An Algorithm for Computing Compacted Voronoi Diagrams Defined by Convex Distance Functions

Thomas C. Kao David M. Mount
Department of Computer Science
University of Maryland
College Park, MD 20742
Internet: kao@cs.umd.edu

Abstract

We present an algorithm for computing a compact representation of the generalized Voronoi diagram for a set of n line segments under a distance function defined by a k-sided convex polygon. Our algorithm runs in $O((\log^2 k)n \log n)$ time improving on the existing $O(kn \log n)$ algorithm and produces a compact representation of the diagram of size $O(k+n)$ from which point location queries can be answered in $O(\log k + \log n)$ time. The algorithm has the interesting property of breaking through the $\Omega(kn)$ output size lower bound for the running time of the algorithm.

1 Introduction

In this paper we reconsider the problem of computing the generalized Voronoi diagram of a set of disjoint polygonal objects in the plane using a distance function defined by a convex polygon. This problem was studied by Leven and Sharir [LS 87], and is motivated by the following motion planning problem: Given a convex polygon C with k sides and a set W of polygonal obstacles, which are composed of a set of n line segments ("walls") and isolated points, plan a purely translational motion of C from a given initial position to a desired final position during which C avoids collision with the obstacles, or determine that no such motion exists. (We assume that these n line segments only intersect at endpoints.) Leven and Sharir showed that this problem could be solved by computing the generalized Voronoi diagram of W with respect to a distance function defined by C, rather than the standard Euclidean distance metric. Such a generalized Voronoi diagram will be referred to as a *C-diagram* [For 85] throughout this paper. The generalized Voronoi diagram was also studied by Chew and Drysdale [CD 85] (for point sets only) and Fortune [For 85], with time bounds similar to Leven and Sharir.

Leven and Sharir assumed that the convex polygonal robot that defines the distance function has a constant number of sides, and they presented an $O(n \log n)$ algorithm for computing the diagram. They observed that if the number of sides k of the convex distance function is taken into account then the running time of their algorithm is $O(kn \log n)$ and uses $O(kn)$ space. Because the generalized Voronoi diagram can consists of $\Theta(kn)$ line segments in general, in the worst case their space bounds are optimal and time bounds are within a logarithmic factor of optimal simply by consideration of output size. (In addition there is the standard $\Omega(n \log n)$ lower bound on the computation of the Voronoi diagram.)

The principal contribution of this paper is a modification to Leven and Sharir's algorithm which runs in $O((\log^2 k)n \log n)$ time and stores the diagram in a compacted form which requires only $O(k+n)$ space. The compacted representation inherits many of the essential query properties of the original diagram, in particular point location queries can be answered in $O(\log k + \log n)$ time, matching the time bound for the standard representation.

From the standpoint of applications, it may seem frivolous to consider convex robots with arbitrarily large numbers of sides. It is not unreasonable to believe that robotics applications, any convex shape can be represented to an acceptable degree of precision by a polygon containing no more than, say, 20 sides. Even though the algorithmic techniques which we present are not sophisticated, there are two reasons that we feel the result is of importance.

- This is among a few nontrivial examples in which a standard geometric structure (in this case the generalized Voronoi diagram), can be stored implicitly using sublinear storage so that queries can be answered in the same time bound (see also [EGHSSSW 89]). This is of methodological interest, because it is tempting to assume naively that output size is a reasonable lower bound for the complexity of a problem, without considering alternative representations or the information theoretic content of the output representation. This caveat was discussed at some length by Guibas and Seidel in their paper on computing convolutions [GS 87]. The notion of finding compact representations and dealing with implicitly defined objects is pervasive throughout computational geometry, but has yet to be addressed within the domain of generalized Voronoi diagrams.

- In practice, moderate multiplicative factors on running time can be tolerated especially when considering preprocessing time, which is what we are doing in computing a Voronoi diagram. However, the space requirements of a search structure must be paid for throughout the lifetime of query processing. Multiplying the space required by a complex data structure by a moderate multiplicative factor (e.g. 20) may be unacceptable in applications where space is limited.

The generalized Voronoi diagram for a collection of n line segments using a convex distance function is a subdivision of the plane into n regions (possibly unbounded). There are $O(n)$ vertices of degree three or greater, which are connected by a collection of "bisectors" (relative to the convex distance function) between pairs of neighboring objects. Each bisector is a simple polygonal arc consisting $O(k)$ segments. In general these bisectors are not convex, but they possess certain star-shaped properties relative to the objects they bisect. Our algorithm is a simple modification of Leven and Sharir's algorithm. In Section 3 we show that by exploiting these properties we can perform the essential primitive tracing and intersection operations needed to compute the Voronoi diagram in $O(\log^2 k)$ time per primitive, improving on the naive $O(k)$ time algorithms for these primitives. In Section 4 we show how to represent the entire diagram in a compact form requiring $O(k+n)$ space, and finally we show how to perform point location queries from this representation.

1.1 Voronoi Diagrams and Extensions

Many proximity problems can be solved using Voronoi diagrams [PS 85]. A natural question to ask is whether the Voronoi diagram can be generalized to other geometric shapes and metrics. These generalization would have many applications. Lee and Wong [LW 80] generalized the Voronoi diagram using L_1 metric and L_∞ metric and pointed out that these diagrams speed up retrieval algorithms for two-dimensional storage systems. Kirkpatrick [Kir 79] and later Yap [Yap 87] proposed an $O(n \log n)$ time algorithm for computing the Voronoi diagram of a set W of polygonal objects, with a total of n edges or vertices.

A natural extension to L_p metrics is the so-called *convex distance functions* introduced by Minkowski [Lay 72]. Given a convex polygon C and an interior reference point c we define the distance function based on C and c as follows. We will refer to the position of C in the plane in which the reference point c of the convex polygon C lies at the origin O as the ***standard position*** of C, and denote this placement by C_o. For a convex body C and real number α, let $\alpha C = \{\alpha p : p \in C\}$ (C grown by the scale factor α) and for vector q, let $q + C = \{q + p : p \in C\}$ (the translate of C by q). Let a and b be two points in the plane. We define the *C-distance* from a to b (or equivalently, the *C-closeness* of a to b), denoted by $d_C(a,b)$, as $d_C(a,b) = \inf\{\alpha \geq 0 : \ a \in b + \alpha C_o\}$. Intuitively, the C-distance from a to b is the smallest (positive) scale factor α needed for a copy C positioned with its reference c at a to enclose the point b.

Note that since the origin O is an interior point of C_o, the C-distance is always finite, positive, continuous, and obeys the triangle inequality, but need not necessarily be symmetric, and thus is not induced by a metric in general [LS 87]. The C-distance from a point p to a given object $W_i \in W$ is defined in a natural way as $d_C(p, W_i) = \inf\{\alpha \geq 0 : \ (p + \alpha C_o) \cap W_i \neq \emptyset\}$.

Using d_C we define the *C-diagram* of W with respect to the convex polygon C as follows. For each $i \neq j$ define $H(i,j) = \{p : d_C(p, W_i) \leq d_C(p, W_j)\}$, as the set of all points in the plane whose C-distance to W_i is no greater than their C-distance to W_j. Then define the ***Voronoi polygon*** (*cell*) $\mathrm{Vor}(W_i)$ associated with W_i to be $\mathrm{Vor}(W_i) = \bigcap_{j \neq i} H(i,j)$, as the set of all points in the plane whose C-closeness to W_i is not greater than their C-closeness to any other element of W. Finally, the ***C-diagram*** is defined to be the set of points which belong to two or more Voronoi polygons. See Figure 1 for an example of C-diagram from [LS 87], where C is a triangle, and the set W consists of two triangles and one quadrilateral. Note that the dashed lines in the figure are the Voronoi edges defined by the three convex objects of W, while the dotted lines are the additional Voronoi edges which result if we consider the (open) line segments and vertices of W as separate objects. We have excluded the portion of the diagram lying in the interior of W.

2 Technical Preliminaries

2.1 Leven and Sharir's Algorithm

Paraphrasing from Leven and Sharir's paper [LS 87], their algorithm for constructing the C-diagram proceeds as fol-

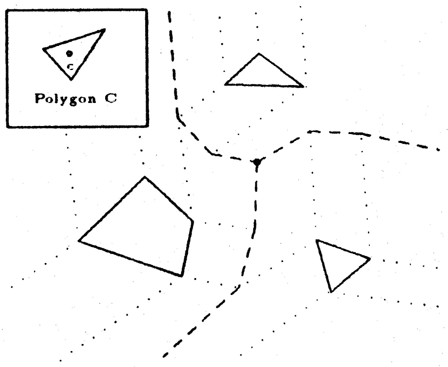

Figure 1: An example of C-diagram

lows. Let P be the set of up to $2n$ leftmost and rightmost points on objects in W. Following [Yap 87], they divide the plane by vertical lines into $2n$ *slabs* such that the interior of each slab contains exactly one point of P. The algorithm runs in stages. In each stage the set of slabs is partitioned into disjoint pairs of adjacent slabs and each such pair is combined into a new, larger, slab. That is, in the first stage the $2n$ slabs are combined in disjoint pairs into n slabs, in second stage these n slabs are combined in pairs into $(n/2)$ slabs and so on. After $O(\log n)$ stages only a single slab remains and the algorithm terminates.

In order to achieve $O(n \log n)$ time and $O(n)$ space bounds for the construction of C-diagram, Leven and Sharir [LS 87] assumed that the polygon C has a fixed number of sides. For such objects certain operations, which are used during the construction of the C-diagram as described below, can be accomplished in *constant time*. Such operations are: (a) computing the C-distance between any two points in the plane or from a given point in the plane to a straight line; (b) calculating the intersection points of the locus of points which are C-equidistant to two "walls" (line segment) or "corners" (vertices) of objects in W and another such curve or a straight line segment. (We denote such a locus as a *bisector*.)

If the number k is taken into account, Leven and Sharir's algorithm uses $O(kn \log n)$ time and $O(kn)$ space. We adopt the convention [Kir 79][Yap 87] that each wall can be decomposed uniquely into an open line segment and two corners from its two endpoints. Therefore, the *sites* in W are either open line segments or points. Such a convention is always plausible and will only increase the complexity of the problem by a (small) constant factor.

We will show in Section 3 that these primitive operations can be performed in $O(\log k)$ and $O(\log^2 k)$ time for items (a) and (b) above, respectively. By implementing these primitive operations carefully and representing the C-diagram *compactly*, we are able to reduced both the time and space bounds. Later in this paper, we will show an $O((\log^2 k) n \log n)$ time algorithm for constructing the C-diagram, using only $O(k+n)$ space.

3 Algorithm Descriptions, Correctness and Bounds

3.1 Representation of Bisectors

Recall that the C-distance from a point p to a site s is denoted as $d_C(p, s)$. The bisector B of two sites (s_1, s_2) is the set of points b (in the plane) that are C-equidistant to both s_1 and s_2. In other word, $B(s_1, s_2) = \{b \mid d_C(b, s_1) = d_C(b, s_2)\}$. We represent a point b on the bisector $B(s_1, s_2)$ by a six-tuple $(b, s_1, s_2, \alpha, c_1, c_2)$, where absolute value of α is the scale factor of C, c_1, c_2 are the *contact points* of $(|\alpha|C)$ with the sites s_1 and s_2 respectively. Such a six-tuple is called a *bisector point descriptor*. We adopt the convention that α is positive (resp. negative) if b lies to the left (resp. right) of the directed segment from c_1 to c_2. We also define the points p_i of $s_i, i = 1, 2$, to be the contact points of the objects s_i with the scaled body $|\alpha|C$. (If s_i makes contact with $|\alpha|C$ along an edge, choose p_i to be the point with the lexicographically smallest coordinates.)

Given two objects s_1 and s_2, we note that if a point b is on $B(s_1, s_2)$, then we can compute the scale factor α in $O(\log k)$ time. Conversely, given a (signed) scale factor α, we can compute a point $b \in B(s_1, s_2)$ in $O(\log^2 k)$ time, or report that no such point exists. These conversions are omitted here to save space.

3.2 Implementing Primitive Operations

Next we consider how to implement the primitive operations needed by Leven and Sharir's algorithm.

Computations of C-distance:

- C-distance $d_C(a, b)$:
  Using binary search find the intersection point x of C (placed so that its reference coincides with a) with the directed ray from a to b. Then $d_C(a, b) = |\overline{ab}|/|\overline{ax}|$. This can be done in $O(\log k)$ time.
- C-distance $d_C(a, L)$:
  First find any fixed *contact point* x of C with the supporting line parallel to L and on the same of C. Note that $\overrightarrow{ax}$ may not be perpendicular to the line L. Compute the intersection point p of L with the ray directed from a to x. Then $d_C(a, L) = |\overline{ap}|/|\overline{ax}|$. This can be done in $O(\log k)$ time.

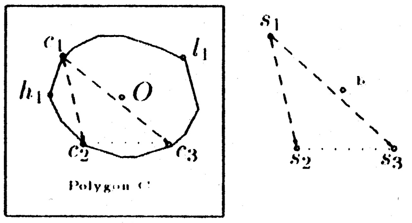

Figure 2: Case PPP

Intersection of bisectors: With loss of generality, assume that the three contact points c_1, c_2 and c_3 are in cyclical order when listed counterclockwise (on the boundary of $|\alpha|C$, for some α). The output will be a pair (b,α), where b is the point that is C-equidistant to to three sites s_1, s_2 and s_3, and α is the (signed) C-distance between b and the three sites. Again, we assume α is positive. Recall that the k vertices of C are stored in counterclockwise order. We also observe that there are four possible combinations of sites to consider: three (open) lines (LLL), one point with two lines (PLL), two points and one line (PPL), and finally three points (PPP). We will only show two of the cases (PLL and PPP) to illustrate to ideas, the rest are left as exercises to save space.

Case PLL: In this case, s_1 is a point, while both s_2 and s_3 are (open) line segments. Note that the fixed contact points $c_i, i = 2, 3$, of the line segments s_i are precomputed once-for-all in $O(\log k)$ time each.

1. Let l_1 be the point c_3. Similarly, let h_1 be the point c_2. We will maintain the invariant that c_1 lies counterclockwise between l_1 and h_1.

2. **if** the points l_1 and h_1 are on the same edge e_1 of C, then go to step 6.
else pick the vertex with index half-way between l_1 and h_1 as the contact point c_1.

3. Compute $\alpha_2 = |\overline{s_1s_2}|/|\overline{c_1c_2}|$. Compute $\alpha_3 = |\overline{s_1s_3}|/|\overline{c_1c_3}|$.

4. **if** $\alpha_2 > \alpha_3$, let $h_1 = c_1$. Go to step 2,
else if $\alpha_2 < \alpha_3$, let $l_1 = c_1$. Go to step 2.

5. (We have found that c_1 lies on the edge e_1 of step 2) Compute the correct c_1 directly (and analytically). Compute the scale factor $\alpha = |\overline{s_1s_2}|/|\overline{c_1c_2}|$. Compute the point $\overrightarrow{b} = s_1 - \alpha(\overrightarrow{Oc_1})$. Exit and output the pair (b,α).

Correctness: see case PPP below.

Case PPP: In this case, all three sites are points (see Figure 2).

1. Compute h_3 (and also l_1) to be a vertex of C that is a parallel tangent to the ray from s_3 to s_1.
Compute h_1 (and also l_2) to be a vertex of C that is a parallel tangent to the ray from s_1 to s_2.
Compute h_2 (and also l_3) to be a vertex of C that is a parallel tangent to the ray from s_2 to s_3.
We will maintain the invariant that c_i lies counterclockwise l_i and h_i for $i = 1, 2, 3$.

2. Let u be the smallest index such that the points l_u and h_u are NOT on the same edge e_u of C.
If u is not equal to 1, 2, or 3, go to step 7.
if $u = 1$, let $v = 2$ and $w = 3$,
else if $u = 2$, let $v = 3$ and $w = 1$,
else if $u = 3$, let $v = 1$ and $w = 2$.

3. Pick the vertex with index half-way between l_u and h_u as the contact point c_u.

4. Locate c_v such that $\overrightarrow{c_uc_v}||\overrightarrow{s_us_v}$. Compute $\alpha_v = |\overline{s_us_v}|/|\overline{c_uc_v}|$.

5. Locate c_w such that $\overrightarrow{c_uc_w}||\overrightarrow{s_us_w}$. Compute $\alpha_w = |\overline{s_us_w}|/|\overline{c_uc_w}|$.

6. **if** $\alpha_v > \alpha_w$, (move c_u clockwise, which would move both c_v and c_w counterclockwise) let $h_u = c_u, l_v = c_v$ and $l_w = c_w$. Go to step 2,
else if $\alpha_v < \alpha_w$, (move c_u counterclockwise, which would move both c_v and c_w clockwise) let $l_u = c_u$, $h_v = c_v$ and $h_w = c_w$. Go to step 2.

7. (We have found that c_1 lies on the edge e_1, c_2 lies on the edge e_2, and c_3 lies on the edge e_3)
Find the correct c_1, c_2 and c_3 directly (and analytically). Compute the scale factor $\alpha = |\overline{s_1s_2}|/|\overline{c_1c_2}|$. Compute the point $\overrightarrow{b} = s_1 - \alpha(\overrightarrow{Oc_1})$. Exit and output the pair (b,α).

Correctness: We first show that the function $f(c_u) = (\alpha_v - \alpha_w)$ defined over the boundary of C between vertices l_u and h_u, has a unique zero c_u^*. Furthermore, $\forall c_u \in (l_u, c_u^*), f(c_u) < 0$, and $\forall c_u \in (c_u^*, h_u), f(c_u) > 0$. To see this observe that by moving c_u counterclockwise between $[l_u..h_u]$, it can be shown that the angle $\theta = \angle(c_uc_vc_w)$ increases. Let $\theta = \angle(c_uc_vc_w)$ and $\phi = \angle(c_wc_uc_v)$. By the law of Sine, the ratio

$$y(\theta) = \frac{|\overline{c_uc_v}|}{|\overline{c_uc_w}|} = \frac{\sin(\pi - \theta - \phi)}{\sin(\theta)} = \frac{\sin(\theta + \phi)}{\sin(\theta)}.$$

With the angle $\phi = \angle(c_wc_uc_v)$ fixed, it can be shown that that $dy/d\theta < 0, \forall\theta \in (0,\pi)$. Note that $f(c_u) = f(\theta) = (|\overline{s_us_v}| - y(\theta) * |\overline{s_us_w}|)/(\overline{c_uc_v}|$. Therefore, the ratio y is monotonically decreasing as c_u moves counterclockwise, and it is plausible to perform a binary search on c_u.

Analysis: This is a nested binary search for c_1, c_2 and c_3. Step 1 is a simple binary search for the vertices l_i and h_i for the valid range of the contact point c_i, for $i = 1, 2, 3$. This can be computed in $O(\log k)$ time each. Step 4 (and step 5) is a simple binary search for c_v (and c_w), for a given value c_u. This takes $O(\log k)$ time each. The main loop is a binary search for c_u, which is performed $O(\log k)$ iterations. Therefore, the total time requirement is $O(\log^2 k)$.

In conclusion, the C-diagram of a set of n line segments for a convex distance function defined by a k-sided convex polygon can be computed in $O((\log^2 k) n \log n)$ time.

4 A Compact Representation for the C-diagram

In the previous section we have described the basic primitive operations needed for the construction of the Voronoi diagram. In this section we explain how to apply these primitives and describe the compact representation for the diagram.

This compact representation is based on an implicit representation for the bisector of two *sites*, each of which can either be a point or a line (segment). A bisector is defined to be the locus of points which are C-equidistant to the two sites. As mentioned earlier, the bisector of two points is a polygonal chain of up to k sides, monotonic with respect to each point site. The bisector of a point and a line (segment) is a *convex* polygonal chain of up to k sides, monotonic with respect to both sites. The bisector of two lines is a third line separating the two lines.

The C-distance from a point p to a site s, denoted $d_C(p, s)$, can be computed in $O(\log k)$ time. Therefore we can determine whether a point lies above or below a bisector (of sites s and t) in $O(\log k)$ time by comparing $d_C(p, s)$ with $d_C(p, t)$. We can compute the intersection of two bisectors in $O(\log^2 k)$ time. This also means that we can compute a point that is C-equidistant to three sites in $O(\log^2 k)$ time.

By representing the bisectors implicitly, we can cut the total space requirement from $O(kn)$ to $O(k + n)$. In essence, we only store one copy of the polygon C and represent each bisector (with up to k edges) by one pseudo edge. The C-diagram will be represented by a planar graph with only $O(n)$ pseudo edges and $O(n)$ faces.

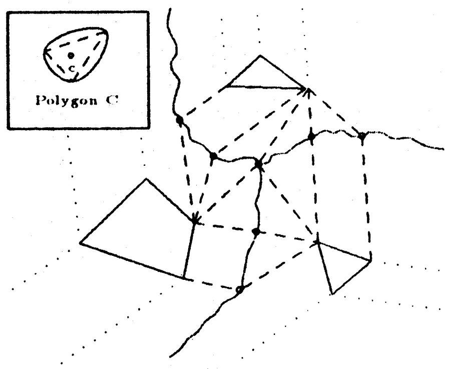

Figure 3: V-vertices and Spokes

4.1 Spokes, Contour Tracing and Point Location

Let L and R are two adjacent slabs with L lies to the left of R. To merge the partial C-diagrams from Vor(L) and Vor(R), we have to compute the locus of points which are equally C-closest to a site in L to a site in R. Following Kirkpatrick [Kir 79], the locus of points is called the ***contour*** separating L from R. According to Lemma 3.1 of [LS 87], it is possible that the contour may extend to infinity and "return from infinity".

Recall that the sites are either points or (open) line segments. There are two kinds of vertices in the C-diagram. The first kind is called the *V-vertices*, which are vertices of degree 3 or more found by the intersections of (three or more) bisectors of sites. There are $O(n)$ such V-vertices and $O(n)$ bisectors. The second kind is called the *B-vertices* which are the vertices of degree 2 along a bisector between two sites. Each bisector has up to k such B-vertices, with a total of $O(kn)$ for the entire C-diagram.

By a technique very similar to that of [LS 87], the contour can be computed with the help of *spokes* defined below. We divide each cell of Vor(L) and Vor(R) into subcells by adding line segments which join each V-vertex μ of the (partial) C-diagram to the three points (p_i) on the three corresponding sites (s_i) in L or R to which μ is C-closest. These line segments are called spokes. Of the bisectors meeting at the V-vertex μ, if a bisector $B(s, t)$ (from elementary sites) with s a point site adjacent to an open line segment t, then the bisector $B(s, t)$ is considered as a spoke from μ to the point site s. See Figure 3, where the robot C is a convex body with three smooth curves to dramatize the effect of pseudo-edges, which appears as solid spline curves. The V-vertices are circled, and the spokes are shown in dashed line. The bisectors from elementary sites that are not considered spokes are shown in dotted line.

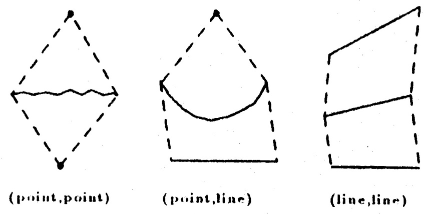

Figure 4: Three types of regions

Therefore, the boundary of each resulting subcell thus consists of (1) two spokes, (2) a connected portion of the bisector between this site and an other, (called a *pseudo-edge*) or possibly two such edges if the subcell is unbounded, and (3) the site itself, which is either a point or a (open) line segment.

The reason for the space saving is because every consecutive chain (of size up to k) of B-vertices that lie on the same bisector B will be represented implicitly as a single pseudo-edge descriptor requiring $O(1)$ space. Each pseudo-edge $\psi(B)$ (for a bisector B) is represented by two bisector point descriptors which mark the starting and ending position on B. Note that there are only $O(n)$ such pseudo-edges. By ignoring the B-vertices in the creation of spokes, the number of subcells and spokes is linear in the number of V-vertices only, namely $O(n)$.

The encoded C-diagram: To facilitate point location query, as discussed below, we need to perform search in the presence of $O(n)$ straight line segments (including spokes) and $O(n)$ pseudo-edges. Let two adjacent subcells $u(s)$ and $v(t)$ be separated by a pseudo-edge $\psi(B)$, where points in subcell u (resp. v) are closer to the site s (resp. t), and let B be the bisector between s and t. We define the *region* $R(u, v)$ as the union of the regions of u and v. Therefore, the boundary of R is the union of the boundaries of u and v, excluding the overlapping pseudo-edge $\psi(B)$. (see Figure 4 for three types of regions) Clearly, there are $O(n)$ such regions. The entire C-diagram is now encoded as a network of straight line segments with $O(n)$ sites, spokes, pseudo-edges, subcells, and regions. Ignoring the pseudo-edges the remaining subdivision is a straight line planar graph of size $O(n)$ which we call $S(W)$.

Point Location Queries: Given a point p in the plane, we show how to adapt standard point location techniques to determine which site is closest to p in $O(\log k + \log n)$ time, using only $O(k + n)$ space. First, apply standard point location to $S(W)$ to locate the region R in the encoded C-diagram that contains p. This can be done in $O(\log n)$ time as the encoded C-diagram is a straight line planar graph of size $O(n)$ [PS 85]. Recall that a region is basically the union of two adjacent subcells that share a pseudo-edge separating two sites, one from each subcell. Let the region R contains the two sites s and t. We can determine which side of the pseudo-edge p lies by testing whether p is closer to the site s or t in $O(\log k)$ time.

5 Bibliography

[CD 85]. L. P. Chew and R. L. Drysdale, III, "Voronoi Diagrams Based on Convex Distance Functions", Proceedings of the First ACM Annual Symp. on Computational Geometry, pp.235-244, 1985.

[EGHSSSW 89] H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink, E. Welzl, "Implicitly representing arrangements of lines or segments", Discrete and Computational Geometry, *4*, pp.433-466, 1989.

[For 85]. S. Fortune, "A Fast Algorithm for Polygon Containment by Translation", Proceedings of the 12th International Colloquium on Automata, Language and Programming, pp.189-198, 1985.

[GS 87]. L. J. Guibas and R. Seidel, "Computing convolutions by reciprocal search", Discrete and Computational Geometry, *2*, pp.175-193, 1987.

[Kir 79]. D. Kirkpatrick, "Efficient Computation of Continuous Skeletons", IEEE 20th Annual Symp. Foundation Comput. Sci., pp.18-27, 1979.

[Lay 72]. S. R. Lay, "Convex Sets and Their applications", Wiley Inc., New York, 1972.

[LS 87]. D. Leven and M. Sharir, "Planning a Purely Translational Motion for a Convex Polygonal Object in Two Dimensional Space Using Generalized Voronoi Diagrams", Discrete Comput. Geom., *2*, pp.9-31, 1987.

[LW 80]. D. T. Lee and C K. Wong, "Voronoi Diagrams in L_1- (L_∞-) Metrics with 2-dimensional Storage Applications", SIAM J. Comput., *9*, pp.201-211, 1980.

[PS 85]. F. P. Preparata and M. I. Shamos, "Computational Geometry, An Introduction", Springer-Verlag New York Inc., 1985.

[Yap 87]. C. K. Yap, "An O($n \log n$) Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments", Discrete Comput. Geom., *2*, pp.365-393, 1987.