A Simple On-line Randomized Incremental Algorithm for Computing Higher Order Voronoi Diagrams*

Franz Aurenhammer
Otfried Schwarzkopf
Institut für Informatik, Fachbereich Mathematik, Freie Universität Berlin,
Arnimallee 2–6, W1000 Berlin 33, Germany

Abstract

We present a very simple algorithm for maintaining order-k Voronoi diagrams in the plane. By using a duality transform that is of interest in its own right, we show that the insertion or deletion of a site involves little more than the construction of a single convex hull in three-space. In particular, the order-k Voronoi diagram for n sites can be computed in time $\mathcal{O}(nk^2 \log n + nk \log^3 n)$ and optimal space $\mathcal{O}(k(n-k))$ by an on-line randomized incremental algorithm whose practicality can be compared with the recent Voronoi diagram algorithm by Guibas, Knuth, and Sharir. The time bound can be improved by a log-factor without losing much simplicity. For $k \geq \log^2 n$, this is optimal for a randomized incremental construction; we show that the expected number of structural changes during the construction is $\Theta(nk^2)$. Finally, by going back to the primal space, we automatically obtain a dynamic data structure that supports k-nearest neighbor queries, insertions, and deletions in a planar set of sites. The structure is very easy to implement, exhibits a satisfactory expected performance, and occupies not more storage than the current order-k Voronoi diagram.

1 Introduction

Ever since the introduction of randomized incremental construction into computational geometry by Clarkson and Shor [CS], the paradigm has gained considerable popularity within the field. It allows for algorithms which are efficient in the expected case — where the expectancy does not depend on an input distribution, but only on random choices made by the algorithm — and which are very often really practical since being conceptually simple and lending themselves to implementation easily. It is thus not astonishing that a considerable number of geometric problems has been shown to be solvable by randomized incremental construction, see e.g. [MMO, S1, S2, W, GKS, BDSTY, BDT, CEGSS].

One of the problems that has eluded efficient implementation by randomized incremental construction is the computation of order-k Voronoi diagrams. Given n points (called *sites*) in the Euclidean plane, and an integer k between 1 and n, the *order-k Voronoi diagram* partitions the plane into regions such that each point within a fixed region has the same k closest sites. These regions are convex polygons since they come from intersecting halfplanes bounded by perpendicular bisectors of sites. The total number of regions and edges is in $\mathcal{O}(k(n-k))$ since the diagram may be viewed as a planar graph with as many vertices; see Lee [L].

An algorithm to compute the order-k Voronoi diagram is called *randomized incremental* if it takes a random permutation $p_1, \ldots, p_n$ of the given sites and considers them in this order. At stage i, it maintains the order-k Voronoi diagram of $\{p_1, \ldots, p_i\}$. Such an algorithm is called *on-line* if it does not consider a site p_j in stage i for $i < j$. An on-line randomized incremental algorithm can thus be used in a dynamic setting, although the time bounds are valid only if sites are added in random order.

There is a recent on-line randomized incremental algorithm by Boissonnat et al. [BDT] which takes expected time $\mathcal{O}(nk^4 \log n)$ and expected space $\mathcal{O}(nk^3)$. Since the output size is in $\mathcal{O}(k(n-k))$, this calls for improvement. In fact, several deterministic (off-line) algorithms have been developed, the fastest (but not simplest) of which for small k is by Aggarwal et al. [AGSS] and achieves a running time of $\mathcal{O}(nk^2 + n \log n)$. Randomized (but not incremental) algorithms taking time $\mathcal{O}(nk^2 + n \log n)$ and $\mathcal{O}(kn^{1+\epsilon})$ have been proposed by Mulmuley [M] and by Clarkson [C], re-

*This research was partially supported by the Deutsche Forschungsgemeinschaft under grant Al 253/1–3, Schwerpunktprogramm "Datenstrukturen und effiziente Algorithmen"and by the ESPRIT II Basic Research Action of the European Community under contract Nos. 3075 and 3299 (project ALCOM and working group "Computing by Graph Transformations"). Furthermore, part of this work was done while the second author was visiting Utrecht University.

spectively.

We investigate why it is so difficult to devise an efficient randomized incremental algorithm for this problem and obtain the following results: If the n sites are added at random while their order-k Voronoi diagram is maintained, the expected number of Voronoi vertices that appear at some intermediate stage during the algorithm is $\Theta(nk^2)$. This implies that we cannot even hope to get somewhere near $\mathcal{O}(nk)$ by randomized incremental construction, since the number of structural changes alone is much bigger. The situation gets even worse if we consider the expected size of the *conflict graph* (introduced by Clarkson and Shor [CS]) which arises during the incremental construction: it is $\Theta(nk^3 \log \frac{n}{k})$. This explains neatly the result by Boissonnat et al. [BDT]: Since they cannot compute the order-k Voronoi diagram alone, but compute all diagrams of order j, for $1 \leq j \leq k$ (a problem shared by most other algorithms cited above), they obtain an additional factor of $\mathcal{O}(k)$.

We use a duality transform to construct the order-k Voronoi diagram via a single convex hull in three dimensions. This releases us from computing all the order-j diagrams for $j < k$ at the same time. In the dual setting, the insertion (and also the deletion) of a site amounts to little more than computing a convex hull in three-space. Exploiting duality further, we are able to apply a simple point location technique to get rid of the conflict graph. What is obtained is an unrivaled simple on-line algorithm for inserting and deleting sites in a planar order-k Voronoi diagram. In particular, the diagram can be computed in $\mathcal{O}(nk^2 \log n + nk \log^3 n)$ expected time by an on-line algorithm which we believe to be highly practical.

By doing things in a bit more sophisticated but still simple fashion, we are able to improve the running time by a log-factor, that is, to $\mathcal{O}(nk^2 + nk \log^2 n)$. This is optimal for a randomized incremental construction if $k \geq \log^2 n$. Sites can now be inserted and deleted in time proportional to the number of structural changes in the diagram, which nicely generalizes a similar result for the usual (order-1) Voronoi diagram [AGSS]. The time for locating a site to be inserted reduces to $\mathcal{O}(k \log^2 n)$.

We then consider the memory space requirements of our algorithms. For an off-line application (all sites are known in advance) we only have to store, at any stage, the current order-k Voronoi diagram and an edge of conflict for each not-yet inserted site. So we can do with optimal $\mathcal{O}(k(n-k))$ space. However, if we implement the algorithms in a manner which is on-line, we have to remember older copies of the diagram in order to facilitate point-location. This seems to lead to $\mathcal{O}(k^2(n-k))$ expected space. We present a simple possibility to reduce this to optimal $\mathcal{O}(k(n-k))$ worst-case space, again making use of our duality transform. This modification does not require any new tools and thus retains the simplicity of the algorithms.

Finally, by going back to the primal space — while still remaining in three dimensions — we automatically get a dynamic data structure for k-nearest neighbor search, allowing both insertions and deletions. We show that the k sites closest to a query point can be collected during the point location, hence keeping the space requirement of this easily implemented structure proportional to the size of the order-k Voronoi diagram of the current set of sites.

2 Complexity of Randomized Incremental Construction

In this section we obtain a general result on the incremental construction of higher order Voronoi diagrams. Let $S = \{p_1, \ldots, p_n\}$ be a set of sites in the plane, and let $k\text{-}Vor(S)$ denote their order-k Voronoi diagram. To simplify the exposition, we assume that the sites in S are in general position, meaning that no three sites are collinear and no four sites are cocircular. We need the following well-known fact.

Fact 1 *Each vertex of k-$Vor(S)$ is the center of a circle with exactly 3 sites of S on the boundary and either $k-1$ or $k-2$ sites of S in its interior.*

Such a vertex v is said to be of the *close-type* if the corresponding circle contains $k-1$ sites, and v is of the *far-type*, otherwise.

We observe that any incremental algorithm that constructs $k\text{-}Vor(S)$ by inserting the sites $p_1, \ldots, p_n$ in this order must create all the vertices appearing in all the intermediate diagrams $k\text{-}Vor(\{p_1, \ldots, p_{k+1}\})$, $\ldots$, $k\text{-}Vor(\{p_1, \ldots, p_n\})$. Note that $k\text{-}Vor(\{p_1, \ldots, p_i\})$ is undefined for $i < k$; it contains only one region covering the whole plane if $i = k$, and it is the furthest-site Voronoi diagram of $\{p_1, \ldots, p_{k+1}\}$ (which has only $\mathcal{O}(k)$ vertices) if $i = k+1$. We will thus assume in the following that the construction starts at stage $k+2$.

Let us now take a random permutation $p_1, \ldots, p_n$ of S and determine the expected number of vertices that are created in stage i, for $i \geq k+2$. These are exactly the vertices of $k\text{-}Vor(\{p_1, \ldots, p_i\})$ that are not vertices of $k\text{-}Vor(\{p_1, \ldots, p_{i-1}\})$. Using Seidel's *backwards analysis* [S1], we can reformulate our question as follows: What is the expected number of vertices in $k\text{-}Vor(\{p_1, \ldots, p_i\})$ that disappear when a random site in $\{p_1, \ldots, p_i\}$ is removed? This number can be expressed as

$$\sum_{v \in k\text{-}Vor(\{p_1, \ldots, p_i\})} Prob\{v \text{ disappears}\}.$$

When does a fixed vertex v disappear? By Fact 1 this is the case iff we remove one of the three sites whose circumcircle defines v, or one of the $k-1$ or $k-2$ sites in that circle. The probability for this event is $\frac{k+1}{i}$ or $\frac{k+2}{i}$, and we sum over $\Theta(k(i-k))$ vertices, according to the complexity of an order-k Voronoi diagram for i sites. This shows that the expected number of vertices created in stage i is $\Theta(k^2(1-\frac{k}{i}))$. Interestingly, this number does not depend much on i. Summing over i, we obtain:

Lemma 2 *The expected number of order-k Voronoi diagram vertices created by a randomized incremental algorithm for a set of n sites in the plane is in $\Theta(k^2(n-k))$.*

This is a disappointing result, since the size of the order-k Voronoi diagram is only $\Theta(k(n-k))$. It is thus not possible to find a randomized incremental algorithm which works in time optimal with respect to the output size. However, most known algorithms for the computation of order-k Voronoi diagrams also take time $\Omega(k^2 n)$, so we might still want to find a randomized incremental algorithm with its typical and desirable properties, namely being simple and perhaps online.

3 A Duality Transform for Order-k Voronoi Diagrams

In this section we introduce a duality transform that relates order-k Voronoi diagrams in the plane to certain convex hulls in three-space. This transform is the heart of the algorithms to be described in this paper. It allows us to insert and delete sites in an order-k Voronoi diagram in a smooth fashion via the computation of convex hulls. It further provides a possibility to determine a Voronoi vertex that gets destroyed when inserting a site by simply locating that site in a triangulation.

The transform works as follows. Recall that a region of the order-k Voronoi diagram is the locus of all points in the xy-plane that have the same k closest sites. Let now $T \subset S$ be any subset of k sites. We transform T — and thus the region it may define in the diagram — into a dual point, $q(T)$, in three-dimensional space by taking the centroid of T and lifting it up in z-direction. More precisely,

$$q(T) = \frac{1}{k}\left(\sum_{p\in T} p\,,\ \sum_{p\in T}\langle p,p\rangle\right).$$

Note that the first term gives the x- and y-coordinates and expresses the *centroid* $\frac{1}{|T|}\sum_{p\in T} p$ of T (also called *center of mass*), while the other term gives the z-coordinate of the point $q(T)$ in three dimensions.

Consider the set, $Q_k(S)$, of all dual points that can be obtained from k-subsets of S in this way. That is, $Q_k(S) = \{q(T) \mid T \subset S, |T| = k\}$. Clearly, $|Q_k(S)| = \binom{n}{k}$. Take the convex hull of $Q_k(S)$ and ignore all of its facets that are invisible from $(0,0,-\infty)$. The remaining *lower convex hull* has the following surprising property.

Lemma 3 *The lower convex hull of $Q_k(S)$ is dual to k-Vor(S).*

A proof has been given in [A1]. Here we prove a stronger result, by mapping each k-subset T of S into a non-vertical plane in three-space,

$$plane(T):\ z = \frac{2}{k}\sum_{p\in T}\langle x,p\rangle - \frac{1}{k}\sum_{p\in T}\langle p,p\rangle,$$

and considering the (unbounded) convex polyhedron, $poly(S)$, that comes from intersecting the halfspaces above all $plane(T)$.

Lemma 4 *k-Vor(S) is the vertical projection of $poly(S)$ onto the xy-plane.*

Proof: Let x be a point in the xy-plane, lying in the region of $T_0 \subset S$ in the order-k Voronoi diagram. We have to show that T_0 is the k-subset whose plane intersects the vertical line through x in the uppermost position.

Let T and T' be two k-subsets of S and suppose them to be of the form $T' = (T\setminus\{s\})\cup\{s'\}$, for $s \neq s'$. We show that if $d(x,s) < d(x,s')$ then $plane(T)$ lies above $plane(T')$ at point x. (d is the Euclidean distance function.) This proves the claim, since for any k-subset $T^* \neq T_0$ we can find a sequence $T_0, T_1, \ldots, T_m = T^*$ such that consecutive subsets are of this special form and the argument can be applied repeatedly, showing that $plane(T_i)$ lies above $plane(T_{i+1})$ at point x.

From $d(x,s) < d(x,s')$ we immediately get $2\langle x,s\rangle - \langle s,s\rangle > 2\langle x,s'\rangle - \langle s',s'\rangle$. Adding

$$2\sum_{p\in T\cap T'}\langle x,p\rangle - \sum_{p\in T\cap T'}\langle p,p\rangle$$

to both sides yields

$$2\sum_{p\in T}\langle x,p\rangle - \sum_{p\in T}\langle p,p\rangle > 2\sum_{p\in T'}\langle x,p\rangle - \sum_{p\in T'}\langle p,p\rangle.$$

If we further divide by k, these terms express the heights of the vertical projections of x onto $plane(T)$ and $plane(T')$. □

Lemma 4 implies Lemma 3 if we apply the point-plane polarity with respect to the paraboloid of revolution $z = \langle x,x\rangle$. This polarity maps a point (a,b,c) into the plane $z = 2ax + 2by - c$ and vice versa, and preserves the relative position between points and planes; see e.g. [A2, E]. In our case, it maps the plane $plane(T)$ into the point $q(T)$. Consequently, the polyhedron $poly(S)$ — and thus its projection k-$Vor(S)$ — are dual to the lower convex hull, $\mathcal{C}(S)$, of $Q_k(S)$. Regions of k-$Vor(S)$ correspond to vertices of $\mathcal{C}(S)$ which will be called *corners* to avoid confusion with the vertices of the diagram. The latter translate to facets of the lower convex hull. By our general position assumption all these facets are triangles. Finally, the dual object of an edge separating two regions of k-$Vor(S)$ is again an edge that now connects the two corresponding corners. It should be noted that for the case $k = 1$, we have just described the usual embedding of a closest-site Voronoi diagram in three-dimensional space.

In order to construct k-$Vor(S)$, we thus could just compute the set $Q_k(S)$ and determine its lower convex hull. Unfortunately, $Q_k(S)$ contains $O(n^k)$ points, only $O(k(n-k))$

of which are corners. However, we can apply our duality transform to parts of the order-k Voronoi diagram where we already know which k-subsets will give rise to corners, and we will use this in the next section.

4 Inserting a Site

In the light of this duality, we choose to maintain the lower convex hull $\mathcal{C}_i$ of $\mathcal{Q}_k(\{p_1, \ldots, p_i\})$ during the incremental construction instead of the order-k Voronoi diagram itself. We show that the insertion of a site involves little more than the computation of the lower convex hull of a certain set of new corners which can be found easily.

Besides the coordinates of the corners and the triangulation representing $\mathcal{C}_i$ the algorithm only needs to store *labels* for the hull edges. Each edge e of $\mathcal{C}_i$ is dual to an edge of $k\text{-}Vor(\{p_1, \ldots, p_i\})$. The latter is a piece of the bisector of two sites $p, q \in \{p_1, \ldots, p_i\}$, and we label e with the pair $\{p, q\}$. This labeling will be maintained with $\mathcal{C}_i$. Note that the labels of a triangle Δ of $\mathcal{C}_i$ are of the form $\{p, q\}, \{q, r\}, \{r, p\}$. The Voronoi vertex corresponding to Δ is just the center of the circumcircle of p, q, and r; see Fact 1. So the vertex and its circle can be restored from the labels in constant time.

The construction is initialized with the determination of $\mathcal{C}_{k+1}$. $\mathcal{Q}_k(\{p_1, \ldots, p_{k+1}\})$ contains $\binom{k+1}{k} = k + 1$ points and can be found in time $\mathcal{O}(k)$, by first calculating the centroid of $p_1, \ldots, p_{k+1}$ (and its height) and subtracting from it each particular site p_i which gives the point $q(\{p_1, \ldots, p_{k+1}\} \setminus \{p_i\})$. We compute $\mathcal{C}_{k+1}$ in time $\mathcal{O}(k \log k)$, using the randomized incremental algorithm by Guibas, Knuth, and Sharir [GKS]. An edge of $\mathcal{C}_{k+1}$ is labeled by $\{p, q\}$ if the corners it connects have been obtained from the centroid above by subtracting p and q, respectively. Note that we have just described an unusual way to compute the furthest-site Voronoi diagram of $k + 1$ sites.

The generic step of the algorithm will be the insertion of site p_i into $\mathcal{C}_{i-1}$, for $i \geq k+2$. Since $\mathcal{Q}_k(\{p_1, \ldots, p_{i-1}\}) \subset \mathcal{Q}_k(\{p_1, \ldots, p_i\})$, we can find $\mathcal{C}_i$ by determining the set $\mathcal{P}$ of all new corners and constructing the lower convex hull of $\mathcal{P}$ and the corners of $\mathcal{C}_{i-1}$. Intuitively speaking, the insertion of p_i looks as follows.

- Identify all triangles of $\mathcal{C}_{i-1}$ destroyed by p_i and cut them out. Let $\mathcal{B}$ be the set of corners on the boundary of the hole.

- Calculate the set $\mathcal{P}$ of all new corners created by p_i.

- Compute the lower convex hull of $\mathcal{P} \cup \mathcal{B}$, using the algorithm by Guibas, Knuth, and Sharir [GKS], and fill the hole. This gives $\mathcal{C}_i$.

- Label the newly constructed edges of $\mathcal{C}_i$.

All these tasks have a simple and nice implementation which is to be described in the remainder of this section.

Let us for the moment assume that we have a point location oracle, which — given a new site p_i — presents us a triangle of $\mathcal{C}_{i-1}$ which is destroyed by p_i. By Lemma 5 below, the triangles of $\mathcal{C}_{i-1}$ that are no longer part of $\mathcal{C}_i$ form a simply connected surface, $\mathcal{U}$, on $\mathcal{C}_{i-1}$. A triangle Δ gets destroyed iff p_i happens to lie within the circumcircle whose center is the dual Voronoi vertex of Δ. According to Fact 1, this vertex then either changes type or disappears. In both cases Δ will not be present in $\mathcal{C}_i$. So we can test in constant time whether Δ will be destroyed. Using the point location oracle and a graph search, the surface $\mathcal{U}$ can thus be removed from $\mathcal{C}_{i-1}$ in time $\mathcal{O}(n_i)$, where n_i is the number of triangles in $\mathcal{U}$. Clearly the set $\mathcal{B}$ of corners on the boundary of $\mathcal{U}$ can be identified during this process.

For a region R of an order-k Voronoi diagram, let $near(R)$ denote its defining k-subset. Let further $near(e) = near(R) \cap near(R')$ if the edge e separates the regions R and R'.

Lemma 5 *The union, $\mathcal{R}$, of all regions R of k-$Vor(\{p_1, \ldots, p_i\})$ with $p_i \in near(R)$ is a polygon which is star-shaped as seen from p_i. The edges of k-$Vor(\{p_1, \ldots, p_{i-1}\})$ in the interior of $\mathcal{R}$ form a connected graph, $\mathcal{G}$.*

Proof: $\mathcal{R}$ is the union of intersections of halfplanes that contain p_i and are bounded by bisectors of p_i and some other site. Since such halfplanes are star-shaped with respect to p_i, $\mathcal{R}$ has the same property.

Assume now that $\mathcal{G}$, the part of $k\text{-}Vor(\{p_1, \ldots, p_{i-1}\})$ lying in $\mathcal{R}$, is disconnected. This implies that there is a region A of $k\text{-}Vor(\{p_1, \ldots, p_{i-1}\})$ which intersects the boundary of $\mathcal{R}$ in a disconnected set Γ. Now, consider the order-$(k+1)$ Voronoi diagram of $\{p_1, \ldots, p_i\}$. There is region A' in that diagram with $near(A') = near(A) \cup \{p_i\}$, and $A \cap \mathcal{R} \subseteq A'$. In fact, $A \cap \mathcal{R}$ is exactly that part of A' where p_i is not the furthest point in $near(A')$. Furthermore, Γ coincides with the boundary of the region of p_i in the furthest-site Voronoi diagram of $near(A')$ within A', which is well known to be connected. $\square$

Figure 1 illustrates the insertion of site p_i into the order-3 Voronoi diagram of 7 sites. The boundary of $\mathcal{R}$ is drawn with bold lines. The graph $\mathcal{G}$ of destroyed edges is shown dashed.

Now we want to determine the set $\mathcal{P}$ of new corners. Luckily, we have the following.

Lemma 6 *Each edge e in k-$Vor(\{p_1, \ldots, p_{i-1}\})$ that gets totally or partially destroyed by the insertion of p_i gives rise to a new region R in k-$Vor(\{p_1, \ldots, p_i\})$. All new regions are generated in this way. In particular, $near(R) = near(e) \cup \{p_i\}$.*

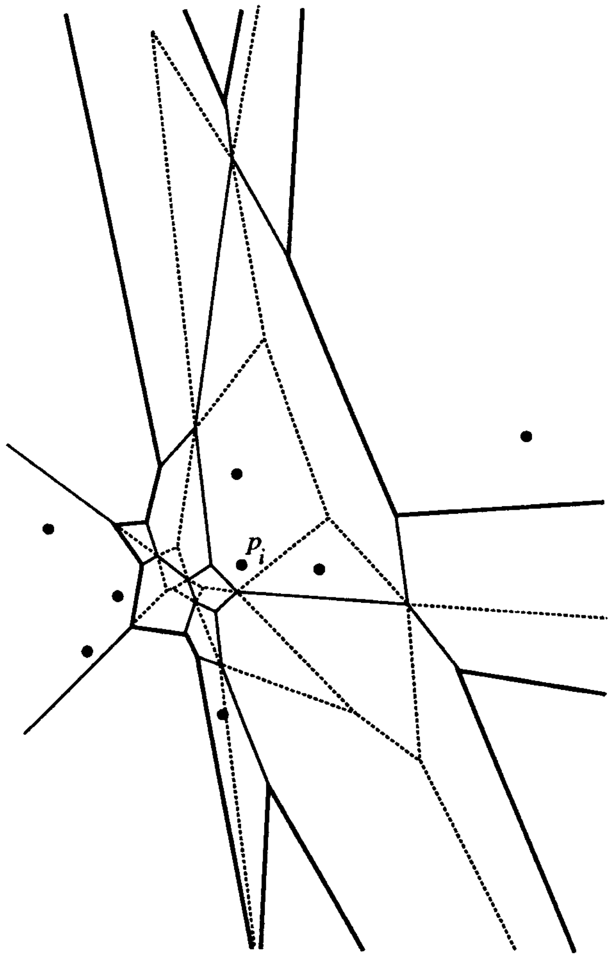

Figure 1: Inserting a site.

Proof: Let x be a point in the part of e which gets destroyed. Then there exists a circle around x that encloses $near(e)$ and p_i but does not enclose any other site in $\{p_1, \ldots, p_i\}$. But this means that x is contained in the region R of $k\text{-}Vor(\{p_1, \ldots, p_i\})$ with $near(R) = near(e) \cup \{p_i\}$.

For the reverse direction, let R be a region in $k\text{-}Vor(\{p_1, \ldots, p_i\})$ with $p_i \in near(R)$. Then there exists a circle enclosing only the sites in $near(R)$. When removing p_i and shrinking the circle until two sites lie on its boundary, its center will be a point on an edge e of $k\text{-}Vor(\{p_1, \ldots, p_{i-1}\})$ with $near(e) = near(R) \setminus \{p_i\}$. ◻

Lemma 6 implies that we can collect $\mathcal{P}$ while walking along the surface $\mathcal{U}$, and it also implies that $|\mathcal{P}| \in \mathcal{O}(n_i)$. Note that we can compute the coordinates of the new corner that arises from an edge e of $\mathcal{U}$ in constant time, by subtracting from one corner of e the appropriate site in the label of e and then adding p_i.

After having computed the lower convex hull of $\mathcal{P} \cup \mathcal{B}$ in time $\mathcal{O}(n_i \log n_i)$ we finally have to label this newly created edges of $\mathcal{C}_i$. It is possible to do this in time $\mathcal{O}(n_i)$, via the coordinates of the corners since we already know the labels of the edges on the boundary of $\mathcal{U}$. Let Δ be a triangle with only one labeled edge e and let c be its corner opposite to e. The *coordinates* of the unknown third site appearing in the labels of Δ are obtained by subtracting from c one corner of e and adding the appropriate site in the label of e. Here it is necessary to know whether Δ is dual to a close-type or a far-type vertex; the type can be read off from e's label, provided we store edge labels as oriented pairs of sites. There might arise numerical problems if we try to identify a site via its coordinates. However, it should be observed that if the sites are given in integer coordinates then also all corners will have this property. (Of course, we would use the coordinates of the corners $q(T)$ only multiplied by k, to avoid unnecessary divisions and roundoff errors.) Anyway, we shall see another — quite simple — technique of labeling all the edges of $\mathcal{U}$ in Section 7.

We have just proved:

Lemma 7 *Given $\mathcal{C}_{i-1}$ we can insert a new site p_i — which was not known before — in time $\mathcal{O}(n_i \log n_i)$, provided we know a triangle of $\mathcal{C}_{i-1}$ that is destroyed by p_i.*

5 The Site Location Problem

We give now a method to find the required starting triangle for our graph search. The usual technique to attack this problem is by maintaining a conflict graph [CS] or by using a related method that is on-line and obtains the same time bounds, see e.g. [BDT]. However, we can show [AS] that the conflict graph of our problem has an expected size of $\mathcal{O}(nk^3 \log \frac{n}{k})$, which would dominate the time for the construction steps. We circumvent this difficulty by using a two-staged point-location structure, which we nearly get for free and which achieves a total running time of $\mathcal{O}(nk \log^3 n)$ for all site location steps. We profit from one more nice property of our duality transform.

Lemma 8 *Let Δ be a triangle of $\mathcal{C}_{i-1}$, let v denote its dual Voronoi vertex, and let O be the circumcircle corresponding to v. Then O encloses the projection of Δ on the xy-plane.*

Proof: Let p, q, and r be the three sites defining O. Depending on whether v is a close-type vertex or a far-type vertex, O encloses a $(k-1)$-subset or a $(k-2)$-subset $T \subset \{p_1, \ldots, p_{i-1}\}$. By definition of $\mathcal{Q}_k(\{p_1, \ldots, p_{i-1}\})$, the projections of the corners of Δ are the centroids of $T \cup \{p\}$, $T \cup \{q\}$, and $T \cup \{r\}$ in the former case, and the centroids of $T \cup \{p, q\}$, $T \cup \{p, r\}$, and $T \cup \{q, r\}$ in the latter. Clearly, these centroids lie within the convex hull of

$T \cup \{p, q, r\}$, which lies within O. This implies the lemma. □

Recall that Δ is destroyed by changing the set of sites enclosed by O. It is thus sufficient for our purposes to locate a new site p_i in the triangulation of the plane given by the projection of $\mathcal{C}_{i-1}$.

We use a technique similar to that used in [GKS, AESW]. We do not remove the triangles of $\mathcal{C}_{i-1}$ that get destroyed by the insertion of p_i from our data structure, but mark them as old. In the moment of becoming old, each triangle gets a pointer to the newly added *cap*, the lower convex hull of $\mathcal{P} \cup \mathcal{B}$ that has been added to $\mathcal{C}_{i-1}$ in order to obtain $\mathcal{C}_i$.

The structure maintained at stage i during the course of the algorithm is thus the family of all lower convex hulls $\mathcal{C}_j$, for $k+1 \leq j \leq i$, which can be imagined as a sequence of caps that have been added to the initial cap $\mathcal{C}_{k+1}$; see Figure 2. For each cap, we need a point location structure for the triangulation it projects to. But recall that we have used the algorithm by Guibas, Knuth and Sharir [GKS] to compute the caps, which can be implemented in such a fashion that it automatically gives a point location structure for these triangulations, with an expected query time of $\mathcal{O}(\log^2 n_j) = \mathcal{O}(\log^2 n)$. So, in order to obtain the two-staged point-location structure just described, nothing has to be added to our algorithm but the establishment of some pointers from triangles to caps.

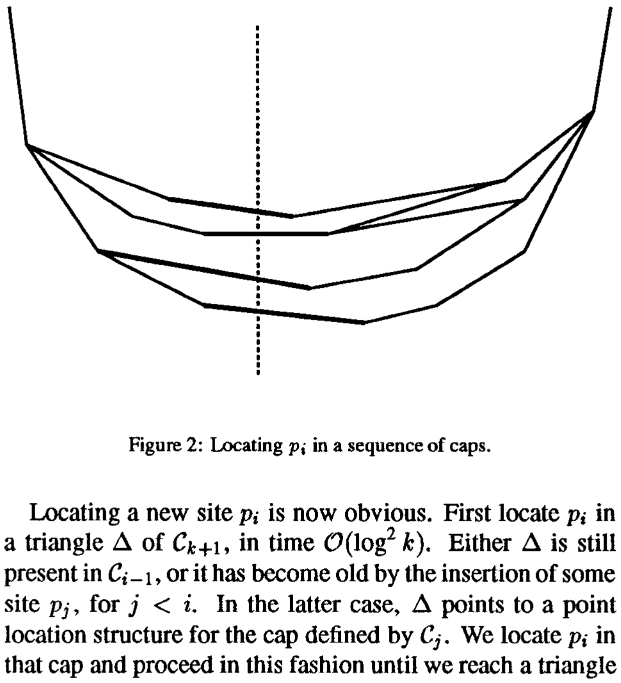

Figure 2: Locating p_i in a sequence of caps.

Locating a new site p_i is now obvious. First locate p_i in a triangle Δ of $\mathcal{C}_{k+1}$, in time $\mathcal{O}(\log^2 k)$. Either Δ is still present in $\mathcal{C}_{i-1}$, or it has become old by the insertion of some site p_j, for $j < i$. In the latter case, Δ points to a point location structure for the cap defined by $\mathcal{C}_j$. We locate p_i in that cap and proceed in this fashion until we reach a triangle of $\mathcal{C}_{i-1}$.

In how many caps do we have to locate a fixed site q? Let Δ_j be the triangle of $\mathcal{C}_j$ containing q in its projection. We are interested in the expected number of indices j such that $\Delta_j \neq \Delta_{j-1}$. This number can be expressed as

$$\sum_{j=k+2}^{n} Prob\{\Delta_j \neq \Delta_{j-1}\}.$$

Note that q is fixed and the probability is taken over all permutations of $\{p_1, \ldots, p_n\}$. Again, we use a backwards analysis to see that this probability is the same as the probability that Δ_j disappears when a random site in $\{p_1, \ldots, p_j\}$ is removed. That probability, however, is $\Theta(\frac{k}{j})$ by the argument of Section 2. This shows that, for a fixed site, the expected number of point locations in caps is $\Theta(k \log \frac{n}{k})$.

According to Lemma 7 and Lemma 2, the expected runtime of our algorithm *without* the point location steps is in

$$\mathcal{O}(k \log k) + \sum_{i=k+2}^{n} \mathcal{O}(n_i \log n_i) = \mathcal{O}(k^2(n-k) \log n).$$

We conclude the following result for the computation of order-k Voronoi diagrams.

Theorem 9 *The order-k Voronoi diagram of n sites in the plane can be computed in expected time $\mathcal{O}(k^2(n-k)\log n + nk \log^3 n)$ by a very simple, randomized, incremental, on-line algorithm.*

6 Reducing the Space Complexity

The simple on-line implementation of our algorithm seems to have a price of $\mathcal{O}(k^2(n-k))$ expected space, which might be prohibitive for large values of k. We suggest the following addition to the algorithm that can be incorporated easily and which does not change the asymptotic running time.

Fix some suitable constant c and reserve $ck(n-k)$ storage for the computation of the order-k Voronoi diagram of $\{p_1, \ldots, p_n\}$. Run the algorithm until this amount of storage is exhausted. Now, discard everything in the memory besides the $\mathcal{O}(k(i-k))$ corners of the actual lower convex hull $\mathcal{C}_i$. Rebuild $\mathcal{C}_i$ by inserting its corners randomly with the algorithm by Guibas, Knuth, and Sharir [GKS], in order to get a point location structure for the projection of $\mathcal{C}_i$. If c has been chosen large enough, this rearrangement of memory happens only $\mathcal{O}(k)$ times in the expected case. (The expected number of new Voronoi vertices per insertion is $\mathcal{O}(k^2)$; see Section 2.) Since every rearrangement can be done in time $\mathcal{O}(k(n-k)\log n)$, the total time required for these steps is $\mathcal{O}(k^2(n-k)\log n)$. After verifying that the point location time can only get better by this modification, we obtain

Theorem 10 *The algorithm of Theorem 9 can be modified to run with optimal $\mathcal{O}(k(n-k))$ space in the same asymptotic running time. The modification does not require any new tools.*

Note that this space bound is only true in the expected case, since the space requirement of the convex hull algorithm in [GKS] has an expected bound only. However, it is easy to convert an expected space bound into a worst case bound by stopping the algorithm if it exceeds its space limit, and restarting it with a new choice of the random permutation. In this fashion every randomized algorithm using expected space S in expected time T can be converted into one using at most $(1+c)S$ space in the worst case in expected time $\frac{1+c}{c}T$, for any $c > 0$.

We further mention that the space requirement of our algorithm also drops to optimal $\mathcal{O}(k(n-k))$ if we implement it as off-line, by maintaining an edge of the conflict graph for every not-yet inserted site by the same technique as before. In this case, we only need to maintain the current lower hull, which has a complexity of $\mathcal{O}(k(n-k))$ in the worst case.

7 Speeding Up the Algorithm

Let us here present a more elaborate (and slightly more complicated) way of inserting a site p_i in time $\mathcal{O}(n_i)$ — instead of $\mathcal{O}(n_i \log n_i)$ — which will finally lead to an optimal randomized incremental on-line algorithm.

As has been observed in the proof of Lemma 5, $k\text{-}Vor(\{p_1,\ldots,p_{i-1}\})$ coincides with the order-$(k+1)$ Voronoi diagram of $\{p_1,\ldots,p_i\}$ in the star-shaped domain influenced by p_i. It is well known that $k\text{-}Vor(\{p_1,\ldots,p_i\})$ can be obtained from the latter diagram by constructing the furthest-site Voronoi diagram of $near(R)$ within each of its regions R. We thus could use the algorithm of Aggarwal et al. [AGSS] to construct in time $\mathcal{O}(n_i)$ these furthest-site diagrams in the domain of influence of p_i, and then could "glue" them together, in order to obtain $k\text{-}Vor(\{p_1,\ldots,p_i\})$.

We find it more simple and elegant to perform a similar task in the dual environment. In particular, we need not glue together newly constructed objects, but fill in some missing convex hull edges instead. To this end, we reconsider the distinction between far-type and close-type vertices.

Any far-type vertex v which gets destroyed by the insertion of p_i reappears as a close-type vertex v' in $k\text{-}Vor(\{p_1,\ldots,p_i\})$. Consider the edges e_1, e_2, e_3 adjacent to v. Since v is a far-type vertex, these edges have different $(k-1)$-subsets $near(e_1)$, $near(e_2)$ and $near(e_3)$ which, by Lemma 6, give rise to the three regions adjacent to v'. We can thus for every destroyed triangle Δ that is dual to a far-type vertex create three new corners and connect them as a triangle Δ'. The labels for the edges of Δ' clearly are the same as those for Δ.

All close-type triangles within the set $\mathcal{P}$ of new corners are created this way. The triangles having one corner in $\mathcal{P}$ and two in the set $\mathcal{B}$ of corners on the boundary of the hole are also of the close-type. They can be constructed and labeled from local information during the graph search. The triangles between two corners in $\mathcal{P}$ and one in $\mathcal{B}$ are of the far-type. We further have:

Fact 11 [L] *Every region of an order-k Voronoi diagram has at least one close-type vertex.*

That is, each corner in $\mathcal{P}$ belongs to at least one close-type triangle. We thus have not only constructed (and labeled) all close-type triangles of the lower convex hull of $\mathcal{P} \cup \mathcal{B}$ but the whole set $\mathcal{P}$ as well. The lemma below implies that we have nearly constructed the lower convex hull $\mathcal{C}_i$.

Lemma 12 *The edges of the close-type triangles of the lower convex hull of $\mathcal{P} \cup \mathcal{B}$ form a single connected component.*

Proof: Assume that there exists a corner $c \in \mathcal{P}$ which is not connected to some corner in $\mathcal{B}$. This implies a closed sequence of adjacent far-type triangles in $\mathcal{C}_i$ surrounding c. Hence in the dual setting there is a cycle E of edges in $k\text{-}Vor(\{p_1,\ldots,p_i\})$ spanned by far-type vertices. Recall that such a far-type vertex v is defined by a subset $T(v) \subset \{p_1,\ldots,p_i\}$ of $k+1$ sites, three of which defining the circumcircle and $k-2$ lying within it. Now consider an edge (v, v') of E. Let this edge be part of the bisector of sites p and q, so $p, q \in T(v) \cap T(v')$. Since we deal with far-type vertices, we have $near(v,v') = T(v) \setminus \{p,q\} = T(v') \setminus \{p,q\}$, which implies $T(v) = T(v')$. So all vertices within E are defined by the same $(k+1)$-subset T. Consequently, the cycle E will also appear in $k\text{-}Vor(T)$. But $|T| = k+1$ so that $k\text{-}Vor(T)$ is just the furthest-site Voronoi diagram of T which is a tree. This is a contradiction. $\square$

There is still a little problem with the computation of set $\mathcal{P}$. According to Lemma 6, the same corner may be generated by more than one destroyed edge. This was no problem in the simple algorithm, since multiple corners were automatically deleted by the convex-hull algorithm. Now, however, we have to make sure that when considering two edges generating the same corner we get pointers to the same new object. Fortunately, this can be taken care of easily during the graph search, due to the following.

Lemma 13 *The edges e of k-Vor$(\{p_1,\ldots,p_{i-1}\})$ fall into several classes with different $near(e)$. The destroyed edges of each such class form a connected graph.*

Proof: Observe that classes change if and only if the edges lie in different regions of $k\text{-}Vor(\{p_1,\ldots,p_i\})$. Within such a region R, the edges of its corresponding class are part of the closest-site Voronoi diagram of $\{p_1,\ldots,p_i\} \setminus near(R)$. This part is well known to be connected. $\square$

We are left with the problem of filling and labeling some not-yet triangulated holes of $\mathcal{C}_i$ the boundaries of which are single polygonal cycles. (In fact, it can be shown that the projection of a hole on the xy-plane is convex.) It is thus easy to form the convex hull of the corners in such a cycle in linear time, using the algorithm by Clarkson and Shor [CS] (but without the conflict graph) or by Guibas, Knuth, and Sharir [GKS] (but without the point location part). Finally, labeling the new edges is straightforward, by starting at the boundary of each hole and exploiting the fact that two edge labels of a triangle already imply the third.

Lemma 14 *$\mathcal{C}_i$ can be constructed from $\mathcal{C}_{i-1}$ in time $\mathcal{O}(n_i)$ if a triangle of $\mathcal{C}_{i-1}$ that is destroyed by p_i is known.*

It is also possible to speed up by a log-factor the rest of the algorithm (i.e., the site location steps and the memory rearrangement) without affecting its optimal space complexity. Since the linear-time construction of the lower convex hull of $\mathcal{P} \cup \mathcal{B}$ does not yield for free a point location structure for its projection, we postprocess this triangulation using the algorithm by Seidel [S2]. This takes expected time $\mathcal{O}(n_i)$ and allows us to locate the triangle containing a new site in expected time $\mathcal{O}(\log n)$. Similarly, we postprocess the current lower convex hull $\mathcal{C}_j$ during the rearranging of memory in expected time $\mathcal{O}(k(j-k))$. This gives us

Theorem 15 *The order-k Voronoi diagram of n sites in the plane can be computed in expected time $\mathcal{O}(k^2(n-k) + kn\log^2 n)$ and worst-case space $\mathcal{O}(k(n-k))$ by a reasonably simple, randomized, incremental, on-line algorithm.*

This complexity is optimal for a randomized incremental algorithm if $k \geq \log^2 n$.

There is an additional feature that may us let prefer this algorithm to the previous one. When computing the convex hull for one of the holes using the algorithm of [GKS], the primitive operation is to check for two adjacent triangles whether they form a convex or a concave angle in space. In our case, the two triangles will be formed by four corners lying on the boundary of a hole.

But the (yet to be filled in) triangles to appear in the hole are all of the far-type. By the proof of Lemma 12 all their corresponding vertices are thus defined by a fixed set T of $k+1$ sites, implying that the corners on the boundary of the hole correspond to k-subsets of the form $T \setminus \{p\}$. Without having to know T itself, we can easily determine the 'missing' site p for every such corner on the boundary from the labels of the edges along the boundary. If we do this before starting to compute the convex hull for such a hole, every primitive triangle test can be considered as a test involving four sites. Furthermore, it is not difficult to verify that this is just the standard primitive for computing Voronoi diagrams in general: Given four sites, does the fourth site lie within the circle spanned by the other three?

This implies that we do not need the *coordinates* of the corners during the insertion step — all we need is the combinatorial structure of the lower convex hull plus the edges labels. This makes this algorithm much more reliable with respect to numerical issues. Of course, we still need the coordinates for the point location step—but the z-coordinates of corners are never used.

8 Deleting a Site

We promised in the Introduction that not only insertions but also deletions of sites can be handled by our methods in a simple and efficient way. Indeed, the surface $\mathcal{U}$ of triangles on $\mathcal{C}_i$ getting destroyed by the deletion of a site p_j, for $j \leq i$, can be identified analogously. It just consists of all triangles whose corresponding circumcircles enclose p_j. A starting triangle with this property can be obtained by finding some edge that uses p_j as a label. (By realizing labels as pointers between edges and sites, this is easy to do.) Note that there must exist such an edge. One can further collect during the graph search in $\mathcal{U}$ all corners that will reappear.

Lemma 16 *For each edge e in k-Vor$(\{p_1,\ldots,p_i\})$ that gets totally or partially destroyed by the deletion of p_j, a region R of k-Vor$(\{p_1,\ldots,p_i\} \setminus \{p_j\})$ will reappear. All such regions are created in this way. In particular, near$(R) =$ (near$(e) \setminus \{p_j\}) \cup \{p,q\}$, where $\{p,q\}$ is the label of e.*

Proof: The argument is similar to that for Lemma 6. □

We are left with the problem of constructing and labeling the lower convex hull of these corners. Clearly this can be done analogously to the insertion step described in Section 4. By the lemma below, also the linear-time method of Section 7 carries over.

Lemma 17 *When deleting p_j from k-Vor$(\{p_1,\ldots,p_i\})$, the edges within the reconstructed star just are the edges within the star constructed by inserting p_j into the order-$(k+1)$ Voronoi diagram of $\{p_1,\ldots,p_i\} \setminus \{p_j\}$.*

Proof: Arguments similar to those used in the proof of the second part of Lemma 5 yield the assertion. □

We conclude:

Lemma 18 *Let n_j denote the number of structural changes in k-Vor$(\{p_1,\ldots,p_i\})$ caused by the deletion of p_j. Then p_j can be deleted from $\mathcal{C}_i$ in time $\mathcal{O}(n_j \log n_j)$ by a very simple algorithm, and in time $\mathcal{O}(n_j)$ by a reasonably simple algorithm.*

9 Dynamic k-Nearest Neighbor Search

In the light of Lemma 4, we can also run our algorithms for inserting and deleting sites in the primal space, by maintaining the projection polyhedron $poly(\{p_1,\ldots,p_i\})$ rather than its dual object $\mathcal{C}_i$. Of course, this can be done by exactly the same algorithm, just by *interpreting* some values in a different way.

However, when we consider the current data structure not as the lower convex hull C_i but as the order-k Voronoi diagram, it would be more useful to implement the site location step in a different fashion. In fact, we would like to be able to locate an arbitrary point in the diagram, thus giving us a query structure for k-nearest neighbors queries. We do this by modifying our algorithm, so that it not only maintains the order-k Voronoi diagram of $\{p_1, \dots, p_i\}$, but the *bottom-vertex-triangulation* of the diagram. For every region of the diagram (which is a convex polygon) we choose the (lexicographically) smallest vertex v and triangulate the region by inserting edges between v and all vertices not adjacent to v.

The analysis of Section 2 then remains essentially true. Now, the basic object to be considered in the analysis is a bottom-vertex-triangle (bv-triangle for short) and not a vertex of the diagram, but the probability that some fixed bv-triangle is destroyed when removing a random site from the diagram is still $\Theta(k/i)$ when there are i sites in the diagram. It follows that the expected number of bv-triangles considered during the course of a randomized incremental algorithm is $\mathcal{O}(k^2(n-k))$ as well.

The site location step is now implemented by locating the new site p_i in the bv-triangulation of the current $k\text{-}Vor(\{p_1, \dots, p_{i-1}\})$. If p_i lies in triangle Δ of this triangulation, then at least one of the vertices of Δ is destroyed by the insertion of p_i.

The location of an (arbitrary) point q in the bottom-vertex-triangulation of $k\text{-}Vor(\{p_1, \dots, p_i\})$ is done in essentially the same fashion as before. Any destroyed bv-triangle is marked as old and gets a pointer to a point location structure for the bv-triangulation of the new part of the diagram. Again, the previous analysis holds, giving us an $\mathcal{O}(nk^2 + nk\log^2 n)$ algorithm for the computation of order-k Voronoi diagrams which at the same time produces a point location structure for the diagram with query time $\mathcal{O}(k\log^2 n)$. (Note that this is not as bad as it seems, since such a query would be used to retrieve the k nearest neighbors, so its output size is k.) This is a generalization to order k of the point location structure for closest-site Voronoi diagrams proposed by Guibas, Knuth, and Sharir [GKS]. Again, we can use our memory rearrangement scheme to reduce the complexity of our data structure to $\mathcal{O}(k(n-k))$. However, to really solve nearest k neighbors-queries, we have to store the subset corresponding to each region of the diagram. If we do this naïvely, the space requirement is $\mathcal{O}(k^2(n-k))$. This can again be reduced to optimal $\mathcal{O}(k(n-k))$ by an idea that we are going to discuss below when having dealt with deletions.

In fact, we wish to give a dynamic algorithm for the maintenance of order-k Voronoi diagrams under (randomized) sequences of insertions and deletions. We employ the model of [Sc]: We are given a base set $\mathcal{U}$ of sites in the plane, and an (infinite) string δ over the alphabet $\{+, -\}$. This string is going to denote the sequence of insertions and deletions: $+$ denoting insertion and $-$ denoting deletion. In this model, the user is free to choose $\mathcal{U}$ and δ. Then, the algorithm will do the following: It starts with a set $S_0 = \emptyset$ and goes through stages $1, 2, 3, \dots$. In stage i, the i^{th} element of δ is considered. If this is a $+$-sign, then S_i is obtained from S_{i-1} by adding a random site in $\mathcal{U} \setminus S_{i-1}$. If the i^{th} element is a $-$, then S_i is obtained by removing a random site in S_{i-1}. The goal is to maintain $k\text{-}Vor(S_i)$ during the course of this algorithm.

By the techniques of [Sc] it is easily shown that in our case, the expected structural change in stage i is $\mathcal{O}(k^2)$. Furthermore, that paper gives two techniques that can be used to ensure that the point location step in our case takes expected time $\mathcal{O}(k\log^2 n)$ even in the presence of deletions. Since we can do updates in time linear in the size of the structural change as soon as we can locate a conflict for newly inserted sites, the expected running time of our algorithm is $\mathcal{O}(k^2 + k\log^2 n)$ per update. Our method of reducing the memory requirement can still be used, giving optimal $\mathcal{O}(k(n-k))$ space where n is the maximum number of sites ever present in S_i at the same time, while the (amortized) time per stage remains the same.

Finally, if we wish to use our structure to do k-nearest neighbor queries, we have to identify the subset associated with a region found by the point location step. If the k-subsets of the regions defined by the n_i sites in the current set were stored explicitly, this would take space $\mathcal{O}(k^2 n_i)$. For small values of k we may even contend ourselves with that bound. The space drops to $\mathcal{O}(k(n_i - k))$ when the following simple idea is used.

During an insertion or deletion step, we augment the newly constructed regions with some 'critical' sites in their k-subsets. When inserting a site p, each such region R is augmented with p. Hence, when proceeding from some region R' to the region R during later point locations, p is the unique site in $near(R) \setminus near(R')$. When deleting a site, each new region R is augmented with all sites which appear in the labels of the edges generating R (see Lemma 16). In addition, each site p appearing in the label of such an edge is associated with the index of the old region R' lying not on p's side of that edge. Hence, when the point location proceeds from R' to R, we can identify p, which again is the unique site in $near(R) \setminus near(R')$.

We thus can collect a set of candidates for the k nearest neighbors of a query point q during the process of locating q, starting at the oldest diagram, $k\text{-}Vor^*$, for whose regions R we store $near(R)$ in order to initialize the candidate set. The final size of this set is $k + t - 1$ if t point locations have been performed. It remains to single out the k sites closest to q (which clearly are in this set) by sorting. Note that the total time for reporting the k nearest neighbors is dominated by the time for locating q.

Again, the space needed for storing all sets $near(R)$ for $k\text{-}Vor^*$ may be prohibitive as n, the current number of sites in $k\text{-}Vor^*$, typically is much larger than k after each rearrangement step. We would like to use only $O(k(n-k))$ space and to have access to each $near(R)$ quickly. This can be achieved by using persistent data structures [DSST] or, in order to retain simplicity, by doing the following after having computed $k\text{-}Vor^*$:

We partition $k\text{-}Vor^*$ into $O(n-k)$ connected domains of $O(k)$ regions each. For every domain, we choose one region R and store $near(R)$ with R. If the query point q falls into some domain, we simply use graph search in this domain and exploit the edge labels to determine from $near(R)$, in $O(k \log k)$ time, the k-subset of the region containing q. (A partitioning into connected domains can easily be obtained by taking any path visiting all $O(k(n-k))$ cells and cutting it into pieces of length at most $O(k)$. Then, the domains would not necessarily be disjoint, but this is no problem for this technique.)

What is obtained is a dynamic data structure capable of handling insertions, deletions, and k-nearest neighbor queries in a set of sites in the plane. This structure is eminently practical to implement and will exhibit a satisfactory expected performance when combined with the technique of memory rearrangement.

10 Conclusions

It is an obvious question whether the methods presented in this paper can be generalized, to higher dimensions or to distance functions that are more general that the Euclidean. In fact, Lemma 3 and Lemma 4 can be shown to hold in arbitrary dimensions, and even for the more general class of *power diagrams*. One reason why this class is important is that the family of all *k-sets* of a finite point set (i.e., those which can be separated from the set by some hyperplane) can be represented by a power diagram in one dimension lower [A2].

References

[AESW] P.K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. Proc. 6th Ann. ACM Symp. Computational Geometry (1990), 203 - 210.

[AGSS] A. Aggarwal, L.J. Guibas, J. Saxe, P.W. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. Discrete Comput. Geometry 4 (1989), 591 - 604.

[A1] F. Aurenhammer. A new duality result concerning Voronoi diagrams. Discrete Comput. Geometry 5 (1990), 243 - 254.

[A2] F. Aurenhammer. Power diagrams: properties, algorithms, and applications. SIAM J. Computing 16 (1987), 78 - 96.

[AS] F. Aurenhammer, O. Schwarzkopf. A Simple On-line Randomized Incremental Algorithm for Computing Higher Order Voronoi Diagrams. To appear as a Technical Report, Freie Universität Berlin, 1991.

[BDSTY] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. Technical report, INRIA, 1990.

[BDT] J.-D. Boissonnat, O. Devillers, M. Teillaud. A randomized incremental algorithm for constructing higher order Voronoi diagrams. Algorithmica, to be published.

[CEGSS] B. Chazelle, H. Edelsbrunner, L.J. Guibas, M. Sharir, J. Snoeyink. Computing a face in an arrangement of line segments. 2nd Ann. ACM-SIAM Symp. SODA, 1991, to be presented.

[C] K.L. Clarkson. New applications of random sampling in computational geometry. Discrete Comput. Geometry 2 (1987), 195 - 222.

[CS] K.L. Clarkson, P.W. Shor. Applications of random sampling in computational geometry, II. Discrete Comput. Geometry 4 (1989), 387 - 421.

[DSST] J.R. Driscoll, N. Sarnak, D.D. Sleator, R.E. Tarjan. Making data structures persistent. J. Comput. System Sci. 38 (1989), 86 - 124.

[E] H. Edelsbrunner. Algorithms in Combinatorial Geometry. Springer, Berlin - Heidelberg, 1987.

[GKS] L.J. Guibas, D.E. Knuth, M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. Springer LNCS 443 (1990), 414 - 431.

[L] D.T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. IEEE Trans. Computers C-31 (1982), 478 - 487.

[MMO] K. Mehlhorn, S. Meiser, C. Ó'Dúnlaing. On the construction of abstract Voronoi diagrams. Discrete Comput. Geometry, to be published.

[M] K. Mulmuley. On levels in arrangements and Voronoi diagrams. Discrete Comput. Geometry, to be published.

[Sc] O. Schwarzkopf. Randomized Incremental Algorithms in a Dynamic Setting. To appear as a Technical Report, Freie Universität Berlin, 1991.

[S1] R. Seidel. Linear programming and convex hulls made easy. Proc. 6th Ann. ACM Symp. Computational Geometry (1990), 211–215.

[S2] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Technical Report B 90–07, Freie Universität Berlin, October 1990.

[W] E. Welzl. Constructing smallest enclosing disks (balls and ellipsoids). To appear as a Technical Report, Freie Universität Berlin, 1991.