1979

Optimal expected-time algorithms for closest-point problems

Jon Louis. Bentley
Carnegie Mellon University

Bruce W. Weide

Andrew Chi-chih. Yao

CMU-CS-79-111

OPTIMAL EXPECTED-TIME ALGORITHMS FOR CLOSEST-POINT PROBLEMS

Jon Louis Bentley
Departments of Computer Science and Mathematics
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Bruce W. Weide
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Andrew C. Yao
Computer Science Department
Stanford University
Stanford, California 94305

Abstract

Geometric *closest-point problems* deal with the proximity relationships in k-dimensional point sets. Examples of closest-point problems include building minimum spanning trees, nearest neighbor searching, and triangulation construction. Shamos and Hoey [1975] have shown how the *Voronoi diagram* can be used to solve a number of planar closest-point problems in optimal worst-case time. In this paper we extend their work by giving optimal *expected-time* algorithms for solving a number of closest-point problems in k-space, including nearest neighbor searching, finding all nearest neighbors, and computing planar minimum spanning trees. In addition to establishing theoretical bounds, the algorithms in this paper can be implemented to solve practical problems very efficiently.

This research was supported in part by the Office of Naval Research under Contract N00014-76-C-0370 and in part by the National Science Foundation under Grant MCS-77-05313.

Table of Contents

1. Introduction

Shamos and Hoey [1975] have collected together and studied a group of problems in computational geometry that they refer to as *closest-point* problems. Problems in this set are usually defined on point sets in Euclidean space and include such computational tasks as nearest-neighbor searching, finding all nearest-neighbor pairs, and constructing Voronoi diagrams. The merits of studying these problems *as a set* have been proven repeatedly since the class was first defined. Not only do the various problems often arise in the same application areas, but time and again we have seen that advances made in the computational efficiency of an algorithm for one of the problems can be applied to increase the computational efficiency of others. In this paper we continue in this spirit by showing how the technique of "cells" can be used to produce optimal expected-time algorithms for many closest-point problems.

All of the closest-point problems that we will study in this paper have as input a set S of n points in Euclidean k-space. The *nearest-neighbor searching* problem calls for organizing the set S into a data structure such that subsequent queries asking for the nearest point in S to a new point can be answered quickly. The *all nearest neighbors* problem is similar: it calls for finding for each point in S its nearest neighbor among the other points of S. Both of these problems arise in statistics, data analysis and information retrieval. A problem similar to the nearest neighbor problems is that of finding the *closest pair* in a point set: that pair of points realizing the minimum interpoint distance in the set. The *minimum spanning tree* (or *MST*) problem calls for finding a tree connecting the points of the set with minimum total edge length. This problem arises in statistics, image processing, and communication and transport networks. The most complicated closest-point problem that we will investigate in this paper is the problem of constructing the *Voronoi diagram* of a point set. This problem, along with its applications, is described in Section 3. All of the problems that we have just mentioned are described in great detail by Shamos [1978]; he also discusses many of their applications.

Much previous work has been done on closest-point problems. The seminal paper in this field is the classic work of Shamos and Hoey [1975] in which the problems are defined and a number of optimal worst-case algorithms for *planar* point sets are given. Algorithms for closest-point problems in higher-dimensional spaces have been given by Bentley [1976], A. Yao [1977], and Yuval [1976]. Randomized algorithms for the closest-pair problem have been given by Rabin [1976] and Weide [1978]; Fortune and Hopcroft [1979] have recently shown that the speedup of the fast closest-pair algorithms was not due to their randomized nature alone but also to the model of computation employed (which allowed floor functions). For

additional results on closest-point problems the reader is referred to Preparata and Hong [1977] and Lipton and Tarjan [1977].

In this paper we study closest-point problems from a viewpoint not taken in any of the above papers. We assume that the point sets are randomly drawn from some "smooth" underlying distribution, and then use the cell technique to give fast expected-time algorithms for many closest-point problems. In Sections 2 and 3 we illustrate this idea by applying it to two fundamental closest-point problems (nearest-neighbor searching and Voronoi diagram construction) under the very restrictive assumption that the points are drawn uniformly from the unit square. In Section 4 we extend these results to other planar closest-point problems and to point sets drawn from a wide class of distributions. In Section 5 we extend our algorithms to problems in Euclidean k-space. Most of the algorithms we present solve a problem on n inputs in expected time proportional to n and all searching structures we give have constant expected retrieval time; these results are therefore optimal by the trivial lower bounds. Having thus resolved the primary theoretical issues, we turn to implementation considerations in Section 6. Conclusions and directions for further research are then offered in Section 7.

2. Nearest Neighbor Searching

The problem that is easiest to state and most clearly illustrates the cell method is *nearest-neighbor searching*, sometimes called the *post office problem*. Given n points in Euclidean space, we are asked to preprocess the points in time P(n) in such a way that for a new *query point* we can determine in time Q(n) which point of the original set is closest to it. There are (complicated) structures available for solving the problem with $P(n) = O(n \log n)$ and $Q(n) = O(\log n)$ in the worst case (Lipton and Tarjan [1977]), but one might expect that on the average we can do better. In fact we will see that for a large class of distributions of points, the expected values of P(n) and Q(n) can be made to be O(n) and O(1), respectively. Although we will initially restrict our attention to this apparently simple problem (and the planar case at that), the techniques used also apply to other closest point problems, which we will investigate in later sections.

We first consider the problem of nearest neighbor searching in the plane, where the points (both the original n points and the query point) are chosen independently from a uniform distribution over the unit square. The idea of the preprocessing step is to assign each point to a small square (*bin* or *cell*) of area C/n, so that the expected number of points in each bin is C. This is easily done by creating an array of size $(n/C)^{1/2}$ by $(n/C)^{1/2}$ that holds pointers to the lists of points in each bin.

When a query point comes in, we search the cell to which it would be assigned. If that cell is empty, then we start searching the cells surrounding it in a spiral-like pattern until a point is found. Once we have one point we are guaranteed that there is no need to search any bin that does not intersect the circle of radius equal to the distance to the first point found and centered at the query point. Figure 1 shows how the spiral search might proceed for the query point in the middle of the circle. Once the point in the northeast neighbor is found, only bins intersecting the circle must be searched. Each of these is marked with an x in Figure 1. In order to make this test easy, we suppose that all bins that lie within that distance of the query point in either coordinate are examined, making the number of cell accesses slightly larger than necessary but simplifying the specification of how the appropriate bins are to be found.

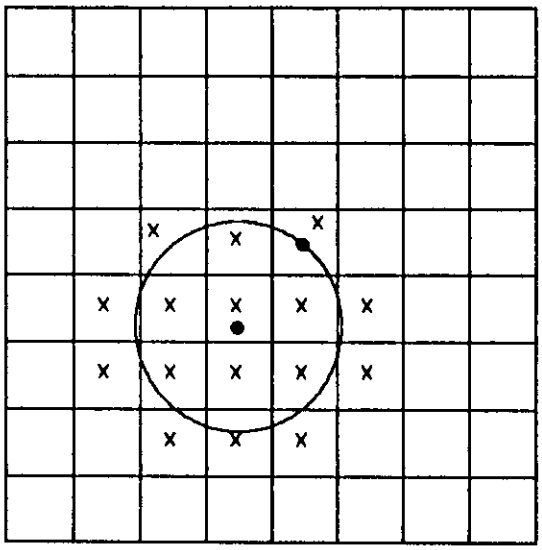

Figure 1. Spiral nearest neighbor search using cells.

It is clear that preprocessing, which consists of assigning each point to the appropriate bin, can be accomplished in linear time if computation of the floor function in constant time is allowed. *This assumption is necessary to solve most of the closest point problems in o(n log n) time because lower bounds proportional to n log n are known for many of them in the "decision tree with linear functions" model of computation.* The following theorem shows that spiral search is indeed a constant-time solution to the nearest neighbor searching problem.

> Theorem 1 - If n points are chosen independently from a uniform distribution over the unit square, then spiral search finds the nearest neighbor of a query point after examining only an expected constant number of the other points.

Proof - Certain notation is required in this proof. We will first define the concept of *layers* of cells surrounding the query point. We say that the cell containing the query point is in

layer 1, the eight cells surrounding that are in layer 2, the sixteen cells surrounding those are in layer 3, and so on. In general, for any $k \geq 1$, the k-th layer contains exactly $8(k-1)$ cells and there are $(2k-3)^2 \leq 4k^2$ cells on or within layer k. We will also use in our proof the constant

$$q = C/n.$$

Since the number of cells in the structure is n/C, q can be thought of as the probability of a point being placed in a certain fixed cell. We are now equipped to proceed to the statistical arguments required to prove Theorem 1.

Let p_{ij} be the probability that the first i cells probed by the search are empty and the (i+1)-st cell contains exactly j points. Simple combinatorial arguments show that

$$p_{ij} = \binom{n}{j} q^j [1 - (i+1)q]^{n-j}.$$

Since the (i+1)-st cell examined can be at most in the $k = (i^{1/2}+1)$-st layer, one need search a total of at most $4k^2 \leq (12i+4)$ cells, or 11i+3 beyond the original i+1 searched. The expected number of points in each cell beyond the original i+1 is $(n-j)/[1/q - (i+1)]$. Combining these counting arguments shows that the expected number of points examined, m, is bounded above by

$$m \leq \sum_{0 \leq i \leq t} \sum_{1 \leq j \leq n} p_{ij} \left[j + (11i+3) \frac{n-j}{1/q - (i+1)} \right]$$

where $t = n/C - 1$ (one less than the total number of cells). Rearranging this expression yields

$$m \leq \sum_{0 \leq i \leq t} \left[\sum_{1 \leq j \leq n} j p_{ij} + (11i+3)q \sum_{1 \leq j \leq n} (n-j) \frac{p_{ij}}{1-(i+1)q} \right]$$

We will now use the binomial theorem and other basic combinatorial identities to expand the two inner sums of the above expression, yielding

$$\sum_{1 \leq j \leq n} j p_{ij} = \sum_{1 \leq j \leq n} j \binom{n}{j} q^j [1 - (i+1)q]^{n-j}$$

$$= nq \sum_{0 \leq k \leq n-1} \binom{n-1}{k} q^k [1 - (i+1)q]^{n-1-k}$$

$$= nq \ (1 - iq)^{n-1}$$

and

$$\sum_{1\le j\le n} (n-j)\,\frac{p_{ij}}{1-(i+1)q} = \sum_{1\le j\le n} (n-j)\binom{n}{j} q^j\,[1-(i+1)q]^{n-1-j}$$

$$= n \sum_{1\le j\le n} \binom{n-1}{j} q^j\,[1-(i+1)q]^{n-1-j}$$

$$= n(1-iq)^{n-1} - n[1-(i+1)q]^{n-1}.$$

Substituting these sums back into the rearranged bound on m gives

$$m \le nq \sum_{0\le i\le t} \Big((1-iq)^{n-1} + (11i+3)\,[1-iq]^{n-1} - (11i+3)\,[1-(i+1)q]^{n-1} \Big)$$

$$= nq \sum_{0\le i\le t} \Big((11i+4)[1-iq]^{n-1} - (11i+3)\,[1-(i+1)q]^{n-1} \Big)$$

$$\le nq \sum_{0\le i\le t} (11i+4)\,(1-iq)^{n-1}$$

$$\le nq \sum_{0\le i\le t} (11i+4)\,e^{-i(n-1)q}$$

$$= C \sum_{0\le i\le t} (11i+4)\,e^{-C(1-1/n)i}$$

$$= O(1). \quad \square$$

Note that this proof is valid for any given point in the unit square. The programming of the spiral search, however, must behave properly when cells on the boundary of the unit square are being examined. This argument shows that the expected number of points examined by the spiral search is bounded above by a constant. Similar arguments can be used to show that the expected number of cells examined is also bounded above by a constant. Since those are the only two time-consuming operations in a spiral search, we have have shown that the expected running time of spiral search is bounded above by a constant, independent of the value of n.

Although the proof of Theorem 1 is rather tedious, the theorem itself can be easily understood on an intuitive level. Phrased very briefly, nearest neighbors are a local phenomenon, and so are cells. The following is a lengthier but more graphic illustration. Suppose you were standing in the middle of a large hall that is covered with tiles that are one-foot by one-foot square; suppose furthermore that the hall has been sprinkled uniformly with pennies, so that there are about a dozen pennies per tile, on the average. How many feet out will you have to look before you find the penny nearest you? Your answer will be independent of the size of the hall, because the density of pennies is the critical issue, and

not their absolute number--whether the hall is one hundred feet square or one mile square is immaterial. This is exactly the phenomenon we exploit in nearest neighbor searching by ensuring that there are a constant number of points per cell on the average.

We will now apply the cell method to a number of other closest-point problems. Although the formal proofs of the algorithms will all have the rather complicated structure of the proof of Theorem 1, the reasons why the algorithms perform efficiently all come back to the same principle: closest-point problems investigate local phenomena, and cells capture locality.

3. The Voronoi Diagram

The *Voronoi diagram* of a point set is a device that captures many of the closeness properties necessary for solving closest-point problems. For any point x in a set S the *Voronoi polygon* of x is defined to be the locus of all points that are nearer x than any other point in S. Notice that the Voronoi polygon of point x is a convex polygon with the property that any point lying in that polygon has x as its nearest neighbor. The union of the edges of all the Voronoi polygons in a set forms the Voronoi diagram of the set. A planar point set and its Voronoi diagram are illustrated in Figure 2. The Voronoi diagram has many fascinating properties that are quite useful computationally. We already mentioned the fact that the nearest neighbor to a new point is that point whose Voronoi polygon contains the new point. This fact can be used to give a fast worst-case algorithm for nearest neighbor searching. Another interesting property of the Voronoi diagram is the fact that the *dual* of the diagram (that is, the graph obtained be connecting all pairs of points that share an edge in their respective Voronoi polygons) is a supergraph of the minimum spanning tree of the set and, furthermore, the dual contains at most 3n - 6 edges. These and many other properties of the Voronoi diagram are discussed by Shamos [1978].

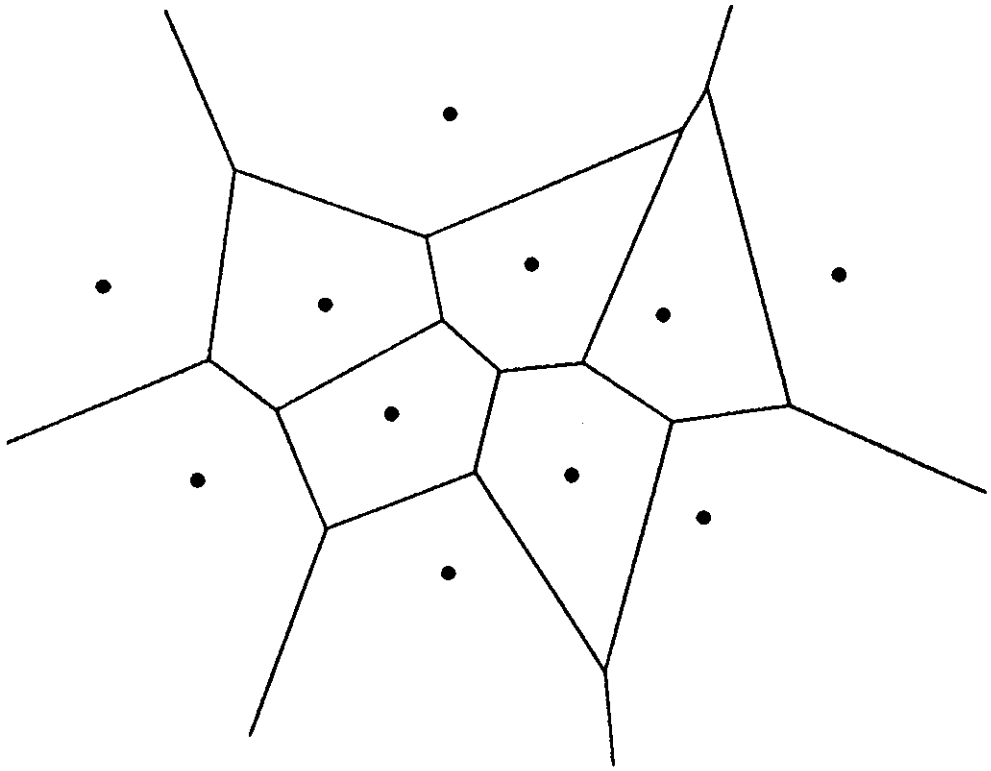

Figure 2. A point set and its Voronoi diagram.

The problem of constructing the Voronoi diagram of a planar point set is somewhat more delicate than nearest-neighbor searching. For each point, we will compute its Voronoi polygon by listing its edges together with the associated Voronoi neighbors in clockwise order. We will show that this can still be accomplished in linear expected time under the same assumption of point sets drawn from a bivariate uniform distribution. The basic idea is to search all cells in a relatively small neighborhood of each point in a spiral-like fashion until at least one point is found in each of the eight octants shown in Figure 3, or we give up having examined O(log n) cells. The *tentative* Voronoi polygon of the center point is that determined by considering just those eight points. Let d be the distance from the center point to the farthest point of its tentative Voronoi polygon. Then no point farther than 2d from the center point can have any affect on the actual Voronoi polygon of that point, which means that the Voronoi polygon of such a point can be constructed by considering only the few (expected constant) number of points which are in the circle of Figure 3.

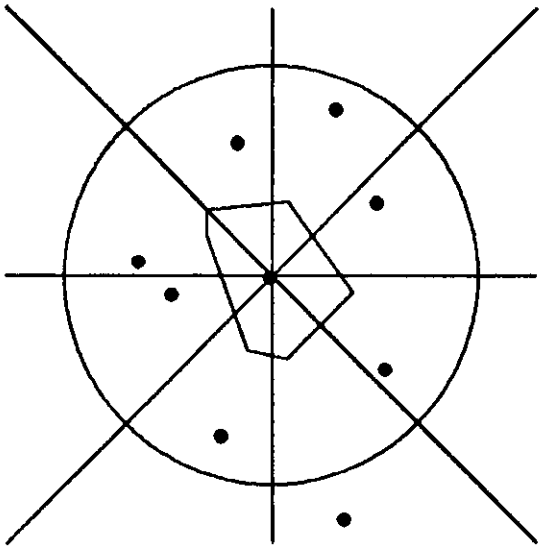

Figure 3. Construction of Voronoi polygon.

In the case that there is at least one point found in each octant before the O(log n) cells are searched, the point is called a *closed* point. A spiral search can be used to determine whether or not a given point is closed in constant expected time, and for a closed point its Voronoi polygon can then be computed in constant expected time. This can be proved by slightly modifying the proof of Theorem 1. The function p_{ij} in that proof remains the same; the only change is in the number of cells that need be searched if the point is found in the i-th cell. Performing the above operations on all points allows us to identify all closed points and compute their Voronoi polygons in linear expected time.

All points in the set that are not closed are called *open* points (note that most of these are near the boundary of the point set). Using methods similar to those of the proof of Theorem 1, it can be shown that the expected number of open points is $O((n \log n)^{1/2})$. Since each open point is identified in O(log n) steps, the total work required to identify all of the open points is $O(n^{1/2}\log^{3/2}n)$. Once the Voronoi polygons of the closed points are constructed and the open points are identified (all of which takes linear expected time) we are left with the problem of computing the Voronoi polygons of the open points. This is accomplished by applying the O(n log n) Voronoi diagram algorithm of Shamos and Hoey [1975] to the set of open points plus the set of closed points that are Voronoi neighbors of some open point. The expected size of this set is $O((n \log n)^{1/2})$, so the expected time required by the O(n log n) worst-case algorithm is $O(n^{1/2}\log^{3/2}n)$. This computes the Voronoi polygons for the open points, and completes our description of a linear expected-time algorithm for constructing the Voronoi diagram.

4. Extensions of the Planar Algorithms

The algorithms of Sections 2 and 3 can be used to solve a number of planar closest-point problems. Given the fast nearest-neighbor searching algorithm we can easily solve the *all nearest-neighbors* problem (which calls for finding the nearest neighbor of each point) in linear expected time, for point sets drawn from uniform distributions. This is accomplished by preprocessing the n points in linear time and then doing n searches, each of expected constant cost. Once we have found all nearest neighbors we can easily find the *closest pair* in the set by taking the minimum of the n distances. Shamos and Hoey [1975] have shown that once we have constructed the Voronoi diagram of a point set we can solve many other problems in linear worst-case time. Together with the Voronoi diagram algorithm of Section 3 this allows us to solve both the minimum spanning tree and *Delaunay triangulation* problems in linear expected time. The details of these algorithms, together with some of the applications areas in which they arise, are discussed by Shamos [1978].

All of the results that we have described so far are valid only for point sets drawn uniformly on the unit square; the algorithms can easily be adapted to work for many known distributions of points. The extension of these results to unknown distributions is a bit tricky. If we proceed for such a distribution as though it were uniform over some bounded region, a query can still be answered in constant expected time under certain conditions. The cells are chosen by first finding the minimum and maximum values in both x and y and then partitioning the rectangle defined by those four values into a number of squares proportional to n. The resulting cells can be represented by a two-dimensional array and our previous algorithms can operate as before. The only restriction on the underlying distribution required to achieve constant expected time is that it satisfy a condition similar to but more restrictive than a *Lipschitz condition*.

> Theorem 2 - Let n points be chosen independently from the distribution F(x,y) over a bounded convex region in the plane, where F satisfies the condition that there exist constants $0<C_1\leq C_2$ such that for any region of area A, the probability assigned to A by F lies between C_1A and C_2A. (Alternatively, F has a density with respect to Lebesgue measure that is bounded above and bounded below away from zero.) Then the same algorithm that was used for nearest neighbor searching in the uniform case answers a query in constant expected time.

Sketch of Proof - The proof of Theorem 1 can be easily modified to prove this theorem. The requirement that the distribution be over some bounded convex region of the plane ensures that some constant proportion of the cells will be used to contain points of the distribution, and the expected number of points per cell will therefore be bounded above and below by constants. The lower bound on density, C_1, guarantees that the expected number

of layers that need be examined before a point is found is small, and the upper bound C_2 guarantees that not many points will be in the neighboring cells when they are investigated. With these details in mind, the modification of Theorem 1 is straightforward. One can even use the same technique to prove a stronger version of Theorem 2--the requirement that the region be convex can be weakened to include non-convex regions with "sufficiently smooth" boundaries, where "sufficiently smooth" is given a precise technical meaning. ∎

Similar arguments show how the above methods can be applied to give linear expected-time algorithms for all of the closest-point problems mentioned above, when the point sets satisfy the conditions of Theorem 2.

5. Extensions to Higher Dimensions

In the previous section we showed how the algorithms of Section 2 and 3 can be used to solve a number of problems with inputs drawn from a wide variety of planar distributions; in this section we will see how the basic results can be extended to solve closest-point problems in k-dimensional space. If the point sets are drawn independently and uniformly from the unit hypercube (that is, $[0, 1]^k$), then we can use the cell technique by dividing the hypercube into n/C cells, each of side $(C/n)^{1/k}$.

The first closest-point problem in k-space that we will examine is that of nearest neighbor searching. Dobkin and Lipton [1976] showed that a nearest neighbor to a new point can be found in worst-case time proportional to k log n; their method requires preprocessing and storage prohibitive for any practical application, however. Friedman, Bentley and Finkel [1977] have described an algorithm with expected search time proportional to log n that has very modest preprocessing and storage costs. We will now examine a cell-based method for nearest neighbor searching that yields constant expected retrieval time. The preprocessing phase of the algorithm consists of placing the points into cells in k-space as described above. To perform a nearest neighbor search we generalize the spiral search of Section 2, starting at the cell holding the query point and searching outwards until we find a non-empty cell. At that point we must search all cells that intersect the ball centered at the query point with radius equal to the distance to the nearest neighbor found so far. Arguments similar to those used in Section 2 can be used to show that the expected work performed in this search is constant. Once we have this result we can solve both the all nearest neighbors and closest pair problems in linear expected time.

The k-dimensional minimum spanning tree problem calls for finding a spanning tree of the point set of minimum total edge length. Straightforward algorithms for this task require $O(n^2)$ time. A. Yao [1977] has shown that there is a subquadratic worst-case algorithm for solving

this problem, but his algorithm is probably slower than the straightforward method for most practical applications. Practical algorithms for this task have been proposed by Bentley and Friedman [1978] and Rohlf [1978], but the analysis of those algorithms remains primarily empirical. We will now investigate the use of the cell technique to solve this problem in fast expected time. We use the method of A. Yao [1977], which calls for finding the nearest neighbor of each of the n points in each of some critical number of generalized orthants. Yao has shown that the resulting graph is a supergraph of the minimum spanning tree of the point set. Since that graph contains a number of edges linear in n (for any fixed dimension k), the minimum spanning tree can be found in O(n log log n) time (see Yao [1975] or Cheriton and Tarjan [1976]). Because the "nearest neighbor in orthant" searching can be accomplished in constant expected time for each point, the total expected running time of this algorithm is O(n log log n).

It appears to be a very difficult task to use the cell method to construct k-dimensional Voronoi diagrams in fast expected time. Generalizing the method of Section 3 allows us to find the Voronoi polytopes of all closed points in linear expected time, but at that point there still remain $O((n \log n)^{1-1/k})$ open points. Since no fast algorithms are known for constructing Voronoi diagrams in k-space, it is not clear how to find the true Voronoi diagram. Notice, however, that we have found the Voronoi polytopes of an increasing fraction of the points (that is, the ratio of open points to n approaches zero as n grows). This technique can be used for "Voronoi polytope searching", which asks for the actual Voronoi polytope containing the query point--our method will succeed in constant time with probability approaching one.

The algorithms we have described above have all been for points drawn uniformly in $[0, 1]^k$. Methods analogous to those used in Section 4 can be used to show that the algorithms can be modified to work for point sets drawn from any distribution over some bounded convex region with density bounded above and away from zero.

6. Implementation Considerations

In this section we will discuss the implementation of the algorithms described in the preceeding sections. The algorithms in those sections share a common structure: in the first phase the points are stored in cells and in the second phase additional processing is done on the points. The implementation of the first phase is trivial. Points can be placed in cells by first finding the cell number (accomplished by a multiplication for scaling and a floor function to find the integer cell index) and then performing an array index. The difficulty of the second phase of processing will depend on the particular problem being solved. In the case of nearest neighbor searching all that is required is a "spiral search" and some distance

calculations; both of these are easy to implement. For the Voronoi diagram, however, the second phase of processing is very complicated. One advantage of the locality inherent in closest-point problems is that very slow algorithms may be used to perform the operations that take place in a local area; this will increase the constant of linearity, but will not slow the asymptotic running time of the algorithms.

It is important to mention one *caveat* that will be inherent to any application of the cell technique: the constant of linearity of most algorithms based on this method will increase *exponentially* with the dimension of the space, k. This is true simply because a cell in k-space has 3^k-1 neighbor cells. It seems, though, that this complexity might be inherent to any algorithm for solving closest-point problems because a point in a high-dimensional space can have many "close" neighbors. (More precisely, the maximum "adjacency" of point sets in k-space can be equated with the number of sphere touchings, which grows exponentially with k.) The practical outgrowth of this observation is that the methods we have described will prove impractical for large k; we estimate that this will happen somewhere for k between five and ten for data sets of less than ten thousand points. Data analysts observed this phenomenon long ago and refer to it as "the curse of dimensionality".

Weide [1978] has described how the empirical cumulative distribution function can be used to decrease the constants of the running times of programs based on cell techniques. We will now briefly discuss the application of his techniques to the case of planar nearest-neighbor searching. If the points to be stored for nearest-neighbor searching indeed come from a uniform distribution on $[0,1]^2$, then the cell technique performs very well. If the points come from a distribution that is not uniform (but still "smooth" enough to satisfy the requirements of Theorem 2), then the cells might perform poorly in the sense of being too large (in dense regions of the plane) or too small (in sparse regions). We would therefore like the cells to adapt their size in different regions of the space. One approximation to this "adaptive" behavior can be achieved with the cell method by incorporating a "conditioning pass" that examines the distribution before the points are placed in cells. This pass might work by finding the 10-th, 20-th, ..., 90-th percentile points in both the x and y marginal distributions. Each set of nine points partitions its dimension into ten "slabs", and the cross product partitions space into one hundred rectangles. Figure 4 illustrates such a partition of a heavily central distribution, such as a bivariate normal truncated at three standard deviations (where 6 points are sampled in each marginal, creating 49 rectangles). For most distributions satisfying the conditions of Theorem 2, the distribution of points within each rectangle will be much smoother than the distribution over the entire space. Because we sampled only a constant number of points in each dimension, we can locate which rectangle to examine in a nearest neighbor search in constant time. The exact number of rectangles to be used depends critically on the "roughness" of the underlying distribution--the smoother the

distribution, the fewer sample points required. These and other adaptation techniques are discussed in detail by Weide [1978].

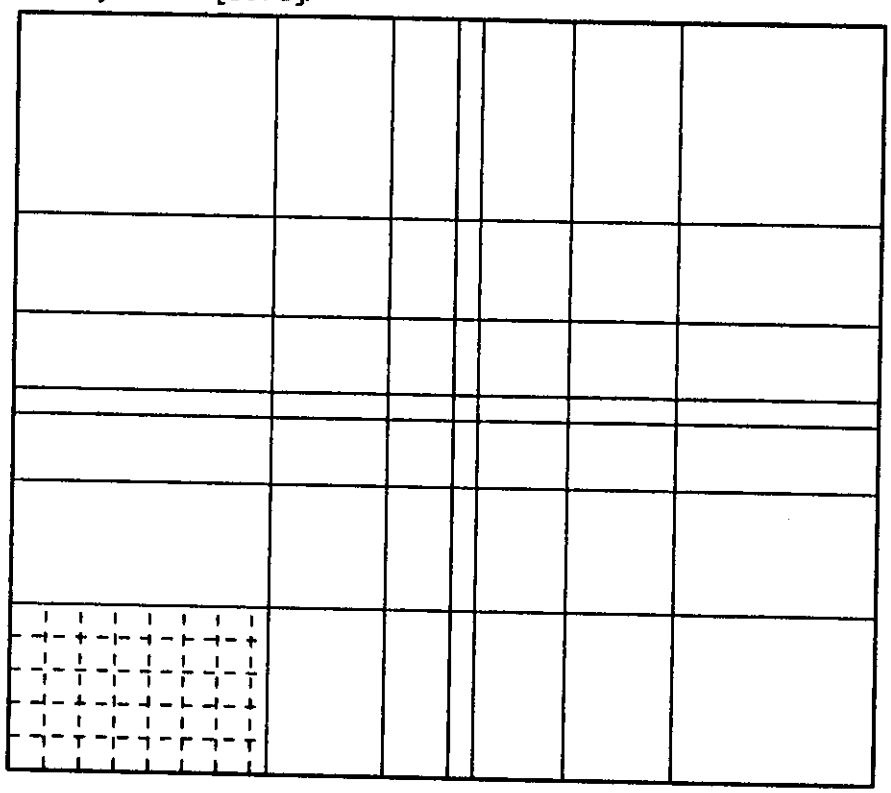

Figure 4. Adapting cell sizes by sampling marginals.

Although the searching structures that we have described in this paper are inherently *static*, they can be modified to become *dynamic*. We will first consider the case in which the nearest-neighbor structure is initially empty and then must support a series of Insert and Search operations. We will use a method to convert our cell structure from static to dynamic that is similar to a method described by Aho, Hopcroft and Ullman [1974, p. 113] for converting a static hash table into a dynamic one. The nearest neighbor structure is initially defined to have a maximum allowable size of (say) eight; we will call this size Max. When a new point is inserted into the structure, it is merely appended to the list of points currently in its cell. Whenever an insertion causes the number of points currently in the structure to exceed Max we perform the following operations: Max is set to twice its current value, a new structure of Max/C cells is created, and the points currently stored in the structure are "re-inserted" into the new structure. Note that for any distribution satisfying the conditions of Theorem 2, the expected number of points per cell is always bounded above and below by constants. Furthermore, analysis shows that the total amount of computation required to insert n elements into this structure is proportional to n. (Whenever a structure of size m is rebuilt, it is because m/2 points were inserted, so the "amortized" cost per point is constant; for a more formal analysis, see Aho, Hopcroft and Ullman [1974].) Monier [1978] has

described a related technique that allows a hash table to support both insertions and deletions intermixed with queries. We can use his idea to give dynamic structures for all of the searching problems discussed in this paper with the following properties: a sequence of n insertions and deletions can be performed in time proportional to n, at any point in this sequence it is possible to perform a search in constant expected time, and the storage used by the structure is always proportional to the number of elements currently stored.

In the above discussion we have described a number of rather exotic extensions to the basic structures of this paper. For many applications, however, the basic structure is all that is needed. We will therefore conclude this section on implementation by mentioning our experience in implementing the nearest neighbor searching algorithm for point sets drawn uniformly on $[0,1]^2$. The implementation in Fortran required approximately 35 lines of code to insert the points into the bins and 40 lines of code to accomplish the spiral search. The observed running times of the resulting routines were respectively linear and constant, as predicted, with very low overhead.

7. Conclusions

In this paper we have seen a number of algorithms for solving multidimensional closest-point problems. The algorithms were all based on the simple idea of cells, and were analyzed under the assumption that the points were drawn from some underlying "smooth" distribution. All of the searching methods we described have linear preprocessing costs and constant expected searching costs; all of the algorithms (with the exception of k-dimensional minimum spanning trees) have linear expected running time. It is clear that these algorithms achieve the trivial lower bounds and are therefore optimal. Although we have described the algorithms primarily as theoretical devices (sacrificing efficiency for ease of analysis), the discussion in Section 6 described how they can be efficiently implemented on a random access computer.

Much further work remains to be done in developing fast expected-time algorithms for closest-point problems. Can the expected complexity of computing minimum spanning trees in k-space be reduced from O(n log log n) to O(n)? A particularly important problem is to extend our results from bounded distributions to unbounded distributions (the multivariate normal, for example). It appears that new algorithms will have to be developed for this problem, taking special care of "outliers". Another very interesting open problem is to describe precisely how much of the efficiency of our algorithms is gained from probabilistic assumptions and how much is gained by use of the floor function. (The recent paper of Fortune and Hopcroft [1979] shows that floor can be used to speed up the computation of closest pair without making the randomization assumptions of Rabin [1976] and Weide

[1978].)

Acknowledgements

Helpful conversations with James B. Saxe and Professor Michael Ian Shamos are gratefully acknowledged.

References

Bentley, J. L. [1976]. Divide and conquer algorithms for closest point problems in multidimensional space, Ph.D. Thesis, University of North Carolina, Chapel Hill, North Carolina (December 1976).

Bentley, J. L. and J. H. Friedman [1978]. "Fast algorithms for constructing minimal spanning trees in coordinate spaces," *IEEE Transactions on Software Engineering C-27*, 2 (February 1978), pp. 97-105.

Cheriton, D. and R. E. Tarjan [1976]. "Finding minimum spanning trees," *SIAM J. Computing* 5, 4 (December 1976), pp. 724-742.

Dobkin, D. and R. J. Lipton [1976]. "Multidimensional searching problems," *SIAM J. Computing* 5, 2 (June 1976), pp. 181-186.

Fortune, S. and J. E. Hopcroft [1978]. "A note on Rabin's nearest-neighbor algorithm," *Information Processing Letters 8*, 1 (January 1979), pp. 20-23.

Friedman, J. H., J. L. Bentley, and R. A. Finkel [1977]. "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software 3*, 3 (September 1977), pp. 209-226.

Knuth, D. E. [1968]. *The Art of Computer Programming, Volume One: Fundamental Algorithms*, Addison-Wesley, Reading, Mass.

Lipton, R. J. and R. E. Tarjan [1977]. "Application of a planar separator theorem," *Eighteenth Symposium on the Foundations of Computer Science* (October 1977), IEEE, pp. 162-170.

Monier, L. [1978]. Personal communication of Louis Monier of the Universite de Paris-Sud to J. L. Bentley (June 1978).

Preparata, F. P. and Hong, S. J. [1977]. "Convex hull of finite sets of points in two and three dimensions," *CACM 20*, 2 (February 1977), pp. 87-93.

Rabin, M. O. [1976]. "Probabilistic algorithms," in *Algorithms and complexity: New directions and recent results*, pp. 21-39, (Ed. J. F. Traub), Academic Press, 1976.

Rohlf, F. J. [1978]. "A probabilistic minimum spanning tree algorithm," *Information Processing Letters 7*, 1 (January 1978), pp. 44-48.

Shamos, M. I. and D. Hoey [1975]. "Closest-point problems," *Sixteenth Symposium on the Foundations of Computer Science* (October 1975), IEEE, pp. 151-162.

Shamos, M. I. [1978]. Computational Geometry, Ph.D. Thesis, Yale University, New Haven, Connecticut (May 1978).

Weide, B. W. [1978]. Statistical Methods in Algorithm Design and Analysis, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania (August 1978). Appeared as CMU Computer Science Report CMU-CS-78-142.

Yao, A. C. [1975]. "An O(|E| log log |V|) algorithm for finding minimum spanning trees," *Information Processing Letters 4*, 1 (September 1975), pp. 21-23.

Yao, A. C. [1977]. On constructing minimum spanning trees in k-dimensional space and related problems, Stanford Computer Science Department Report STAN-CS-77-642 (December 1977).

Yuval, G. [1976]. "Finding nearest neighbors," *Information Processing Letters 5*, 3 (August 1976), pp. 63-65.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CMU-CS-79-111	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) OPTIMAL EXPECTED-TIME ALGORITHMS FOR CLOSEST-POINT PROBLEMS		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J.L. Bentley, B.W. Weide and A.C. Yao		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0370
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Department Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE March 1979 13. NUMBER OF PAGES 20
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as above		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)