Computational Geometry II

D. T. Lee
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
e-mail: dtlee@ece.nwu.edu

1 Introduction

This is a follow up on the previous Chapter dealing with geometric problems and their efficient solutions. The classes of problems that we address in this Chapter include proximity, optimization, intersection, searching, point location, and some discussions of geometric software that is under development.

2 Proximity

Geometric problems pertaining to the questions of how *close* two geometric entities are among a collection of objects or how *similar* two geometric patterns match each other abound. For example, in pattern classification and clustering, features that are *similar* according to some metric, are to be clustered in a group. The two aircrafts that are *closest* at any time instant in the air space will have the largest likelihood of collision with each other. In some cases one may be interested in how *far apart* or how *dissimilar* the objects are. Some of these *proximity* related problems will be addressed in this section.

2.1 Closest Pair

Consider a set S of n points in $\Re^k$. The *closest pair problem* is to find in S a pair of points whose distance is the minimum, *i.e.*, find p_i and p_j, such that $d(p_i, p_j) = \min_{k \neq l}\{d(p_k, p_l)$, for all points $p_k, p_l \in S\}$, where $d(a, b)$ denotes the Euclidean distance between a and b. (The result below holds for any distance metric in Minkowski's norm.) Enumerating all pairs of distances to find the pair with the minimum distance would take $O(d \cdot n^2)$ time. As is well-known, in 1-dimension one can solve the problem much more efficiently: Since the closest pair of points must occur consecutively on the real line, one can sort these points and then scan them in order to solve the closest pair problem in $O(n \log n)$ time. The time complexity turns out to be best possible, since the problem has a lower bound of $\Omega(n \log n)$, following from a linear time transformation from the *element uniqueness problem*[85].

But unfortunately there is no total ordering for points in $\Re^k$ for $k \geq 2$, and thus sorting is not applicable. We will show that using *divide-and-conquer* approach one can solve this problem in $O(n \log n)$ optimal time. Let us consider the case when $k = 2$. In the following we only compute the minimum distance of the closest pair; the actual identity of the closest pair that realizes the minimum distance can be found easily by some straightforward bookkeeping operations. Consider a vertical separating line $\mathcal{V}$ that divides S into S_1 and S_2 such that

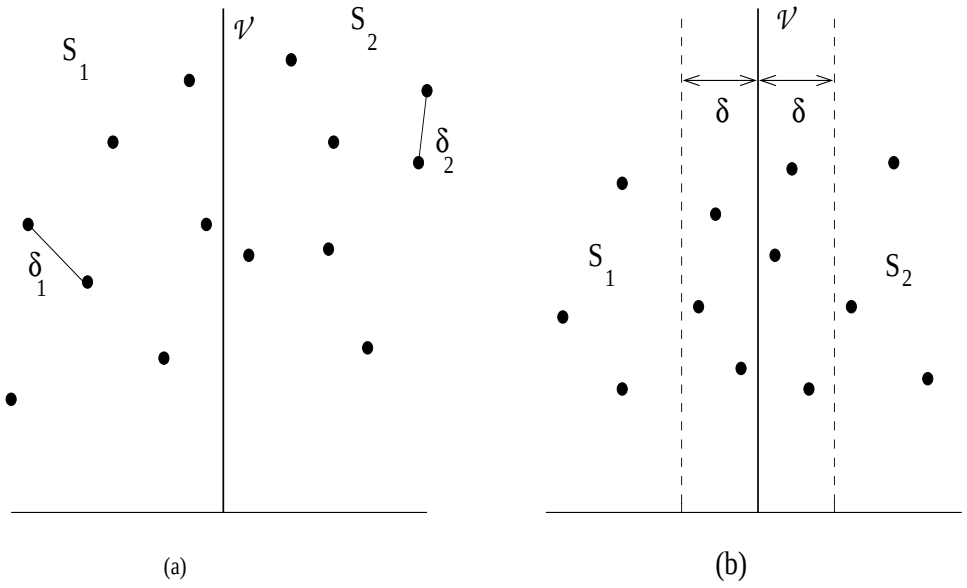


Figure 1: Divide-and-Conquer scheme for closest pair problem.

$|S_1| = |S_2| = n/2$. Let δ_i denote the minimum distance defined by the closest pair of points in $S_i, i = 1, 2$. Observe that the minimum distance defined by the closest pair of points in S is either δ_1, δ_2, or $d(p, q)$ for some $p \in S_1$ and $q \in S_2$. In the former case, we are done. In the latter, points p and q must lie in the vertical strip of width $\delta = \min\{\delta_1, \delta_2\}$ on each side of the separating line $\mathcal{V}$. (Fig. 1) The problem now reduces to that of finding the closest pair between points in S_1 and S_2 that lie inside the strip $\mathcal{L}$ of width 2δ. This subset $\mathcal{L}$ of points possesses a special property, known as *sparsity*, *i.e.*, for each square box[1] of length 2δ the number of points in $\mathcal{L}$ is bounded by a constant $c = 4 \cdot 3^{k-1}$, since in each set S_i, there exists no point that lies in the interior of the δ-ball centered at each point in $S_i, i = 1, 2$[85]. (Fig. 2). It is this sparsity property that enables us to solve the *bi-chromatic closest pair problem* in $O(n)$ time.

The bi-chromatic closest pair problem is defined as follows. Given two sets of red and blue points, denoted R and B, find the closest pair $r \in R$ and $b \in B$, such that $d(r, b)$ is minimum among all possible distances $d(u, v), u \in R, v \in B$. Let $\overline{\mathcal{S}}_i \subseteq S_i$ denote the set of points that lie in the vertical strip. In 2-dimensions, the sparsity property ensures that for each point $p \in \overline{\mathcal{S}}_1$ the number of candidate points $q \in \overline{\mathcal{S}}_2$ for the closest pair is at most six (Fig. 2). We therefore can scan these points $\overline{\mathcal{S}}_1 \cup \overline{\mathcal{S}}_2$ in order along the separating line $\mathcal{V}$ and compute the distance between each point in $\overline{\mathcal{S}}_1$ (resp. $\overline{\mathcal{S}}_2$) scanned and its six candidate points in $\overline{\mathcal{S}}_2$ (resp. $\overline{\mathcal{S}}_1$). The pair that gives the minimum distance δ_3 is the bi-chromatic closest pair. The minimum distance of all pairs of points in S is then equal to $\delta_S = min\{\delta_1, \delta_2, \delta_3\}$.

Since the merge step takes linear time, the entire algorithm takes $O(n \log n)$ time. This idea generalizes to higher dimensions, except that to ensure the sparsity property of the set $\mathcal{L}$, the separating hyperplane should be appropriately chosen so as to obtain an $O(n \log n)$ time algorithm[85], which is asymptotically optimal.

We note that the bi-chromatic closest pair problem is in general more difficult than the closest pair problem. Edelsbrunner and Sharir[46] showed that in 3-dimensions the

[1] A box is a hypercube in higher dimensions.

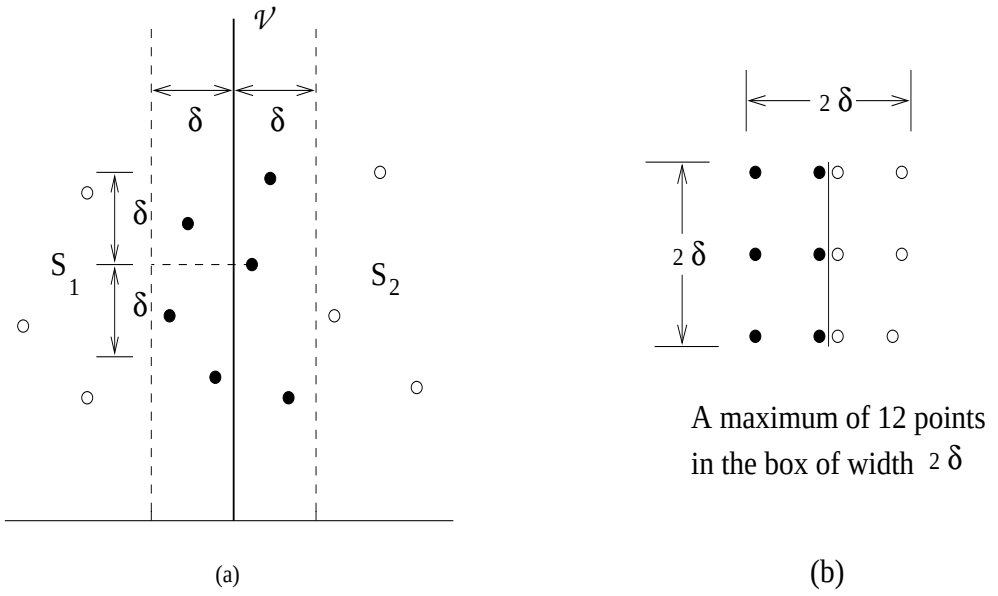


Figure 2: The box of width 2δ dissected by the separating line has at most 12 points; each point in S_2 needs to examine at most 6 points in S_1 to find its closest neighbor.

number of possible closest pairs is $O((|R| \cdot |B|)^{2/3} + |R| + |B|)$. Agarwal *et al.*[2] gave an $O(n^{2(1-1/(\lceil k/2 \rceil + 1))+\epsilon})$ time algorithm and a randomized algorithm with an expected running time of $O(n^{4/3} \log^c n)$ for some constant c, where $n = |R| + |B|$. Only when the two sets possess the sparsity property defined above can the problem be solved in $O(n \log n)$ time, where $n = |R| + |B|$. A more general problem, known as *fixed radius all nearest-neighbor problem in a sparse set*[85], *i.e.*, given a set M of points in $\Re^k$ that satisfies the sparsity condition, find all pairs of points whose distance is less than a given parameter δ, can be solved in $O(|M| \log |M|)$ time[85].

The closest pair of vertices u and v of a simple polygon P such that $\overline{u,v}$ lies totally within P can be found in linear time[57]; $\overline{u,v}$ is also known as a diagonal of P.

2.2 Voronoi Diagrams

The Voronoi diagram $\mathcal{V}(S)$ of a set S of points, called *sites*, $S = \{p_1, p_2, \ldots, p_n\}$ in $\Re^k$ is a partition of $\Re^k$ into Voronoi cells $V(p_i)$, $i = 1, 2, \ldots, n$, such that each cell contains points that are closer to site p_i than to any other site $p_j, j \neq i$, *i.e.*,

$$V(p_i) = \{x \in R^k | d(x, p_i) \leq d(x, p_j) \forall p_j \in R^k, j \neq i.\}$$

In 2-dimensions, $\mathcal{V}(S)$ is a planar graph and is of size linear in $|S|$. In dimensions $k \geq 2$, the total number of d-faces of dimensions $d = 0, 1, \ldots, k-1$, in $\mathcal{V}(S)$ is $O(n^{\lceil d/2 \rceil})$.

Figure 3(a) shows the Voronoi diagram of 16 point sites in 2-dimensions. Figure 3(b) shows the straight-line dual graph of the Voronoi diagram, which is called the Delaunay triangulation (cf. Section 5.2 of the previous Chapter). In this triangulation the vertices are the sites, and two vertices are connected by an edge if their Voronoi cells are adjacent.

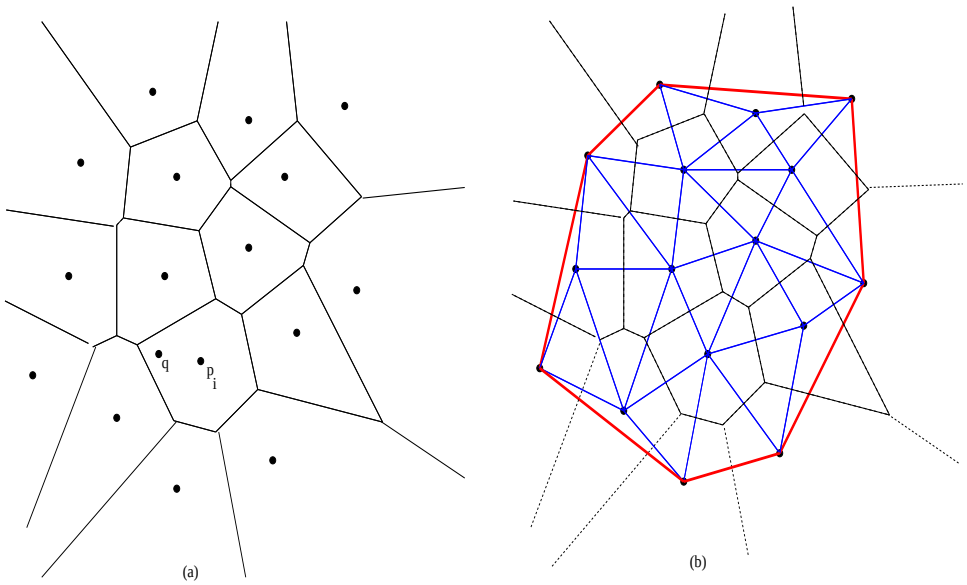


Figure 3: The Voronoi diagram of a set of 16 points in the plane.

2.2.1 Construction of Voronoi Diagrams in 2-Dimensions

The Voronoi diagram possesses many proximity properties. For instance, for each site p_i, the closest site must be among those whose Voronoi cells are adjacent to $V(p_i)$. Thus the closest pair problem for S in $\Re^2$ can be solved in linear time after the Voronoi diagram has been computed. Since this pair of points must be adjacent in the Delaunay triangulation, all one has to do is to examine all adjacent pairs of points and report the pair with the smallest distance. A divide-and-conquer algorithm to compute the Voronoi diagram of a set of points in the L_p-metric for all $1 \leq p \leq \infty$ is known[64]. There are a rich body of literature concerning the Voronoi diagram. The interested reader is referred to recent surveys[41, 88].

We give below a brief description of a plane-sweep algorithm, known as the *wavefront approach*, due to Dehne and Klein[39]. Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of point sites in $\Re^2$ sorted in ascending x-coordinate value, *i.e.*, $x(p_1) < x(p_2) < \ldots < x(p_n)$. Consider that we sweep a vertical line $\mathcal{L}$ from left to right and as we sweep $\mathcal{L}$, we compute the Voronoi diagram $\mathcal{V}(S_t)$, where

$$S_t = \{p_i \in S | x(p_i) < t\} \cup \{\mathcal{L}_t\}.$$

Here $\mathcal{L}_t$ denotes the vertical line whose x-coordinate equals t. As is well known, $\mathcal{V}(S_t)$ will contain not only straight line segments, which are portions of perpendicular bisectors of two point sites, but also parabolic curve segments, which are portions of **bisectors** of one point site and $\mathcal{L}_t$. The *wavefront* W_t, consisting of a sequence of parabolae, called *waves*, is the boundary of the Voronoi cell $V(\mathcal{L}_t)$ with respect to S_t. Figs. 4(a) and (b) illustrate two instances, $\mathcal{V}(S_t)$ and $\mathcal{V}(S_{t'})$. Those Voronoi cells that do not contribute to the wavefront are final, whereas those that do will change as $\mathcal{L}$ moves to the right. There are two possible *events* at which the wavefront needs an *update*. One, called *site event*, is when a site is hit by $\mathcal{L}$ and a *new* wave appears. The other, called *spike event*, is when an old wave disappears. Let p_i and p_j be two sites such that the associated waves are adjacent in W_t. The bisector of p_i and p_j defines an edge of $\mathcal{V}(S_t)$ to the left of W_t. Its extension into the cell $V(\mathcal{L}_t)$

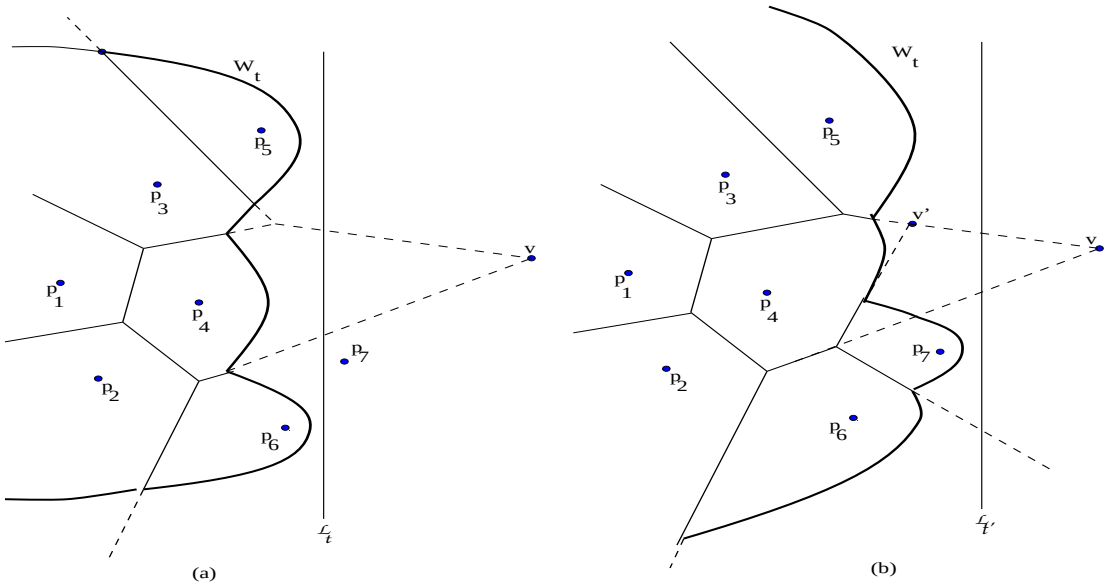


Figure 4: The Voronoi diagrams of $\mathcal{V}(S_t)$ and $\mathcal{V}(S_{t'})$

is called a *spike*. The spikes can be viewed as tracks along which two neighboring waves travel. A wave disappears from W_t, once it has reached the point where its two neighboring spikes intersect. In Fig. 4(a) dashed lines are spikes and v is a potential spike event point. Without p_7 the wave of p_3 would disappear first and then the wave of p_4. After p_7, a site event, has occurred, a new point v' will be created and it defines an earlier spike event than v. v' will be a spike event point at which the wave of p_4 disappears and waves of p_5 and p_7 become adjacent. Note that the spike event corresponding to v' does not occur at $\mathcal{L}_t$, when $t = x(v')$. Instead, it occurs at $\mathcal{L}_t$, when $t = x(v') + d(v', p_4)$. If there is no site event between $\mathcal{L}_{x(p_7)}$ and $\mathcal{L}_t$, then the wave of p_4 will disappear. It is not difficult to see that after all site events and spikes events have been processed at time τ, $\mathcal{V}(S)$ is identical to $\mathcal{V}(S_\tau)$ with the wavefront removed.

Since the waves in W_t can be stored in a **height-balanced binary search tree** and the site events and spike events can be maintained as a **priority queue**, the overall time and space needed are $O(n \log n)$ and $O(n)$ respectively.

Although $\Omega(n \log n)$ is a lower bound for computing the Voronoi diagram for an arbitrary set of n sites, this lower bound does not apply to special cases, *e.g.*, when the sites are on the vertices of a convex polygon. In fact the Voronoi diagram of a convex polygon can be computed in linear time[6]. This demonstrates further that additional properties of the input can sometimes help reduce the complexity of the problem.

2.2.2 Construction of Voronoi Diagrams in Higher Dimensions

The Voronoi diagrams in $\Re^k$ are related to the convex hulls $\Re^{k+1}$ via a **geometric duality** transformation. Consider a set S of n sites in $\Re^k$, which is the hyperplane $\mathcal{H}^0$ in $\Re^{k+1}$ such that $x_{k+1} = 0$, and a paraboloid $\mathcal{P}$ in $\Re^{k+1}$ represented as $x_{k+1} = x_1^2 + x_2^2 + \cdots + x_k^2$. Each site $p_i = (\mu_1, \mu_2, \ldots, \mu_k)$ is transformed into a hyperplane $\mathcal{H}(p_i)$ in $\Re^{k+1}$ denoted as $x_{k+1} = 2\sum_{j=1}^{k} \mu_j x_j - (\sum_{j=1}^{k} \mu_j^2)$. That is, $\mathcal{H}(p_i)$ is tangent to the paraboloid $\mathcal{P}$ at point

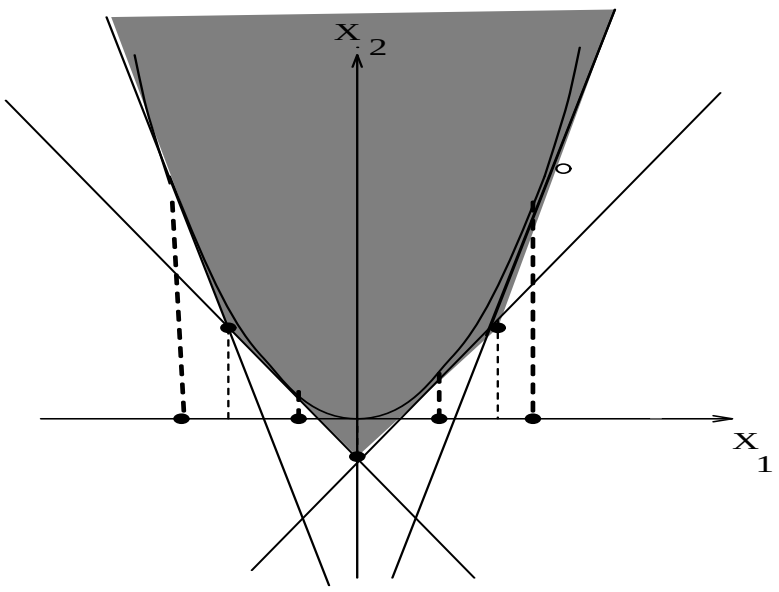


Figure 5: The paraboloid transformation of a site in 1-dimension to a line tangent to a parabola.

$\mathcal{P}(p_i) = (\mu_1, \mu_2, \ldots, \mu_k, \mu_1^2 + \mu_2^2 + \cdots + \mu_k^2)$, which is just the vertical projection of site p_i onto the paraboloid $\mathcal{P}$. See Fig. 5 for an illustration of the transformation in 1-dimension. The half-space defined by $\mathcal{H}(p_i)$ and containing the paraboloid $\mathcal{P}$ is denoted as $\mathcal{H}^+(p_i)$. The intersection of all half-spaces $\bigcap_{i=1}^{n} \mathcal{H}^+(p_i)$ is a convex body and the boundary of the convex body is denoted $CH(\mathcal{H}(S))$. Any point $q \in \Re^k$ lies in the Voronoi cell $V(p_i)$ if the vertical projection of q onto $CH(\mathcal{H}(S))$ is contained in $\mathcal{H}(p_i)$. The distance between point q and its closest site p_i can be shown to be equal to the square root of the vertical distance between its vertical projection $\mathcal{P}(q)$ on the paraboloid $\mathcal{P}$ and on $CH(\mathcal{H}(S))$. Moreover every κ-face of $CH(\mathcal{H}(S))$ has a vertical projection on the hyperplane $\mathcal{H}^0$ equal to the κ-face of the Voronoi diagram of S in $\mathcal{H}^0$.

We thus obtain the result which follows from the theorem for the convex hull in the previous Chapter.

Theorem 1 *The Voronoi diagram of a set S of n points in $\Re^k$, $k \geq 2$ can be computed in $O(n \log \mathcal{H})$ time for $k = 2$, and in $O(n \log \mathcal{H} + (n\mathcal{H})^{1-1/(\lfloor (k+1)/2 \rfloor + 1)} \log^{O(1)} n)$ time for $k > 2$, where $\mathcal{H}$ is the number of i-faces, $i = 0, 1, \ldots, k$.*

It has recently been shown that the Voronoi diagram in $\Re^k$, for $k = 3, 4$, can be computed in $O((n + \mathcal{H}) \log^{k-1} \mathcal{H})$ time[11].

2.2.3 Farthest Neighbor Voronoi Diagram

The Voronoi diagram defined in Section 2.2 is also known as the *nearest neighbor* Voronoi diagram. The *nearest neighbor* Voronoi diagram partitions the space into cells such that each site has its own cell, which contains all points that are closer to this site than to any other site. A variation of this partitioning concept is a partition of the space into cells, each of which is associated with a site, and contains all points that are *farther* from the site than

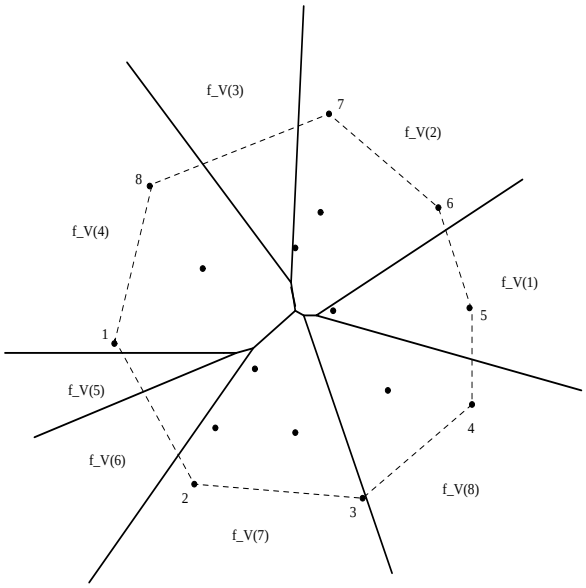


Figure 6: The farthest neighbor Voronoi diagram of a set of 16 sites in the plane.

from any other site. This diagram is called the *farthest neighbor* Voronoi diagram. Unlike the *nearest neighbor* Voronoi diagram, a farthest neighbor Voronoi diagram only has a subset of sites which have a Voronoi cell associated with them. Those sites that have a non-empty Voronoi cell are those that lie on the convex hull of S. A similar partitioning of the space is known as the order κ-*nearest neighbor* Voronoi diagram, in which each Voronoi cell is associated with a subset of κ sites in S for some fixed integer κ such that these κ sites are the closest among all other sites. For $\kappa = 1$ we have the *nearest neighbor* Voronoi diagram, and for $\kappa = n - 1$ we have the *farthest neighbor* Voronoi diagram. The construction of the order κ-*nearest neighbor* Voronoi diagram in the plane can be found in *e.g.*,[85]. The order κ Voronoi diagrams in $\Re^k$ are related to the levels of hyperplane arrangements in $\Re^{k+1}$ using the paraboloid transformation discussed in Section 2.2.2. See *e.g.*, [1] for details. Below is a discussion of the *farthest neighbor* Voronoi diagram in 2-dimensions.

Given a set S of sites $s_1, s_2, \ldots, s_n$, the f-neighbor Voronoi cell of site s_i is the locus of points that are farther from s_i than from any other site $s_j, i \neq j$, *i.e.*,

$$f\_V(s_i) = \{p \in \Re^2 | d(p, s_i) \geq d(p, s_j), s_i \neq s_j\}.$$

The union of these f-neighbor Voronoi cells is called the *farthest neighbor* Voronoi diagram of S. Fig. 6 shows the *farthest neighbor* Voronoi diagram for a set of 16 sites. Note that only sites that are on the convex hull $CH(S)$ will have a non-empty f-neighbor Voronoi cell[85] and that all the f-neighbor Voronoi cells are unbounded.

Since the *farthest neighbor* Voronoi diagram in the plane is related to the convex hull of the set of sites, one can use the *divide-and-marriage-before-conquest* paradigm to compute the *farthest neighbor* Voronoi diagram of S in 2-dimensions in time $O(n \log \mathcal{H})$, where $\mathcal{H}$ is the number of sites on the convex hull. Once the convex hull is available, the linear time algorithm[6] for computing the Voronoi diagram for a convex polygon can be applied.

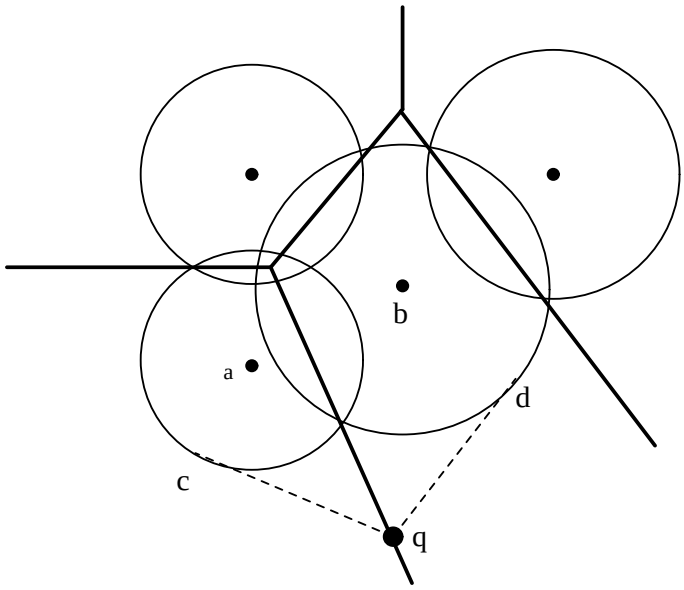


Figure 7: The power diagram in 2-dimensions. $\delta(q,a) = \delta(q,b) =$ length of $\overline{q,c}$.

2.2.4 Weighted Voronoi Diagrams

When the sites have weights such that the distance from a point to the sites is weighted, the structure of the Voronoi diagram can be drastically different than the unweighted case. We consider a few examples.

Example 1 Power Diagrams

Suppose each site s in $\Re^k$ is associated with a non-negative weight, w_s. For an arbitrary point p in $\Re^k$ the weighted distance from p to s is defined as

$$\delta(s,p) = d(s,p)^2 - w_s^2$$

If w_s is positive, and if $d(s,p) \geq w_s$, then $\sqrt{\delta(s,p)}$ is the length of the tangent of p to the ball, $b(s)$, of radius w_s and centered at s. $\delta(s,p)$ is also called the *power of* p with respect to the ball $b(s)$. The locus of points p equidistant from two sites $s \neq t$ of equal weight, will be a hyperplane called the *chordale* of s and t. See Fig. 7. Point q is equidistant to sites a and b and the distance is the length of the tangent line $\overline{q,c} = \overline{q,d}$.

The power diagram in 2-dimensions can be used to compute the contour of the union of n disks, and the connected components of n disks in $O(n \log n)$ time, and in higher dimensions it can be used to compute the union or intersection of n axis-parallel cones in $\Re^k$ with apexes in a common hyperplane in time $O(CH_{k+1}(n))$, the multiplicative weighted nearest-neighbor Voronoi diagram (defined below) for n points in $\Re^k$ in time $O(CH_{k+2}(n))$, and the Voronoi diagrams for n spheres in $\Re^k$ in time $O(CH_{k+2}(n))$, where $CH_\ell(n)$ denotes the time for constructing the convex hull of n points in $\Re^\ell$[88]. For the best time bound for $CH_\ell(n)$ consult Section 2 of the previous Chapter. For more results on the union of spheres and the volumes see [45].

Example 2 Multiplicative-Weighted Voronoi Diagrams

Suppose each site $s \in \Re^k$ is associated with a positive weight w_s. The distance from a point $p \in \Re^k$ to s is defined as

$$\delta_{multi-w}(s,p) = d(p,s)/w_s$$

In 2-dimensions, the locus of points equidistant to two sites $s \neq t$ is a disk, if $w_s \neq w_t$, and a perpendicular bisector of line segment $\overline{s,t}$, if $w_s = w_t$. Each cell associated with a site s consists of all points closer to s than to any other site and may be disconnected. In the worst case the multiplicative-weighted nearest neighbor Voronoi diagram of a set S of n points in 2-dimensions can have $O(n^2)$ regions and can be computed in $O(n^2)$ time. But in 1-dimension, the diagram can be computed optimally in $O(n \log n)$ time. On the other hand the multiplicative-weighted farthest neighbor Voronoi diagram has a very different characteristic. Each Voronoi cell associated with a site remains connected, and the size of the diagram is still linear in the number of sites. An $O(n \log^2 n)$ time algorithm for constructing such a diagram is given in [69]. See [89] for more applications of the diagram.

Example 3 Additive-Weighted Voronoi Diagrams

Suppose each site $s \in \Re^k$ is associated with a positive weight w_s. The distance of a point $p \in \Re^k$ to a site s is defined as

$$\delta_{add-w}(s,p) = d(p,s) - w_s$$

In 2-dimensions, the locus of points equidistant to two sites $s \neq t$ is a branch of a hyperbola, if $w_s \neq w_t$, and a perpendicular bisector of line segment $\overline{s,t}$ if $w_s = w_t$. The Voronoi diagram has properties similar to the ordinary unweighted diagram. For example, each cell is still connected and the size of the diagram is linear. If the weights are positive, the diagram is the same as the Voronoi diagram of a set of spheres centered at site s and of radius w_s and in 2-dimensions this diagram for n disks can be computed in $O(n \log n)$ time[15, 81] and in $k \geq 3$ one can use the notion of power diagram (cf. Example 1) to compute the diagram[88].

2.2.5 Generalizations of Voronoi Diagrams

We consider two variations of Voronoi diagrams that are of interest and have applications.

Example 4 Geodesic Voronoi Diagrams

The nearest neighbor geodesic Voronoi diagram is a Voronoi diagram of sites in the presence of obstacles. The distance from point p to a site s, called the *geodesic* distance between p and s, is the length of the shortest path from p to s avoiding all the obstacles (cf. Section 6.1). The locus of points equidistant to two sites s and t is in general a collection of hyperbolic segments. The cell associated with a site is the locus of points whose geodesic distance to the site is shorter than to any other site[84]. The farthest neighbor geodesic Voronoi diagram can be similarly defined. Efficient algorithms for computing either kind of geodesic Voronoi diagram for k point sites in an n-sided simple polygon in $O((n+k)\log(n+k))$ time can be found in [84]. Fig. 8 illustrates the geodesic Voronoi diagram of a set of point sites within a simple polygon; the whole shaded region is $V(s_i)$.

Example 5 Skew Voronoi Diagrams

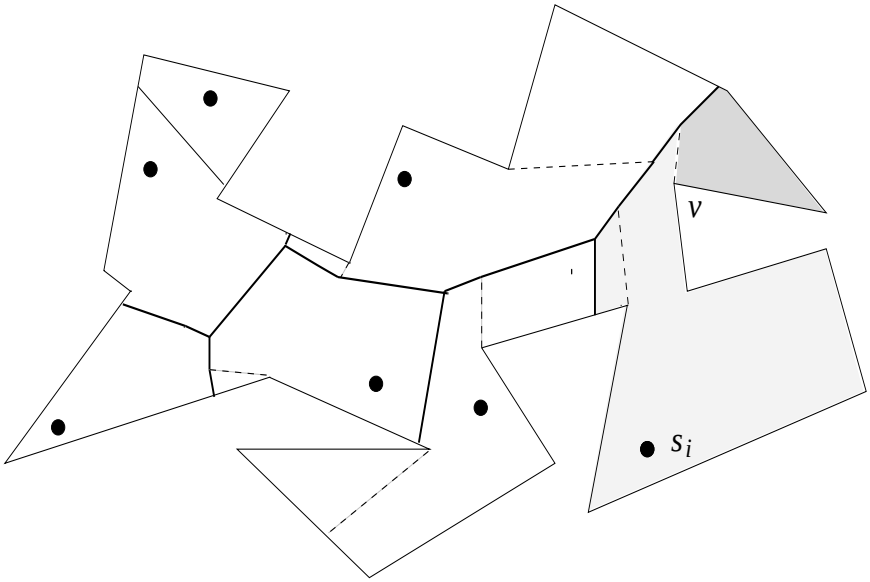


Figure 8: The geodesic Voronoi diagram within a simple polygon.

Most recently a *directional* distance function between two points in the plane is introduced in [8] that models a more realistic distance measure. The distance, called *skew distance*, *from* point p *to* point q is defined as

$$\tilde{d}(p,q) = d(p,q) + k \cdot d_y(p,q),$$

where $d_y(p,q) = y(q) - y(p)$, and $k \geq 0$ is a parameter. This distance function is *asymmetric* and satisfies $\tilde{d}(p,q) + \tilde{d}(q,p) = 2d(p,q)$, and the triangle inequality. Imagine we have a tilted plane $\mathcal{T}$ obtained by rotating the xy-plane by an angle α about the x-axis. The height (z-coordinate) $h(p)$ of a point p on $\mathcal{T}$ is related to its y-coordinate by $h(p) = y(p) \cdot \sin\alpha$.

The distance function defined above reflects the cost that is proportional to the difference of their heights; the distance is *smaller* going *downhill* than going *uphill*. That is, the distance from p to q defined as $\tilde{d}(p,q) = d(p,q) + \kappa \cdot (h(q) - h(p))$ for $\kappa > 0$ serves this purpose. $\tilde{d}(p,q)$ is less than $\tilde{d}(q,p)$ if $h(q)$ is smaller than $h(p)$.

Because the distance is *directional*, one can define two kinds of Voronoi diagrams defined by the set of sites. A skew Voronoi cell *from* a site p, $\mathcal{V}_{from}(p)$ is defined as the set of points that are closest to p than to any other site. That is,

$$\mathcal{V}_{from}(p) = \{x | \tilde{d}(p,x) \leq \tilde{d}(q,x)\}$$

for all $q \neq p$. Similarly one can define a skew Voronoi cell *to* a site p as follows.

$$\mathcal{V}_{to}(p) = \{x | \tilde{d}(x,p) \leq \tilde{d}(x,q)\}$$

for all $q \neq p$.

The collection of these Voronoi cells for all sites is called the *skew* (or *directional*) Voronoi diagram.

For each site p we define a r-disk centered at p, denoted $from_r(p)$ to be the set of points to which the skew distance from p is r. That is, $from_r(p) = \{x | \tilde{d}(p,x) = r\}$. Symmetrically

we can also define a r-disk centered at p, denoted $to_r(p)$ to be the set of points from which the skew distance to p is r. That is, $to_r(p) = \{x | \tilde{d}(x,p) = r\}$. The subscript r is omitted, when $r = 1$. It can be shown that $to_r(p)$ is just a mirror reflection of $from_r(p)$ about the horizontal line passing through p. We shall consider only the skew Voronoi diagram which is the collection of the cells $\mathcal{V}_{from}(p)$ for all $p \in S$.

Lemma 1 *For $k > 0$, the unit-disk from(p), is a conic with focus p, directrix the horizontal line at y-distance $1/k$ above p, and eccentricity k. Thus from(p) is an ellipse for $k < 1$, a parabola for $k = 1$, and a hyperbola for $k > 1$. For $k = 0$, from(p) is a disk with center p(which can be regarded as an ellipse of eccentricity zero).*

Note that when k equals 0, the skew Voronoi diagram reduces to the ordinary nearest neighbor Voronoi diagram. When $k < 1$, it leads to known structures: By Lemma 1, the skew distance $\tilde{d}$ is a *convex distance function* and the Voronoi diagrams for convex distance functions are well-studied (see *e.g.,* [88]). They consist of $O(n)$ edges and vertices, and can be constructed in time $O(n \log n)$ by divide-and-conquer.

When $k \geq 1$, since the unit disks are no longer bounded, the skew Voronoi diagrams have different behavior from the ordinary ones. As it turns out, some of the sites do not have nonempty skew Voronoi cells in this case. In this regard, it looks like ordinary farthest neighbor Voronoi diagram discussed earlier.

Let $L_0(p,k)$ denote the locus of points x such that $\tilde{d}(p,x) = 0$. It can be shown that for $k = 1$ $L_0(p,k)$ is a vertical line emanating downwards from p; and for $k > 1$, it consists of two rays, emanating from and extending below p, with slopes $1/(\sqrt{k^2-1})$ and $-1/(\sqrt{k^2-1})$ respectively. Let $N(p,k)$ denote the area below $L_0(p,k)$ (for $k > 1$). Let the *0-envelope*, $E_0(S)$, be the upper boundary of the union of all $N(p,k)$ for $p \in S$. $E_0(S)$ is the upper envelope of the graphs of all $L_0(p,k)$, when being seen as functions of the x-coordinate. For each point u lying above $E_0(S)$, we have $\tilde{d}(p,u) > 0$ for all $p \in S$, and for each point v lying below $E_0(S)$, there is at least one $p \in S$ with $\tilde{d}(p,v) < 0$. See Figure 9 for an example of a 0-envelope (shown as the dashed polygonal line) and the corresponding skew Voronoi diagram. Note that the skew Voronoi cells associated with sites q and t are empty. The following results are obtained[8].

Lemma 2 *For $k > 1$, the 0-envelope $E_0(S)$ of a set S of n sites can be computed in $O(n \log \mathcal{H})$ time and $O(n)$ space, where $\mathcal{H}$ is the number of edges of $E_0(S)$.*

Lemma 3 *Let $p \in S$ and $k > 1$. Then $\mathcal{V}_{from}(p) \neq \emptyset$ if and only if $p \in E_0(S)$. $\mathcal{V}_{from}(p)$ is unbounded if and only if p lies on the upper hull of $E_0(S)$. For $k = 1$ $\mathcal{V}_{from}(p)$ is unbounded for all p.*

Theorem 2 *For any $k \geq 0$ the skew Voronoi diagram for n sites can be computed in $O(n \log \mathcal{H})$ time and $O(n)$ space, where $\mathcal{H}$ is the number of non-empty skew Voronoi cells in the resulting Voronoi diagram.*

The sites mentioned so far are point sites. They can be of different shapes. For instance, they can be line segments, or polygonal objects. The Voronoi diagram for the edges of a simple polygon P that divides the interior of P into Voronoi cells is also known as the *medial axis*, or *skeleton* of P[85]. The distance function used can also be convex distance function or other norms.

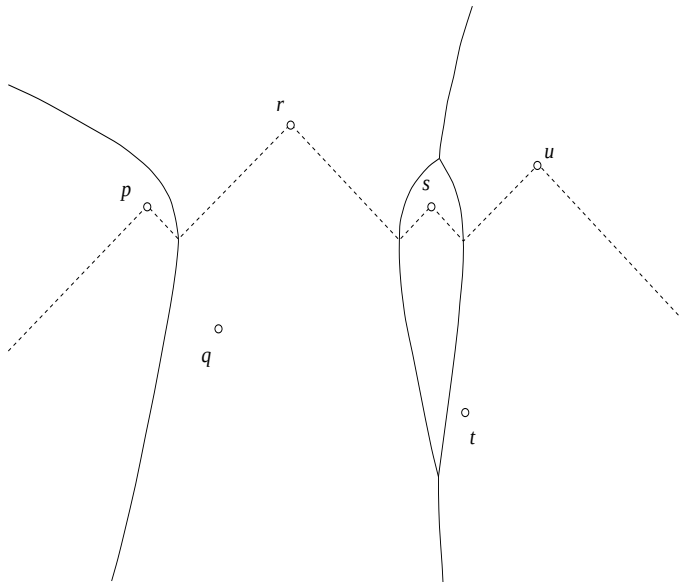


Figure 9: The 0-envelope and the skew Voronoi diagram when $k = 1.5$

3 Optimization

The geometric optimization problems arise in operations research, VLSI layout, and other engineering disciplines. We give a brief description of a few problems in this category that have been studied in the past.

3.1 Minimum Cost Spanning Tree

The minimum (cost) spanning tree, *MST*, of an undirected, weighted graph $G(V, E)$, in which each edge has a nonnegative weight, is a well studied problem in graph theory and can be solved in $O(|E| \log |V|)$ time[85]. When cast in the Euclidean or other L_p-metric plane in which the input consists of a set S of n points, the complexity of this problem becomes different. Instead of constructing a *complete* graph with edge weight being the distance between its two endpoints, from which to extract an *MST*, a sparse graph, known as the *Delaunay triangulation* of the point set, is computed. The Delaunay triangulation of S, which is a planar graph, is the straight-line dual of the Voronoi diagram of S. That is, two points are connected by an edge if and only if the Voronoi cells of these two sites share an edge. (cf. Section 5 of the previous Chapter) It can be shown that the *MST* of S is a subgraph of the Delaunay triangulation. Since the *MST* of a planar graph can be found in linear time[85], the problem can be solved in $O(n \log n)$ time. In fact, this is asymptotically optimal, as the closest pair of the set of points must define an edge in the *MST*, and the closest pair problem is known to have an $\Omega(n \log n)$ lower bound, as mentioned in Section 2.1.

This problem in dimensions 3 or higher can be solved in subquadratic time, *e.g.*, in 3-dimensions, $O((n \log n)^{1.5})$ time is sufficient and in $k \geq 3$ dimensions $O(n^{2(1-1/(\lceil k/2 \rceil + 1))+\epsilon})$ time suffices[2]. Interesting enough, if we want to find an *MST* that spans at least k nodes in a planar graph (or in the Euclidean plane), for some parameter $k \leq n$, then the problem,

called *k-MST* problem, is NP-hard[86]. Approximation algorithms for the *k-MST* problem can be found in [19, 52].

3.2 Steiner Minimum Tree

The *Steiner minimum tree*, *SMT*, of a set of vertices $S \subseteq V$ in an undirected weighted graph $G(V, E)$ is a spanning tree of $S \cup Q$ for some $Q \subseteq V$ such that the total weight of the spanning tree is minimum. This problem differs from *MST* in that we need to identify a set $Q \subseteq V$ of so-called *Steiner vertices* so that the total cost of the spanning tree is minimized. Of course if $S = V$, *SMT* is the same as *MST*. It is the identification of the Steiner vertices that makes this problem intractable. In the plane, we are given a set S of points and are to find a shortest tree interconnecting points in S, while additional Steiner points are allowed. Both Euclidean and rectilinear (L_1-metric) SMT problems are known to be NP-hard. In the geometric setting, the rectilinear SMT problem arises mostly in VLSI net routing, in which a number of terminals need to be interconnected using horizontal and vertical wire segments using the shortest wire length. As this problem is intractable, heuristics are proposed. For more information, the reader is referred to a recent special issue of *Algorithmica* on Steiner trees, edited by Hwang[59]. Most heuristics for the L_1 SMT problem are based on a classical theorem, known as the *Hanan grid theorem*, which states that the Steiner points of an SMT must be at the grid defined by drawing horizontal and vertical lines through each of the given points. However, when the number of orientations permitted for routing is greater than 2, the Hanan grid theorem no longer holds true. In [68] Lee and Shen established a *multi-level grid* theorem, which states that the Steiner points of an SMT for n points must be at the grid defined by drawing λ lines in the feasible orientation recursively for up to $n-2$ levels, where λ denotes the number of orientations of the wires allowed in routing. That is, the given points are assumed to be at the 0-th level. At each level, λ lines in the feasible orientations are drawn through each *new* grid points created at the previous level. In this λ-geometry plane feasible orientations are assumed to make an angle $i\pi/\lambda$ with the positive x-axis. For the rectilinear case, $\lambda = 2$. Fig 10 shows that Hanan grid is insufficient for determining a Steiner SMT for $\lambda = 3$. Steiner point s_3 does not lie on the Hanan grid. $\lambda = 4$.

Definition 1 *The performance ratio of any approximation $\mathcal{A}$ in metric space $\mathcal{M}$ is defined as*

$$\rho_{\mathcal{M}}(\mathcal{A}) = \inf_{P \in \mathcal{M}} \frac{L_s(P)}{L_{\mathcal{A}}(P)}$$

where $L_s(P)$ and $L_{\mathcal{A}}(P)$ denote, respectively, the lengths of a Steiner minimum tree and of the approximation $\mathcal{A}$ on P in space $\mathcal{M}$. When the MST is the approximation, the performance ratio is known as the Steiner ratio, denoted simply as ρ.

It is well-known that the Steiner ratios for the Euclidean and rectilinear SMTs are $\frac{\sqrt{3}}{2}$ and $\frac{2}{3}$ respectively[59]. The λ-*Steiner ratio*[2] for the λ-geometry SMTs is no less than $\frac{\sqrt{3}\cos(\pi/2\lambda)}{2}$. The following interesting result regarding Steiner ratio is reported in [68], which shows that the Steiner ratio is not an increasing function from $\frac{2}{3}$ to $\frac{\sqrt{3}}{2}$, as λ varies from 2 to ∞.

[2]The λ-*Steiner ratio* is defined as the greatest lower bound of the length of SMT over the length of MST in the λ-*geometry* plane.

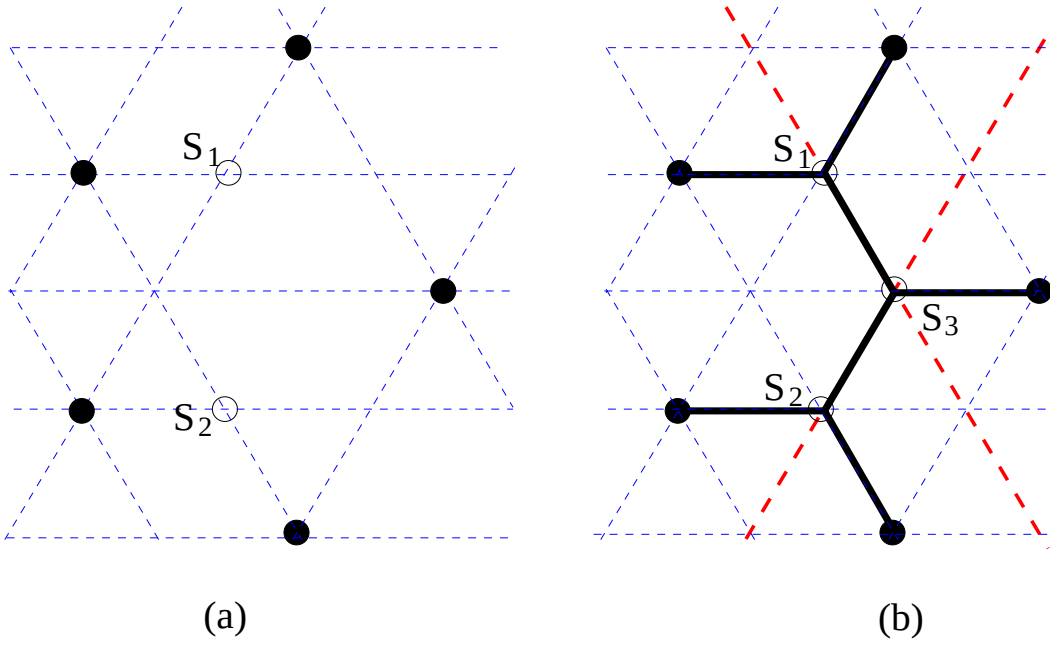


Figure 10: Hanan grid theorem fails for $\lambda = 3$. Steiner point s_3 does not lie on the Hanan grid.

Theorem 3 *The λ-Steiner ratio is $\frac{\sqrt{3}}{2}$, when λ is multiple of 6, and $\frac{\sqrt{3}\cos(\pi/2\lambda)}{2}$ when λ is multiple of 3 but not multiple of 6.*

3.3 Minimum Diameter Spanning Tree

The minimum *diameter* spanning tree *MDST* of an undirected weighted graph $G(V, E)$ is a spanning tree such that its **diameter**, *i.e.*, total weight of the longest path in the tree, is minimum. This arises in applications to communication network where a tree is sought such that the maximum delay, instead of the total cost, is to be minimized. Using a graph-theoretic approach one can solve this problem in $O(|E||V| \log |V|)$ time. However, by the triangle inequality one can show that there exists an *MDST* such that the longest path in the tree consists of no more than *three* segments[58]. Based on this an $O(n^3)$ time algorithm was obtained.

Theorem 4 *Given a set S of n points, the minimum diameter spanning tree for S can be found in $\theta(n^3)$ time and $O(n)$ space.*

We remark that the problem of finding a spanning tree whose total cost **and** the diameter are both bounded is NP-complete[58]. In [86] the problem of finding a *minimum diameter cost spanning tree* is studied. In this problem for each pair of vertices v_i and v_j there is a weighting function $w_{i,j}$ and the *diameter cost* of a spanning tree is defined to be the maximum over $w_{i,j} * d_{i,j}$, where $d_{i,j}$ denotes the distance between vertices v_i and v_j. To find a spanning tree with minimum diameter cost as defined above is shown to be NP-hard[86].

Another similar problem that arises in VLSI clock tree routing is to find a tree from a source to multiple sinks such that every source-to-sink path is a *shortest* rectilinear path and

the total wire length is to be minimized. This problem, also known as *rectilinear Steiner arborescence problem* (see [59]). is still not known to be solvable in polynomial time. The problem is widely believed to be NP-hard. Recently we have shown that the problem of finding a minimum spanning tree such that the longest source-to-sink path is bounded by a given parameter is NP-complete.

3.4 Minimum Enclosing Circle

Given a set S of points the problem is to find a smallest disk enclosing the set. This problem is also known as the (unweighted) 1-center problem. That is, find a center such that the maximum distance from the center to the points in S is minimized. More formally, we need to find the center $c \in \Re^2$ such that $\max_{p_j \in S} d(c, p_j)$ is minimized. The weighted 1-center problem, in which, the distance function $d(c, p_j)$ is multiplied by the weight w_j, is a well-known min-max problem, also referred to as the *emergency center problem* in operations research. In 2-dimensions, the 1-center problem can be solved in $O(n)$ time. The minimum enclosing ball problem in higher dimensions is also solved by using linear programming technique[97]. The general p-center problem, *i.e.*, finding p circles whose union contains S such that the maximum radius is minimized, is known to be NP-hard. For a special case when $p = 2$ Eppstein[48] gave an $O(n \log^2 n)$ randomized algorithm based on parametric search technique. For the problem of finding a minimum enclosing ellipsoid for a point set in $\Re^k$ and other types of geometric *location* problem see [47, 97].

3.5 Largest Empty Circle

This problem, in contrast to the minimum enclosing circle problem, is to find a circle centered in the interior of the convex hull of the set S of points that does not contain any given point and the radius of the circle is to be maximized. This is mathematically formalized as a max-min problem, the minimum distance from the center to the set is maximized. The weighted version is also known as the *obnoxious center* problem in facility location. For the unweighted version the center must be either at a vertex of the Voronoi diagram for S in the convex hull, or at the intersection of a Voronoi edge and the boundary of the convex hull. $O(n \log n)$ time is sufficient for this problem. Following the same strategy one can solve the largest *empty square* problem for S in $O(n \log n)$ time as well, using the Voronoi diagram in the L_∞-metric[64]. The time complexity of the algorithm is asymptotically optimal, as the maximum gap problem, *i.e.,* finding the maximum gap between two consecutive numbers on the real line, which requires $\Omega(n \log n)$ time, is reducible to this problem[85]. In contrast to the minimum enclosing ellipsoid problem, the largest empty ellipsoid problem has also been studied[43].

3.6 Largest Empty Rectangle

In Section 2.4 of the previous Chapter we mentioned the *smallest enclosing rectangle* problem. Here we look at the problem of finding a largest rectangle that is empty. That is, given a rectangle containing a set S of n points find the largest area sub-rectangle with sides parallel to those of the original rectangle, whose interior contains no points from S[7]. The problem also arises in document analysis of printed-page layout[16] in which *white* space in the black-and-white image of the form of a maximal empty rectangle is to be recognized. A related problem, called *largest empty corner rectangle* problem is that given two subsets S_l and S_r of S separated by a vertical line, find the largest rectangle containing no other

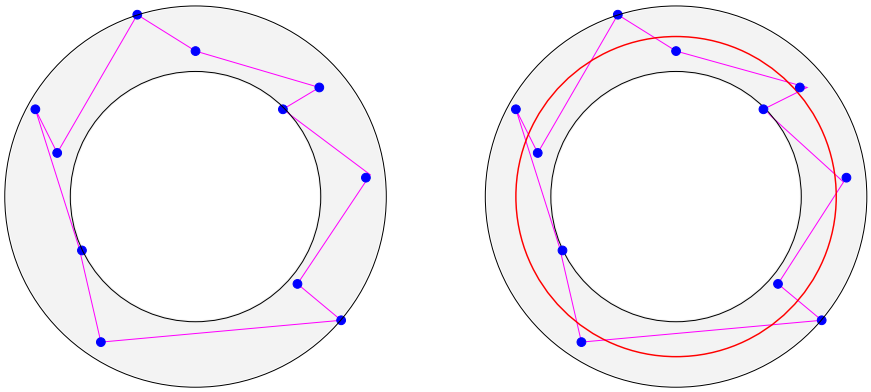
points in S such that lower left corner and upper right corner of the rectangle are in S_l and S_r respectively. This problem can be solved in $O(n \log n)$ time, where $n = |S|$, using fast matrix searching technique (cf. Section 4 of the previous Chapter). With this as a subroutine, one can solve the largest empty rectangle problem in $O(n \log^2 n)$ time[7]. When the points define a rectilinear polygon that is orthogonally convex, the largest empty rectangle that can fit inside the polygon can be found in $O(n\alpha(n))$ time, where $\alpha(n)$ is the slowly growing inverse of Ackermann's function using a result of Klawe and Kleitman[63]. When the polygon P is arbitrary and may contain *holes*, Daniels *et al.*[37] gave an $O(n \log^2 n)$ algorithm, for finding the largest empty rectangle in P, which matches the best known result when P is a rectilinear polygon[7].

3.7 Minimum-Width Annulus

Given a set of S of n points find an annulus (defined by two concentric circles) whose center lies internal to the convex hull of S such that the *width* of the annulus is minimized. This problem arises in dimensional tolerancing and metrology which deals with the specification and measurement of error tolerances in geometric shapes. To measure if a manufactured circular part is *round*, an **ANSI** standard is to use the width of an annulus covering the set of points obtained from a number of measurements. This is known as the *roundness* problem[94, 51]. It can be shown that the center of the annulus can be located at the intersection of the nearest neighbor and the farthest neighbor Voronoi diagrams as discussed in Section 2.2[51]. The center can be computed in $O(n \log n)$ time[51]. If the input is defined by a simple polygon P with n vertices, then the problem is to find a minimum-width annulus that contains the boundary of P. The center of the smallest annulus can be located at the medial axis of P[94]. In particular, the problem can be solved in $O(n \log n + k)$, where k denotes the number of intersection points of the medial axis of the simple polygon and the farthest neighbor Voronoi diagram of the vertices of P. In [94] k is shown to be $\theta(n^2)$. However, if the polygon is convex, one can solve this problem in linear time[94]. Note that the minimum-width annulus problem is equivalent to the *best circle approximation* problem, in which a circle approximating a given shape (or a set of points) is sought such that the *error* is minimized. The error of the approximating circle is defined to be the maximum over all distances between points in the set and the approximating circle. To be more precise, the error is equal to one half of width of the smallest annulus. See Fig. 11.

If the center of the smallest annulus of a point set can be arbitrarily placed, the center *may* lie at infinity and the annulus degenerates to a pair of parallel lines enclosing the set of points. When the center is to be located at infinity, the problem becomes the well known *minimum-width* problem, which is to find a pair of parallel lines enclosing the set such that the distance between them is minimized. The *width* of a set of n points can be computed in $O(n \log n)$ time, which is optimal[4]. In 3-dimensions the *width* of a set is also used as a measure for flatness of a *plate*, a so-called *flatness* problem in computational metrology. Chazelle *et al.*[29] gave an $O(n^{8/5+\epsilon})$ time algorithm for this problem, improving over a previously known algorithm that runs in $O(n^2)$ time.

If, instead of annulus width, the annulus area is to be minimized, then this problem can be solved in linear time via a reduction to fixed-dimensional linear programming[4]. Shermer and Yap[92] introduced the notion of *relative roundness*, where one wants to minimize the ratio of the annulus width and the radius of the inner circle. An $O(n^2)$ algorithm was presented. Duncan *et al.*[42] define another notion of roundness, called *referenced roundness*, which becomes equivalent to the flatness problem when the radius of the reference circle



(a) (b)

Figure 11: Minimum-width annulus and best circle approximation.

is set to infinity. Specifically given a reference radius ρ of an annulus A that contains S, *i.e.*, ρ is the mean of the two concentric circles defining the annulus, find an annulus of a minimum width among all annuli with radius ρ containing S, or for a given $\epsilon > 0$, find an annulus containing S whose width is upper bounded by ϵ. They presented an $O(n \log n)$ algorithm for 2-dimensions and a near quadratic-time algorithm for 3-dimensions.

4 Geometric Matching

Matching in general graphs is one of the classical subjects in combinatorial optimization and has applications in operations research, pattern recognition and VLSI design. Only geometric versions of the matching problem are discussed here. For graph-theoretic matching problems see [82].

Given a weighted undirected complete graph on a set of $2n$ vertices, a *complete matching* is a set of n edges such that each vertex has exactly one edge incident on it. The weight of a matching is the sum of the weights of the edges in the matching. In a metric space, the vertices are points in the plane and the weight of an edge between two points is the distance between them. The *Euclidean minimum weight matching problem* is that given $2n$ points, find n matching pairs of points (p_i, q_i) such that $\sum d(p_i, q_i)$ is minimized.

It was not known if geometric properties can be exploited to obtain an algorithm that is faster than the $\theta(n^3)$ algorithm for general graphs (see [82]). Vaidya[96] settled this question in the affirmative. His algorithm is based on a well-studied primal-dual algorithm for weighted matching. Making use of additive weighted Voronoi diagram discussed in Section 2.2.4 and the range search tree structure (see Section 7.1), Vaidya solved the problem in $O(n^{2.5} \log^4 n)$ time. This algorithm also generalizes to $\Re^k$ but the complexity is increased by a $\log^k n$ factor.

The *bipartite minimum weight matching problem* is defined similarly, except that we are given a set of red points $R = \{r_1, r_2, \ldots, r_n\}$ and a set of blue points $B = \{b_1, b_2, \ldots, b_n\}$ in the plane, and look for n matching pairs of points $(r, b) \in R \times B$ with minimum cost. In [96] Vaidya gave an $O(n^{2.5} \log n)$ time algorithm for Euclidean metric and an $O(n^2 \log^3 n)$

algorithm for L_1-metric.

If these $2n$ points are given as vertices of a polygon, the problems of minimum weight matching and bipartite matching, can be solved in $O(n \log n)$ time if the polygon is convex and in $O(n \log^2 n)$ time if the polygon is simple. In this case the weight of each matching pair of vertices is defined to be the geodesic distance between them[73]. However, if a *maximum* weight matching is sought, an $\log n$ factor can be shaved off[73].

Because of the triangle inequality, one can easily show that in a minimum weight matching the line segments defined by the matched pairs of points cannot intersect one another. Generalizing this *non-intersecting* property the following *geodesic minimum matching* problem in the presence of obstacles can be formulated. Given $2m$ points and polygonal obstacles in the plane, find a matching of these $2m$ points such that the sum of the geodesic distances between matched pairs is minimized. These m paths must not cross each other (they may have portions of the paths overlapping each other). There is no efficient algorithm known to date, except for the obvious method of reducing it to a minimum matching of a complete graph, in which the weight of an edge connecting any two points is the geodesic distance between them. Note that finding a geodesic matching without optimization is trivial, since these m noncrossing paths can always be found. This geodesic minimum matching problem in the general polygonal domain seems nontrivial. The *noncrossing* constraint and the optimization objective function (minimizing total weight) makes the problem hard.

When the matching of these $2m$ points is given *a priori*, finding m noncrossing paths minimizing the total weight, seems very difficult. This resembles global routing problem in VLSI for which m 2-terminal nets are given, and a routing is sought that optimizes a certain objective function, including total wire length, subject to some capacity constraints[71]. The noncrossing requirement is needed when single layer routing or planar routing model is used. Global routing problems in general are NP-hard. Since the paths defined by matching pairs in an optimal routing cannot cross each other, paths obtained by earlier matched pairs become *obstacles* for subsequently matched pairs. Thus the *sequence* in which the pairs of points are matched is very crucial. In fact, the path defined by a matched pair of points need not be the shortest. Thus to route the matched pairs in a greedy manner sequentially does not give an optimal routing. Consider the configuration shown in Fig. 12 in which $R = X, Y, Z$, $B = x, y, z$, and points (X, x), (Y, y) and (Z, z) are to be matched. Note that in this optimal routing, none of the matched pairs is realized by a shortest path, *i.e.*, a straight line. This problem is referred to as the *shortest k-pair noncrossing path* problem. However, if the m matching pairs of points are on the boundary of a simple polygon, and the path must be confined to the interior of the polygon, Papadopoulou[83] gave an $O(n + m)$ algorithm for finding an optimal set of m noncrossing paths, if a solution exists, where n is the number of vertices of the polygon.

Atallah and Chen[14] consider the following bipartite matching problem: Given n red and n blue disjoint isothetic rectangles in the plane, find a matching of these n red-blue pairs of rectangles such that the *rectilinear* paths connecting the matched pairs are noncrossing and monotone. Surprisingly enough, they show that such a matching satisfying the constraints always exists and give an asymptotically optimal $O(n \log n)$ algorithm for finding such a matching.

To conclude this section we remark that the *min-max* versions of the general matching or bipartite matching problems are open. In the red-blue matching if one of the sets is allowed to translate, rotate or scale we have a different matching problem. In this setting we often look for the *best match* according to min-max criterion, *i.e.*, the maximum *error* in the matching is to be minimized. A dual problem can also be defined, *i.e.*, given a maximum

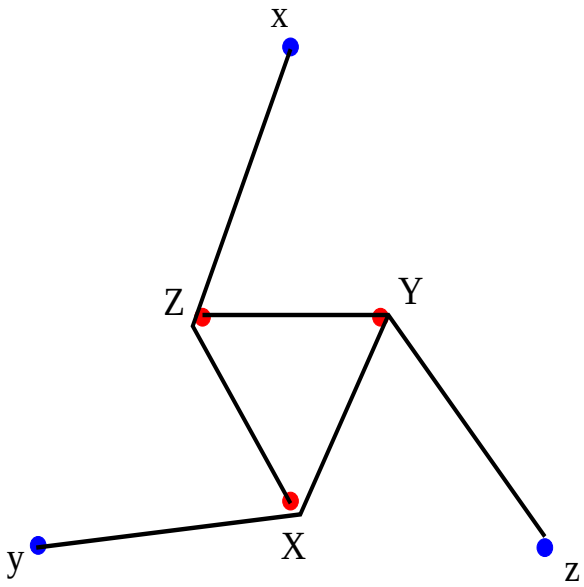


Figure 12: An instance of 3 noncrossing pair matching problem.

error bound, determine if a matching exists, and if so, what kind of *motions* are needed. In [60] Imai *et al.* called this problem *geometric fitting*.

5 Planar Point Location

Planar point location is a fundamental problem in computational geometry. Given a planar subdivision, and a query point, we want to find the region that contains the query point. Fig. 13 shows an example of a planar subdivision. This problem arises in geographic information systems, in which one often is interested in locating, for example, a certain facility in a map. Consider the skew Voronoi diagram, discussed earlier is Section 2.2.5, for a set S of emergency dispatchers. Suppose an emergency situation arises at a location q and that the nearest dispatcher p is to be called so that the distance $\tilde{d}(p,q)$ is the smallest among all distances $\tilde{d}(r,q)$, for $r \in S$. This is equivalent to locating q in the Voronoi cell $\mathcal{V}_{from}(p)$ of the skew Voronoi diagram that contains q. In situations like this it is vital that the nearest dispatcher be located quickly. We therefore address the point-location problem under the assumption that the underlying planar map is *fixed* and the main objective is to have a *fast* response time to each query. Toward this end we preprocess the planar map into a suitable structure so that it would facilitate the point-location task.

An earlier preprocessing scheme is based on the *slab method*[85], in which parallel lines are drawn through each vertex, thus partitioning the plane into parallel slabs. Each parallel slab is further divided into subregions by the edges of the subdivision that can be linearly ordered. Any given query point q can thus be located by two binary searches; one to locate among the $n+1$ horizontal slabs the slab containing q, and followed by another to locate the region defined by a pair of consecutive edges which are ordered from left to right. We use a three tuple, $(P(n), S(n), Q(n))$= (preprocessing time, space requirement, query time) to denote the performance of the search strategy. The slab method gives an

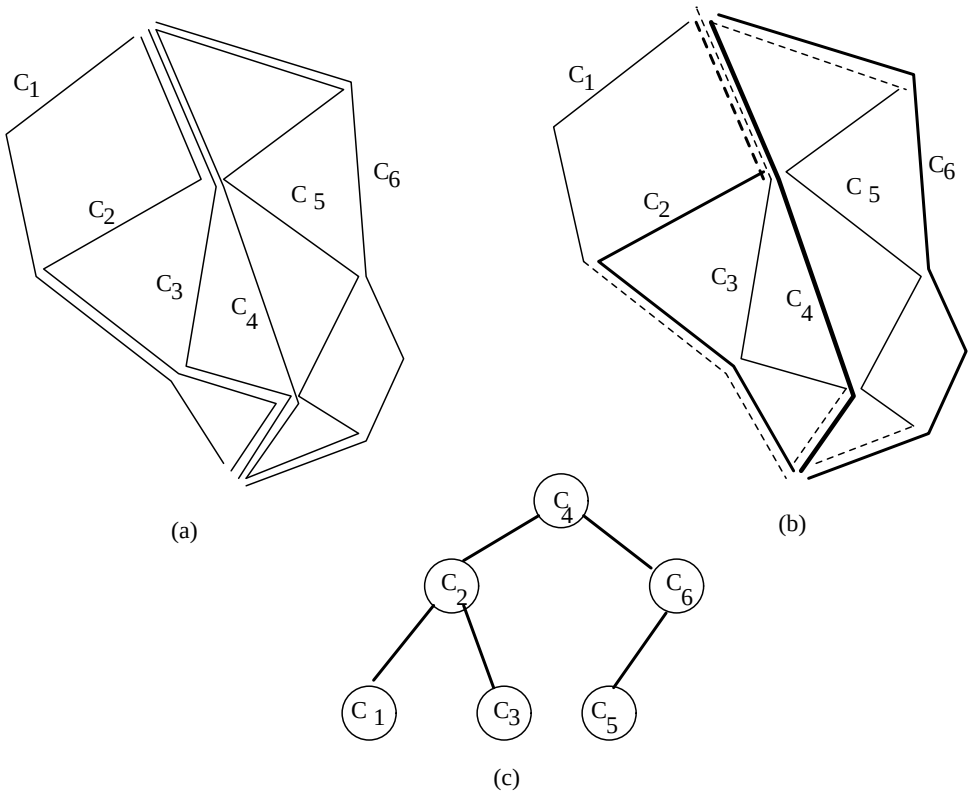


Figure 13: Chain decomposition method for a planar subdivision.

$(O(n^2), O(n^2), O(\log n))$ algorithm. Since preprocessing time is only performed once, the time requirement is not as critical as the space requirement, which is permanently engaged. The primary goal of any search strategy is to minimize the query time and the space required. Lee and Preparata[85] first proposed a *chain decomposition* method to decompose a monotone planar subdivision with n points into a collection of $m \leq n$ monotone chains organized in a complete binary tree. Each node in the binary tree is associated with a monotone chain of at most n edges, ordered in y-coordinate. This set of monotone chains forms a totally ordered set partitioning the plane into collections of regions. In particular, between two adjacent chains there are a number of disjoint regions. The point location process begins with the root node of the complete binary tree. When visiting a node, the query point is compared with the node, hence the associated chain, to decide on which side of the chain the query point lies. Each chain comparison takes $O(\log n)$ time, and total number of nodes visited is $O(\log m)$. The search on the binary tree will lead to two adjacent chains and hence identify a region that contains the point. Thus the query time is $O(\log m \log n) = O(\log^2 n)$. Unlike the slab method in which each edge may be stored as many as $O(n)$ times, resulting in $O(n^2)$ space, it can be shown that with an appropriate chain assignment scheme, each edge in the planar subdivision is stored only once. Thus the space requirement is $O(n)$. For example, in Fig. 13 the edges shared by the root chain C_4 and its descendant chains are assigned to the root chain; in general any edge shared by two nodes on the same root-to-leaf path will be assigned to the node that is an ancestor of the other node. The chain decomposition scheme gives rise to an $(O(n \log n), O(n), O(\log^2 n))$ algorithm. The binary search on the chains is not efficient enough. Recall that after each *chain comparison*, we will move down the binary search tree to perform the next chain comparison and start over another binary search on the *same* y-coordinate of the query point to find an edge of the chain, against which a comparison is made to decide if the point lies to the left or right of the chain. A more efficient scheme is to be able to perform a binary search of the y-coordinate at the root node and to spend only $O(1)$ time per node as we go down the chain tree, shaving off an $O(\log n)$ factor from the query time. This scheme is similar to the ones adopted by Chazelle and Guibas[38, 85] in fractional cascading search paradigm and by Willard[38] in his range tree search method. With the linear time algorithm for triangulating a simple polygon (cf. Section 5 of the previous Chapter) we conclude with the following optimal search structure for planar point-location.

Theorem 5 *Given a planar subdivision of n vertices, one can preprocess the subdivision in linear time and space such that each point location query can be answered in $O(\log n)$ time.*

The point location problem in arrangements of hyperplanes is also of significant interest. See *e.g.,* [31]. **Dynamic** versions of the point location problem, where the underlying planar subdivision is subject to changes (insertions and deletions of vertices or edges) have also been investigated. See [34] for a survey of dynamic computational geometry.

6 Path Planning

This class of problems is mostly cast in the following setting. Given are a set of obstacles O, an object, called *robot*, and an initial and final position, called source and destination respectively. We wish to find a path for the robot to move from the source to the destination avoiding all the obstacles. This problem arises in several contexts. For instance, in robotics this is referred to as the *piano movers' problem* or *collision avoidance* problem, and in VLSI

design this is the *routing* problem for 2-terminal nets. In most applications we are searching for a collision avoidance path that has a *shortest* length, where the distance measure is based on the Euclidean or L_1-metric. For more information regarding motion planning see *e.g.*, [10].

6.1 Shortest Paths in 2-Dimensions

In 2-dimensions, the Euclidean shortest path problem in which the robot is a point, and the obstacles are simple polygons, is well studied. A most fundamental approach is by using the notion of *visibility graph*. Since the shortest path must make turns at polygonal vertices, it is sufficient to construct a graph whose vertices include the vertices of the polygonal obstacles, the source and the destination, and whose edges are determined by vertices that are mutually *visible*, *i.e.*, the segment connecting the two vertices does not intersect the interior of any obstacle. Once the visibility graph is constructed with edge weight equal to the Euclidean distance between the two vertices, one can then apply the Dijkstra's shortest path algorithms[85] to find a shortest path between the source and destination. The Euclidean shortest path between two points is referred to as the *geodesic* path and the distance as the *geodesic* distance. The visibility graph for a set of polygonal obstacles with a total of n vertices can be computed trivially in $O(n^3)$ time. The computation of the visibility graph is the dominating factor for the complexity of any visibility graph based shortest path algorithm. Research results aiming at more efficient algorithms for computing the visibility graph and for computing the geodesic path in time proportional to the size of the graph have been obtained. For example, in [53] Ghosh and Mount gave an output sensitive algorithm that runs in $O(\mathcal{F} + n \log n)$ time for computing the visibility graph, where $\mathcal{F}$ denotes the number of edges in the graph.

Mitchell[75] used the so-called *continuous Dijkstra* wavefront approach to the problem for general polygonal domain of n obstacle vertices and obtained an $O(n^{3/2+\epsilon})$ time algorithm. He constructed a *shortest path map* that partitions the plane into regions such that all points q that lie in the same region have the same vertex sequence in the shortest path from the given source to q. The shortest path map takes $O(n)$ space and enables us to perform shortest path queries, *i.e.*, find a shortest path from the given source to any query points, in $O(\log n)$ time. Hershberger and Suri[56] on the other hand used plane subdivision approach and presented an $O(n \log^2 n)$-time and $O(n \log n)$-space algorithm to compute the *shortest path map* of a given source point. They later improved the time bound to $O(n \log n)$. If the source-destination path is confined in a simple polygon with n vertices, the shortest path can be found in $O(n)$ time[38].

In the context of VLSI routing one is mostly interested in rectilinear paths (L_1-metric) whose edges are either horizontal or vertical. As the paths are restricted to be rectilinear, the shortest path problem can be solved more easily. Lee *et al.*[70] gave a survey on this topic.

In a two-layer routing model, the number of segments in a rectilinear path reflects the number of *vias*, where the wire segments change layers, which is a factor that governs the fabrication cost. In robotics, a straight line motion is not as costly as *making turns*. Thus the number of segments (or *turns*) has also become an objective function. This motivates the study of the problem of finding a path with the least number of segments, called the *minimum link path problem*[78, 93].

These two cost measures, length and number of links, are in conflict with each other. That is, a shortest path may have far too many links, whereas a minimum link path may

be arbitrarily long compared with a shortest path. A path that is optimal in both criteria is called a *smallest path*. In fact it can be easily shown that in a general polygonal domain, a smallest path does not exist. However, a smallest rectilinear path in a simple rectilinear polygon exists, and can be found in linear time. Instead of optimizing both measures *simultaneously* one can either seek a path that optimizes a linear function of both length and the number of links, known as the *combined* metric[98] or optimizes them in a lexicographical order. For example, we optimize the length first, and then the number of links, *i.e.*, among those paths that have the same shortest length, find one whose number of links is the smallest and vice versa. In rectilinear case see *e.g.,* [98]. In [77] algorithms for computing a shortest (in L_2-norm) k-link path in a polygon and for approximating shortest k-link paths in polygons with holes were presented.

A generalization of the collision avoidance problem is to allow collision with a cost. Suppose each obstacle has a weight which represents the cost if the obstacle is *penetrated*[76]. Lee *et al.* (see [70]) studied this problem in the rectilinear case. They showed that a shortest rectilinear path between two given points in the presence of weighted rectilinear polygons can be found in $O(n \log^{3/2} n)$ time and space. Chen *et al.*[32] showed that a data structure can be constructed in $O(n \log^{3/2} n)$ time and $O(n \log n)$ space that enables one to find a shortest path from a given source to any query point in $O(\log n + \mathcal{H})$ time, where $\mathcal{H}$ is the number of links in the path. Another generalization is to include in the set of obstacles some subset $F \subset O$ of obstacles, whose vertices are *forbidden* for the solution path to make turns. Of course, when the weight of obstacles is set to be ∞, or the forbidden set $F = \emptyset$, these generalizations reduce to the ordinary collision avoidance problem.

6.2 Shortest Paths in 3-Dimensions

The Euclidean shortest path problem between two points in a 3-dimensional polyhedral environment turns out to be much harder than its 2-dimensional counterpart. Consider a convex polyhedron P with n vertices in 3-dimensions and two points s, d on the surface of P. A shortest path from s to d on the surface will cross a sequence of edges, denoted $\xi(s, d)$. $\xi(s, d)$ is called the *shortest path edge sequence* induced by s and d and consists of distinct edges. For given s and d, the shortest path from s to d is not unique. However, $\xi(s, d)$ is unique. If $\xi(s, d)$ is known, the shortest path between s and d can be computed by a planar unfolding procedure so that these faces crossed by the path lie in a common plane and the path becomes a straight line segment.

Shortest paths on the surface of a convex polyhedron P possess the following topological properties. (i) They do not pass through vertices of P and do not cross an edge of P more than once, (ii) They do not intersect themselves, *i.e.,* they must be simple, and (iii) Except for the case of two shortest paths sharing a common subpath, they intersect transversely in at most one point, *i.e.,* they cross each other. If the shortest paths are grouped into *equivalent classes* according to the sequences of edges that they cross, then the number of such equivalent classes, denoted $|\xi(P)|$ is $\theta(n^4)$, where n is the number of vertices of P. These equivalent classes can be computed in $O(|\xi(P)|n^3 \log n)$ time. Chen and Han[33] gave an $O(n^2)$ algorithm for finding a shortest path between a fixed source s and any destination d, where n is the number of vertices and edges of the polyhedron, which may or may not be convex. If s and d lie on the surface of two different polyhedra, $O(N^{O(k)})$ algorithm suffices for computing the shortest path between them amidst a set of k polyhedra, where N denotes the total number of vertices of these obstacles.

The crux of the problem lies in the fact that the number of possible edge sequences may

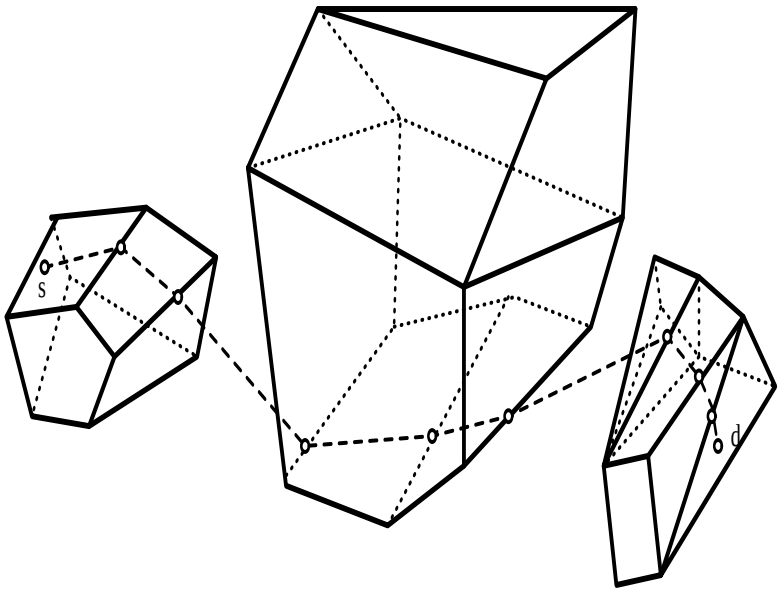


Figure 14: A possible shortest path edge sequence between points s and d.

be exponential in the number of obstacles, if s and d lie on the surface of different polyhedra. It was established that the problem of determining the shortest path edge sequence is indeed NP-hard[10]. Fig. 14 shows an example of a possible shortest path edge sequence induced by s and d in the presence of 3 convex polyhedra. Approximation algorithms for this problem can be found in *e.g.,* Choi *et al.*[35].

7 Searching

This class of problems is cast in the form of query-answering. Given is a collection of objects, with preprocessing allowed, one is to find objects that satisfy the queries. The problem can be **static** or dynamic, depending on if the database to be searched is allowed to change over the course of query-answering sessions, and it is studied mostly in two modes, *count-mode* and *report-mode*. In the former case only the *number* of objects satisfying the query is to be answered, whereas in the latter the actual identity of the objects is to be reported. In the report-mode the query time of the algorithm consists of two components, *search time* and *retrieval time*, and expressed as $Q_{\mathcal{A}}(n) = O(f(n) + \mathcal{F})$, where n denotes the size of the database, $f(n)$ a function of n, and $\mathcal{F}$ the size of output. Sometimes we may need to perform some semigroup operations to those objects that satisfy the query. For instance, we may have weights $w(v)$ assigned to each object v, and we want to compute $\sum w(v)$ for all $v \cap q \neq \emptyset$. This is referred to as *semigroup range searching*[22]. The semigroup range searching problem is the most general form: if the semigroup operation is set union, we get report-mode range searching problem and if the semigroup operation is just addition (of uniform weight), we have the count-mode range searching problem. We will not discuss the semigroup range searching here. It is obvious that algorithms that handle the report-mode queries can also handle the count-mode queries ($\mathcal{F}$ is the answer). It seems *natural* to expect that the algorithms for count-mode queries would be more efficient (in terms of the

order of magnitude of the space required and query time), as they need not search for the objects. However, it was argued that in the report-mode range searching, one could take advantage of the fact that since reporting takes time, the more to report, the *sloppier* the search can be. For example, if we were to know that the ratio $n/\mathcal{F}$ is $O(1)$, we could use a sequential search on a linear list. This notion is known as *filtering search*[85]. In essence more objects than necessary are identified by the search mechanism, followed by a filtering process leaving out unwanted objects. As indicated below the count-mode range searching problem is harder than the report-mode counterpart[23].

7.1 Range Searching

This is a fundamental problem in database applications. We'll discuss this problem and the algorithm in the 2-dimensional space. The generalization to higher-dimensions is straightforward using a known technique, called multidimensional divide-and-conquer[85]. Given is a set of n points in the plane, and the ranges are specified by a product $(l_1, u_1) \times (l_2, u_2)$. We would like to find points $p = (x, y)$ such that $l_1 \leq x \leq u_1$ and $l_2 \leq y \leq u_2$. Intuitively we want to find those points that lie inside a query rectangle specified the the range. This is called *orthogonal range searching*, as opposed to other kinds of range searching problems discussed below, *e.g.,* half-space range searching, and simplex range searching, etc. Unless otherwise specified, a range refers to an orthogonal range. We discuss the static case, as this belongs to the class of **decomposable searching problems**, the **dynamization transformation** techniques can be applied. We note that the range tree structure mentioned below can be made dynamic by using a weight-balanced tree, called $BB(\alpha)$ tree.

For count-mode queries it can be solved by using the locus method as follows. Divide the plane into $O(n^2)$ cells by drawing horizontal and vertical line through each point. The answer to the query q, *i.e.*, find the number of points dominated by q, (those points whose x- and y-coordinates are both no greater than those of q) can be found by locating the cell containing q. Let it be denoted by $Dom(q)$. Thus the answer to the count-mode range-queries can be obtained by some simple arithmetic operations of $Dom(q_i)$ for the four corners, q_1, q_2, q_3, q_4 of the query rectangle. Let q_4 be the northeast corner and q_2 be the southwest corner. The answer will be $Dom(q_4) - Dom(q_1) - Dom(q_3) + Dom(q_2)$. Thus in $\Re^k$ we have $Q(k, n) = O(k \log n)$, $S(k, n) = P(k, n) = O(n^k)$. To reduce space requirement at the expense of query time has been a goal of further research on this topic. Bentley introduced a data structure, called *range trees*[85].Using this structure the following results were obtained: for $k \geq 2$, $Q(k, n) = O(\log^{k-1} n)$, $S(k, n) = P(k, n) = O(n \log^{k-1} n)$ (see [67] for more references).

For report-mode queries, by using filtering search technique the space requirement can be further reduced by a $\log \log n$ factor. If the range satisfies additional conditions, *e.g., grounded* in one of the coordinates, say $l_1 = 0$, or the aspect ratio of the intervals specifying the range is fixed, less space is needed. For instance, in 2-dimensions, the space required is linear (a saving of $\log n / \log \log n$ factor) for these two cases. By using the so-called functional approach to data structures Chazelle[22] developed a *compression* scheme to reduce further the space requirement. Thus in k-dimensions, $k \geq 2$, for the count-mode range queries we have $Q(k, n) = O(\log^{k-1} n)$ and $S(k, n) = O(n \log^{k-2} n)$ and for report-mode range queries $Q(k, n) = O(\log^{k-1} n + \mathcal{F})$, and $S(k, n) = O(n \log^{k-2+\epsilon} n)$ for some $0 < \epsilon < 1$.

As regards the lower bound of range searching in terms of space-time tradeoffs, Chazelle[23] showed that in k-dimensions, if the query time is $O(\log^c n + \mathcal{F})$ for any constant c, the space

required is $\Omega(n(\log n/\log\log n)^{k-1})$ for pointer machine models and the bound is tight for any $c \geq 1$ if $k = 2$, and any $c \geq k - 1 + \epsilon$ (for any $\epsilon > 0$) if $k > 2$. See also [24, 95] for more lower bound results related to orthogonal range searching problems.

7.2 Other Range Searching Problems

There are other range searching problems, called simplex range searching problem, and the half-space range searching problems that have been well-studied. A simplex range in $\Re^k$ is a range whose boundary is specified by $k+1$ hyperplanes. In 2-dimensions it is a triangle. For this problem there is a lower bound on the query time for simplex range queries: let m denote the space required, $Q(k,n) = \Omega((n/\log n)m^{1/k}), k > 2$ and $Q(2,n) = \Omega(n/\sqrt{m})$[38].

The report-mode half-space range searching problem in the plane can be solved optimally in $Q(n) = O(\log n + \mathcal{F})$ time and $S(n) = O(n)$ space, using geometric duality transform[38]. But this method does not generalize to higher dimensions. In [3] Agarwal and Matoušek obtained a general result for this problem: for $n \leq m \leq n^{\lfloor k/2 \rfloor}$, with $O(m^{1+\epsilon})$ space and preprocessing, $Q(k,n) = O((n/m^{1/\lfloor k/2 \rfloor})\log n + \mathcal{F})$. As half-space range searching problem is also decomposable standard dynamization techniques can be applied.

A general method for simplex range searching is to use the notion of *partition tree*. The search space is partitioned in a hierarchical manner using cutting hyperplanes[25], and a search structure is built in a tree structure. Using a **cutting theorem** of hyperplanes Matoušek[74] showed that for k-dimensions, there is a linear space search structure for the simplex range searching problem with query time $O(n^{1-1/k})$, which is optimal in 2-dimensions and within $O(\log n)$ factor of being optimal for $k > 2$. For more detailed information regarding geometric range searching see [74].

The above discussion is restricted to the case in which the database is a collection of points. One may consider also other kinds of objects, such as line segments, rectangles, triangles, etc., whatever applications may take. The inverse of the orthogonal range searching problem is that of *point enclosure searching problem*. Consider a collection of isothetic rectangles. The point enclosure searching is to find all rectangles that contain the given query point q. We can cast these problems as *intersection searching* problem, *i.e.*, given a set S of objects, and a query object q, find a subset $\mathcal{F}$ of S such that for any $f \in \mathcal{F}, f \cap q \neq \emptyset$. We have then the rectangle enclosure searching problem, rectangle containment problem, segment intersection searching problem, etc. Janardan and Lopez[61] generalized the intersection searching in the following manner. The database is a collection of *groups* of objects, and the problem is to find all groups of objects intersecting a query object. A group is considered to be intersecting the query object if any object in the group intersects the query object. When each group has only one object, this reduces to the ordinary searching problems.

8 Intersection

This class of problems arises in, for example, architectural design, computer graphics, etc. In an architectural design no two objects can share a common region. In computer graphics the well-known hidden-line or hidden-surface elimination problems[40] are examples of intersection problems. This class encompasses two types of problems, *intersection detection* and *intersection computation*.

8.1 Intersection Detection

The intersection detection problem is of the form: Given a set of objects, do any two intersect? For instance, given n line segments in the plane, are there two that intersect? The intersection detection problem has a lower bound of $\Omega(n \log n)$[85].

In 2-dimensions the problem of detecting if two polygons of r and b vertices intersect was easily solved in $O(n \log n)$ time, where $n = r + b$ using the red-blue segment intersection algorithm[30]. However, this problem can be reduced in linear time to the problem of detecting the self intersection of a polygonal curve. The latter problem is known as the *simplicity* test and can be solved optimally in linear time by Chazelle's linear time triangulation algorithm (cf. Section 5 of the previous Chapter). If the two polygons are convex, then $O(\log n)$ suffices in detecting if they intersect[27]. Note that although detecting if two convex polygons intersect can be done in logarithmic time, detecting if the boundary of the two convex polygons intersect requires $\Omega(n)$ time[27]. Mount[79] investigated the intersection detection of two simple polygons and computed a separator of m links in $O(m \log^2 n)$ time if they don't intersect.

In 3-dimensions, detecting if two convex polyhedra intersect can be solved in linear time[27] by using a hierarchical representation of the convex polyhedron, or by formulating it as a linear programming problem in 3 variables.

8.2 Intersection Reporting/Counting

One of the simplest such intersecting reporting problems is that of *reporting* pairwise intersection, *e.g.*, intersecting pairs of line segments in the plane. An earlier result due to Bentley and Ottmann[85] used the plane sweep technique that takes $O((n + \mathcal{F}) \log n)$ time, where $\mathcal{F}$ is the output size. This is based on the observation that the line segments intersected by a vertical sweep-line can be ordered according to the y-coordinates of their intersection with the sweep-line, and the sweep-line status can be maintained in logarithmic time per event point, which is either an endpoint of a line segment or the intersection of two line segments. It is not difficult to see that the lower bound for this problem is $\Omega(n \log n + \mathcal{F})$; thus the above algorithm is $O(\log n)$ factor from the optimal. This segment intersection reporting problem has been solved optimally by Chazelle and Edelsbrunner[28] who used several important algorithm design and data structuring techniques, as well as some crucial combinatorial analysis. In contrast to this asymptotically time-optimal *deterministic* algorithm, a simpler randomized algorithm was obtained[36] for this problem which is both time- and space-optimal. That is, it requires only $O(n)$ space (instead of $O(n + \mathcal{F})$ as reported in [28]). Balaban[17] most recently reported a deterministic algorithm that solves this problem optimally both in time and space.

On a separate front, the problem of finding intersecting pairs of segments from two different sets was considered. This is called *bi-chromatic line segment* intersection problem.

Chazelle *et al.*[30] used *hereditary segment trees* structure and fractional cascading and solved both segment intersection reporting and counting problems optimally in $O(n \log n)$ time and $O(n)$ space. (The term $\mathcal{F}$ should be included in case of reporting.) If the two sets of line segments form connected subdivisions, then merging or overlay of these two subdivisions can be computed in $O(n+\mathrm{F})$[50]. See [5] for more applications of the bichromatic intersection problem.

The *rectangle intersection reporting* problem arises in the design of VLSI circuitry, in which each rectangle is used to model a certain circuitry component. These rectangles

are isothetic, *i.e.*, their sides are all parallel to the coordinate axes. This is a well-studied classical problem and optimal algorithms ($O(n \log n + \mathcal{F})$ time) have been reported (see [67] for references). The k-dimensional hyperrectangle intersection reporting (respectively, counting) problem can be solved in $O(n^{k-2} \log n + \mathcal{F})$ time and $O(n)$ space (respectively, in time $O(n^{k-1} \log n)$ and space $O(n^{k-2} \log n)$). Gupta *et al.*[55] gave an $O(n \log n \log \log n + \mathcal{F} \log \log n)$ time and linear space algorithm for the *rectangle enclosure reporting* problem that calls for finding all enclosing pairs of rectangles.

8.3 Intersection Computation

Computing the actual intersection is a basic problem, whose efficient solutions often lead to better algorithms for many other problems.

Consider the problem of computing the common intersection of half-planes by divide-and-conquer. Efficient computation of intersection of two convex polygons is required during the merge step. The intersection of two convex polygons can be solved very efficiently by plane-sweep in linear time, taking advantage of the fact that the edges of the input polygons are ordered. Observe that in each vertical strip defined by two consecutive sweep-lines, we only need to compute the intersection of two trapezoids, one derived from each polygon[85].

The problem of intersecting two convex polyhedra was first studied by Muller and Preparata[85], who gave an $O(n \log n)$ algorithm by reducing the problem to the problems of intersection detection and convex hull computation. Following this result one can easily derive an $O(n \log^2 n)$ algorithm for computing the common intersection of n half-spaces in 3-dimensions by divide-and-conquer. However, using geometric duality and the concept of separating plane, Preparata and Muller[85] obtained an $O(n \log n)$ algorithm for computing the common intersection of n half-spaces, which is asymptotically optimal. There appears to be a difference in the approach to solving the common intersection problem of half-spaces in 2- and 3-dimensions. In the latter we resorted to geometric duality instead of divide-and-conquer. This *inconsistency* was later resolved. Chazelle[26] combined the hierarchical representation of convex polyhedra, geometric duality and other ingenious techniques to obtain a linear time algorithm for computing the intersection of two convex polyhedra. From this result several problems can be solved optimally: (1) the common intersection of half-spaces in 3-dimensions can now be solved by divide-and-conquer optimally, (2) the merging of two Voronoi diagrams in the plane can be done in linear time by observing the relationship between the Voronoi diagram in 2-dimensions and the convex hull in 3-dimensions (cf. Section 2 of the previous Chapter), and (3) the medial axis of a simple polygon or the Voronoi diagram of vertices of a convex polygon[6] can be solved in linear time.

9 Research Issues and Summary

We have covered in this chapter a number of topics in computational geometry, including proximity, optimization, planar point location, geometric matching, path planning, searching and intersection. These topics discussed here and in the previous chapter are not meant to be exhaustive. New topics arise as the field continues to flourish.

In Section 3 we have discussed the problems of smallest enclosing circle and the largest empty circle. These are the two extremes: either the circle is empty or it contains *all* the points. The problem of finding a *smallest* (resp. *largest*) circle containing at least (resp. at most) k points for some integer $0 \leq k \leq n$ is also a problem of interest. Moreover, the shape

of the object is not limited to circles. A number of open problems remain. What is the complexity of the rectilinear Steiner arborescence problem? Given two points s and t in a simple polygon, is it NP-complete to decide whether there exists a path with at most k links and of length at most L? What is the complexity of the shortest k-pair noncrossing path problem discussed in Section 4? How fast can one solve the geodesic minimum matching problem for $2m$ points in the presence of polygonal obstacles? Can one solve the largest empty rectangle problem for a rectilinear polygon in $O(n \log n)$ time? The best known algorithm to date runs in $O(n \log^n)$ time[7]. Is $\Omega(n \log n)$ a lower bound of the minimum-width annulus problem? Can the technique used in [51] be applied to the polygonal case to yield an $O(n \log n)$ time algorithm for the minimum-width annulus problem?

Recently researchers in computational geometry begin to address the issues concerning the actual running times of the algorithms and their robustness when the computations in their implementations are not *exact*. It is understood that the real-RAM computation model with an implicit infinite precision arithmetic is unrealistic in practice. In addition to the robustness issue concerning the accuracy of the output of an algorithm, one needs to find a *new* cost measure to evaluate the efficiency of an algorithm. In the infinite-precision model, the asymptotic time complexity was accepted as an adequate cost measure. However, when the input data have a finite precision representation and computation time varies with the precision required, an alternative cost measure is warranted. The notion of the **degree** of a geometric algorithm could be an important cost measure for comparing the efficiency of algorithms when they are actually implemented[72]. This could play a similar role as the asymptotic time complexity has in the past for the real-RAM computation model.

On the applied side there are new efforts put into development of geometric software. A library of geometric software including visualization tools, and application programs, is under development. A website containing geometric software is maintained at the Geometry Center, University of Minnesota, at http://www.geom.umn.edu/software/cglist. A project, known as the CGAL project[49], is under way by researchers in Europe to organize a system library containing primitive geometric abstract data types useful for geometric algorithm developers. This is concurrent with the LEDA[80] project at the Max-Planck-Institut für Informatik, Saarbrücken, Germany. Other projects related to the efforts of building geometric software or problem solving environment, include GASP, GeoLab, GeoMAMOS, XYZ Geobench, etc. See [66] for more information.

10 Defining Terms

ANSI: American National Standards Institute.
Bisector: A bisector of two elements e_i and e_j is defined to be the locus of points equidistant from both e_i and e_j. That is, $\{p | d(p, e_i) = d(p, e_j)\}$. For instance, if e_i and e_j are two points in the Euclidean plane, the bisector of e_i and e_j is the perpendicular bisector to the line segment $\overline{e_i, e_j}$.
Cutting theorem: This theorem[25] states that for any set $\mathcal{H}$ of n hyperplanes in $\Re^k$, and any parameter r, $1 \leq r \leq n$, there always exists a $(1/r)$-cutting of size $O(r^k)$. In 2-dimensions, a $(1/r)$-cutting of size s is a partition of the plane into s disjoint triangles, some of which are unbounded, such that no triangle in the partition intersects more than n/r lines in $\mathcal{H}$. In $\Re^k$, triangles are replaced by simplices. Such a cutting can be computed in $O(nr^{k-1})$ time.
Decomposable searching problems: A searching problem with query Q is decompos-

able if there exists an efficiently computable associative, and commutative binary operator @ satisfying the condition: $Q(x, A \cup B) = @(Q(x, A), Q(x, B))$. In other words, one can decompose the searched domain into subsets, find answers to the query from these subsets, and combine these answers to form the solution to the original problem.

Degree of an algorithm or problem: Assume that each input variable is of arithmetic degree 1 and that the arithmetic degree of a polynomial is the common arithmetic degree of its monomials, whose degree is defined to be the sum of the arithmetic degrees of its variables. An algorithm has degree d if its test computation involves evaluation of multivariate polynomials of arithmetic degree d. A problem has degree d if any algorithm that solves it has degree *at least* d[72].

Diameter of a graph: The distance between two vertices u and v in a graph is the sum of weights of the edges of a shortest path between them. (For unweighted graph, it is the number of edges of a shortest path.) The diameter of a graph is the maximum among all distances between all possible pairs of vertices.

Dynamic versus Static: This refers to cases when the underlying problem domain can be subject to updates, *i.e.,*. insertions, and deletions. If no updates are permitted, the problem or data structure is said to be static; otherwise, it is said to be dynamic.

Dynamization transformation: A data structuring technique that can transform a static data structure into dynamic one. In so doing, the performance of the dynamic structure will exhibit certain space-time tradeoffs. See *e.g.,*[65, 85] for more references.

Geometric duality: A transform between a point and a hyperplane in $\Re^k$, that preserves incidence and order relation. For a point $p = (\mu_1, \mu_2, \ldots, \mu_k)$, its dual $\mathcal{D}(p)$ is a hyperplane denoted by $x_k = \sum_{j=1}^{k-1} \mu_j x_j - \mu_k$; for a hyperplane $\mathcal{H} : x_k = \sum_{j=1}^{k-1} \mu_j x_j + \mu_k$, its dual $\mathcal{D}(\mathcal{H})$ is a point denoted by $(\mu_1, \mu_2, \ldots, -\mu_k)$. There are other duality transformations. What is described in the text is called the *paraboloid* transform. See [44, 38, 85] for more information.

Height-balanced binary search tree: A data structure used to support membership, insert/delete operations each in time logarithmic in the size of the tree. A typical example is the *AVL* tree or *red-black* tree.

Orthogonally convex rectilinear polygon: A rectilinear polygon P is orthogonally convex if every horizontal or vertical segment connecting two points in P lies totally within P.

Priority queue: A data structure used to support insert and delete operations in time logarithmic in the size of the queue. The elements in the queue are arranged so that the element of the *minimum* priority is always at one end of the queue, readily available for delete operation. Deletions only take place at that end of the queue. Each delete operation can be done is constant time. However, since restoring the above property after each deletion takes logarithmic time, we often say that each delete operation takes logarithmic time. A heap is a well known priority queue.

References

[1] P. K. Agarwal, M. de Berg, J. Matoušek and O. Schwarzkopf, "Constructing Levels in Arrangements and Higher Order Voronoi Diagrams," *Proc. 10th Annual ACM Symp. Comput. Geometry*, 1994, 67-75.

[2] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, E. Welzl, "Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs," *Discrete & Comput. Geometry.*, 6,5 (1991), 407-422.

[3] P. K. Agarwal and J. Matoušek, "Dynamic Half-Space Range Reporting and Its Applications," *Algorithmica*, 13,4 (April 1995), 325-345.

[4] P. K. Agarwal, M. Sharir and S. Toledo, "Applications of Parametric Searching in Geometric Optimization," *J. Algorithms*, 17 (1994), 292-318.

[5] P. K. Agarwal and M. Sharir, "Red-Blue Intersection Detection Algorithms, with Applications to Motion Planning and Collision Detection," *SIAM J. Comput.*, 19,2 (1990), 297-321.

[6] A. Aggarwal and L. J. Guibas and J. Saxe and P. W. Shor, "A Linear-Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon," *Discrete & Comput. Geometry*, 4,6 (1989), 591-604.

[7] A. Aggarwal and S. Suri, "Fast Algorithms for Computing the Largest Empty Rectangle," *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, 1987, 278-290.

[8] O. Aichholzer, F. Aurenhammer, D. Z. Chen, D. T. Lee, A. Mukhopadhyay and E. Papadopoulou, "Voronoi Diagrams for Direction-Sensitive Distances," *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, 418-420.

[9] H. Alt and O. Schwarzkopf, "The Voronoi Diagram of Curved Objects," *Proc. 11th Annual ACM Symp. Comput. Geometry*, 1995, 89-97.

[10] H. Alt and C. K. Yap, "Algorithmic Aspect of Motion Planning: A Tutorial, Part 1 & 2," *Algorithms Review,* 1 (1990), 43-77.

[11] N. M. Amato and E. A. Ramos, "On Computing Voronoi Diagrams by Divide-Prune-and-Conquer," *Proc. 12th Annual ACM Symp. Comput. Geometry*, 1996, 166-175.

[12] H. Aonuma, H. Imai, K. Imai and T. Tokuyama, "Maximin Location of Convex Objects in a Polygon and Related Dynamic Voronoi Diagrams," *Proc. 6th Annual ACM Symp. Comput. Geometry*, 1990, 225-234.

[13] M. J. Atallah, "Parallel Techniques for Computational Geometry," *Proceedings of IEEE*, 80,9 (Sept. 1992), 1435-1448.

[14] M. J. Atallah and D. Z. Chen, "Applications of a Numbering Scheme for Polygonal Obstacles in the Plane," *Proc. 7th Int'l Symp. Algorithms and Computation,* Dec. 1996, 1-24.

[15] F. Aurenhammer, "A Relationship between Gale Transforms and Voronoi Diagrams," *Discrete Appl. Math.*, 28 (1991), 83-91.

[16] H. S. Baird, S. E. Jones and S. J. Fortune, "Image Segmentation by Shape-Directed Covers," *Proc. 10th Int'l Conf. Pattern Recognition*, 1990, 820-825.

[17] I. J. Balaban, "An Optimal Algorithm for Finding Segments Intersections," *Proc. 11th Annual Symp. Comput. Geometry*, June 1995, 211-219.

[18] S. N. Bespamyatnikh, "An Optimal Algorithm for Closest Pair Maintenance," *Proc. 11th Annual Symp. Comput. Geometry*, June 1995, 152-161.

[19] A. Blum, R. Ravi and S. Vempala, "A Constant-factor Approximation Algorithm for the k-MST Problem," *Proc. 28th Symp. Theory of Comput.,* May 1996.

[20] J.-D. Boissonnat, M. Sharir, B. Tagansky and M. Yvinec, "Voronoi Diagrams in Higher Dimensions Under Certain Polyhedra Distance Functions," *Proc. 11th Annual ACM Symp. Comput. Geometry*, 1995, 79-88.

[21] P. Callahan and S. R. Kosaraju, "Algorithms for Dynamic Closest Pair and n-Body Potential Fields," *Proc. 6th ACM-SIAM Symp. Discrete Algorithms*, 1995, 263-272.

[22] B. Chazelle, "A Functional Approach to Data Structures and Its Use in Multidimensional Searching," *SIAM J. Computing*, 17,3 (June 1988), 427-462.

[23] B. Chazelle, "Lower Bounds for Orthogonal Range Searching, I: The Reporting Case," *J. ACM*, 37,2 (April 1990), 200-212.

[24] B. Chazelle, "Lower Bounds for Orthogonal Range Searching, II: The Arithmetic Model," *J. ACM*, 37,3 (July 1990), 439-463.

[25] B. Chazelle, "Cutting Hyperplanes for Divide-and-Conquer," *Discrete & Comput. Geometry*, 9,2 (1993), 145-158.

[26] B. Chazelle, "An Optimal Algorithm for Intersecting Three-Dimensional Convex Polyhedra," *SIAM J. Comput.*, 21,4 (1992), 671–696.

[27] B. Chazelle and D. P. Dobkin, "Intersection of Convex Objects in Two and Three Dimensions," *J. ACM*, 34,1 (1987), 1-27.

[28] B. Chazelle and H. Edelsbrunner, "An Optimal Algorithm for Intersecting Line Segments in the Plane," *J. ACM*, 39,1 (1992), 1-54.

[29] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir, "Diameter, Width, Closest Line Pair, and Parametric Searching," *Discrete & Computational Geometry*, 8 (1993), 183-196.

[30] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir, "Algorithms for Bichromatic Line-Segment Problems and Polyhedral Terrains," *Algorithmica*, 11,2 (Feb. 1994), 116-132.

[31] B. Chazelle and J. Friedman, "Point Location among Hyperplanes and Unidirectional Ray-Shooting," *Comput. Geom. Theory Appl.*, 4 (1994) 53-62.

[32] D. Chen, K. S. Klenk and H.-Y. T. Tu, "Shortest Path Queries among Weighted Obstacles in the Rectilinear Plane," *Proc. 11th Annual ACM Symp. Comput. Geometry*, June 1995, 370-379.

[33] J. Chen and Y. Han, "Shortest Paths on a Polyhedron, Part I: Computing Shortest Paths," *Int'l J. Comput. Geometry & Applications*, 6,2 (1996), 127-144.

[34] Y.-J. Chiang and R. Tamassia, "Dynamic Algorithms in Computational Geometry," *Proceedings of IEEE*, 80,9 (Sept. 1992), 1412-1434.

[35] J. Choi, J. Sellen and C. K. Yap, "Approximate Euclidean Shortest Path in 3-Space," *Proc. 10th Symp. on Computational Geometry*, 1994, 41-48.

[36] K. L. Clarkson and P. W. Shor, "Applications of Random Sampling in Computational Geometry II," *Discrete & Computational Geometry*, 4 (1989), 387-421.

[37] K. Daniels, V. Milenkovic and D. Roth, "Finding the Largest Area Axis-Parallel Rectangle in a Polygon," *Computational Geometry: Theory & Appl.*, 7,1-2 (Jan. 1997), 125-148.

[38] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, Computational Geometry Algorithms and Applications, Springer-Verlag, 1997.

[39] F. Dehne and R.Klein, "The Big Sweep: On the Power of the Wavefront Approach to Voronoi Diagrams," *Algorithmica*, 17,1 (Jan. 1997), 19-32.

[40] S. E. Dorward, "A Servey of Object-Space Hidden Surface Removal," *Int'l J. Computational Geometry & Applications* 4,3 (Sept. 1994), 325-362.

[41] D. Z. Du and F. K. Hwang, Computing in Euclidean Geometry, Eds., World Scientific Publishing Co., Singapore, 1992.

[42] C. A. Duncan, M. T. Goodrich, and E. A. Ramos, "Efficient Approximation and Optimization Algorithms for Computational Metrology," *Proc. 8th ACM-SIAM Symp. Discrete Algorithms*, 1997, 121-130.

[43] R. A. Dwyer and W. F. Eddy, "Maximal Empty Ellipsoids," *Proc. 5th ACM-SIAM Symp. Discrete Algorithms*, 1994, 98-102.

[44] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag, 1987.

[45] H. Edelsbrunner, "The Union of Balls and Its Dual Shape," *Proc. 9th Annual ACM Symp. Comput. Geometry*, 1993, 218-231.

[46] H. Edelsbrunner and M. Sharir, "A Hyperplane Incidence Problem with Applications to Counting Distances," *Applied Geometry and Discrete Mathematics. The Victor Klee Festschrift*, 253–263, ed.: P. Gritzmann and B. Sturmfels, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1991.

[47] A. Efrat and M. Sharir, "A Near-Linear Algorithm for the Planar Segment Center Problem," *Proc. 5th ACM-SIAM Symp. Discrete Algorithms*, 1994, 87-97.

[48] D. Eppstein, "Fast Construction of Planar Two-Centers," *Proc. i8th Annu. ACM-SIAM Symp. Discrete Alg.*, 1997, 131-138.

[49] A. Fabri, G. Giezeman, L. Kettner, S. Schirra and S. Schönherr, "The CGAL Kernel: A Basis for Geometric Computation," Applied Computational Geometry, Lin and Manocha (Eds.) Springer-Verlag, 1996.

[50] U. Finkle andK. Hinrichs, "Overlaying simply connected planar subdivision in linear time," *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, 119-126.

[51] J. Garcia-Lopez and P. A. Ramos, "Fitting a Set of Points by a Circle," *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, 139-146.

[52] N. Garg and D. S. Hochbaum, "An $O(\log k)$ approximation algorithm for the k Minimum Spanning Tree Problem in the Plane," *Proc. 26th Symp. Theory of Comput.,* May 1994.

[53] S. K. Ghosh and D. M. Mount, "An Output-Sensitive Algorithm for Computing Visibility Graphs," *SIAM J. Comput.* 20,5 (Oct. 1991), 888-910.

[54] J. E. Goodman and J. O'Rourke, (eds.) The Handbook of Discrete and Computational Geometry, CRC Press LLC, Boca Raton, FL, 1997.

[55] P. Gupta, R. Janardan, M. Smid and B. Dasgupta, "The Rectangle Enclosure and Point-Dominance Problems Revisited," *Proc. 11th Annu. ACM Sympos. Comput. Geom.* 1995, 162-171.

[56] J. Hershberger and S. Suri, "Efficient Computation of Euclidean Shortest Paths in the Plane", *34th Symp. on Foundations of Computer Science*, 1993, 508- 517.

[57] Hershberger, J. and S. Suri, "Finding a Shortest Diagonal of a Simple Polygon in Linear Time," *Computational Geometry: Theory and Applications*, 7,3 (Feb. 1997), 149-160.

[58] J. M. Ho, C. H. Chang, D. T. Lee and C. K. Wong, "Minimum Diamter Spanning Tree and Related Problems," *SIAM J. Comput.,* 20,5 (October 1991), 987-997.

[59] F. K. Hwang, "Foreword," *Algorithmica* 7,2/3 (1992), 119-120.

[60] K. Imai, S. Sumino and H. Imai, "Minimax Geometric Fitting of Two Corresponding Sets of Points," *Proc. 5th Annual ACM Symp. Comput. Geometry*, 1989, 266-275.

[61] R. Janardan and M. Lopez, "Generalized Intersection Searching Problems," *Int'l J. Comput. Geom. & Appl.*, 3,1 (March 1993), 39-69.

[62] S. Kapoor and M. Smid, "New Techniques for Exact and Approximate Dynamic Closest-Point Problems," *SIAM J. Computing*, 25,4 (Aug. 1996), 775-796.

[63] M. M. Klawe and D. J. Kleitman, "An almost linear time algorithm for generalized matrix searching" *SIAM J. Discrete Math.*, 3,1 (Feb. 1990), 81-97.

[64] D. T. Lee, "Two Dimensional Voronoi diagrams in the L_p-metric," *J. ACM*, 27 (1980), 604-618.

[65] D. T. Lee, "Computational Geometry," Computer Science and Engineering Handbook, Ed. A. Tucker, CRC Press, 1996, 111-140.

[66] D. T. Lee, "Geometric Algorithm Visualization, Current Status and Future," Applied Computational Geometry, Lin and Manocha (Eds.) Springer-Verlag, 1996, 45-50.

[67] D. T. Lee and F. P. Preparata, "Computational Geometry: A Survey," *IEEE Trans. Comput.,* C-33,12 (Dec. 1984), 1072–1101.

[68] D. T. Lee, and C. F. Shen, "The Steiner Minimal Tree Problem in the λ-*geometry* Plane," *Proc. 7th Int'l Symposium on Algorithms and Computation*, Osaka, Japan, Dec. 1996, 247-255.

[69] D. T. Lee and V. B. Wu, "Multiplicative Weighted Farthest Neighbor Voronoi Diagrams in the Plane," *Proc. Int'l Workshop on Discrete Mathematics and Algorithms*, Hong Kong, Dec. 1993, 154-168.

[70] D. T. Lee, C. D. Yang and C. K. Wong, "Rectilinear Paths among Rectilinear Obstacles," *Perspectives in Discrete Applied Math.*, K. Bogart, Ed., 70 (1996), 185-215.

[71] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, John Wiley & Sons, 1990.

[72] G. Liotta, F. P. Preparata and R. Tamassia, "Robust Proximity Queries: an Illustration of Degree-driven Algorithm Design," *Proc. 13th Annual ACM Symp. Comput. Geometry*, 1997, 156-165.

[73] O. Marcotte and S. Suri, "Fast Matching Algorithms for Points on a Polygon," *SIAM J. Comput.*, 20,3 (June 1991), 405-422.

[74] J. Matoušek, "Geometric Range Searching," *ACM Computing Survey,* 26,4 (1994), 421-461.

[75] J. S. B. Mitchell, 'Shortest Paths among Obstacles in the Plane", *Int'l J. Computational Geometry & Applications,* 6,3 (Sept. 1996), 309-332.

[76] J. S. B. Mitchell and C. H. Papadimitriou, "The Weighted Region Problem: Finding Shortest Paths through a Weighted Planar Subdivision", *J. ACM*, 38,1 (Jan. 1991), 18-73.

[77] J. S. B. Mitchell, C. Piatko, and E. M. Arkin, "Computing a Shortest k-Link Path in a Polygon," *Proc. 33rd Annual Symp. Foundations of Comput. Sci.,* Oct. 1992, 573-582.

[78] J. S. B. Mitchell, G. Rote and G. Wōginger, "Minimum Link Path among Obstacles in the Planes," *Algorithmica*, 8 (1992), 431-459.

[79] D. M. Mount, "Intersection Detection and Separators for Simple Polygons" *Proc. 8th Annual ACM Symp. Comput. Geometry*, 1992, 303–311

[80] S. Näher, "LEDA – A Library of Efficient Data Types and Algorithms", Max-Planck-institut für informatik; http://www.mpi-sb.mpg.de/LEDA/leda.html.

[81] A. Okabe, B. Boots and K. Sugihara, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, John Wiley & Sons, Chichester, England, 1992.

[82] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Engelwood Cliffs, NJ, 1982.

[83] E. Papadopoulou, "k-Pairs Non-Crossing Shortest Paths in a Simple Polygon," *Proc. 7th Int'l Symp. Algorithms and Computation,* Dec. 1996, 305-314.

[84] E. Papadopoulou and D. T. Lee, "Efficient Computation of the Geodesic Voronoi Diagram of Points in a Simple Polygon," *Algorithmica*, to appear.

[85] F. P. Preparata and M. I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, 1988.

[86] R. Ravi, R.Sundaram, M. V. Marathe, D. J. Rosenkrantz and S. S. Ravi, "Spanning Trees Short or Small," *SIAM J. Discrete Math.*, 9,2 (May 1996) 178-200.'

[87] H. Rosenberger, "Order k Voronoi Diagrams of Sites with Additive Weights in the Plane," *Algorithmica*, 6 (1991), 153-181.

[88] , J. Sack and J. Urrutia, Handbook of Computational Geometry, Elsevier Sci. Publishers, Amsterdam, 1997.

[89] B. F. Schaudt and R. L. Drysdale, "Multiplicatively Weighted Crystal Growth Voronoi Diagrams," *Proc. 7th Annual ACM Symp. Comput. Geometry*, 1991, 214-223.

[90] B. F. Schaudt and R. L. Drysdale, "Higher-Dimensional Voronoi Diagrams for Convex Distance Functions," *Proc. 4th Canad. Conf. Comput. Geometry*, 1992, 274-279.

[91] C. Schwartz, M. Smid, and J. Snoeyink, "An Optimal Algorithm for the On-Line Closest-Pair Problem," *Algorithmica*, 12,1 (July 1994), 18-29.

[92] T. Shermer and C. Yap, "Probing Near Centers and Estimating Relative Roundness," *Proc. ASME Workshop on Tolerancing and Metrology*, 1995.

[93] S. Suri, "On Some Link Distance Problems in a Simple Polygon," *IEEE Trans. Robotics and Automation,* 6, 1 (1990), 108-113.

[94] K. Swanson, D. T. Lee, and V. L. Wu, "An Optimal Algorithm for Roundness Determination on Convex Polygons," *Comput. Geometry: Theory & Appl.*, 5,4 (Nov. 1995), 225-235.

[95] P. M. Vaidya, "Space-Time Tradeoffs for Orthogonal Range Queries," *SIAM J. Computing*, 18,4 (August 1989) 748-758.

[96] P. M. Vaidya, "Geometry Helps in Matching," *SIAM J. Computing*, 18, 6 (Dec. 1989) 1201-1225.

[97] E. Welzl, "Smallest Enclosing Disks, Balls and Ellipsoids," in New Results and New Trends in Computer Science, H. Maurer, Ed., LNCS, Vol. 555, Springer-Verlag, 1991, 359-370.

[98] C. D. Yang, D. T. Lee and C. K. Wong, "Rectilinear Path Problems Among Rectilinear Obstacles Revisited," *SIAM J. Computing*, 24,3, (June 1995), 457-472.

[99] F. F. Yao, "Computational Geometry," in Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity, J. van Leeuwen, Ed., 1994, 343-389.

[100] Yap, C., "Exact Computational Geometry and Tolerancing," in *Snapshots of Computational and Discrete Geometry*, Avis and Bose, eds., School of Comput. Sci., McGill University, 1995.

11 Further Information

Additional references about various variations of closest pair problems can be found in [18, 21, 62, 91]. For additional results concerning the Voronoi diagrams in higher dimensions and the duality transformation see [15]. For more information about Voronoi diagrams for sites other than points, in various distance functions or norms, see [12, 9, 20, 81, 87, 88, 90]. A recent textbook by de Berg *et al.*[38] contains a very nice treatment of computational geometry in general. More information can be found in [54, 65, 88, 99] The reader who is interested in parallel computational geometry is referred to [13]. For current research activities and results, the reader may consult the Proceedings of the Annual ACM symposium on Computational Geometry, and the following three journals, *Discrete & Computational Geometry*, *International Journal of Computational Geometry & Applications,* and *Computational Geometry: Theory and Applications.* The **ftp** site /pub/geometry/geombib.tar.gz at ftp.cs.usask.ca contains close to ten thousand entries of bibliography in this field.

Those who are interested in the implementations or would like to have more information about available software, can consult http://www.geom.umn.edu/software/cglist/.

The following WWW page on *Geometry in Action* maintained by David Eppstein at http://www.ics.uci.edu/~eppstein/geom.html and computational geometry page by J. Erickson at http://www.cs.duke.edu/~jeffe/compgeom give a comprehensive description of research activities of computational geometry.