `spoke()`. A total cost of $O(\log k + T_s)$ is attributed to each polygon.

The costs of expanding the set T of spoke regions with which P conflicts and updating the DAG are attributed to Voronoi vertices. Let v be a Voronoi vertex of some spoke region $A \in T$ where v has not been tested for a conflict with P yet. The cost of extending (or not extending) T through the spoke regions incident to v arises from calculating the distance from v to P in $O(T_s)$ time. This cost is attributed to the other Voronoi vertex of A. Since a spoke region is added to T only when we have tested one of its Voronoi vertices and that vertex does not appear in $V(\mathcal{P} \cup \{P\})$, this expansion cost is attributed to a Voronoi vertex at most once.

When the algorithm partitions the conflicting spoke regions to update the history DAG, the $O(T_v)$ cost of computing new Voronoi vertices is associated with each vertex created, and the cost of adding a spoke region to the conflict history DAG is allocated to the new Voronoi vertex of the new spoke region.

Since each Voronoi vertex with which P 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 created in the history of the RIC accounts for a total cost of $O(T_s + T_v)$.

As with Klein, Mehlhorn, and Meiser [24], the algorithm expects to com-

pute $O(k)$ Voronoi vertices. These vertices are each charged a cost of $O(T_s+T_v)$ while the k polygons are each charged an expected cost of $O(\log k + T_s)$. This gives a total expected time for the RIC algorithm of $O(k(\log k + T_v + T_s))$. ■

3.5 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 tree 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 [14, 21].

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 their Voronoi vertex where the cells of A, B and C occur in counterclockwise (ccw) order. Return the Voronoi vertex, 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. The 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 takes $O(\log n)$ time under the Euclidean metric and $O(\log n \log m)$ time under d_M.

Some of our routines make use of Kirkpatrick and Snoeyink's tentative prune-and-search technique [21].

Theorem 3.12 (From [21]) *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 [20] and [21].

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 $q \in M_p^A$ and $a \in A$ that share parallel tangents, and where the ray from p through the point q intersects A at the point a. By first computing outer common tangents of M_p^A and A, we restrict our search to two polygonal chains that share a single pair of parallel tangents and satisfy the ray intersection property.

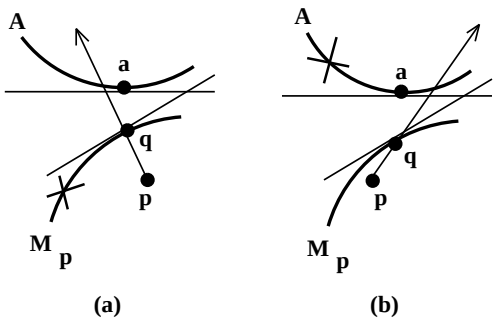

Figure 15: Half of either chain A or chain M can be discarded.

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 q to a; the opposite configuration

is handled with a symmetric argument. There are two cases, depending on whether a is right or left of the ray $\overrightarrow{pq}$ as illustrated in figure 15. If a is to the right (figure 15a), then any ray $\overrightarrow{pq'}$ with q' counterclockwise (ccw) from q on M_p will intersect A at a point a' clockwise (cw) of A. Because tangents at q' and a' will intersect to the right of $\overrightarrow{q'a'}$, we can discard the portion of M_p that is ccw of q. If a is on the left of $\overrightarrow{pq}$ (figure 15b) then consider $a' \in A$ cw of a and $q' \in M_p$ on $\overline{pa'}$. Again, tangents to a' and q' intersect right of $\overrightarrow{q'a'}$, so we can discard the part of A cw of a. (If a is on $\overline{pq}$ then we can perform both discards.) ■

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 6.)

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 [14] 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 their 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 the points b and $\tau_a \cap \tau_b$, discard the half of B away from $\tau_a \cap \tau_b$; otherwise discard the half of A. ■

If $O(m)$ preprocessing on M is allowed to identify vertices with parallel tangents, then one can compute the desired homothet in $O(\log n + \log m)$ time by extending the tentative prune-and-search approach [1].

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 $f: A \to B$ that maps $a \in A$ to the intersection of the normal to A at a with B. If a is a vertex of A, then we consider all vectors from a that lie between the normals to the edges adjacent to a as a set of normal vectors at a to intersect with B and f maps a to a range on B. Theorem 3.12 allows us to compute a fixed point and Voronoi vertex in $O(\log n)$ time. See Kirkpatrick and Snoeyink [21] for more detail.

For the convex distance function d_M, we start with the same common 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. ∎

3.6 Lower bounds

Let d_E be the Euclidean distance function and let d_M be the convex distance function defined by an arbitrary m-gon. The algorithm for finding the compact diagram has $O(k \log n)$ and $O(k \log n \log m)$ time bounds under the d_E and d_M distance functions respectively. In this section we prove lower bounds on the cost

of computing the compact diagram associated with k sites of total complexity n. For d_E, the lower bound matches our algorithm's worst-case time complexity; for d_M, the lower bound is $\Omega(k(\log n + \log m))$ operations.

Theorem 3.16 *To compute the compact diagram of k sites having a total of n vertices under d_E requires $\Omega(k \log n)$ operations; under d_M it requires $\Omega(k(\log n + \log m))$ operations.*

Proof: The proof examines the cases where each of k, n, and m is the dominating factor for the compact diagram.

For a set of k point sites under d_E, the Voronoi diagram can be constructed from the spoke diagram in $\Theta(k)$ time. Thus the problem of finding the compact diagram inherits the Voronoi diagram's $\Omega(k \log k)$ lower bound [34, 29]. The proof argument can be easily modified to give an $\Omega(k \log k)$ lower bound for k point sites under d_M.

In a similar fashion, the following $\Omega(k(\log n + \log m))$ bound for finding the compact diagram under d_M can be modified to give an $\Omega(k \log n)$ bound for the same task under d_E.

We use reductions from a variant of the element uniqueness problem to obtain the bounds. An instance of the *vertex query problem* is a polygon P with t distinct vertices all of which lie on the unit circle and a set Q of q query points on the unit circle. The problem asks if any point of Q is a vertex of P. It is straightforward to demonstrate an $\Omega(q \log t)$ lower bound for the vertex query problem on the fixed order algebraic decision tree model [4].

The vertex query problem can be reduced to q disjoint instances of the Voronoi vertex problem: each instance takes one query points and two distinct vertices of P and asks if the Voronoi centre with respect to the convex distance function defined by P lies at the origin chosen for P. The q instances can be sufficiently separated in the plane so that the spoke diagram includes the particular Voronoi vertices. This implies an $\Omega(k \log m)$ lower bound on computing the compact diagram.

The vertex query problem can also be reduced to q disjoint instances of the Voronoi vertex problem in another manner. Each instance takes as one site the polygon P and as the two other sites two vertices from a fixed convex triangle T tangent to the unit circle at the query point and asks if their Voronoi

centre under the convex distance function defined by T lies at the origin of T. By suitably spacing the q instances in the plane, we produce a collection of $3q$ sites of total complexity $q(t+2)$ whose spoke diagram encodes a solution to the original vertex query problem. Thus, the $\Omega(q \log t)$ lower bound when $t > q$ implies an $\Omega(k \log n)$ lower bound on the cost of computing the compact diagram.

Taking the maximum of these lower bounds and noting that n is $\Omega(k)$ yields the theorem's result. ■

It would be interesting to close the gap between the upper and lower 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 this representation by a general algorithm, where the dependence on site and distance function complexity is restricted to subroutines that are called $O(k)$ times.

Of greatest interest is the extension of this representation to higher dimensions, analogous to the work of de Berg et al. [12]. Our 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. We also thank the referees for improvements to the paper arising from their comments and suggestions.

References

[1] H. Alt, D. Hsu, and J. Snoeyink. Computing the largest inscribed isothetic rectangle. Submitted to the, 1994.

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

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

[4] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[5] 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.

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

[7] 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.

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

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

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

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

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

[13] 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.

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

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

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

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

[18] T. C. Kao and D. M. Mount. An algorithm for computing compacted Voronoi diagrams defined by convex distance functions. In *Proc. 3rd Can. Conf. Comp. Geom.*, pages 104–109, Simon Fraser University, Vancouver, 1991.

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

[20] D. Kirkpatrick and J. Snoeyink. Computing constrained segments: Butterfly wingspans in logarithmic time. In *Proc. 5th Can. Conf. Comp. Geom.*, pages 163–168, Waterloo, Canada, 1993.

[21] D. Kirkpatrick and J. Snoeyink. Tentative prune-and-search for computing fixed-points with applications to geometric computation. *Fund. Infor.*, 1994.

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

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

[24] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Comput. Geom. Theory Appl.*, 3(3):157–184, 1993.

[25] 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.

[26] K. Mehlhorn, S. Meiser, and C. Ó'Dúnlaing. On the construction of abstract Voronoi diagrams. *Disc. & Comp. Geom.*, 6:211–224, 1991.

[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] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams. In *Report 260*, pages 178–191. IIG-TU, Graz, Austria, 1988.

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

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

[36] K. Sugihara and M. Iri. Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic. *Proc. IEEE*, 80(9):1471–1484, Sept. 1992.

[37] 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.