Solving Query-Retrieval Problems by Compacting Voronoi Diagrams

(Extended Abstract)

Alok Aggarwal[1,2] *Mark Hansen*[2] *Tom Leighton*[2]

[1]IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York 10598

[2]Mathematics Department and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

In this paper, we describe a new technique for solving a variety of query-retrieval problems in optimal time with optimal or near-optimal space. In particular, we use the technique to construct algorithms and data structures for circular range searching, half-space range searching, and computing k-nearest neighbors in a variety of metrics. For each problem and each query, the response to the query is provided in $O(k)$ or $O(k + \log n)$ time where k is the size of the response and n is the size of the problem. (E.g., for the n-point k-nearest neighbors problem, the k-nearest neighbors of any query point are provided in $O(k + \log n)$ steps.) Depending on the problem being solved, the space required for the data structure is either linear or $O(n \log n)$. Hence, the time bounds are optimal and the space bounds are optimal or near-optimal. Previously known data structures for these problems required a factor of $\Omega(\log n(\log \log n)^2)$ or $\Omega(\log n \log \log n)$ more space and/or more time to answer each query.

Our compaction technique incorporates planar separators, filtering search, and the probabilistic method for discrepancy problems. The fundamental idea is that k^{th}-order Voronoi diagrams (and other suitable proximity diagrams) can be compacted from $k^{O(1)}n$ space to $O(n)$ space and still retain all the information that is essential for solving query problems. This result is of independent interest and may be useful in improving the memory space requirement or the query-time bound for other query-retrieval problems.

1 Introduction

1.1 Background

Query-retrieval problems have received considerable attention in the computational geometry literature [7, 9, 10, 11, 12, 14, 16, 18, 19]. Typically, a query-retrieval problem consists of a set of n geometric objects (e.g. points in the plane), and an unlimited sequence of queries (e.g. "of the n points in the plane, find the k points that are closest to position (x, y)"). The task is to preprocess the n objects and devise a data structure so that each query can be answered as quickly as possible.

The most important measures of efficiency in a query-retrieval problem are the space required by the data structure and the time needed to answer each query. For example, typical query-retrieval problems require $\Omega(n)$ space and $\Omega(k + \log n)$ time per query where k is the size of the response to the query. The algorithms and data structures described in this paper all achieve the optimal time bound and use $\Theta(n)$ or $\Theta(n \log n)$ space, depending on the problem being solved. Preprocessing time and space (used to construct the data structure) are also of concern, but are usually not as important as data structure space and query response time. All of the algorithms described in this paper use polynomial preprocessing time and space. Using probabilistic methods, the preprocessing time and space can be reduced to $O(n \log^2 n)$.

1.2 Main Results

In this paper, we describe a new technique for solving query-retrieval problems in optimal time with optimal or near-optimal space. The technique incorporates planar separators, filtering search, and the probabilistic method to compact k^{th}-order Voronoi diagrams (and/or other suitable proximity diagrams) from $k^{O(1)}n$ space to $O(n)$ space without losing any of the information that

This research was supported by Air Force contract AFOSR 89-0271, Army contracts DAAL-03-86-K-0171, and DARPA contract N00014-89-J-1988. Mark Hansen is supported by the DARPA Research Program in Parallel Processing. The research of the 3rd author was initiated while visiting A.T. & T. Bell Labs.

is essential for solving query problems. As examples, we use the technique to construct algorithms and data structures for k-nearest neighbor search, circular range search, and half-space range search. A brief description of each of these problems and of our results is provided below.

Planar k-Nearest Neighbor Search

Input: A set P of n points in the Euclidean plane E^2 and an integer $k \geq 0$.
Query: Find the k points in P that are closest to a query point q.

We show how to solve this problem using $O(n)$ space and $O(k + \log n)$ time per query. If k is not fixed, but is given as a part of the query (i.e., the query is a pair (q, k) with $q \in E^2$ and $k \leq n$), then our solution uses $O(n \log n)$ space and $O(k + \log n)$ time per query. The best previously known solution to this problem is due to Chazelle, Cole, Preparata, and Yap [11] who construct a data structure using $\Theta(n(\log n \log\log n)^2)$ space.

We also consider the k-nearest neighbors problem in some weighted metrics. For example, we show how to construct data structures and algorithms with equivalent performance in the power-distance metric and the additive-weight metric. In the *power-distance metric*, a weight w_p is provided with each point $p \in P$, and the distance between p and q is defined to be $d(p,q)^2 - w_p^2$ where $d(p,q)$ is the Euclidean distance between p and q. In the *additive-weight metric*, the distance between p and q is defined to be $d(p,q) + w_p$. Power-distance and additive-weight metric Voronoi diagrams are well known and have several applications [3, 4, 5, 6, 17, 16, 30, 32], but the k-nearest neighbor searching problem in these metrics does not appear to have been studied previously. As an example of the applications of these metrics, we use our technique for computing k-nearest neighbors in the power-distance metric to solve the 3-dimensional half-space range search problem.

In the special case that each query point q belongs to a restricted set Q of $O(n)$ points (e.g., $Q = P$), the query response time of our algorithms can be improved to $O(k)$ without increasing the space. This restricted searching result could be useful in conjunction with heuristics for large traveling salesman problems that repeatedly compute the k nearest neighbors of various cities along the tour [25]. For small problem instances, it is possible to simply precompute the k-nearest neighbors of every node and store them, thereby using $O(kn)$ space. For large n, however, this approach is not possible, and the techniques described in this paper become more appropriate. In this application, our approach uses $O(n)$ space and $O(k)$ query time, both of which are optimal.

Circular Range Search

Input: A set P of n points in the Euclidean plane E^2.
Query: Find all points of P contained in a disk in E^2 with radius r centered at q.

We show how to solve this problem using $O(n \log n)$ space and $O(k + \log n)$ time per query where k is the number of points in the disk. Several algorithms for this problem have been reported in the literature [7, 9, 11, 12]. The best previously known algorithm (due to Chazelle, Cole, Preparata, and Yap [11]) uses $\Theta(n(\log n \log\log n)^2)$ space and $O(k + \log n)$ time per query. We transform the circular range search problem into the planar k-nearest neighbors search problem and apply the filtering search technique of [9] to obtain the result.

If the centers of the query disks are known ahead of time and if the number of centers is $O(n)$ then the query time can be improved to $O(k)$ without increasing the space.

Half-Space Range Search in Three Dimensions

Input: A set P of n points in Euclidean space E^3.
Query: Find all points of P that are contained in the half space defined by $ax + by + cz \leq d$.

We show how to solve this problem with $O(n \log n)$ space and $O(k + \log n)$ query time, where k is the number of points that are contained in the half-space. Previously, Chazelle and Preparata [16] had given a data structure for this problem using $O(n \log^5 n \log\log^4 n)$ space which was later improved to $O(n \log^2 n \log\log n)$ space by Clarkson and Shor [18]. We transform the range search problem into the k-nearest neighbor search problem in the power-distance metric and use filtering search to obtain our solution.

1.3 Outline

The remainder of this paper is divided into three sections. Section 2 presents our compaction technique by constructing an $O(n)$-space data structure for k-nearest neighbor search in the Euclidean plane. Initially a randomized construction is given, and later in Section 2.5 we show how to use Spencer's method [31] to give a deterministic construction. We also indicate how the compaction technique can be applied to nearest neighbor search problems in other metrics. Section 3 briefly illustrates how we can build a linear data structure for k-nearest neighbor search with significantly improved preprocessing time by re-introducing some randomness into the deterministic construction. Finally, Section 4 discusses applications of the compaction technique to circular range search problems in the plane and half-space range search problems in three dimensions. We conclude this section with a suggestion for generalizing

the technique so that it can be applied to other retrieval problems.

2 An Optimal Data Structure for k-Nearest Neighbor Queries in L_2

Let V be the triangulated k^{th}-order Voronoi diagram for a set P of n points in the plane under the standard L_2 (Euclidean) metric. We assume that the points of P are in general position. Then V is a planar graph with $O(kn)$ edges and $O(kn)$ faces [27]. Using Kirkpatrick's results [26] on planar point location, it is well known how to preprocess V to answer k-nearest neighbor queries in $O(\log n + k)$ time. Unfortunately, the resulting data structure occupies $O(kn)$ space. In this section we construct an optimal data structure using only $O(n)$ space which achieves the $O(\log n + k)$ time bound for answering k-nearest neighbor queries.

2.1 Planar Separator Techniques

We begin by taking the dual of V: $\tilde{V}$. Note that $\tilde{V}$ is a planar, degree 3 graph with $O(kn)$ edges and nodes. A node v of $\tilde{V}$ corresponds to a region of the plane. Any query point in this region has the same set S_v of k-nearest neighbors in P. It follows from basic properties of the Voronoi diagram for points in general position that whenever v and w are adjacent nodes in $\tilde{V}$, S_v and S_w differ in at most one point [27]. For each edge $e = (v, w)$ in $\tilde{V}$, label e with the set of points $L_e = S_v \bigcap S_w$. So L_e contains either $k-1$ or k points. Now fix a point $p \in P$ and consider the faces of V which contain p in their list of k-nearest neighbors. These faces correspond to a set of nodes in $\tilde{V}$. Let $(\tilde{V})_p$ consist of these nodes, together with the edges e such that $p \in L_e$. Then we have the following observation:

Observation 1 *For all $p \in P$, $(\tilde{V})_p$ is connected.*

Proof: Consider a node $v \in (\tilde{V})_p$ and the corresponding face F_v in V. Draw a straight line from any spot in F_v to p. Moving along this line, we are always getting closer to p, so each face of V which we pass through contains p as one of its k-nearest neighbors. Furthermore, since they are adjacent, these faces which the line passes through correspond to a connected subset of nodes in $(\tilde{V})_p$. Let w be the node in $(\tilde{V})_p$ whose corresponding face in V contains p. Then we have shown that v is in the same connected component as w for all $v \in (\tilde{V})_p$. □

At this point, we would like to partition $\tilde{V}$ into groups containing $\sim k^6$ nodes. Each connected component of a group will then correspond to a contiguous group of faces in the triangulated Voronoi diagram. Using the Lipton-Tarjan planar separator theorem, we can split $\tilde{V}$ in half by removing $O(\sqrt{kn})$ edges. Continue to apply the separator theorem, at each level splitting the existing groups in half until the point is reached where groups contain $\sim k^6$ nodes. At this point, $O(\frac{n}{k^2})$ edges will have been removed. Let $\tilde{V}_1, \ldots, \tilde{V}_l$ be the resulting $l = O(\frac{n}{k^6})$ pieces of $\tilde{V}$. Then define P_i to be the union of the S_v such that $v \in \tilde{V}_i$.

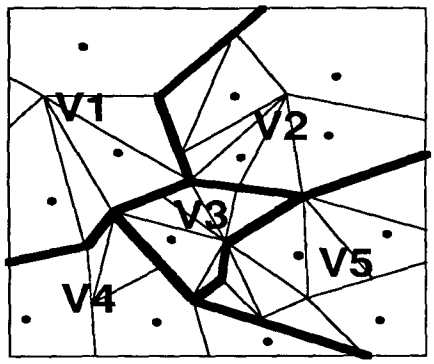

Figure 1: The triangulated Voronoi diagram V with darker edges defining $V_1, \ldots, V_l$

Observation 2 *For any query point q, the k-nearest neighbors of q are completely contained in some P_i.*

Proof: Let F_q be the face in V which contains q and v_q be the corresponding node in the dual $\tilde{V}$. Then $v_q \in \tilde{V}_i$ for some i and the set of k-nearest neighbors to q is $S_{v_q} \subset P_i$. □

For all i, let V_i be the region of the plane defined by $\tilde{V}_i$. See Figure 1. Notice that collectively there are at most $O(\frac{n}{k^2})$ connected subregions among the regions $V_1, \ldots, V_l$. These connected subregions are defined by the $O(\frac{n}{k^2})$ edges of $\tilde{V}$ that are removed during the planar separator process. (Note that any V_i may be the union of numerous connected subregions.) Using Kirkpatrick's planar point location algorithm [26], preprocess this set of edges to create a data structure which on input q, can locate the region V_i containing q. Call this the *locator tree* for $V_1, \ldots, V_l$. This tree has space that is linear in the number of edges needed to bound the V_i's.

Observation 3 *The locator tree uses only $O(\frac{n}{k^2})$ space.*

Proof: Each boundary edge of a region V_i corresponds to a removed edge in the dual. We have removed at most $O(\frac{n}{k^2})$ edges. □

Once we have located the V_i which contains q we only need to search P_i to find the k-nearest neighbors of q. Since P_i was defined from $\sim k^6$ sets of points S_v for $v \in \tilde{V}_i$, and since each S_v contains k points, $\|P_i\| \leq k^7$. Hence by traversing the locator tree in $O(\log n)$ time, we can reduce the original query problem to one of finding the k-nearest neighbors out of a set P_i with only $O(k^7)$ points. At this point, one should notice that the sets $P_1, \ldots, P_l$ are **not** pairwise disjoint. In fact, we have $\sim \frac{n}{k^6}$ P_i's, each of size $O(k^7)$, and a naive analysis would suggest that $O(kn)$ memory is required to store the P_i's. In order to obtain an $O(n)$ upper bound on

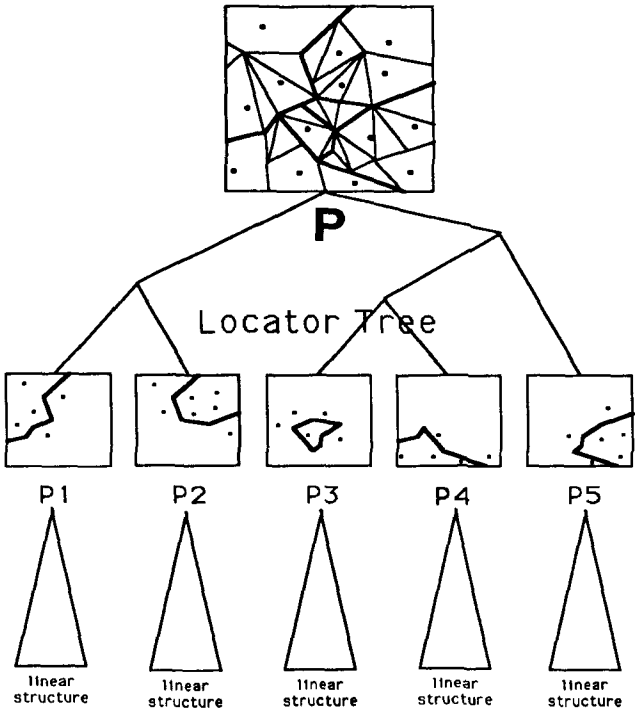

Figure 2: Linear structures for the P_i's give a linear structure for P

the size of the data structure we are building, it is necessary to give a strict upper bound on the number of duplications of points which occur among the P_i's. Toward this end, we show that the total number of points, counting duplications, which need to be stored in the data structure is bounded above by $(1 + O(\frac{1}{k}))n$:

Lemma 4 $\sum_{i=1}^{l} \|P_i\| \leq (1 + O(\frac{1}{k}))n$.

Proof: We begin by assigning each point $p \in P$ to the node $v_p \in \tilde{V}$ whose corresponding face in V contains p. Now we argue that the number of duplicates of points in P which are contained in the P_i's is bounded by k times the number of edges cut during the planar separator stage of the construction. Let $x \in P_i$. Then $x \in S_v$ for some $v \in \tilde{V}_i$. So $v \in (\tilde{V})_x$ which is connected by Observation 1. Then either $(\tilde{V})_x$ is completely contained in $\tilde{V}_i$ (and x occurs uniquely in P_i) or else an edge e of $(\tilde{V})_x$ was cut during the construction of $\tilde{V}_i$ (i.e., a node of $\tilde{V}_i$ must be incident to a cut edge e for which $x \in L_e$). Since L_e contains at most k points for any e and at most $O(\frac{n}{k^2})$ edges have been cut, the total number of duplicate points in all of the P_i's is $O(\frac{n}{k})$. □

2.2 Probabilistic Techniques

Each of the set of points P_i will now be treated separately. It follows from Lemma 4 that by creating a linear size data structure for k-nearest neighbor queries on each P_i, we can build a linear size data structure for k-nearest neighbor queries on P using the locator tree discussed above. See Figure 2 for a picture of such a data structure.

For each P_i, create a set of $s = \frac{k}{\log^2 k}$ buckets $B_i^1, \ldots, B_i^s$. We assign the points of P_i to these buckets with equal probability. That is, a point $p \in P_i$ is assigned to bucket B_i^j with probability $\frac{\log^2 k}{k}$. Our intention is to search each bucket for the $\sim \log^2 k$ nearest neighbors of q, and in this manner capture the k-nearest neighbors of q in P_i. To do this, we need to show that for any q, the k-nearest neighbors of q are evenly distributed among the buckets. Consider the k^{th}-order Voronoi diagram on P_i. This diagram has $m = O(k^8)$ regions corresponding to all possible sets of k-nearest neighbors to q. Call the corresponding sets of k points: $C_1, \ldots, C_m$ the *constraints*. We say that a constraint C_r is *satisfied* if each bucket $B_i^1, \ldots, B_i^s$ contains less than $\log^2 k + O(\log^{\frac{3}{2}} k)$ points from C_r.

Lemma 5 *With probability* $\geq 1 - \frac{1}{k}$ *every constraint is simultaneously satisfied by a random assignment of* P_i *to the buckets.*

Proof: This is a direct application of Chernoff bounds. In particular, let $X_1, \ldots, X_k$ be independent $0-1$ random variables with probability q of being 1. Then if $\chi = \sum_1^k X_i$, and $\beta \geq 1$, $\Pr[\chi \geq \beta qk] \leq exp((\beta - 1 - \beta \ln \beta)qk)$. This inequality is a well known Chernoff bound and can be easily derived using moment generating functions. Now choose a constraint $C_r = \{p_1, \ldots p_k\}$ and for $l = 1 \ldots k$ let $X_l = 1$ if $p_l \in B_i^j$. In this case $q = \frac{\log^2 k}{k}$. We say that C_r is satisfied if less that $\log^2 k + \gamma \log^{\frac{3}{2}} k$ points from C_r are contained in any one bucket. We will show that γ can be chosen to satisfy the lemma. Now let $\beta = 1 + \frac{\gamma}{\sqrt{\log k}}$. Then C_r is satisfied if less than βqk points from C_r are contained in any one bucket. Since there are $O(k^8)$ constraints and $O(k)$ buckets, from the Chernoff bound given above, we conclude that the probability that some constraint is not satisfied is less than $k^9 \Pr[$ the number of points from C_r in bucket $B_i^j \geq \log^2 k + \gamma \log^{\frac{3}{2}} k] \leq k^9 exp((\beta - 1 - \beta \ln \beta)qk) \leq k^9 exp(-\frac{\gamma^2 \log k}{2} + \frac{\gamma^3 \sqrt{\log k}}{2})$. Hence, in order to satisfy the lemma, it suffices to pick γ such that $k^9 exp(-\frac{\gamma^2 \log k}{2} + \frac{\gamma^3 \sqrt{\log k}}{2}) \leq \frac{1}{k}$. When k is sufficiently large, $\gamma \geq 4$ is good enough. □

For each set P_i, assign the points randomly to buckets and then check all of the constraints to see that they are satisfied. Lemma 5 tells us that with probability $\geq 1 - \frac{1}{k}$ the assignment to P_i will check out on the first try. We can build the k^{th}-order Voronoi diagram on P_i in $O(k^2\|P_i\| + \|P_i\| \log(\|P_i\|))$ time [1, 27]. By Lemma 5 we expect to check at most 2 assignments in order to successfully divide P_i into buckets. Hence the expected time to assign all of the P_i's to buckets is $O(k^2 n + n \log n)$. Here we have described a Las Vegas algorithm to divide up the regions P_i into buckets.

The assignments to buckets are guaranteed to satisfy all constraints, but the running time of the algorithm is not guaranteed. However, with high probability this algorithm will terminate in $O(k^2n + n\log n)$ time. In the next section we show how to assign the P_i's to buckets deterministically.

2.3 The Compacted Data Structure

At this point, we have a data structure which divides P up into sets $P_1, \ldots, P_l$ with very little duplication of points. Given a query point, we can locate the proper set P_i which contains q's k-nearest neighbors in $O(\log n)$ time. Each P_i is further divided up into $\frac{k}{\log^2 k}$ buckets. We will retrieve the $\log^2 k + 4\log^{\frac{3}{2}} k$ nearest neighbors to q from each bucket $B_i^1, \ldots, B_i^s$. The idea now is to recursively build a data structure for retrieving $\log^2 k + 4\log^{\frac{3}{2}} k$ nearest neighbors in each bucket. At the bottom level, where we are searching buckets for the c (a fixed constant which does not vary with k) nearest neighbors of q, we simply construct c^{th}-order Voronoi diagrams on the points in these buckets and process these diagrams using Kirkpatrick's planar point location techniques to get linear space, $O(c)$ time structures for finding the c-nearest neighbors to q. The c points retrieved from each leaf are stored in a linear array of possible neighbors for q. In what follows, we will show that at most $O(k)$ points are stored in this array. This array of $O(k)$ points is guaranteed to contain the k-nearest neighbors of q. To find the k-nearest neighbors, we will begin by finding the k^{th}-nearest point, x, to q in this array in $O(k)$ time by using an order statistic algorithm [8]. Then simply compare each point to x and keep those which are closer or equal in distance to q.

Below we give an analysis of the data structure to show that it occupies $\Theta(n)$ space. We also solve the necessary recurrence to prove that the structure reports a set of $O(k)$ points containing the k-nearest neighbors of a query point q in $O(\log n + k)$ time.

The data structure we have constructed is a tree with levels alternating between locator trees and pointers to buckets. Notice that at level i of this structure we are breaking up a search problem for k_i neighbors into a number of new search problems for $k_{i+1} = \log^2 k_i + O(\log^{\frac{3}{2}} k_i)$ neighbors. Hence at level i, $\Theta(\overbrace{\log\log\ldots\log}^{i} k)^2$ is a lower bound on the range of the neighbor searches being performed. Now notice from Observation 3 that the locator trees at level i will occupy $O(\sum_j \frac{m_j}{k_{i-1}^2})$ space, where m_j are the sizes of the buckets on level $i-1$. By repeated applications of Lemma 4: $\sum_j m_j \leq (1+\frac{1}{k})\cdots(1+\frac{1}{k_{i-1}})n$. Hence, the total amount of space occupied by the locator trees is: $O(\frac{n}{k^2} + \frac{(1+\frac{1}{k})n}{(\log k)^2} + \frac{(1+\frac{1}{k})(1+\frac{1}{\log^2 k})n}{(\log\log k)^2} + \cdots)$ $= O(n)$ space. Since the number of bucket nodes increases geometrically every other level, the space used to store the bucket nodes is dominated by the space used to store the leaves of the data structure. Each leaf holds a constant sized Voronoi diagram. Hence the total amount of space used to store this structure is bounded by a constant times the total number of points (including duplicates) which appear in the Voronoi diagrams in the leaves. By Lemma 4 this number is $O(((1+\frac{1}{k})(1+\frac{1}{(\log k)^2})(1+\frac{1}{(\log\log k)^2})\cdots)n) = O(n)$.

The upper bound on running time for extracting the k-nearest neighbors is equal to the time required to traverse the first level locator tree, plus upper bounds on the times required to search recursively through the first level buckets:

$$\begin{aligned} T(n,k) &\leq \alpha\log n + \frac{k}{\log^2 k}T(k^7, \log^2 k + 4\log^{\frac{3}{2}} k) \\ T(n,c) &\leq \alpha c \end{aligned}$$

This can be solved by substitution using the following expression for $T(n,k)$:

$$\alpha\log n + \frac{7\alpha k}{\log k} + \frac{7\alpha(1+\frac{4}{\sqrt{\log k}})k}{\log\log k} + \frac{7\alpha(1+\frac{4}{\sqrt{\log k}})(1+\frac{4}{\sqrt{\log\log k}})k}{\log\log\log k} + \cdots$$

Hence, we have $T(n,k) = O(\log n + k)$. In the case that the query point q is restricted to a set of $O(n)$ points, then this time can be reduced to $O(k)$. To see this, let $c_1, \ldots, c_m$ be the possible values for q where $m = O(n)$. Eliminate the locator tree at the first level of the compacted data structure and replace it with an array $A[1\ldots m]$ where $A[j]$ contains a pointer to the node on the second level which handles the set P_i containing the k-nearest neighbors of c_j. On input c_j, begin traversing the compacted data structure at the node indicated by the pointer $A[j]$. In the running time analysis this replaces the $\alpha\log n$ term with $O(1)$, and the recurrence for $T(n,k)$ now has solution $O(k)$.

Lastly, we must check that at most $O(k)$ points are reported. Notice that points are reported only in the leaves of this structure and are not *passed up* the tree and filtered at each level. In fact, if we did compute order statistics and filter points at each level, the resulting report time would be $\Omega(\log n + k\log^* k)$. Implicit in the recurrence given above is the fact that the superset of the k-nearest neighbors collectively reported by the leaves has size at most: $((1+\frac{4}{\sqrt{\log k}})(1+\frac{4}{\sqrt{\log\log k}})(1+\frac{4}{\sqrt{\log\log\log k}})\cdots)k = O(k)$.

2.4 Extensions to Voronoi Diagrams in Other Metric Spaces

The ideas used above to construct a linear-sized data structure for optimal-time k-nearest neighbor search-

ing are general enough to apply to Voronoi diagrams constructed using a wide variety of distance functions. In fact, it is clear from our discussion that the techniques work whenever the k^{th}-order Voronoi diagram is planar and contains $O(k^\beta n)$ edges and faces for some constant β. In such cases, we can always apply the planar separator theorem to build a locator tree leading to sets P_i containing $O(k^{\beta+6})$ points. At this point, we will have $O(k^{2\beta+6})$ constraints imposed on the buckets for P_i. As in the L_2 case, these constraints can be satisfied with high probability. The resulting data structure obtained by applying these techniques recursively satisfies the same recurrences as above with slightly different constants. Applying these ideas to non-Euclidean Voronoi diagrams (such as those generated by the power-distance metric or weighted metric with additive weights) is important in the applications sections appearing below. Note that the Voronoi diagrams for these metrics have $k^{O(1)}$ edges and the connectivity property described in Observation 1.

2.5 Removing Randomness from the Construction

To make the above construction deterministic, we use the well known *probabilistic method* described in [31]. For each P_i, we will proceed by first assigning the points to two buckets B_0 and B_1. Then each of these buckets will be split into two buckets, etc., until after $O(\log k)$ iterations the points of P_i have been split among $\frac{k}{\log^2 k}$ buckets. During the initial iteration, the points of P_i are taken in some order $p_1, p_2, \ldots$ and assigned to B_0 or B_1. The probabilistic method allows us to ensure that each bucket ends up containing $\leq \frac{k}{2} + O(\sqrt{k \log k})$ points from each constraint C_j. Central to the methods in [31] for solving this problem is the computation of the conditional probabilities of the form: Pr[at least $\frac{k}{2} + O(\sqrt{k \log k})$ points from C_j are assigned to B_0 | $p_1 \in B_0, p_2 \in B_1, \ldots, p_j \in B_0$]. Such a conditional probability can be computed exactly as a binomial series with at most k terms. To do this efficiently, it will suffice to precompute all $\binom{k'}{j}$ for $k' \leq k$ and $j \leq k'$. This can be accomplished iteratively in $O(k^2)$ time. Then the conditional probabilities needed are all of the form: $\frac{1}{2^u} \sum_{l=v}^{u} \binom{u}{l}$. These can now be computed and stored iteratively using the previously stored binomial coefficients. The total time required is again $O(k^2)$.

Each iteration of the bucket assignment algorithm requires looking up conditional probabilities for $O(k||P_i||)$ constraints with k points per constraint. Hence, each iteration of the bucket assignment algorithm can be accomplished in $O(k^2||P_i||)$ time and, since there are $O(\log k)$ iterations, the entire assignment process for the first level of the structure takes $O(k^2 n \log k)$ time. This bound then decreases geometrically for each subsequent level. Hence, the maximum number of points from any constraint C_i in a given bucket obeys the following recurrence: $m_j \leq \frac{m_{j-1}}{2} + O(\sqrt{m_{j-1} \log k})$ where $m_0 = k$. It follows by substitution that $m_j \leq \frac{k}{2^j} + O(\sqrt{\frac{k}{2^j} \log k})$ for all j such that $\frac{k}{2^j} = \Omega(\log k)$. Hence, at the $\log(\frac{k}{\log^2 k})$ stage, we have produced roughly equal sized buckets $P_1, \ldots P_{\frac{k}{\log^2 k}}$ such that each bucket contains at most $\log^2 k + O(\log k \sqrt{\log k})$ points from any C_i.

2.6 Total Preprocessing Time

The total computation time for building the compressed Voronoi diagram is given by the sum of the times for the following operations. The time given in each case is the sum of the associated operations performed at every level:

1. Compute all Voronoi diagrams: $O(k^2 n + n \log n)$ [1, 27].
2. Compute planar separators to build P_i's: $O(kn \log \frac{n}{k^6})$ [28].
3. Build all of the locator trees: $O(n \log n)$ [26].
4. A deterministic assignment of the P_i's to buckets: $O(k^2 n \log k)$ [Section 2.5].
5. A randomized assignment of the P_i's to buckets: $O(k^2 n + n \log n)$ expected time [Section 2.2].

The dominating term in the sum of these time bounds varies with k, but it is clear that the construction can always be accomplished in $O(k^2 n \log k + kn \log n)$ deterministic time or $O(k^2 n + kn \log n)$ randomized time.

3 A Monte Carlo Construction Achieving $O(n \log^2 n \log\log n)$ Preprocessing Time

For $k \leq \log n$, the deterministic construction has preprocessing time $O(n \log^2 n \log\log n)$. For $k \geq \log n$, the preprocessing time for the above deterministic construction is dominated by the computation of Voronoi diagrams and the assignment of points to buckets *at the one or two top levels.* However, when $k \geq \log n$, we can discard the initial computation of Voronoi diagrams, and assign points directly to buckets in a randomized manner. In what follows we show that with high probability this approach builds a correct data structure for solving the k-nearest neighbors problem in $O(n \log^2 n \log\log n)$ time. Furthermore, each time we query this data structure we get a *witness* which tells us whether or not the answer provided to our query is correct.

When $k \geq \log n$, skip the initial planar separator stage, and at the first level assign the points directly to $\frac{k}{\log n}$ buckets using $O(n)$ time. As we demonstrated in

the previous analysis of probabilistic bucket assignment, with high probability, in order to capture the k-nearest neighbors of q it suffices to report the $O(\log n)$ nearest neighbors of q in each bucket. This can be accomplished using the above deterministically constructed data structures using $O(n \log^2 n \log\log n)$ total preprocessing time. Hence the total preprocessing time to build this Monte Carlo structure is $O(n \log^2 n \log\log n)$. If we use a randomized assignment of points with checking (as presented in Section 2.2) at the first level, this time bound then reduces to $O(n \log^2 n)$.

Given a query q to this new data structure, keep track of the $O(\log n)$ points reported from each of the first level buckets. We are certain that each bucket reported correctly. Now let $p_1, \ldots, p_k$ be the final report from this new structure. If our top level assignment of points to buckets is correct, there should be at least one point reported from each bucket that does not make the final list $p_1, \ldots, p_k$. These $\frac{k}{\log n}$ discarded points are our witnesses that the final list is correct. If all of the points reported from some bucket are contained in the final list, then that bucket contains more than $\log^2 k + 4\log^{\frac{3}{2}} k$ of the nearest neighbors to q and the top level assignment of points to buckets violates at least one constraint.

4 Applications of the Compaction Technique

4.1 k-Nearest Neighbors

Given a set of n points in the Euclidean plane E^2, and a fixed k, the technique given in Section 2 provides an $O(n)$ space data structure for computing the k-nearest neighbors of a query point q. However, if k is not fixed, but rather given as a part of the query, then we simply construct $\log n - \log\log n$ linear size data structures $D_1, \ldots, D_l$, where the D_i can be used to obtain the $2^i \log n$-nearest neighbors of q. These $O(\log n)$ structures take a total of $O(n \log n)$ space. Now, given a pair (q, k), we simply compute the smallest j such that $2^j \geq \lfloor \frac{k}{\log n} \rfloor$ and use D_j to compute the $2^j \log n$-nearest neighbors of q in $O(\log n + 2^j \log n) = O(\log n + k)$ time. From this set of $O(k + \log n)$ neighbors, filter out the k-nearest in $O(k + \log n)$ additional time by finding the k^{th} order statistic x [8] and keeping the points which are as close to q as x.

4.2 Circular Range Search

To solve the circular range search problem we also use the $\log n - \log\log n$ data structures described above. When given a query disk with center c and radius r, use q to query $D_1, D_2, \ldots$ until some structure D_j provides output which contains a point x farther than distance r from q. (Observe that this is a straightforward application of the filtering search technique [9]). Now, let j be the smallest index such that D_j outputs a point x outside the query disk. Then the time to execute these j stages is simply: $O(\sum_{i=0}^{j}(2^i + 1)\log n) = O(2^j \log n + \log n) = O(k + \log n)$ since $k = O(2^j \log n)$. Finally, search through the $2^j \log n$ points given by D_j and output those which are contained inside the query disk.

4.3 Half Space Range Search and Weighted Diagrams

Given a set P of n points in E^2, the power distance from any point q in E^2 to a point $p \in P$, denoted by $d_P(p, q)$, is taken to be equal to $d^2(p, q) - w_p^2$ where $d(p, q)$ is simply the Euclidean distance between p and q, and w_p is the weight assigned to p in the power-distance metric. If $w_p \geq 0$, then the following geometric interpretation can be given to $d_P(p, q)$: Let L be the line through q that is tangent to a disk with radius w_p^2 and center p. Then $\sqrt{d_P(p, q)}$ is the length of the segment between q and the point on L tangent to the disk.

Given two points $p, s \in P$, the locus of points with equal power distance from both p and q can be shown to be a straight line. Hence, the concept of k^{th}-order Voronoi diagram generalizes nicely to this metric. Aurenhammer [3, 4] and Edelsbrunner [20] discuss some properties of k^{th}-order Voronoi diagrams in the power-distance metric, and they call these diagrams k^{th}-order power diagrams. Aurenhammer and Edelsbrunner also provide a correspondence between the arrangements of planes in 3 dimensions and power diagrams in 2 dimensions. This correspondence allows us to transform the half-space range search problem into a nearest neighbor search problem in the power distance metric. This transformation is well known to many researchers in computational geometry and we only outline it below.

Recall that for the half-space range search problem in 3 dimensions, we are given a set P of n points in E^3 and a query half-space. We are asked to report all points of P that lie inside (or on the boundary of) this half-space. In what follows, we show how this problem can be transformed into a nearest neighbor search problem in the power-distance metric. To begin, we show how to transform the range search problem into a ray-stabbing problem using geometric duality [15, 23].

Geometric Duality In the half-space range searching problem, we are given a set of points in 3-space (a_i, b_i, c_i) for $i = 1 \ldots n$. Then for each query half-space $\alpha x + \beta y + \gamma z \geq 1$ we are asked to return those points which are contained in the half-space or on its boundary. By using geometric duality, for each point (a_i, b_i, c_i) we have a half-space in the dual given by the equation $a_i x + b_i y + c_i z \geq 1$. Similarly, the query half-space has a point (α, β, γ) in the dual space. Then we have the following duality result:

Observation 6 *A point (a_i, b_i, c_i) lies inside the half-space $\alpha x + \beta y + \gamma z \geq 1$ iff the dual point (α, β, γ) lies*

inside the dual half-space $a_i x + b_i y + c_i z \geq 1$.

Proof: Both sides of the "iff" are equivalent to: $\alpha a_i + \beta b_i + \gamma c_i \geq 1$. □

Such simple duality results have powerful consequence in computational geometry [15]. For our purposes, Observation 6 indicates that the transformation from a half-space range searching problem to a ray stabbing problem can be accomplished in $O(n)$ time and space. After the transformation, we are given a set of half-spaces defined by $a_i x + b_i y + c_i z \geq 1$ for $i = 1 \ldots n$ and a query point (α, β, γ). Our algorithm should report those half-spaces which contain this query-point.

Without loss of generality, rotate the coordinate axes so that none of the planes defining our half-spaces is parallel to the z-axis. Each half-space now intersects the z-axis. Define those half-spaces which contain a segment of the z-axis going to $+\infty$ to be *upwardly oriented*. Define the others to be *downwardly oriented*. We will split these groups into two sets and treat them separately. This will cause us to run the algorithm twice, once on the upwardly oriented half-spaces and once on the downwardly oriented half spaces. Within a constant factor, however, this will not affect our running time. So, without loss of generality, we will assume that all of the half-spaces are upwardly oriented. Hence, if we consider a ray which originates at point (α, β, γ), extends downward to $z = -\infty$, and that runs parallel to the z-axis, then the point (α, β, γ) is contained in exactly those half-spaces whose boundary planes the ray intersects. In this manner, we have transformed the half-space range query problem into a vertical ray-stabbing problem.

In order to describe how to solve the ray-stabbing problem using k^{th}-order power diagrams we need to understand more about the relationship between power diagrams and planes in Euclidean 3-space.

Power Diagrams and Planes in 3-Space Aurenhammer and Edelsbrunner [4, 3, 21] provide a correspondence between the arrangements of planes in 3 dimensions and power diagrams in 2 dimensions. This correspondence hinges on the notion of a k-set. Given a set of n planes in E^3, the *k-set* is the locus of all points in E^3 that satisfy the following properties:

1. Each point in the k-set belongs to one of the n planes.
2. For every point in the k-set there are exactly $n - k$ planes passing above it in the vertical dimension (i.e., the positive z-direction).

Two points are said to be in the same *face* of a k-set if they lie in the same plane and have exactly the same planes passing above them.

In the above definition of a k-set, we say that a plane lies above a point (a, b, c), if for some positive λ, the point $(a, b, c + \lambda)$ belongs to the plane. Notice that the boundary of a k-set is formed by the lines defining the intersection of pairs of planes. Let Δ be a face in a k-set. Then we define the projection of Δ to be the image of Δ projected onto a plane at $z = -\infty$: $\{(x, y): (x, y, z) \in \Delta \text{ for some } z\}$. Now fix some k_0. It is not hard to see that the projection of all the faces of the k_0-set gives a division of the plane into convex sub-regions.

We now wish to establish a correspondence between the weighted points which define a power diagram, and planes in 3-space. For proofs and more detailed explanations of the results appearing below, we refer the reader to the work of Aurenhammer and Edelsbrunner [4, 3, 20, 21]. Recall that each point p_i in the set of points defining a power diagram has an associated weight: $w(p_i)$. Define a plane $\pi(p_i)$: $z = 2(x, y)^T p_i - p_i^T p_i + w(p_i)$ where $(x, y)^T p_i$ is the dot product of (x, y) and (x_i, y_i) and $p_i = (x_i, y_i)$. Furthermore, for any two points p_i, p_j we define $\text{chor}(p_i, p_j)$ to be the locus of points equidistant from p_i and p_j in the power distance metric with weights $w(p_i)$ and $w(p_j)$. It is not hard to see that $\text{chor}(p_i, p_j)$ is a straight line. Furthermore, we have the following observation:

Observation 7 *The projection of* $\pi(p_i) \bigcap \pi(p_j)$ *onto the plane is* $chor(p_i, p_j)$.

In fact, a much stronger result can be proven. Consider the k-sets formed by the set of planes $\Gamma = \pi(p_1), \pi(p_2), \ldots \pi(p_n)$. Then for a fixed k we have:

Theorem 8 *The projection of the faces of the k-set of Γ onto the plane at $z = -\infty$ is the k^{th}-order power diagram on the points $p_1, \ldots, p_n$ with weights $w(p_i)$. Furthermore, if $\pi(r_1)$ is the plane containing one of the faces Δ of the k-set, and $\pi(r_2), \ldots, \pi(r_k)$ are the planes below Δ, then the k-nearest neighbors to the points in the projection of Δ are $r_1, \ldots, r_k$.*

Theorem 8 can be used to give an upper bound on the size of the k^{th}-order power diagram. Since the maximum number of faces in a k-set in 3 dimensions has been shown to be $O(k^2 n)$ (see Clarkson and Shor [18]), the number of edges in a k^{th}-order power diagram is $O(k^2 n)$. Consequently, if k is known ahead of time, then our compaction techniques can be used to obtain an $O(n)$ data structure which can be used to compute the k-nearest neighbors in the power metric in $O(k + \log n)$ time. On the other hand, if k is not fixed, but given as a part of the query, then we can construct $\log n - \log\log n$ linear data structures as in the previous section and obtain the k-nearest neighbors of q in $O(\log n + k)$ time. Again, this structure requires $O(n \log n)$ space.

Returning to the vertical ray-stabbing problem, $p_1, \ldots, p_n$ and $w(p_1), \ldots, w(p_n)$ are the points and weights corresponding to the power diagram generated by the projection of Γ. We are given a query point (α, β, γ), and are asked to determine the planes that this point lies above. Let $D_1, D_2, \ldots, D_{\log n - \log\log n}$ be the

compacted data structures described above. First, we query D_1 with the point (α, β) to determine the $\log n$-nearest points $r_1, \ldots, r_{\log n}$ in the power-distance metric. This takes $O(\log n)$ time. Then $\pi(r_1), \ldots, \pi(r_{\log n})$ are the lowest $\log n$ planes which might appear below (α, β, γ). We can then check each of these planes $\pi(r_i)$ in constant time to see if (α, β, γ) appears above $\pi(r_i)$. At this point, we have still used $O(\log n)$ time. If all of $\pi(r_1), \ldots, \pi(r_{\log n})$ appear below (α, β, γ), then continue on to check D_2 and retrieve the $2 \log n$ nearest neighbors to (α, β). Proceed in this manner, using the filtering search techniques presented in the previous section. At some D_i, we will find the first plane $\pi(r_j)$ which lies above (α, β, γ). At this point, stop and report all of the planes from the set $\pi(r_1), \ldots, \pi(r_{\log n 2^i})$ which lie below (α, β, γ). Let us say that k planes are reported. Then the set of planes retrieved from D_i has size at most $2k$. Since at a minimum D_i gives us $\log n$ planes to search through, it is clear that this algorithm runs in $O(\log n + k)$ time.

4.4 k-Nearest Neighbors in the Weighted Metric

The k-nearest neighbor searching in the weighted metric (with additive weights) can also be performed using data structures built from the compaction techniques given in this paper. As described earlier, the k^{th}-order Voronoi diagram for this metric has only $O(kn)$ edges. However, since these edges are defined by second degree curves we cannot use Kirkpatrick's planar point location algorithm to build the locator trees in the compact data structure. In this case, we use the planar point location data structure given by Edelsbrunner, Guibas, and Stolfi [22]. With this change, the resulting compact Voronoi diagrams in the additive-weight metric give the same optimal time and space complexity performance for finding k-nearest neighbors as the euclidean structures. When k is fixed, the data structure occupies $O(n)$ space and responds to k-nearest neighbor queries in $O(\log n + k)$ time.

4.5 Further Extensions of the Compaction Technique

The technique given in Section 2 for the k-nearest neighbors problem may have applications to other retrieval problems. For example, this technique can be modified to obtain an $O(n)$ space data structure that can be used to solve a retrieval problem in $\Theta(k + \log n)$ time for any class of objects with the following properties:

1. Each object in the class is a bounded or unbounded region of the plane and its boundary is some Jordan curve.

2. If we consider any n objects in this class then the number of disjoint *k-regions* is $O(k^\beta n)$ (for some constant $\beta > 0$). We define a k-region as a connected region of the plane such that any point contained in that region belongs to exactly k objects. Furthermore, each k-region must be adjacent to at most $O(k^\alpha)$ other regions for some constant $\alpha > 0$.

3. The partitioning of the plane into k-regions by the set of Jordan curves J defining the n objects must satisfy certain point location properties: given any sub-partitioning of the plane by some $O(m)$ sized subset of J, there exists a data structure of $O(m)$ size which can perform planar point location in $O(\log m)$ time.

It is not hard to see that most geometric objects (circles, ellipses, polygonal curves with at most a constant number of edges) satisfy these properties. Consequently, for most of these problems, given a fixed value of k, we can obtain an $O(n)$ data structure that reports the $\Theta(k)$ objects containing a query point q in $O(\log n + k)$ time. (If more or less than $\Theta(k)$ objects contain q then the structure would report the empty set in $O(\log n)$ time.) However, for all such cases, these results are not new since the techniques given in [12, 14] can be used to obtain simpler $O(n)$ space data structures. In a similar vein, we can use this technique together with filtering search to solve the 3-dimensional dominance reporting problem in $O(n \log n)$ space and $O(\log n + k)$ time, but again a simpler data structure that achieves the same space and time bounds has been given by Gabow et al. [24].

5 Acknowledgments

The authors thank Bernard Chazelle, Herbert Edelsbrunner, and David Johnson for several useful discussions.

References

[1] A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor, "A Linear Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon," Proc. of the 19^{th} Ann. ACM Symp. on the Theory of Computing, 1987, pp. 39 - 47.

[2] P.F. Ash, E.D. Bolker, "Generalized Dirichlet tesselations," Geometriae Dedicata 20 (1986), pp. 209 - 243.

[3] F. Aurenhammer, "Voronoi Diagrams - A Survey," Technical Report, Institutes for Information Processing, Graz Technical University, Austria, 1988.

[4] F. Aurenhammer, "Power Diagrams: properties, algorithms, and applications," SIAM J. Computing 16 (1987), pp. 78 - 96.

[5] F. Aurenhammer, H. Edelsbrunner, "An optimal algorithm for constructing the weighted Voronoi diagram in the plane," Pattern Recognition 17 (1984), pp. 251 - 257.

[6] F. Aurenhammer, H. Imai, "Geometric relations among Voronoi diagrams," Geometriae Dedicata 27 (1988), pp. 65 - 75.

[7] J.L. Bentley, H.A. Maurer "A note on Euclidean near neighbor searching in the plane," Inf. Process. Lett. 8 (1979) pp. 133 - 136.

[8] M. Blum, R.W. Floyd, V. Pratt, R. Rivest, R.E. Tarjan, "Time bounds for selection", Journal of Computer and System Sciences, vol. 7, no. 4, Aug. 1973, pp. 448 - 461.

[9] B.M. Chazelle, "Filtering search: a new approach to query-answering," SIAM J. Comput., Vol. 15, No. 3, Aug. 1986, pp. 703 - 724.

[10] B.M. Chazelle, "A functional approach to data structures and its use in multidimensional searching," SIAM J. Comput. Vol. 17, No. 3, June 1988, pp. 427 - 462.

[11] B.M. Chazelle, R. Cole, F.P. Preparata, C.K. Yap, "New upper bounds for neighbor searching," Information and Control 68 (1986), pp. 105 - 124.

[12] B.M. Chazelle, H. Edelsbrunner, "Optimal solutions for a class of point retrieval problems," J. Symbolic Computation (1985) Vol. 1, pp. 47 - 56.

[13] B.M. Chazelle, H. Edelsbrunner, "An improved algorithm for constructing k^{th}-order Voronoi diagrams," IEEE Trans. Computing C-36 (1987) pp. 1349 - 1354.

[14] B.M. Chazelle, H. Edelsbrunner, "Linear space data structures for two types of range search," Discrete and Computational Geometry Vol. 2 (1987) pp. 113 - 126..

[15] B.M. Chazelle, L.J. Guibas, D.T. Lee,"The power of geometric duality," BIT 25 (1985), pp. 76 - 90.

[16] B.M. Chazelle, F.P. Preparata, "Half-space range search: an algorithmic application of k-sets," Discrete and Computational Geometry, Vol. 1, No.1 (1986) pp. 83 - 94.

[17] L.P. Chew, R.L. Drysdale, "Voronoi diagrams based on convex distance functions," Proc. 1^{st} Ann. ACM Symp. Computational Geometry (1985) pp. 235 - 244.

[18] K.L Clarkson and P.W. Shor, "Applications of random sampling in computational geometry," Discrete and Computational Geometry, Vol. 4, No. 5, (1989) pp. 387 - 422.

[19] R. Cole, C.K. Yap, "Geometric retrieval problems," Proc. 24th Annual IEEE Symp. on Foundations of Computer Science, Nov. 1983, pp. 112 - 121.

[20] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag, Berlin, West Germany, 1986.

[21] H. Edelsbrunner, "Edge-skeletons in arrangements with applications," Algorithmica 1 (1986) pp. 93 - 109.

[22] H. Edelsbrunner, L.J. Guibas, J. Stolfi, "Optimal point location in a monotone subdivision," SIAM J. Computing 15 (1986) pp. 317 - 340.

[23] H. Edelsbrunner, J. O'Rourke, R. Seidel, "Constructing arrangements of lines and hyperplanes with applications," SIAM J. Computing 15 (1986) pp. 341 - 363.

[24] H.N. Gabow, J.L Bentley, R.E. Tarjan, "Scaling and related techniques for geometry problems," Proc. of the 16th Annual ACM Symp. of Theory of Computing (1984) pp. 135 - 143.

[25] D.S. Johnson, *personal communication* 1989.

[26] D.G. Kirkpatrick "Optimal search in planar subdivisions.", SIAM J. Comput. 12(1983), pp. 28 - 35.

[27] D.T. Lee, "On k-nearest neighbor Voronoi diagrams in the plane," IEEE Trans. on Computers, Vol. C-31, 1982, pp. 478 - 487.

[28] R.J. Lipton and R.E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal on Applied Mathematics,* Vol. 36, No. 2 (April 1979), pp. 177 - 189.

[29] D. Marcotte and S. Suri, "Fast matching algorithms for points on a polygon," Proc. of the 30th Annual IEEE Conf. on Foundations of Computer Science (1989) pp. 60 - 65.

[30] H. Rosenberger, "Order-k Voronoi diagrams for sites with additive weights in the plane. Rep. UIUCDCS-R-88-1431, Dept. Comput. Sci, Univ. Illinois, Urbana, IL, 1988.

[31] J. Spencer *Ten Lectures on the Probabilistic Method,* SIAM, 1987.

[32] P. Vaidya, "Geometry helps in matching," Proc. of the 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 422 - 425.

[33] F.F. Yao "A 3-space partition and its applications," Proc. of the 15th Annual ACM Symp. on Theory of Computing (1983) pp. 258 - 263.