Cloning Voronoi Diagrams via Retroactive Data Structures

Matthew T. Dickerson[1], David Eppstein[2], and Michael T. Goodrich[2]

[1] Dept. of Math and Computer Sci., Middlebury College, Middlebury, Vermont, USA
[2] Computer Science Department, University of California, Irvine, USA

Abstract. We address the problem of replicating a Voronoi diagram $V(S)$ of a planar point set S by making proximity queries:
1. the exact location of the nearest site(s) in S
2. the distance to and label(s) of the nearest site(s) in S
3. a unique label for every nearest site in S.

In addition to showing the limits of nearest-neighbor database security, our methods also provide one of the first natural algorithmic applications of retroactive data structures.

1 Introduction

In the *algorithmic data-cloning framework* [13], a *data querier*, Bob, is allowed certain types of queries to a data set S that belongs to a *data owner*, Alice. Once Alice has determined the kinds of queries that she will allow, she must correctly answer every valid query from Bob. The information security question, then, is to determine how many queries and how much processing time is needed for Bob to clone the entire data set. We define a *full cloning* of S to mean that Bob can answer any validly-formed query as accurately as Alice could. In an ϵ*-approximate cloning* of S, Bob can answer any validly-formed query to within an accuracy of $\epsilon > 0$.

In this paper, we are interested in data sets consisting of a set S of n points in the plane, where n and the contents of S are initially unknown. We study the risks to S when Alice supports planar nearest-neighbor queries on S. We assume that all the sites in S are inside a known bounding box, B, which, without loss of generality, can be assumed to be a square with sides normalized to have length 1. Since planar nearest-neighbor queries define a Voronoi diagram in the plane (e.g., see [7]), we can view Bob's goal in this instance of the algorithmic data-cloning framework as that of trying to determine the Voronoi diagram of S inside the bounding box B. We consider three types of responses (in decreasing order of information content):

1. the exact location of the nearest site(s) to p in S
2. the distance and label(s) of the nearest site(s) to p in S
3. a unique label identifying each nearest site to p in S.

With all three cases, we want to know how difficult it is to compute the Voronoi diagram, or an approximation of it, from a set of queries.

M. de Berg and U. Meyer (Eds.): ESA 2010, Part I, LNCS 6346, pp. 362–373, 2010.

Related work. Motivated by the problem of having a robot discover the shape of an object by touching it [6], there is considerable amount of related work in the computational geometry literature on discovering polygonal and polyhedral shapes from probing (e.g., see [1,2,4,9,10,14,19,18]). We refer the interested reader to the survey and book chapter by Skiena [17,20], and simply mention that, with the notable exception of work by Dobkin *et al.* [9], this prior work is primarily directed at discovering obstacles in a two-dimensional environment using various kinds of contact probes. Translated into this context, their method results in a scheme that would use $7n - 5$ queries to clone a Voronoi diagram, with a time overhead that is $\Theta(n^2)$.

In the framework of *retroactive data structures* [3,8,12], each update operation o to a data structure D, such as an insertion or deletion of an element, comes with a unique numerical value, t_o, specifying a *time value* at which the operation o is assumed to take place. The order in which operations are presented to the data structure is not assumed to be the same as the order of these time values. Just like update operations, query operations also come with time values; a query with time value t should return a correct response with respect to a data structure on which all operations with $t_o < t$ have been performed. Thus, an update operation, o, having a time value, t_o, will affect any subsequent queries having time values greater than or equal to t_o. In a *partially retroactive* data structure the time for a query must be at least as large as the maximum t_o seen so far, whereas in a *fully retroactive* data structures there is no restriction on the time values for queries. Demaine *et al.* [8] show how a general comparison-based ordered dictionary (with successor and predecessor queries) of n elements (which may not belong to a total order, but which can always be compared when they are in D for the same time value) can be made fully retroactive in $O(n \log n)$ space and $O(\log^2 n)$ query time and amortized $O(\log^2 n)$ update time in the pointer machine model. Blelloch [3] and Giora and Kaplan [12] improve these bounds, for numerical (totally ordered) items, showing how to achieve a fully retroactive ordered dictionary in $O(n)$ space and $O(\log n)$ query and update times in the RAM model. These latter results do not apply to the general comparison-based partially-ordered setting, however.

Our Results. Given a set S of n points in the plane, with an API that supports nearest-neighbor queries, we show how queries of Type 1 and Type 2 allow an exact cloning of the Voronoi diagram, $V(S)$, of S with $O(n)$ queries and $O(n \log^2 n)$ processing time. Our algorithms are based on non-trivial modifications of the sweep-line algorithm of Fortune [11] (see also [7]) so that it can construct a Voronoi diagram correctly in $O(n \log^2 n)$ time while tolerating unbounded amounts of backtracking. We efficiently accommodate this unpredictable backtracking through the use of a fully retroactive data structure for general comparison-based dictionaries. In particular, our method is based on our showing that the dynamic point location method of Cheng and Janardan [5] can be adapted into a method for achieving a general comparison-based fully retroactive ordered dictionary with $O(n)$ space, $O(\log n)$ amortized update times, and $O(\log^2 n)$ query times. We also provide lower bounds that show that, even with an adaptation of the Dobkin *et al.* [9] approach optimized for nearest-neighbor searches, there is a sequence of query responses that requires $\Omega(n^2)$ overhead for their approach applied to these types of *exact* queries. Nevertheless, we show that it is possible to clone $V(S)$ using only $3n$ queries. We prove that queries of Type 3 can never exactly clone $V(S)$, however, nor even determine with certainty the value of $n = |S|$. Nevertheless, we show that with $n \log(\frac{1}{\epsilon})$ queries we can construct an ϵ-approximate cloning of $V(S)$ that will support approximate nearest neighbor queries guaranteeing a response that is a site within (additive) $\epsilon > 0$ distance of the exact nearest neighbor of the query point.

2 A Fully-Retroactive Ordered Dictionary

In this section, we develop a fully retroactive ordered dictionary data structure using $O(n)$ space, $O(\log n)$ amortized update time, and $O(\log^2 n)$ query time, based on a dynamic point location method of Cheng and Janardan [5]. The main idea is to construct an interval tree, B, over the intervals between the insertion and deletion times of each item in the dictionary, and to maintain B as a BB[α]-tree [16].

Each item x is stored at the unique node v in B such that x's insertion time is associated with v's left subtree and x's deletion time is associated with v's right subtree. We store x in two priority search trees [15], $L(v)$ and $R(v)$, associated with node v. These two priority search trees are both ordered by the dictionary ordering of the items stored in them; all such items are active at the time value that separates v's left and right subtrees, so they are all comparable to each other. The priority search trees differ, however, in how they prioritize their items. As with priority search trees more generally, each node in $L(v)$ and $R(v)$ stores two items, one that is used as a search key and another that has the minimum or maximum priority within its subtree. In $L(v)$, the insertion time of an item is used as a priority, and a node in $L(v)$ stores the item that has the minimum insertion time among all items within the subtree of descendants of that node. In $R(v)$, the deletion time of an item is used as a priority, and a node in $R(v)$ stores the item that has the maximum deletion time within its subtree.

An insertion of an item x in D is done by finding the appropriate node v of the interval tree and inserting x into $L(v)$ and $R(v)$, and a deletion is likewise done by deletions in $L(v)$ and $R(v)$. Updates that cause a major imbalance in the interval tree structure are processed by rebalancing, which implies, by the properties of BB[α]-trees [16], that updates run in $O(\log n)$ amortized time.

Queries are done by searching the interval tree for the nodes with the property that the retroactive time specified as part of the query could be contained within one of the time intervals associated with that node. For each matching interval tree node v, we perform a search in either $L(v)$ or $R(v)$ depending on the relation between the query time and the time that separates the left and right children of v. The search method of Cheng and Janardan [5] allows us to find the successor of the query value, among the nodes stored in $L(v)$ or $R(v)$ with time intervals that contain the query time, in time $O(\log n)$. The result of the overall query is then formulated by comparing the results found at each interval tree node and choosing the one that is closest to the query value. Thus, the query takes $O(\log n)$ time to identify the interval tree nodes associated with the query time, $O(\log^2 n)$ to query each of logarithmically many priority search trees, and $O(\log n)$ time to combine the results, for a total of $O(\log^2 n)$ time.

Theorem 1. *One can maintain a fully-retroactive general comparison-based dictionary on n elements, using $O(n)$ space, so that updates run in $O(\log n)$ amortized time and predecessor and successor queries run in $O(\log^2 n)$ time.*

3 Exact Query Probes

We begin our study of Voronoi diagram cloning with the strongest sort of queries—Type 1. Given a query point p, a Type-1 query returns the site q in S nearest to p, that is, it returns the geometric location of q, p's nearest-neighbor in S. In the event that p has more than one nearest neighbors in S, all nearest neighbors are returned. We show that only $O(n)$ queries and $O(n \log^2 n)$ processing time is needed to completely clone $V(S)$—which, as implied, also means we explicitly have determined both S and n.

Overview of Our Algorithm. Our algorithm is adapted from the plane sweep Voronoi diagram algorithm of Fortune [11], with a significant modification to allow for unbounded and unpredictable amounts of backtracking. The fundamental difference is that the Fortune algorithm begins with the set of sites, S, completely known; in our case, the only thing we know at the start is a bounding box containing S. Using the formulation of de Berg *et al.* [7], Fortune's algorithm uses an event queue to controls a sweep line that moves in order of decreasing y coordinates, with a so-called "beach line"—an x-monotone curve made up of parabolic segments following above the sweep line. The plane above the beach line is partitioned into cells according to the final Voronoi diagram of S. There are two types of events, caused when the sweep line crosses point sites in S and Voronoi vertices in $V(S)$; the latter points are determined as the algorithm progresses. In our version, we need to find both the sites and the Voronoi vertices as the plane sweep advances. And because not all sites are known in advance, we will need to verify *tentative* Voronoi vertex events as we sweep across them, at times backtracking our sweep line when our queries reveal new sites that invalidate tentative Voronoi vertices and introduce new events that are actually above our sweep line. We will show that each query discovers a feature in the Voronoi diagram, that the number of times we backtrack is bounded by the number of these features, and these facts imply that the number of queries and updates we perform in a retroactive dictionary used to implement our sweep-line algorithm is $O(n)$. In fact, we will prove that the number of probes is at most $4n$.

We begin with an overview of our algorithm. The algorithm begins by finding all the Voronoi regions and edges that intersect the top edge of the bounding box, B. If there are k such regions (and thus $k-1$ edges), this can be accomplished in $O(n)$ time with $2k-1$ queries. This step initializes our event queue with k of the point sites in S.

The algorithm then proceeds much as the Fortune algorithm, but with the following two important changes. Whenever we reach a point site event for some site $q \in S$ (i.e., when q is removed from the event queue), we do a nearest-neighbor query on the point of the beach line directly above q—that is, the point with the same x-coordinate as q and a y coordinate on the beach line that exists for the time value when the sweep line hits q. The position of this query point can be determined by using a retroactive dictionary queried with respect to the components of the beach line for the time value (in the plane sweep) associated with the point q. (See Fig. 1.) Querying this point will either confirm a Voronoi edge known to be part of the final Voronoi diagram (in which case we proceed with the sweep) or it will discover a new site r in S (in which case the sweep line restores point q to the event queue and backtracks to r).

The second type of event is a tentative Voronoi vertex event. We do a nearest-neighbor query at the point believed to be a Voronoi vertex, which either confirms the vertex and all of its adjacent Voronoi edges above it, or it discovers a new site in S. This new site must be above the sweep line: at the time that Fortune's algorithm processes a Voronoi vertex event its sweep line must be as far from the Voronoi vertex as the three sites generating the vertex, so undiscovered sites below the sweep line cannot be nearest neighbors to the tentative vertex. If the algorithm discovers a new site above the sweep line, then again we backtrack and process that new site. In either case the Voronoi vertex is removed from the event queue—either added to the Voronoi diagram being constructed as a validated vertex, or ignored as a false vertex.

Correctness and Complexity. Both the correctness and the analysis of this algorithm make use of the following important observation. Though the algorithm backtracks at certain "false" events—or tentative events that are proven false—it never completely removes any Voronoi components that have been confirmed by probes. Voronoi edges can only be added in two ways: the addition of a new site that creates one new edge, or the addition of a Voronoi vertex that terminates two edges and creates one new edge. In both cases, the edge is verified as an actual edge using a query before it is added to the Voronoi diagram being constructed, thus the diagram never contains edges that could later be falsified. (See Fig. 2.)

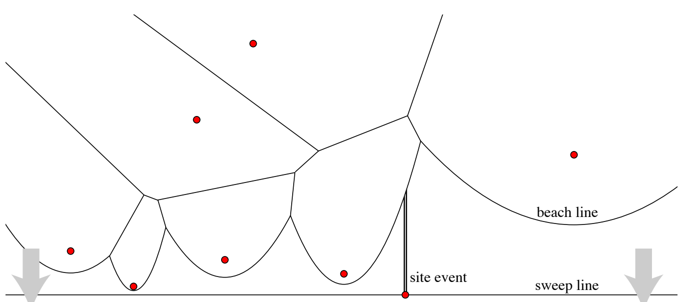

Fig. 1. Illustrating the sweep-line algorithm for constructing a Voronoi diagram

The insertion of a new site begins a new edge directly above it, where the parabola of the site being added to the tree T—a degenerate line-segment parabola at the instant it is added—intersects the existing parabola above it, thus replacing one leaf in the tree with three. But before this site is inserted with its edge, the edge is tested with a query into the existing Voronoi diagram. The other time an edge may be added is at a Voronoi vertex where two existing edges meet and a third new one is created. But all tentative Voronoi vertices are also verified by queries before they become circle events.

There are a few key observations that will lead to the analysis of the algorithm's run time and total number of queries. First, the sweep line will only backtrack when a new site in S is discovered, and so there are at most n backtracks. Second, every time we have a tentative Voronoi vertex that turns out to be unverified—that is, an event that turns out not to be part of the final diagram—we have also discovered a new site in S, and thus we have at most $O(n)$ phony events that are processed. It follows that the run time of the algorithm is asymptotically equivalent to the original Fortune algorithm, modulo the time needed for our retroactive data structure queries. Furthermore, after our initialization stage, every nearest-neighbor query either finds a site, verifies a site by looking at the Voronoi edge above it, or verifies a Voronoi vertex. The initialization, as noted, requires $2k-1$ queries if there are k sites initially discovered. Queries discover $n-k$ more sites, verify n sites, and find at most $2n-2-k$ Voronoi vertices, for a total of $4n-3$ which is less than $4n$ queries.

The algorithm requires the same run time as the original Fortune plane sweep except for the processing of the tentative Voronoi vertices that prove to be phony, and all the backtracking (which is implemented using our retroactive dictionary). As noted, there are $O(n)$ of these backtracking steps, since these can only occur once for each previously undetected site in S. So the overall number of updates and queries in our sweep-line-with-backtracking algorithm is $O(n)$; hence, the running time of our algorithm is $O(n \log^2 n)$. Thus, we have the following.

Theorem 2. *Given a set S of n points in $\mathbf{R}^2$, we can construct a copy of $V(S)$ using at most $4n$ Type-1 queries and $O(n \log^2 n)$ time.*

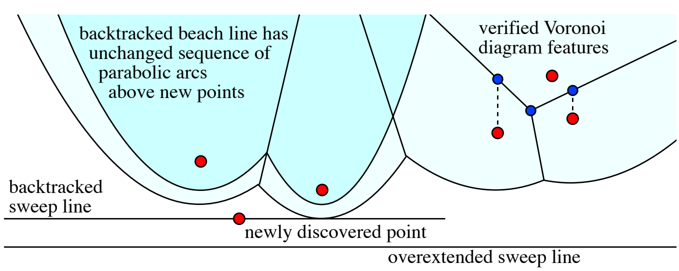

Fig. 2. Backtracking the sweep-line

An Alternate Algorithm Using More Time and Fewer Queries. This second algorithm follows an incremental construction paradigm, based on the general approach of Dobkin *et al.* [9] for discovering a 3-dimensional convex polyhedron using finger probes. The alternative algorithm begins by querying each of the four corners of the bounding box. There are three cases to consider: these probes may discover one, two, or more than two sites in S. If we discover more than two sites, then we construct the Voronoi diagram of all 3 or 4 of the sites discovered by these four queries, but we mark each Voronoi vertex as tentative and put it into a queue. The algorithm then proceeds as follows until the queue is empty. Remove a tentative Voronoi vertex from the queue, and query it. If the query reveals that it is a Voronoi vertex—that is, it has the three expected nearest neighbors—then we confirm the vertex and continue. If it is not a Voronoi vertex, then it must be closer to some previously undiscovered site in S, that will be returned by the query. We add that site to our list of known sites and update the Voronoi diagram in worst case $O(n)$ time using incremental insertion. When the queue is empty, we have a complete Voronoi diagram. Every probe except possibly one of the four corner probes discovered either a new site in S or confirmed a Voronoi vertex, and so the total number of queries is at most $n + (2n - 5) + 1 < 3n$. If the four corner queries discovered only two sites, then we compute the Voronoi edge that would be shared by these two sites if they were the only two sites in S, and we query both intersections of this edge with the bounding box. If we confirm both edges, then $n = 2$ and we are done. We have used $3n = 6$ queries. If at least one of these two additional queries discovers another site, then we have at least three known sites and we proceed as with the previous case. Every query except at most three of the initial queries either confirmed a Voronoi vertex or discovered a new point site, and so the total number of queries is at most $n + (2n - 5) + 3 < 3n$. All four corners will belong to the same Voronoi region if and only if there is only one site in the bounding box, in which case 4 queries was sufficient.

Theorem 3. *Given a set S of n points in $\mathbf{R}^2$, we can construct a copy of $V(S)$ using at most $3n + 1$ Type-1 queries and $\Theta(n^2)$ time.*

Proof. We have already established the quadratic upper bound. For the sake of a lower bound, imagine that we have a set S' of $n/2$ points on the bottom boundary of B, all within distance δ of the point $(0, 0)$, for a small parameter δ with $0 < \delta \leq 1/2^n$. These points, by themselves, construct a Voronoi diagram with parallel edges. Suppose further that there is a single point, $p_0 = (\delta, 1 - \delta)$, near the top boundary of B. The Voronoi region for p_0 intersects the Voronoi region of every point in S'. The above algorithm therefore, after discovering the boundary points in S', would next query a vertex on the Voronoi diagram $V(S')$ of $\mathcal{B}$, which will discover p_0. Next it will probe at vertex that is equi-distant to p_0 and a point in S'. Suppose that this probe discovers a point $p_1 = (\delta, 1/2)$. This point is closer to every point in S' than p_0; hence, updating the Voronoi diagram to go from $V(S' \cup \{p_0\})$ to $V(S' \cup \{p_0, p_1\})$ takes $\Omega(n)$ time. Now, suppose querying a tentative vertex of $V(S' \cup \{p_0, p_1\})$, which will be equi-distant from p_1 and a point in S', discovers a point $p_2 = (\delta, 1/2^2)$. Again, updating the

Voronoi diagram takes $\Omega(n)$ time. Suppose, therefore, that we continue in this way, with each newly-discovered point $p_i = (\delta, 1/2^i)$ requiring that we spend $\Omega(n)$ time to update the current Voronoi diagram. After discovering $p_{n/2-1}$, the n-th point in the set $S = S' \cup \{p_0, p_1, \ldots, p_{n/2-1}\}$, we will have spent $\Omega(n^2)$ time in total to discover the Voronoi diagram $V(S)$. □

4 Distance Query Probes

We next consider our Voronoi diagram cloning algorithm for the case when we use use only distance query probes—probes that return the distance to and label of the nearest site(s). We begin by describing how we can find those sites whose Voronoi regions (and thus also edges) intersect the top boundary of the bounding box where the sweep-line begins. We will speak of a *probe circle* as the set of possible locations of a site returned by a probe p: it is the circle of radius d centered at p, where d is the distance returned to the nearest site.

Initializing the Sweep Line. We begin the initialization process by probing at the two top corners of the bounding box. If both probes return the same site p, then by convexity of Voronoi regions the entire top edge of the box belongs to the Voronoi region $V(p)$. Furthermore, both probes also return a distance d to the site p, and so p must fall on an intersection of two circles of radius d centered at the two corner probe locations. Since one of these intersections is above the bounding box, the remaining intersection gives the exact location of p.

Assume that the two corner probes p_l and p_r on the left and right respectively return different sites q_l and q_R respectively. We know the distance p_l to q_l and p_R to q_R, but we don't know the exact locations of the two sites, nor do we know if there are any other sites with regions intersecting the bounding box. The segment between these two probes is therefore not fully classified. We will describe a recursive procedure for classification.

Let L be an unclassified segment, with the probes p_l and p_r on the left and right sides of the segments respectively returning different sites q_l and q_r. First, we probe the midpoint p_m of L. The probe returns either one of the two known sites, or a new site q_m. If it returns a new site, then we divide L into two segments that are both unclassified, but which have classified endpoints q_l, q_m, and q_r, and we recursively classify them.

Suppose query p_m returns one of the already known sites. If this probe returns q_r, then we can immediately compute the exact location of q_r from the two probes p_r and p_m, since only one of the intersections of the probe circles is inside the bounding box. We also have classified the segment L_r between probes p_r and p_m as being fully inside the Voronoi region of p_r. The other half of the segment, L_l, however, is not classified; we only know the Voronoi regions of its endpoints are the regions of q_r and q_l. Here it would be tempting to again probe the midpoint of the segment L_l; however that could lead to an unbounded number of probes as we repeatedly divide the segment in half because the midpoint of the remaining unclassified segment L_l could still belong to q_r and so we would gain no new information about q_l. What we do instead is use the known location of q_r, which must be outside the probe circle at p_l, to find a probe location p_{l2} close enough to p_l that it is guaranteed not to gives us q_r. This is possible since the perpendicular bisector between q_r (which is a known point) and any point on the probe circle from p_l—which is the set of candidate locations for q_l—must fall between p_m and p_l on a finite segment computable in $O(1)$ time, and thus anything between that range and p_l is closer to q_l than to q_r.

So this new probe p_{l2} returns either q_l or a new site. If it returns a new site, then we divide the unclassified segment into two unclassified segments and recursively classify them. If this new probe gives us q_l then we now know the exact location of q_l from two probes. From the exact locations of q_r and q_l, we can compute and probe where their Voronoi edge ought to cross the bounding box, either confirming that Voronoi edge—which means that the entire edge is now classified—or we discover a new site. If the probe gives us a new site, then again we divide the unclassified segment into two unclassified segments and recursively classify them.

Lemma 1. *The initialization stage for the sweep line requires $O(k)$ time and $3k-1$ probes where k is the number of sites whose Voronoi regions intersect the top of the bounding box.*

Proof. Note that every probe either identifies a previously undiscovered site (k probes), provides a second probe with more information on an already discovered site enabling the exact location of this site to be computed (k probes), or confirms a Voronoi edge ($k-1$ probes for k regions). So the total number of probes in this section is $3k-1$ where k is the number of sites whose Voronoi regions intersect the top of the bounding box. Each probe is processed in $O(1)$ time. □

Processing the Sweep Line. The previous subsection explains how to initialize the sweep line. The algorithm making use of distance-only queries now proceeds as with the exact query probe version of the previous section, except that a slightly different approach requiring more probes will be needed to process tentative site events.

As with the algorithm of the previous section, there are two types of tentative events: a tentative Voronoi vertex for three known sites, and a tentative Voronoi edge that falls directly above a known site and is determined by one other known site. Both of these events need to be verified by probe–that is, we need to determine if these events are actually real, or whether there is some other site closer to the events. In both cases, we use a probe p where the tentative Voronoi feature *should* be. If that problem returns the correct three or two site labels (at the correct distance), then the verification is complete, and we proceed as with the algorithm of the previous section.

However, these probes may discover a new site q; in this case, they give only the distance to that site and not its actually location. We need two more probes that return the same site in order to discover its exact location—but these probes may instead return yet other new sites. We now describe how to choose the locations of these probes so that no work is wasted, and each probe either verifies a Voronoi vertex, verifies a Voronoi edge above a known site, or is one of three probes that exactly locates a site.

Let p_1 be a probe during the sweep line, that attempts to verify a Voronoi vertex or edge, and instead discovers a new site q_1 that was not previously known. Let $d = d(p_1, q_1)$ be the distance returned from probe p_1 to its site q_1, and let e be the distance from p_1 to the nearest previously known sites—that is, the two or three sites whose tentative Voronoi vertex or edge it was seeking to verify. Since probe p_1 returned q_1, we know that $d < e$. Let p_2 be any probe location such that $d(p_1, p_2) < \frac{e-d}{2}$. By the triangle inequality, we know $d(p_2, q_1) < d + \frac{e-d}{2} = \frac{e+d}{2}$, while $d(p_2, r) > e - \frac{e-d}{2} = \frac{e+d}{2}$, where r is any of the two or three previously known closest sites to p_1. It follows immediately that probe p_2 cannot return any previously known site except q_1 which was first discovered by probe p_1. We can choose any probe location meeting this restriction, $d(p_1, p_2) < \frac{e-d}{2}$, which is computable in $O(1)$ time.

So there are two possibilities with probe p_2: either it returns site q_1 again, or it returns a new site q_2. If p_2 returns q_1, we now have two probes returning that site, and distances to that site, so its location is one of at most two intersections between the two probe circles. We can now probe either one of those two intersections, and from the result we determine the exact location of q_1 because the probe either returns q_1 at distance 0, or it returns some other site, or it returns q_1 at a distance > 0.

If p_2 returns a new site q_2, then we now have two sites that have been discovered, but whose exact locations are not known. We can discover the exact location using the recursive method of the previous subsection, treating the segment p_1p_2 as an unclassified segment, probing its midpoint, and continuing. However, once we have received a site as the result of two probes, we still require a third probe to exactly locate it since both intersections of the first two probe circles might be inside the bounding box.

Lemma 2. *Processing the remaining events (after the initialization) for the sweep line requires at most $6n - 3k - 5$ probes where k is the number of sites whose Voronoi regions intersect the top of the bounding box.*

Proof. There are $n - k$ sites to be discovered, n sites that need to have an edge verified above them, and at most $2n - 5$ Voronoi vertices in the Voronoi diagram of n sites. Every probe accomplishes one of five things: it verifies a Voronoi vertex ($2n-5$ probes), verifies a Voronoi edge directly above a site (n probes), or is one of exactly three probes used to discover and then exactly locate a new site ($3(n-k)$ probes.) The total number of probes required is therefore at most $6n-3k-5$. □

Theorem 4. *Given a set S of n points in $\mathbf{R}^2$, we can construct a copy of $V(S)$ using at most $6n - 6$ Type-2 queries and $O(n \log^2 n)$ time.*

5 Label-Only Query Probes

Using queries of the third type, it is impossible to *exactly* clone $V(S)$ or even to determine with certainty the value of $n = |S|$. However even with this minimal query information, we construct an approximate Voronoi diagram $V(S')$ which, without *explicitly* storing the locations of the sites in S, will still support later arbitrary approximate proximity queries to $V(S)$. We now show that an approximate Voronoi diagram can be constructed to answer nearest-neighbor queries, with a probing process that uses $O(N\log(1/\epsilon))$ queries and $O(N(\log N+\log(1/\epsilon)))$ time, where $N\le n$ is the number of discovered sites in S. (Any two sites separated by at least ϵ will be distinguished and discovered.) The main idea of the algorithm is to build an approximation to the Voronoi cell of each known site, using $O(\log(1/\epsilon))$ queries per feature of the cell. This sequence of queries either finds a sufficiently accurate approximation for the location of that feature or discovers the existence of another site label. We begin by querying each corner of our bounding box to find the label of the site in whose region that corner belongs. For any side of the box whose corners are in different Voronoi regions, we do a binary search to find, within a distance of ϵ^2, the edge of the Voronoi region for each different site. This may discover new Voronoi regions. For each new region discovered, we also do a binary search to discover its edges to within ϵ^2. Each binary search requires $O(\log\frac{1}{\epsilon})$ queries and time. The result is an ordered list of Voronoi edges crossing each side of the bounding box.

A second similar search a distance of 2ϵ from each side of the bounding box will find the same Voronoi edges—or will discover a new Voronoi region, indicating that the Voronoi edge has ended. For those Voronoi edges that have not ended within 2ϵ, we compute an approximation of the line containing the Voronoi edge—that is, the perpendicular bisector of the two sites whose labels we know. An argument using similar triangles shows that our approximation of this edge is accurate enough that we can determine to within a distance of $<\epsilon$ where this edge crosses the far boundary. We do a doubling search of queries out along each discovered Voronoi edge, and then a binary search back once we have moved past the end of the edge, to find where it ends. Thus, three binary searches of $O(\log(\frac{1}{\epsilon}))$ queries and time each suffice to discover complete approximations of each Voronoi edge intersecting the bounding box, including an approximate location of the Voronoi vertex terminating these edges. In the worst case, our approximation is within ϵ. A constant number of queries in the vicinity of each Voronoi vertex will discover the other edge or edges coming out of the Vertex. We repeat this process for each new Voronoi edge as it is discovered, until every Voronoi edge has both ends terminated at Voronoi vertices, at which time the approximate Voronoi diagram is complete.

Theorem 5. *Given a set S of n points in $\mathbf{R}^2$, we can construct a planar subdivision, V', using $O(N\log\frac{1}{\epsilon})$ Type-3 queries and $O(N(\log N+\log\frac{1}{\epsilon}))$ time, where $N<n$ is the number of discovered sites in S, such that any two sites separated by at least ϵ will be distinguished and discovered and each point on the 1-dimensional skeleton of V is within distance ϵ of a point on the 1-dimensional skeleton of the Voronoi diagram, $V(S)$, of S.*

References

1. Alevizos, P.D., Boissonnat, J.-D., Yvinec, M.: Non-convex contour reconstruction. J. Symbolic Comput. 10, 225–252 (1990)
2. Aoki, Y., Imai, H., Imai, K., Rappaport, D.: Probing a set of hyperplanes by lines and related problems. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1993. LNCS, vol. 709, pp. 72–82. Springer, Heidelberg (1993)
3. Blelloch, G.E.: Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In: SODA 2008: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 894–903. Society for Industrial and Applied Mathematics, Philadelphia (2008)
4. Boissonnat, J.-D., Yvinec, M.: Probing a scene of non-convex polyhedra. Algorithmica 8, 321–342 (1992)
5. Cheng, S.W., Janardan, R.: New results on dynamic planar point location. SIAM J. Comput. 21, 972–999 (1992)
6. Cole, R., Yap, C.K.: Shape from probing. J. Algorithms 8(1), 19–38 (1987)
7. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Berlin (1997)
8. Demaine, E.D., Iacono, J., Langerman, S.: Retroactive data structures. ACM Trans. Algorithms 3(2), 13 (2007)
9. Dobkin, D.P., Edelsbrunner, H., Yap, C.K.: Probing convex polytopes. In: Proc. 18th Annu. ACM Sympos. Theory Comput., pp. 424–432 (1986)
10. Edelsbrunner, H., Skiena, S.S.: Probing convex polygons with x-rays. SIAM J. Comput. 17, 870–882 (1988)
11. Fortune, S.J.: A sweepline algorithm for Voronoi diagrams. Algorithmica 2, 153–174 (1987)
12. Giora, Y., Kaplan, H.: Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. ACM Trans. Algorithms 5(3), 1–51 (2009)
13. Goodrich, M.T.: The mastermind attack on genomic data. In: IEEE Symposium on Security and Privacy. IEEE Press, Los Alamitos (2009) (to appear)
14. Joseph, E., Skiena, S.S.: Model-based probing strategies for convex polygons. Comput. Geom. Theory Appl. 2, 209–221 (1992)
15. McCreight, E.M.: Priority search trees. SIAM J. Comput. 14(2), 257–276 (1985)
16. Mehlhorn, K.: Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry. EATCS Monographs on Theoretical Computer Science, vol. 3. Springer, Heidelberg (1984)
17. Skiena, S.S.: Problems in geometric probing. Algorithmica 4, 599–605 (1989)
18. Skiena, S.S.: Probing convex polygons with half-planes. J. Algorithms 12, 359–374 (1991)
19. Skiena, S.S.: Interactive reconstruction via geometric probing. Proc. IEEE 80(9), 1364–1383 (1992)
20. Skiena, S.S.: Geometric reconstruction problems. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, ch. 26, pp. 481–490. CRC Press LLC, Boca Raton (1997)