Algorithms for Closest-Point Problems

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

by
Kenneth Lee Clarkson
October 1984

This research and/or preparation was supported in part by the National Science Foundation under grant MSC-83-08109.

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Andrew C. Yao

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Donald E. Knuth

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Ernst W. Mayr

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies & Research

ABSTRACT

This dissertation reports a variety of new algorithms for solving closest-point problems. The input to these algorithms is a set or sets of points in d-dimensional space, with an associated L_p metric. The problems considered are:

- **The all nearest neighbors problem.** For point set A, find the nearest neighbors in A of each point in A.
- **The nearest foreign neighbor problem.** For point sets A and B, find the closest point in B to each point in A.
- **The geometric minimum spanning tree problem.** For point set A, find the minimum spanning tree for the complete weighted undirected graph associated with A, where the vertices of the graph correspond to the points of A, and the weight of an edge is the distance between the points defining the edge.

These problems arise in routing, statistical classification, data compression, and other areas. Obvious algorithms for them require a running time quadratic in n, the number of points in the input. In many cases they can be solved with algorithms requiring $O(n \log^{O(1)} n)$ time.

In this work, approximation algorithms for some cases of these problems have been found. For example, for the minimum spanning tree problem with the L_1 metric, an algorithm has been devised that requires $O(n \log^d(1/\rho))$ time to find a spanning tree with weight within $1 + \rho$ of the minimum. Several other algorithms have been found with time bounds dependent on $\log(1/\rho)$ for attaining error ρ.

Algorithms have also been found that require linear expected time, for independent identically distributed random input points with a probability density function satisfying weak conditions. One such algorithm depends on the fact that under certain conditions, values that are identically distributed, but dependent, can be bucket sorted in linear expected time.

An algorithm has been found for the all nearest neighbors problem that requires $O(n \log n)$ expected time for any input set of points, where the expectation is on the random sampling performed by the algorithm. This algorithm involves the construction of a new data structure, a compressed form of digital trie.

CHAPTER 1.

INTRODUCTION

This dissertation presents a variety of algorithms for closest-point problems. In §1.1, several such problems, their applications, and some of the previous work on algorithms for them will be briefly described. The model of computation and complexity measures used are discussed in §1.2, and the specific results obtained are summarized in §1.3.

§1.1 Closest-Point Problems and Their Applications

1.1.1 The post office problem

The most fundamental closest-point problem is perhaps the *post office problem*:

> Given a set S of n points in d-dimensional space, organize them into a data structure so that given a point q, the closest point in S to q can be found quickly.

For example, if S corresponds to the locations of a set of cities with post offices, and you want to mail a letter to an address at site q, then the answer to this problem is the city to which the letter should probably be sent. For clarity, the points in S will be termed *sites*, and the point q will be termed a *query point*. While the measure of distance used is often the L_2 norm, or Euclidean distance, this is not always the case, and frequently different algorithms can be used for other L_p norms.

The post office problem has many applications, such as data base queries, interactive computer graphics, statistical classification, and data compression. It also arises as a subproblem in other applications.

There are many ways that the construction of a data structure for the sites, or *preprocessing*, can be done. The simplest kind of preprocessing might be to keep the sites in a list, and find the closest site to a query point by simply looking through all of the sites for the closest. In this case, the *preprocessing time* P_n is $O(n)$ (or just zero), and the *query time* Q_n is $O(n)$. Thus if k queries are performed, the

total time required to answer them is $P_n + kQ_n$, or $O(kn)$. When the number of queries is large, it is worthwhile to put more work into the preprocessing phase, to make the query time as short as possible.

For the planar ($d = 2$) case, algorithms are available for the post office problem requiring only $O(n \log n)$ preprocessing, for an $O(\log n)$ query time. Several algorithms with roughly this performance have been described, all based on the *Voronoi diagram* for the sites. The Voronoi diagram for a set of sites is a collection of regions, one for each site. (An example is shown in Figure 1.1.) The region for each site consists of those points that are closer to that site than to any other. For the planar case, Voronoi regions for several L_p metrics are bounded by straight line segments, and only $O(n)$ space is required to store descriptions of these segments in a useful form. Shamos [Sha] devised an $O(n \log n)$ algorithm for computing Voronoi diagrams for the planar Euclidean case. Brown [Bro] has used the relationship between planar Voronoi diagrams and three-dimensional convex hulls to devise new algorithms for such Voronoi diagrams. Lee and Wong [LW] have found $O(n \log n)$ algorithms for some other L_p norms.

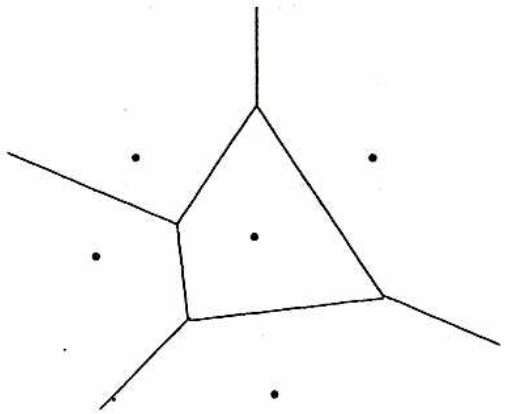

Figure 1.1. A Voronoi diagram for a set of sites.

When the Voronoi diagram for a set of sites is available, the post office queries for those sites may be solved by determining the Voronoi region containing the query point. Lipton and Tarjan [LT] and Kirkpatrick [Kir] have described asymptotically optimal, but complex, algorithms for this *point location* problem. Bilardi and Preparata [BP] have given an algorithm requiring linear expected space and $O(n \log n)$ space guaranteed in the worst case. Their approach stems from work due to Dobkin and Lipton [DoL] and Preparata [P]. Refining an approach due to Shamos [Sha2], and tuned by Lee and Preparata [LP], an asymptotically optimal algorithm has been found by Edelsbrunner *et al.* [EGS].

While Voronoi diagrams are quite helpful for the planar case, they are less so for $d \geq 3$. For these cases they require $\Omega(n^2)$ storage, and fast point location algorithms for high dimensional Voronoi diagrams are not known. However, Dobkin and Lipton

[DoL] have described a searching technique for the post office problem that requires $P_n = O(n^{2^{d+1}})$ preprocessing time and $O(\log n)$ query time.

1.1.2 The nearest foreign neighbor problem

When the number of post office queries is small, it is not worthwhile to spend a lot of time on preprocessing, since this time may be larger than the time spent on queries. For example, when there are n queries, spending more than $O(n^2)$ time on preprocessing is not worthwhile. However, Yao [Yao] has observed that it is still possible to obtain $o(n^2)$ time algorithms for this case, by using grouping: the sites are split into r blocks of size no more than $\lceil n/r \rceil$, and $P_{\lceil n/r \rceil, d}$ preprocessing time is used for each block. Answering a post office query then requires $rQ_{\lceil n/r \rceil}$ time, so the total time is $O(rP_{\lceil n/r \rceil} + nrQ_{\lceil n/r \rceil})$. By choosing r appropriately, this implies that the algorithm of Dobkin and Lipton can be used to obtain an $O(n^{2-1/2^{d+1}} \log^{1-1/2^{d+1}} n)$ algorithm for this limited version of the post office problem.

It may be that not only is the number of post office queries relatively small, but such queries may also be batched, that is, not answered until they are all received. This version of the post office problem is also known as the *nearest foreign neighbor* (NFN) problem. For many applications the post office problem may be solved in this off-line manner. Sometimes only the closest of the query points is desired, leading to the following problem:

> Given sets of sites (points) S and T in d-dimensional space, find the closest pair of sites $x \in S$ and $y \in T$.

We will call this the problem of finding an *NFN pair*.

Little previous work has been done on either the NFN or NFN pair problems, and Yao's grouping technique is the best method known for solving this problem exactly, for higher dimensions ($d > 2$).

1.1.3 Geometric minimum spanning trees

One problem closely related to those above is the geometric minimum spanning tree (MST) problem, which is:

> Given a set S of n sites (points) in d-dimensional space, find a minimum spanning tree for the complete undirected weighted graph G defined by these sites.

The graph G has n vertices, each vertex identified with a site, with the weight of an edge given by the distance between the two sites defining that edge. (See for example Tarjan's monograph [Tar2] for graph-theoretic terminology.)

This problem has many applications, including wire routing, statistical pattern classification, and heuristics for the traveling salesman problem. As with the post office and NFN problems, it is in fact a family of problems, depending on the dimension and on the distance measure used to determine the edge weights.

Two simple facts are crucial for developing algorithms that find minimum spanning trees:

Fact 1.1. For any subset V' of the vertex set V of a graph H, any minimum weight edge connecting a vertex in V' with one in $V \setminus V'$ appears in some minimum weight spanning tree of H.

Fact 1.2. For any maximal weight edge on a simple cycle of a graph H, there is a minimum spanning tree of H that does not contain that edge.

Proofs of these facts can be found in Tarjan's monograph ([Tar2], p. 71). Observe that the problem of determining a minimum weight edge between two sets of vertices is an instance of the NFN pair problem, for our geometrical version of the problem.

We will also need the following lemma, similar to Fact 1.2.

Fact 1.3. Suppose edge e connecting sites a and b is the heaviest edge on a simple cycle C of graph H, so that the weight of e is larger than the weight of any other edge on that cycle. Then e is not in any MST of H.

Suppose edge e' connecting sites c and d is on a simple cycle C' with e. Suppose also that e and e' have equal weight, and this value is larger than the weights of any other edges on C'. Then if T is a set of edges forming an MST of graph H, with $e \in T$, then $T \setminus \{e\} \cup \{e'\}$ is also an MST of H.

In other words, the uniquely heaviest edge in a simple cycle is not in any MST, and if there are two heaviest edges, only one may be in an MST, and either may be arbitrarily chosen.

Proof. To prove the first claim, suppose e is in a minimum spanning tree T. Then deleting e from T results in two trees T_a and T_b, as shown in Figure 1.2. The rest of C forms a path from a to b, and there is some edge f on C with one endpoint in T_a, and the other endpoint in T_b. If e is the uniquely heaviest edge, then f has less weight than e, and it may be used with T_a and T_b to form a tree with less weight than T. This would contradict the assumption that T is an MST. Therefore, e cannot be in any MST.

Similarly, if the second situation in the lemma holds, then the edge f must be in fact e', so that e may be removed and e' added to form another MST. ∎

Since the complete weighted graph G induced by a set of n sites has $\Theta(n^2)$ edges, the problem of finding an MST of G can be solved in $O(n^2)$ time by using any one of a number of available algorithms ([Tar2], Chapter 6). However, it is possible to improve upon this time bound by quickly finding a subgraph G' of G that is a supergraph of a minimum spanning tree of G. That is, G' has the same vertices

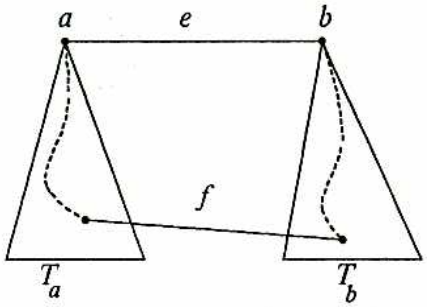

Figure 1.2. A minimum spanning tree T.

as G, but the edge set of G' is a subset of the edge set of G. Most algorithms for geometric MSTs find such an MST supergraph with $O(n)$ edges, and use $o(n^2)$ time. For the planar case, one way to use this approach is to compute the Voronoi diagram for the sites. An MST supergraph can then be found easily in linear time using information from this diagram. Another approach in the planar case is to determine the *relative neighborhood* graph [Sup]. Unfortunately, neither of these approaches is helpful for $d > 2$. However, Yao has found another approach to finding a sparse MST supergraph that does generalize to higher dimensions. This approach involves the concept of a *geographic neighbor* (GN) graph [Yao].

A geographic neighbor graph is specified by a family of cones F whose union is $\Re^d$, and with the apex of each cone at the origin. For each site $q \in S$, and each cone $C \in F$, put an edge in the GN graph from q to the closest site to q in $S \cap (C + q)$, that is, to the closest site contained in the translation of C to have apex at q. These edges form all the edges of a geographic neighbor graph. (Note that a GN graph has $O(n)$ edges.) We will call the problem of finding a GN graph the *nearest geographic neighbor* (NGN) problem. Yao has shown that if the cones in F are *narrow*, then the resulting GN graph will be a supergraph of an MST.

A cone $C \in F$ is narrow if

$$\max\{\|a\|_p, \|b\|_p\} > \|a - b\|_p,$$

for every two points a and b in C not at the origin 0. That is, in the triangle with vertices at a, b, and the origin, the edge ab is not the longest.

Suppose that edge e is the edge in the GN graph for a site q and cone C. Then any edge from site q to a site $s \in S \cap (C + q)$ is a longest edge in a 3-cycle of G containing e. By Fact 1.3 above, for any MST containing that edge, there is another with e instead. Thus only e need be included in an MST supergraph. Thus the GN graph is an MST supergraph.

For example, such a GN graph results for $d = 2$ when the cones used are the eight regions resulting from splitting the quadrants about the origin with the lines $y = x$ and $y = -x$ (See Figure 1.3).

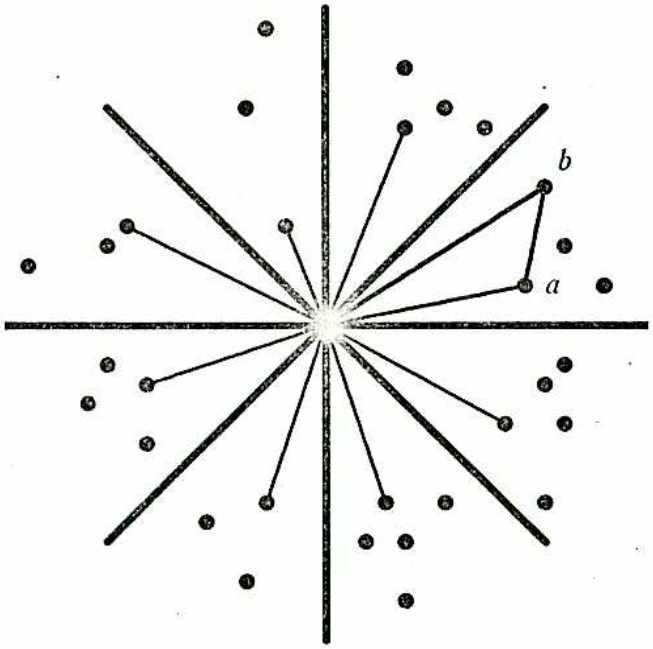

Figure 1.3. The regions for a GN graph, $d = 2$.

Using this reduction of the MST to the NGN problem, the grouping approach mentioned above, and the post office algorithm of Dobkin and Lipton, Yao has devised algorithms for the MST problem that require $O(n^{2-1/2^{d+1}} \log^{1-1/2^{d+1}} n)$ time [Yao]. These algorithms are applicable for the cases where the L_1, L_∞, and L_2 norms are used. Quite recently, Bentley, Gabow, and Tarjan [BGT] have found a way to compute an NGN graph for the L_1 case in $O(n \log^d n \log\log n)$ time, with a similar result for the L_∞ case.

Having found an MST supergraph with $O(n)$ edges, it remains to actually determine the MST. Fredman and Tarjan [FT] have recently found a remarkable algorithm for this task that requires $O(n \log^* n)$ time in the worst case. Alternatively, the edges may be sorted by weight, and Kruskal's algorithm ([Tar2], p. 73) used with the resulting sorted edge list. With the edges sorted, Kruskal's algorithm requires $O(n\alpha(m,n))$ time, where m is the number of edges, and $\alpha(m,n)$ is a very slow growing function, arising from the use of a union/find algorithm ([Tar2], Chapter 2). This approach has the advantage of the use of simple algorithms. Also, in some cases sorting the edges by weight may be done more quickly than in $O(n \log n)$ time.

1.1.4 The all nearest neighbors problem

Another batched form of the post office problem is the *all nearest neighbors* (ANN) problem:

> Given a set S of n sites in d-dimensional space, find the nearest neighbors in set S of each site in S.

The ANN problem has applications in statistical classification. In such applications, the sites correspond to objects to be classified, and the coordinates of the sites correspond to various qualities of those objects, measured quantitatively. The distance between two sites is thus a measure of the resemblance of the corresponding objects.

In hierarchic clustering, groups of sites that cluster together are sought, in an attempt to find groups of similar objects. One approach to such clustering, described by Murtaugh [Mur], is to find *reciprocal* neighbors, those pairs of sites that are nearest neighbors to each other. Each such pair is a cluster, as the two sites are close to each other and not to other sites. Such clusters may be merged, and represented by a single representative site. The clustering process is then repeated. Each phase of this algorithm requires the solution of the all nearest neighbors problem.

Conversely, results from the solution of the ANN problem have been used to show that a set of sites is random, and doesn't contain clusters. Hopkins [Hop] describes a test in which the distribution of distances from a random point to the closest sites to it is compared with the distribution of nearest neighbor distances among the sites. If these distributions match, then the sites may be said to be random.

The clusters that result from hierarchic clustering have some resemblance to a minimum spanning tree of the sites. Indeed, every pair of sites with one site a nearest neighbor of the other corresponds to an edge in a minimum spanning tree, using Fact 1.1. Thus algorithms for the ANN problem may be helpful for the MST problem.

Shamos [Sha] describes other applications of the ANN problem in geography, mathematical ecology, and molecular physics.

While algorithms for the post office problem may be applied to the ANN problem, as noted above many of these algorithms are only suitable for the case $d = 2$, while the applications mentioned require d to be large. Bentley [Ben] has found an $O(n \log^{d-1} n)$ time algorithm for solving the ANN problem, and Zolnowsky [Zol] has shown that an algorithm using a version of k-d trees requires $\Theta(n \log^d n)$ time to solve this problem.

§1.2 Computation Models and Performance Bounds

All of the results mentioned above concern the worst-case performance of algorithms that are exact, and use very simple primitive arithmetic operations. These three characteristics are typical of many, if not most, of the results in the algorithms literature. In recent years, interest has grown in computational models and performance bounds that differ from these. All of the new results described in this dissertation lack at least one of these characteristics. In this section, such issues will be discussed, along with the general nature of the new results.

While it is sometimes necessary to guarantee that an algorithm will run quickly, for all possible problem instances, more often it is sufficient to know that the algorithm is fast on the average, or that only in rare cases is the runtime excessive. In Chapter 4, algorithms are described that have nearly linear expected running times, for random input data. The sites are assumed to be independently identically distributed random variables, with a probability density function that is unknown, but satisfies certain weak conditions.

It is not always possible to know whether or not input data, even if random, satisfy assumptions regarding their distribution. However, if an algorithm itself performs randomization, then we know that expected-time performance estimates will be satisfied. In Chapter 3, such a randomized, or probabilistic, algorithm will be described for constructing a data structure that is then used to solve the all nearest neighbor problem.

The input values for the algorithms described here are known only to a limited precision, and the complexity of the algorithms depends on that precision. In Chapter 2, several approximation algorithms are described with complexities that depend on the bits of precision of accuracy desired. The result is that when site coordinate values are given in fixed point notation, the dependence of the running time of the algorithms on the total size of the input is relatively mild. However, when the the input coordinate values are given in, for example, floating point, this dependence may be excessive. Furthermore, on many machines most arithmetic operations are primitives taking effectively constant time for the input values in a practical range. In this context, counting the total number of operations performed is a reasonable measure of complexity, and this is done in Chapters 3 and 4. Furthermore, the floor, or least integer, function is assumed to require constant time, and is used in the algorithms in Chapter 4. The algorithms of Chapter 3 use also the logarithm and the bitwise exclusive-or functions, both assumed to require constant time. These functions are generally available on many machines; They may be avoided if numeric representations can be readily accessed.

§1.3 Summary of Results

The following is a summary of the specific results reported in this dissertation. The notation used is:

- S is a set of n points in d-dimensional space.
- T is a set of m points in d-dimensional space.
- σ is the problem scale, the ratio of the distance between the farthest pair of sites to the distance between the closest pair of sites.
- α is the absolute error in the answer returned, as a fraction of the distance between the farthest pair of sites.
- ρ is the relative error in the answer returned.
- $\alpha(j,k)$ is a *very* slowly growing function of j and k, appearing in the analysis of union/find algorithms ([Tar2], Chapter 2).

Unless otherwise indicated, the input to the algorithms is S. The asymptotic bounds are as $n \to \infty$. The constant factor depends on the dimension d, and in general will be exponential in d.

Chapter 2 is concerned with approximation algorithms:

- An algorithm is given for solving the all nearest neighbor problem for the Euclidean case in $O(n \log \sigma)$ time. This algorithm can be used to find all the sites at distance within α of the closest to each site, requiring $O(n \log \alpha)$ time to do so.

- An algorithm is given for solving the nearest foreign neighbor problem for S and T, with the L_1 norm used. The algorithm requires $O((n+m)\log^d \alpha)$ time for finding a site pair at distance within α of the closest pair.

- An algorithm is given for solving the minimum spanning tree problem for S, with the L_1 norm, in $O(n \log^{d-1} \rho \log \sigma)$ time. A spanning tree is found with weight within a factor of $1+\rho$ of the minimum possible.

- An approximation algorithm is given for the minimum spanning tree problem, for $d = 3$ and the Euclidean norm, that requires $O(n(\log n + \log \sigma)/\rho)$ time.

In Chapter 3, a probabilistic algorithm for the all nearest neighbors problem is described:

- An algorithm is given for building a cell-tree, a compressed form of digital trie, in $O(n \log n)$ probabilistic time. The logarithm, floor, and bitwise exclusive-or functions are assumed available at unit cost.

- An algorithm is given for solving the all nearest neighbors problem in $O(n)$ worst-case time, given a cell-tree for the input sites.

In Chapter 4, algorithms that are fast on the average are reported:

- An algorithm is given for building a cell-tree in $O(n)$ expected time, for independently identically distributed (IID) sites with a probability density function $f(x)$ that is $O(1/\|x\|^{d+1})$, as $\|x\| \to \infty$. The density function must be bounded, except for a finite number of poles, where $f(x) = O(1/\|x-p\|^{d-\beta})$, as $\|x-p\| \to 0$, for point p and some $\beta > 0$. The algorithm uses the floor function.
- An algorithm is given for finding a minimum spanning tree supergraph, for the L_1 norm and $d = 2$, in $O(n)$ expected time, when the cell-tree for the input sites is available.
- An algorithm is given for solving the minimum spanning tree problem exactly, for all L_p norms and dimensions d, in $O(n\alpha(m,n))$ expected time, for IID sites that are uniformly distributed in a d-cube. Here $m = O(n)$ is the number of edges in the MST supergraph found. The floor and exponential (e^x) functions are required.

CHAPTER 2.

APPROXIMATION ALGORITHMS

In this chapter several *approximation algorithms* for closest-point problems will be described. Several features are shared by these algorithms. For instance, they all require running times linear in the number of points, and these running times also depend on the logarithms of certain numerical parameters, such as the accuracy achieved, or the problem scale.

Algorithms with these general characteristics have of course been described before, but more commonly in the context of numerical analysis, and not for problems with a combinatorial nature. Radix-based sorting and digital tree searching methods ([Knu], §5.2.5, §6.3) are notable exceptions. The use of binary search in algorithms for cost-to-time ratio cycle network problems [Law] results in a logarithmic dependence on parameters of the input. Approximation algorithms requiring time polynomial in $1/\rho$ to achieve ρ relative error have been described for NP-complete problems ([GJ], §6.1-2). Brown [Bro] found algorithms for determining the diameter of a planar point set to relative error ρ in $O(n/\sqrt{\rho})$ and in $O(n + 1/\rho)$ time. Bentley, Faust, and Preparata [BFP] have described an $O(n + 1/\rho)$ algorithm for finding convex hulls.

The basic idea for the algorithms given here is like that for many iterative numerical procedures: Knowledge of the solution to some precision is used to determine a solution at double that precision. An algorithm for solving the original problem is created from one for solving an "improvement" problem. Unlike numerical procedures, however, a solution is not a value or a vector, but a combinatorial object, such as a set of nearest neighbors. This approach seems in several cases to result in relatively simple and even practical algorithms, as well as improvements in asymptotic running time.

Quite recently, Gabow [Gab] has described *scaling algorithms* for network optimization problems. These algorithms share many properties with those described here, and indeed Bentley, Gabow, and Tarjan [BGT] have applied this approach to closest-point problems, finding similar algorithms. Our algorithms are also similar in many ways to the "coarse to fine" algorithms of multiresolution image processing [Ros].

§2.1 The All Nearest Neighbor Problem

2.1.1 Quadtrees

We will begin with a description of the simplest of the approximation algorithms, that for the all nearest neighbors problem. First, some terminology will be given for *quadtrees*, which will be heavily used in the algorithms to follow. Quadtrees are geometric search trees that are analogs of digital search tries. Defining them more precisely:

U_d is the unit d-cube $[0,1)^d$, assumed to contain the sites.

The cells forming a quadtree are from the hierarchy defined as follows: U_d is in the hierarchy, and if a cell C is in the hierarchy, so are the 2^d equal-sized cubic cells obtained by cutting the cell in half in each coordinate. Such a cell D is a quadtree child of C if it contains at least one site. The set of quadtree children of C will be denoted $Children(C)$. A quadtree cell is a cube

$$\bigotimes_{1 \le i \le d} [a_i/2^k, (a_i+1)/2^k),$$

for some integers $k \ge 0$, $0 \le a_i < 2^k$, $1 \le i \le d$.

$Children(S)$ will be used to denote the set of children of a set of quadtree cells S. $Parent(S)$ is defined analogously.

A quadtree cell containing only one site will be termed a *singleton* cell.

The *quadtree* for a set of sites is the rooted tree with root U_d. A node C in the quadtree has $Children(C)$ as its children in that rooted tree. Generally, we will consider the descendents of the root only to the *singleton-cell size*, the largest size at which all cells are singletons. Assuming that the farthest pair distance for the sites is within a constant factor of the side length of U_d, such descendents are $O(\log \sigma)$ generations from the root.

Quadtrees were originally defined for the planar case, in which a node has as many as four children. For simplicity, the term will be used for the higher dimensional cases. Quadtrees have been used frequently in solving geometric problems. Samet ([Ros], p. 212) surveys a number of their previous uses. An example of a quadtree for a set of sites is shown in Figure 2.1.

2.1.2 Algorithm ANN_a

Algorithm ANN_a, for solving the all nearest neighbor problem, will now be described. In the general step, a set *Cells* of quadtree cells of a particular size is considered. For each $C \in Cells$, an upper bound $NN\_Distance(C)$ is known for the nearest neighbor distances to sites in *Cells*. That is, for any site s in C, the closest

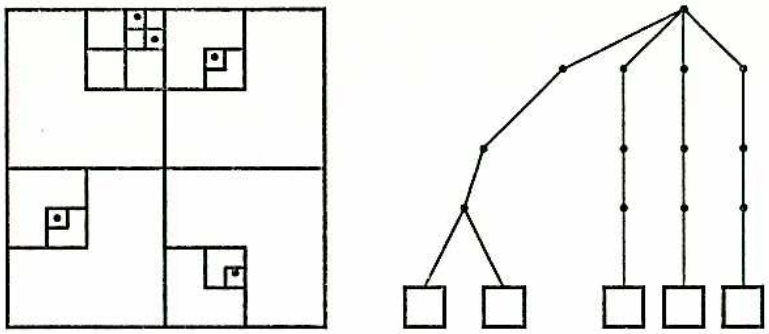

Figure 2.1. An example quadtree.

sites in *Cells* to s are no farther than $NN\_Distance(C)$. Also known for each C is $NN\_Set(C)$, the cells whose closest distance to C is no more than $NN\_Distance(C)$, and therefore may contain nearest neighbors for sites in C. Using this information, the general step determines the corresponding information for $Children(Cells)$. In determining $NN\_Set(C')$ for such a cell $C' \in Children(Cells)$, only the cells in $Children(NN\_Set(Parent(C')))$ need be examined.

For example, a set of three quadtree cells is shown in Figure 2.2 below. Cells A and B have some number of sites greater than 1, and singleton cell C has only one site. The nearest neighbor set for all three of these cells is $\{A, B\}$. (Note that the nearest neighbor set for A contains A, since A has more than one site, while C is not in its own nearest neighbor set.) In determining the nearest neighbor set for the children of these cells, therefore, only the children of A and B need be considered.

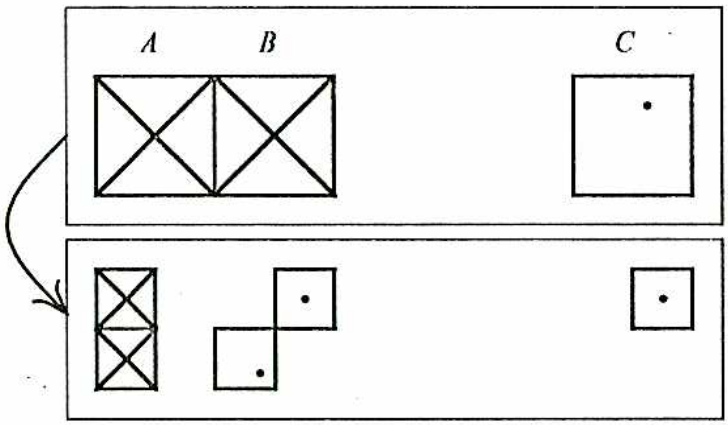

Figure 2.2. One refinement step of ANN_a.

Algorithm ANN_a may be viewed as an approximation scheme, with accuracy dependent on the smallest quadtree cell considered, and hence on the number of

```
procedure ANN_a(Sites : Set_of_Points);
co The number of sites is assumed to be greater than 1.
 The sites are assumed to be scaled to be contained in U_d. oc;
begin
 Contents(U_d) ← Sites;
 NN_Set(U_d) ← {U_d}; Cells ← {U_d};
 while not all cells in Cells are singletons do
  New_Cells ← Children(Cells);
  co The sites forming the contents of each child are determined at this step oc;
  for each C' ∈ New_Cells do
   C ← Parent(C');
   possible_NN_Set(C') ← Children(NN_Set(C));
   if |Contents(C')| = 1 then possible_NN_Set(C') ← possible_NN_Set(C') \ {C'} fi;
   NN_distance(C') ← min{d_max(D', C') | D' ∈ possible_NN_Set(C')};
   NN_Set(C') ← {D' | D' ∈ possible_NN_Set(C'), d_min(D', C') ≤ NN_distance(C')};
   Cells ← New_Cells;
  od;
 od;
end;
```

Figure 2.3. Algorithm ANN_a.

iterations of the general step. We will present it as an exact algorithm, in which quadtree cells are considered down to the size such that all cells of that size are singletons.

The algorithm is given in pseudocode in Figure 2.3. Some additional notation used:

> For a quadtree cell C, the value $Diameter(C)$ is the distance between the farthest pair of points on its boundary. For the L_2 norm, this is $\sqrt{d}$ times the side length of C.
>
> For quadtree cells A and B, the value $d_{\max}(A, B)$ is an estimate of the farthest distance between sites in A and in B. The value of $d_{\max}(A, A)$ will be defined to be $Diameter(A)$. When $A \neq B$, and each has more than one site, $d_{\max}(A, B)$ is defined as the farthest distance between the points on the boundaries of A and B. If A has one site and B has more than one, $d_{\max}(A, B)$ will be the distance of the site in A to the farthest point in B, which will be on the boundary of B. If both A and B are singleton cells, $d_{\max}(A, B)$ will be the distance between the sites in A and B.
>
> The function $d_{\min}(A, B)$ is defined analogously to $d_{\max}(A, B)$.

First, we show that ANN_a is correct:

> **Theorem 2.1.** Algorithm ANN_a returns all and only the nearest neighbors for each site in *Sites*.

Proof. The invariant established is that at the beginning of the main loop, any nearest neighbor to a site s in a cell $C \in Cells$ is in $NN\_Set(C)$. This implies that $possible\_NN\_Set(C')$ contains all such nearest neighbors, when $s \in C'$ and $C' \in Children(C)$. (If C' is a singleton cell, however, and contains only s, then C' itself does not contain a nearest neighbor to s.) The value $NN\_Distance(C')$ thus

provides an upper bound on the nearest neighbor distance to s, and the computation of $NN\_Set(C')$ preserves the invariant. Therefore, for a singleton cell C, the invariant implies that $NN\_Set(C)$ is a superset of the set of nearest neighbors of the site in C. Every cell in *New_Cells* is a singleton cell during the last execution of the body of the main loop, so that the last evaluations of $d_{\min}$ and $d_{\max}$ are performed between singleton cells. These evaluations return the true distances between the sites, so that only the nearest neighbors of a site s are contained in the *NN_Set* for its cell. Therefore, that *NN_Set* is the set of nearest neighbors of s. ∎

Theorem 2.2. Algorithm ANN_a requires $O(n \log \sigma)$ time.

Proof. Clearly the statements prior to the main loop require $O(n)$ time. Determining $Children(Cells)$ and the *Contents* of those children requires $\Theta(n)$ time, since each site must be examined. The time required in the inner loop is proportional to

$$\sum_{C' \in New\_Cells} |possible\_NN\_Set(C')| .$$

This is no more than 2^{2d} times the sum

$$\sum_{C \in Cells} |NN\_Set(C)| .$$

By Lemma 2.3 below, this quantity is linear in $|Cells|$. Since $|Cells| < n$, it follows that each iteration of the main loop requires $O(n)$ time. Furthermore, the number of steps is $O(\log \sigma)$: By appropriate scaling, the diameter of the set of sites is proportional to the side length of the root quadtree cell, and quadtree cells with diameter smaller than the closest distance between any pair of sites must be singleton cells. The total time required is therefore $O(n \log \sigma)$. ∎

Lemma 2.3. At each refinement step,

$$\sum_{C \in Cells} |NN\_Set(C)|$$

is linear in $|Cells|$.

The lemma follows from the fact that the number of cells for which a given cell is an approximate nearest neighbor is bounded by a constant dependent on the dimension. Roughly speaking, this is due to the fact that if a cell is a nearest neighbor of too many cells, those cells must be closer to each other than to the given cell. This observation simply extends to equal-sized cells the observation made by Bentley about points [Ben]. For example, in Figure 2.4 below, a cell C is shown together with sites for which it contains a nearest neighbor, and similarly a site p is shown. Associated with each of the neighbor sites is its *nearest neighbor circle*, the circle centered at the site with radius equal to its nearest neighbor distance upper bound. By definition, such a circle cannot contain other sites. Furthermore, each

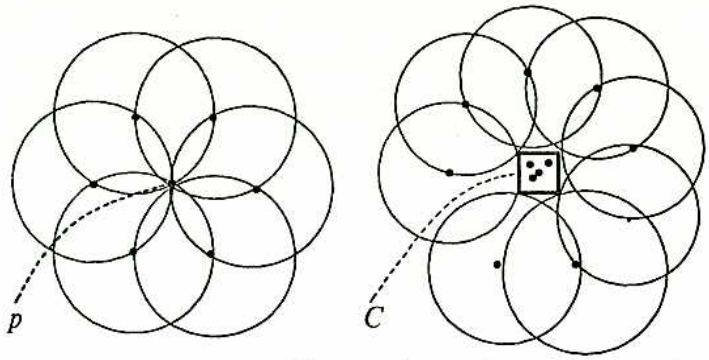

Figure 2.4. Nearest neighbor spheres for points and cells.

NN circle must at least touch C (or p). Only a limited number can do so without containing another site.

Proof of Lemma 2.3.

Let $NN\_Set^{-1}(C) = \{D \mid C \in NN\_Set(D)\}$. Clearly

$$\sum_{C \in Cells} |NN\_Set(C)| = \sum_{C \in Cells} \left|NN\_Set^{-1}(C)\right|.$$

We will bound the latter sum by bounding $\left|NN\_Set^{-1}(C)\right|$ for a given cell C. Let NN_{near} denote the cells D in $NN\_Set^{-1}(C)$ with $d_{\min}(C, D) \leq 2\,Diameter(C)$, and let NN_{far} denote $NN\_Set^{-1}(C) \setminus NN_{\text{near}}$. Then $|NN_{\text{near}}|$ is $O(1)$, with a constant dependent on the dimension, since the cells in NN_{near} have a diameter equal to $Diameter(C)$, and are confined to a region of volume proportional to $Diameter(C)^d$.

To bound the number of cells in $NN\_Set^{-1}(C)$, it remains only to bound the number of cells in NN_{far}. Note that any cell $D \in NN_{\text{far}}$ is a singleton cell: If $|Contents(D)| > 1$, then

$$NN\_Distance(D) \leq d_{\max}(D, D) = Diameter(D),$$

so C cannot be in $NN\_Set(D)$. We will denote the singleton cell containing a site s by D_s. Let $r = Diameter(C)/2$, and $R = 5\,Diameter(C)/2$. Choose a coordinate system in which the center of C is at the origin. Then for every $D_s \in NN_{\text{far}}$ we know that $\|s\| \geq R$. Also, for every $D_s, D_t \in NN_{\text{far}}$,

$$\|s\| - r \leq d_{\min}(C, D_s) \leq d_{\max}(C, D_s) \leq NN\_Distance(D_s) \leq \|s - t\|. \qquad (2.1)$$

Let NN'_{far} be the set of sites

$$\{s' \mid s' = s\frac{R}{\|s\|}, D_s \in NN_{\text{far}}\}.$$

Then $\|s'\| = R$ for any $s' \in NN'_{\text{far}}$. We will show, for every $s', t' \in NN'_{\text{far}}$, that $R - r = \|s'\| - r \leq \|s' - t'\|$. This implies that $\left|NN'_{\text{far}}\right|$ is bounded, since every s'

in NN'_{far} is the center of a ball of radius at least $R - r$ that contains no other site in NN'_{far}. Equivalently, every s' is the center of a ball of radius $(R - r)/2$ not intersecting any other such ball. Only a finite number of such balls can fit into the ball of radius $R + (R - r)/2$ centered at the origin.

To show that $\|s' - t'\| \geq R - r$ for all s', t' in NN'_{far}, we will assume otherwise and show that this contradicts (2.1). Now

$$\left\| s\frac{R}{\|s\|} - t\frac{R}{\|t\|} \right\| < R - r$$

implies that

$$\left\| s - t\frac{\|s\|}{\|t\|} \right\| < \|s\| - r\frac{\|s\|}{R} \leq \|s\| - r.$$

Without loss of generality, we can assume that $\|t\| \geq \|s\|$, so that

$$\left\| t - t\frac{\|s\|}{\|t\|} \right\| = \|t\| - \|s\|.$$

By the triangle inequality,

$$\|s - t\| < (\|s\| - r) + (\|t\| - \|s\|) = \|t\| - r.$$

This contradicts (2.1). Therefore,

$$|NN_{\text{far}}| = \left|NN'_{\text{far}}\right| = O(1),$$

so that $\left|NN\_Set^{-1}(C)\right| = O(1)$. Thus

$$\sum_{C \in Cells} |NN\_Set(C)| = \sum_{C \in Cells} \left|NN\_Set^{-1}(C)\right| = \sum_{C \in Cells} O(1) = O(|Cells|).$$

This is the fact desired. ∎

Note that this proof does not depend on the properties of the L_p metric used.

2.1.3 Further results and variations

Algorithm ANN_a may be modified in a few ways to provide possible improvements in running time. When the NN_Set of a singleton cell C consists of singleton cells, then the nearest neighbors of C are known exactly, and no further processing of $NN\_Set(C)$ need be done. Also, the definitions of $d_{\max}$ and $d_{\min}$ could be modified so that exact diameter and distance values are used when the number of sites in the cells considered is smaller than some constant value.

Recently Bentley, Gabow, and Tarjan [BGT] have independently devised an algorithm essentially the same as ANN_a. They were the first to show that this algorithm has an $O(n)$ time bound for all L_p metrics. Their proof of an analog of Lemma 2.3 involves the use of the concept of a narrow region, mentioned in §1.1.3. From the definition of a narrow region, a site s cannot be a nearest neighbor to more than one site in any narrow region associated with s. This observation, and the fact that a bounded number of narrow regions will cover $\Re^d$, implies the analog of Lemma 2.3.

Similar reasoning can be used to show that an $O(kn \log \sigma)$ algorithm can be devised for the all kth nearest neighbor problem. In this variation of the all nearest neighbors problem, all sites that are within kth closest to a given site are desired. The analog of Lemma 2.3 in this case is the fact that a site cannot be kth nearest neighbor to more than k sites that are in a given narrow region associated with the site. Suppose that there are $k+1$ sites in a narrow region for which this is true. Then the site farthest from s is closer to the other k sites than it is to s, a contradiction.

§2.2 The Nearest Foreign Neighbor Problem

In this section some approximation algorithms for the nearest foreign neighbor problem will be described. These algorithms are applicable when the L_1 distance measure is used. For this case, a restricted class of NFN problems is easy, and it will be shown that any L_1 NFN problem can be reduced, at least approximately, to a series of these easy problems. In §2.2.1, an algorithm for a planar case of the NFN pair problem is informally described. In §2.2.2, the same approach is used in an algorithm for the NFN problem in higher dimensions.

The algorithms described yield values for the nearest foreign neighbor distances that have α absolute error, relative to the problem scale. That is, if the sites fit in a cell of side length 1, then NFN distances will be returned that have an absolute error no more than α.

2.2.1 The planar NFN pair problem

To motivate the algorithm for NFNs to be described, we quote an observation by Guibas and Stolfi [GS]:

> **Lemma 2.4.** Let S and T be two sets of sites in the plane that are separated by a horizontal line and a vertical line. Then the L_1 closest site in S to any site in T is the one closest to the intersection of the two lines.

The situation is like that in Figure 2.5. The lemma follows from the fact that any Manhattan path between a site in S and a site in T can be replaced by a path of equal length passing through the intersection p of the separating lines. Such a path is shortest only if its sections from S to p and from p to T are shortest.

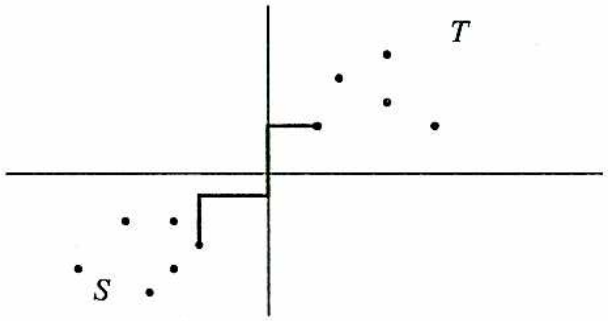

Figure 2.5. An easy L_1 NFN problem.

Using this lemma, therefore, the NFN problem for two sets of sites S and T in this configuration can be solved easily. The site in T that is closest to every site in S can be found in linear time, as can the closest site in S to those in T. These two sites form an NFN pair for the two sets.

For applications of NFN algorithms to finding MSTs, we will need to solve NFN problems for sets of sites in quadtree cells. For example, S may be contained in a quadtree cell A, with T in a quadtree cell B. When A and B are in the same generation of quadtree cells, and so are of equal size, their relationship in terms of Lemma 2.4 is quite simple. Either A and B (and so S and T) can be separated by a horizontal line, or else their boundaries are within the same range of y coordinates. A similar relationship holds true for the x coordinate, and in general:

> Two (equal-sized) quadtree cells A and B are *separated* in a given dimension x_j if there is a $x_j = a$ plane with A on one side and B on the other. If A and B are not separated in the x_j dimension, they are *aligned* in that dimension.

In this terminology, if two quadtree cells are separated in both dimensions, then by Lemma 2.4, the NFN problem may be solved for their sites in linear time.

On the other hand, the input sets S and T may be in quadtree cells that are aligned in some dimension, as in Figure 2.6 below. In this case, Lemma 2.4 does not immediately apply. However, suppose that the cells in the quadtrees rooted at A and B are preprocessed so that for each cell in those quadtrees, the site closest to each corner of that cell is known. An NFN pair can be found recursively as follows: If the two (equal-sized) cells that are input for finding an NFN pair are separated in both dimensions, then simply find the NFN pair in constant time. Otherwise, find the NFN pairs between all pairs of children of the input cells and return the minimum distance pair from among all of those. Some of these children will be separated in both dimensions, and their NFN pairs can be found immediately, as for the "dashed" pairs in Figure 2.6. The other pairs, which are aligned in the x coordinate, must have their NFN distances determined recursively.

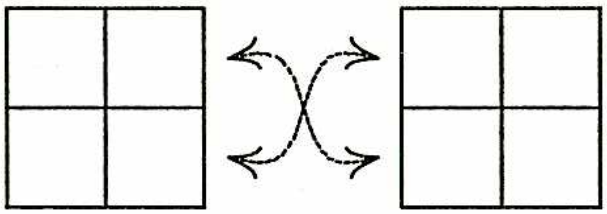

Figure 2.6. Diagonally located cells.

In an equivalent iterative procedure, at each iterative step pairs of cells of a given size that are separated in both dimensions are processed using Lemma 2.4. Some pairs of cells from A and B will be aligned, and indeed there will be rows of cells from A and B, with each cell from A aligned with all the cells in a row of B, and vice versa. At each iteration, when processing the children of the cells in such a row of A, only the children of the cells in the corresponding row of B need be examined. For any cell considered at a step, note that the closest site to its sites can be known exactly, among those sites not in its horizontal row. Furthermore, along such rows, all but the two cells closest to the corresponding opposite row can be eliminated from consideration, as these cells must have closer sites to those in the opposite cell. Thus the NFN pair distance is known at each step, with an accuracy within twice the side lengths of the current cells, and only cells containing a site with distance within that estimate need be considered further.

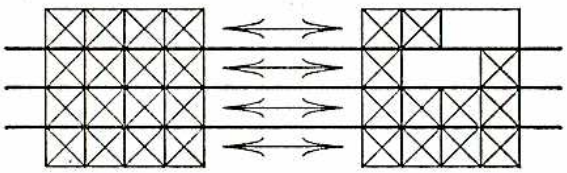

Figure 2.7. Aligned horizontal rows.

The time bound for this procedure depends on the number of iterations, which determines the side length α of the smallest cells processed. This side length is half of the accuracy α to which the NFN pair distance will be known. It is easy to see that this time bound is $O((j+k)\log(1/\alpha))$, for finding the NFN pair between cells with j and with k sites.

2.2.2 The general L_1 NFN problem

A generalization of Lemma 2.4 can be stated as:

> **Lemma 2.4′.** Given sets of sites S and T, and point $p = (c_1, c_2, \ldots, c_d)$, if the hyperplanes $x_1 = c_1$, $x_2 = c_2$, ..., $x_d = c_d$ all separate S and T, then a closest site $s \in S$ to p is a closest site to T. That is, s is as close to any site in T as is any site in S.

The proof of this is analogous to that for Lemma 2.4.