The idea here is that with probability at least ϕ_m, the problem of building a celltree is split into two problems, each for at least k_m and no more than $n - k_m$ objects. The smaller m is, the more evenly the problem is split up, and the smaller the m term in the time bound becomes. However, as m approaches 1, the lower bound available for $\hat{\phi}_m$ will approach 0. Therefore, the value of m must be chosen with care: We will show that $m \approx 2^d$ is about right.

Proof of Lemma 3.3. Split the sum bounding T_n by

$$T_n \leq \sum_{\substack{1<k<k_m \\ \text{or} \\ n-k_m<k<n}} \alpha_k[T_k + T_{n-k+1}] + \sum_{k_m \leq k \leq n-k_m} \alpha_k[T_k + T_{n-k+1}] + C_1 n.$$

We will use proof by induction, and assume the lemma holds for all $n' < n$. The left term is bounded by $(1 - \phi_m)C_2 n \ln n$, when n is large enough that this bounds the time required when *Build_Celltree_Deterministically* is called. To bound the right term, we use the fact that $a \ln a + (s - a + 1)\ln(s - a + 1)$ is decreasing in a for $a < (s + 1)/2$ and increasing in a for $a > (s + 1)/2$. This implies that for any k with $k_m \leq k \leq n - k_m$,

$$k \ln k + (n - k + 1)\ln(n - k + 1) \leq k_m \ln k_m + (n - k_m + 1)\ln(n - k_m + 1).$$

Therefore

$$T_n \leq C_1 n + (1 - \phi_m)C_2 n \ln n + C_2 \phi_m[k_m \ln k_m + (n - k_m + 1)\ln(n - k_m + 1)].$$

We want to show that the latter quantity is less than $C_2 n \ln n$. That is, we want to show that

$$n \ln n \geq \frac{C_1}{C_2 \phi} n + k_m \ln k_m + (n - k_m + 1)\ln(n - k_m + 1).$$

It will suffice to show that

$$n \ln n \geq \frac{C_1}{C_2 \hat{\phi}_m} n + k_m \ln k_m + (n - k_m + 1)\ln(n - k_m + 1),$$

or equivalently

$$n \ln n \geq \frac{C_1}{C_2 \hat{\phi}_m} n + k_m[\ln k_m - \ln(n - k_m + 1)] + (n + 1)\ln(n - k_m + 1).$$

Assuming that $\epsilon/m \leq 1/2$, the middle term can be dropped, so this will be true when

$$n \ln n \geq \frac{C_1}{C_2 \hat{\phi}_m} n + (n + 1)\ln(n - k_m + 1),$$

or equivalently

$$C_2 \ln\left(1 + \frac{k_m - 1}{n - k_m + 1}\right) - C_2 \frac{\ln(n - k_m + 1)}{n} \geq \frac{C_1}{\hat{\phi}_m}.$$

Since

$$\ln\left(1 + \frac{k_m - 1}{n - k_m + 1}\right) \geq \frac{k_m - 1}{n - k_m + 1}\left(1 - \frac{1}{2}\frac{k_m - 1}{n - k_m + 1}\right),$$

and since for $n \geq 16m/\epsilon \ln n$ with $\epsilon/m \leq 1/2$,

$$\frac{1}{4}\frac{\epsilon}{m} \leq \frac{k_m - 1}{n - k_m + 1} \leq \frac{1}{2},$$

it is sufficient to have

$$C_2 \geq 16 \frac{m}{\epsilon} \frac{C_1}{\hat{\phi}_m}$$

to bound T_n by $C_2 n \ln n$, for n large enough. ∎

We want to show that $\hat{\phi}_m$ is bounded away from zero. Define a *light* cell as follows:

> A quadtree cell is light if it contains less than ϵ/m of the input objects, and is a maximal such cell with respect to containment.

If all of the light cells each contain less than an ϵ proportion of the sample objects, then the splitting cell found cannot contain fewer than $n\epsilon/m$ objects. Suppose also that the minimal cell H, with more than a $1 - \epsilon/m$ proportion of the input objects, contains more than a $1 - \epsilon$ proportion of the sample objects. In this case the splitting cell found will also have less than $(1 - \epsilon/m)n$ objects. The probability of these events occurring is clearly positive and independent of n, and provides a positive lower bound for $\hat{\phi}_m$.

A more precise lower bound can be given for $\hat{\phi}_m$. The following simple lemma will be needed.

> **Lemma 3.4.** Given any two light cells, the probability that one of them will contain at least ϵ of the sample objects is increased by moving an object from the lighter to the heavier of the two (or either way if they have an equal number of objects).

Proof. Observe that if one light cell contains γn objects and another contains δn objects, with $\gamma > \delta$, then the chance $f(\gamma)$ that both will contain less than ϵ of the sample objects is decreasing in γ, for fixed $\gamma + \delta$. We can see this as follows. If the two cells contain a total of k sample objects, then

$$f(\gamma) = \sum_{k - \epsilon s < j < \epsilon s} \binom{k}{j} \left(\frac{\gamma}{\gamma + \delta}\right)^j \left(\frac{\delta}{\gamma + \delta}\right)^{k-j}.$$

(It can be assumed that $\lfloor \epsilon s \rfloor \leq k < 2\lfloor \epsilon s \rfloor - 1$, since otherwise the given probability is 1 or 0.) The derivative of this quantity with respect to γ, after removing a positive factor and rearranging, is

$$f'(\gamma) = \sum_{k-\epsilon s<j<\epsilon s} \binom{k-1}{j} k\gamma^{j-1}\delta^{k-j} - \binom{k-1}{j-1} k\gamma^{j}\delta^{k-1-j},$$

which telescopes to

$$\binom{k-1}{k-\lfloor \epsilon s \rfloor} k\gamma^{k-\lfloor \epsilon s \rfloor}\delta^{k-\lfloor \epsilon s \rfloor}\left(\delta^{2\lfloor \epsilon s \rfloor-1-k} - \gamma^{2\lfloor \epsilon s \rfloor-1-k}\right).$$

This quantity is negative, for $\gamma > \delta$, and further $f''(\gamma)$ is negative. The lemma follows. ∎

(Continuing the proof of Theorem 3.2) From Lemma 3.4, it follows that ϕ_m is minimized when all but at most one of the light cells contain $\lfloor \epsilon n/m \rfloor$ of the objects.

We can now prove a lower bound for $\hat{\phi}_m$. Note that $\hat{\phi}_m$ is at least as large as the probability that H contains all of the sample objects, times the probability that each of the light cells $L_1 \ldots L_v$ contained in H has less than ϵ of the sample objects. That is, letting h be the number of objects in H:

$$\hat{\phi}_m \geq (h/n)^s \sum_{0\leq k_i<\epsilon s} \binom{s}{k_1, k_2, \ldots, k_q} \lambda^{k_1}\lambda^{k_2}\ldots\lambda^{k_v},$$

where $\sum_{1\leq i\leq v} k_i = s$, and $\lambda = \lfloor \epsilon n/m \rfloor/h$. It follows that

$$\hat{\phi}_m \geq \sum_{0\leq k_i<\epsilon s} \binom{s}{k_1, k_2, \ldots, k_v} \left(\frac{1}{v}\right)^s,$$

since v, the number of light cells, is at most $\lceil n/\lfloor \epsilon n/m \rfloor \rceil$. Further,

$$\begin{aligned}
\hat{\phi}_m &\geq \frac{s!}{v^s} \sum_{0\leq k_i<\epsilon s} \frac{1}{k_1!k_2!\ldots k_v!} \\
&\geq \frac{s!}{v^s} \sum_{0\leq k_i<2} \frac{1}{k_1!k_2!\ldots k_v!} \\
&= \frac{s!}{v^s}\binom{v}{s} \\
&= \frac{v!}{(v-s)!v^s}.
\end{aligned}$$

By using Stirling's approximation and rearranging, this is greater than

$$\exp\left(\frac{-s^2}{2(v-s)}\left(1+O\left(\frac{1}{(v-s)}\right)\right)\right),$$

as $v - s \to \infty$. If v is chosen as $(s^2 + s)/2$, then $\hat{\phi}_m \geq e^{-1}(1 + O(1/s))$, as $s \to \infty$. Since $s \geq 2/\epsilon = 2(2^{d+1} + 1)$, this implies that with probability $\hat{\phi}_m$, the sample objects will be chosen in such a way that the splitting cell chosen will have at least $n\epsilon/m$ and no more than $n(1 - \epsilon/m)$ sites, where $\epsilon/m = \Omega(2^{-2d})$, as $d \to \infty$. By Lemma 3.3, therefore, T_n is $O(n \log n)$, with the constant factor no worse than exponentially dependent on the dimension d. ∎

§3.2 Using a Celltree

Having built a celltree, we want to use it to solve the all nearest neighbors problem. In this section, modifications to algorithm ANN_a are described that result in ANN_c, an algorithm that solves the all nearest neighbors problem in linear worst-case time, given the celltree for the input sites. This algorithm is described and shown correct in §3.2.1.

With a celltree available, each step of the refinement process of ANN_a can be performed in a time linear in the number of quadtree cells processed at that step. It is no longer necessary to examine every site to determine the cell it occupies. In §3.2.2, it is shown that indeed only $O(1)$ time is required for every quadtree cell that ANN_c examines.

However, a refinement step cannot simply use the celltree children of the current set of cells, rather than the quadtree children. If this were done, NN candidate sets would be maintained between cells that are of arbitrarily different sizes, and Lemma 2.3 could not be guaranteed. In ANN_c, it will be necessary in the course of a computation to process more quadtree cells than are in the input celltree. That is, the tree of cells examined by ANN_c will be less "branchy" than a celltree. On the other hand, as shown in §3.2.2, if there are k internal nodes (cells) at some level of the computation tree, then there will be $(1 + \beta)k$ nodes after a constant number of generations, for some $\beta > 0$. (By "internal nodes" is meant cells containing more than one site, that is, cells that are not leaves.) This fact implies that ANN_c examines only $O(n)$ quadtree cells, for input with n sites. These two features of the algorithm – the guaranteed linear size of the computation tree and the constant amount of work per internal cell – yield the linear time bound.

3.2.1 Algorithm ANN_c

As with procedure *Find_MST_Edges*, we will use the idea of a neighbor-connected component of equal-sized quadtree cells. Recall that two cells are neighbor-connected if they share at least one corner, and a set of cells is neighbor-connected when there is a path of neighbor-connected cells between any two cells in the set. Suppose that at a certain generation in the refinement process, with a set S of equal-sized quadtree cells, we determine the neighbor-connected components of S. Then any such component P is isolated from the rest of the cells, in the sense that every cell in P is at least one cell side length away from all the cells not in P. This isolation allows us to split up the ANN problem between neighbor-connected components, due to the following fact:

> **Lemma 3.5.** Let A and B be quadtree cells with the same side-length r. Let V denote those sites in A that have nearest neighbors in B. If $d_{\min}(A, B) \geq r$, then $|V| = O(1)$, with the constant depending on the dimension. Furthermore, a superset of V may be found in $O(1)$ time.

Proof. An example is shown in Figure 3.4. Suppose that C is a descendent cell of A with $Diameter(C) < r$. If C contains more than 1 site, then all of its sites have nearest neighbors closer than r, and therefore not in B. Let $Looking\_In(A, B)$ be the set of all leaf cells descendent from A with diameter no smaller than r. Then the set of sites in $Looking\_In(A, B)$ is a superset of V, contains $O(1)$ sites, and can be found in $O(1)$ time. ∎

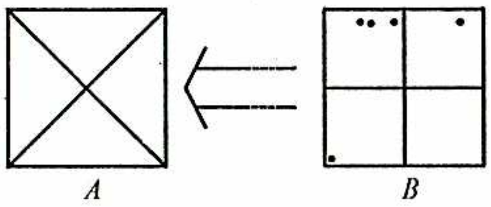

Figure 3.4. Distant cells form almost separate problems.

This lemma implies that

$$P \cup [\cup_{C \in P, D \notin P, D \in S} Looking\_In(D, C)]$$

is "self-contained," and only sites in that set have nearest neighbors in P. This observation forms the basis of ANN_c, shown in Figure 3.5. The algorithm is called with $\{T\}$ as input, for a celltree root T. It proceeds recursively on the components of quadtree children of the input component.

The basic processing of ANN_c is the same as that of ANN_a. However, the computations are put in terms of $NN\_Set^{-1}(C)$, for each cell C in a component. As before, this set of cells contains all sites that might have nearest neighbors in C. Also defined for each cell C is $NN\_Distance(C)$, an upper bound on the distance to the nearest neighbors of sites in C. It is assumed that when the algorithm is called for celltree root T, $NN\_Set^{-1}(T)$ has been set to $\{T\}$, and $NN\_Distance(T) = Diameter(T)$. (Here assuming that T contains more than one site.)

Just prior to a recursive call for a component, the procedure *Shrink* is called for it. Note that in ANN_c, the quadtree children of a cell are used, and not the celltree children. Clearly these quadtree children may be found in $O(1)$ time per cell, given

```
procedure $ANN_c(P : Set\_of\_Cells)$;
co assumes celltrees for cells in $P$ constructed oc;
co assumes $NN\_Sites(D)$ set to empty for all leaf cells $D$ oc;

begin
  Make neighbor-connected components $P_i$ of quadtree children of $P$;

  co Initialize $NN\_Set^{-1}$ for children. oc;
  for each $P_i$ do for $C \in P_i$ do
    $NN\_Set^{-1}(C) \leftarrow Children(NN\_Set^{-1}(Parent(C)))$;
    if $C$ is a leaf then $NN\_Set^{-1}(C) \leftarrow NN\_Set^{-1}(C) \setminus \{C\}$fi;
    if $NN\_Set^{-1}(C)$ is empty then $P_i \leftarrow P_i \setminus \{C\}$;
    $NN\_Distance(C) \leftarrow NN\_Distance(Parent(C))$;
  od;od;

  co Find sites in other components looking in to each cell. oc;
  for each $P_i$ do for $C \in P_i$ do for $D \in NN\_Set^{-1}(C)$ do
    if $D \in P_j$ and $i \neq j$ then $NN\_Set^{-1}(C) \leftarrow NN\_Set^{-1}(C) \cup Looking\_In(D, C) \setminus \{D\}$;
  od;od;od;

  co Update nearest neighbor distances. oc;
  for each $P_i$ do for $C \in P_i$ do for $D \in NN\_Set^{-1}(C)$ do
    $New\_NN\_Distance(D) \leftarrow \min\{d_{\max}(D, D') \mid D' \in NN\_Set^{-1}(C)\}$;
    $New\_NN\_Distance(D) \leftarrow \min\{NN\_Distance(D), New\_NN\_Distance(D), d_{\max}(C, D)\}$;
  od;od;od;

  co Reset $NN\_Site$ sets. oc;
  for each $P_i$ do for $C \in P_i$ do for $D \in NN\_Set^{-1}(C)$ do
    if $New\_NN\_Distance(D) < NN\_Distance(D)$ then
      $NN\_Distance(D) \leftarrow New\_NN\_Distance(D)$;
      if $D$ is a leaf then $NN\_Sites(D) \leftarrow \{\}$ fi;
    fi;
  od;od;od;

  co Prune $NN\_Set^{-1}$. oc;
  for each $P_i$ do for $C \in P_i$ do for $D \in NN\_Set^{-1}(C)$ do
    if $d_{\min}(C, D) > NN\_Distance(D)$ then $NN\_Set^{-1}(C) \leftarrow NN\_Set^{-1}(C) \setminus \{D\}$;
    else if $D$ is a leaf and $C$ is a leaf then
      $NN\_Set^{-1}(C) \leftarrow NN\_Set^{-1}(C) \setminus \{D\}$;
      $NN\_Sites(D) \leftarrow NN\_Sites(D) \cup \{C\}$
    fi fi;
  od;od;od;

  for each nonempty $P_i$ do $Shrink(P_i)$; $ANN_c(P_i)$; od;
end;
```

Figure 3.5. Algorithm ANN_c.

the celltree. Associated with each quadtree cell considered is a celltree cell within it. When a quadtree cell is strictly larger than its associated celltree cell, that quadtree cell has only one child. Thus a component P may have associated with it a set of celltree cells P' that are much smaller, resulting from nonbranching paths in the quadtree. Roughly speaking, the purpose of *Shrink* is to skip over these nonbranching paths when conveniently possible. Let K be the diameter of the largest cell in P'. Then the cells resulting from $Shrink(P)$ are those quadtree cells of diameter K that have the cells in P' as quadtree descendents. These cells can be substituted for those in P, preserving the information contained in the NN_Set^{-1} values. As a result of *Shrink*, the tree of cells touched by ANN_c has $O(n)$ nodes. This fact is proven in §3.2.2 below.

Lemma 3.6. Algorithm ANN_c is correct.

Proof. The following fact will always hold for each component P input to ANN_c:

For all $C \in P$, the set

$$NN\_Set^{-1}(C) \cup \{D \mid C \in NN\_Sites(D)\}$$

contains all sites that might have a nearest neighbor in C.

Here $NN\_Sites(D)$ denotes the leaf cells that are within $NN\_Distance(D)$ of D. The invariant is certainly true initially, when $P = \{T\}$. It must be shown that when it holds for a component, it holds when the component's child components are input. Let P' be such a child component, and C a cell in P'. We will show that before ANN_c is called for P', this invariant will hold for C. Because the invariant holds for P, the analogous fact holds for C when $NN\_Set^{-1}(C)$ is first defined. (This is certainly true for $Parent(C)$ not a leaf cell, hence not in any $NN\_Sites(D)$. If $Parent(C)$ is a leaf cell, we assume $C = Parent(C)$, in the sense that only the size parameters of the parent are adusted to determine C. Thus C is in some $NN\_Sites(D)$ if its parent is.)

We will show that a cell D is removed from $NN\_Set^{-1}(C)$ only when:

The cell $D = C$ and C is a leaf. A site is not its own nearest neighbor.

It is replaced by $Looking\_In(D, C)$. This preserves the invariant when P' is the input, by Lemma 3.5.

The value of $d_{\min}(C, D)$ is greater than $NN\_Distance(D)$. By its computation, $NN\_Distance(D)$ is always no smaller than the nearest neighbor distance for any site in D. The removal of D in this case therefore preserves the invariant.

D and C are leaf cells. In this case, C is added to $NN\_Sites(D)$, preserving the invariant.

The cell C is removed from $NN\_Sites(D)$ only when $NN\_Distance(D)$ is to be made smaller than a previous value: Note that D has $NN\_Sites(D)$ defined only when it is a leaf, and only C is added to $NN\_Sites(D)$ only when it is a leaf cell and $d_{\min}(D, E) \leq NN\_Distance(D)$. In this case, however, $d_{\min}(C, D)$ is the exact distance between the sites in C and in D. By previous computation, $NN\_Distance(D)$ is no more than that distance. Therefore, $d_{\min}(C, D)$ is equal to the current nearest neighbor estimate for D when C is added to $NN\_Sites(D)$. When C is deleted, the value of that estimate has improved, and it is known that C does not contain a nearest neighbor of the site in D.

The above invariant therefore holds for any $C \in P'$, and is always true for any component input.

Since eventually the cells E with $D \in NN\_Set^{-1}(E)$ will all be leaf cells, it follows inductively that after execution of ANN_c for a component P, the set $NN\_Sites(D)$ will contain all leaf cells E in P for which $d_{\min}(E,D) = NN\_Distance(D)$.

It also follows inductively that after execution of ANN_c for a component P, the value of $NN\_Distance(D)$ for any leaf cell D is no larger than the distance between the site in D and its nearest neighbors in P. Here D is any leaf cell in P or in $NN\_Set^{-1}(C)$ for some $C \in P$. Therefore, after execution of ANN_c for initial input $\{T\}$, the set $NN\_Sites(D)$ will contain exactly those sites at $NN\_Distance(D)$ to the site in D. This distance is the nearest neighbor distance to the site in D. Therefore, ANN_c is correct. ∎

3.2.2 Running time analysis of ANN_c

As discussed in the introduction to this section, to bound the running time of ANN_c we will first prove the following.

Lemma 3.7. Algorithm ANN_c requires $O(1)$ time for every quadtree cell it examines.

Proof. The main fact required here is that when ANN_c is called for a component P, the size of $NN\_Set^{-1}(C)$ is $O(1)$ for every $C \in P$. If this is true initially, it will be true for the initial value of the corresponding sets for the children of the cells in P. Let D be a cell in $NN\_Set^{-1}(C)$, for C a child of a cell in P. Then the value of $NN\_Distance(D)$ is computed as an upper bound on the distance of D to its nearest neighbors in $NN\_Set^{-1}(C)$. If the distance of D to C exceeds that upper bound, then D is removed from $NN\_Set^{-1}(C)$. Therefore, by Lemma 2.3, the size of $NN\_Set^{-1}(C)$ is bounded by a constant dependent on the dimension.

This fact allows an easy bound on the work performed for each internal node examined by the algorithm. The work done for each cell $C \in P$ is at most quadratic in $|NN\_Set^{-1}(C)|$, and so is $O(1)$. (The work of finding quadtree children, evaluating *Shrink*, and finding neighbor-connected components is clearly linear.) Each internal cell is examined in a call to ANN_c at most twice, as a child and then as a parent. Therefore, the work done for an internal cell, as a member of some neighbor-connected component, is $O(1)$.

On the other hand, a leaf cell may be examined many times. As noted, however, the work done for a leaf cell in some $NN\_Set^{-1}(E)$, for some internal node E, may be charged to that node. The other work done for a leaf cell must be bounded, however. If C in a component is a leaf cell, and D is a leaf cell in $NN\_Set^{-1}(C)$, then $O(1)$ work is done before C is placed in $NN\_Sites(D)$. (A constant amount of work may also be required when C is removed from $NN\_Sites(D)$.) Since leaf cell C is removed from a component if $NN\_Set^{-1}(C)$ is empty, it follows that a leaf cell is in a component at most twice, unless there is some internal cell E with $E \in NN\_Set^{-1}(C)$. In this case, the work done for C may be charged to E. Since internal cell E has $O(1)$ cells as nearest neighbors, the total charge to internal cells in this way is $O(1)$.

Thus the work charged to either leaf cells or internal cells is $O(1)$, and the lemma follows. ∎

To complete the proof of a linear time bound for ANN_c, we need to show that the use of *Shrink* implies that the number of cells examined is $O(n)$. This will be shown to follow from the fact that if a component P has k cells, then *Shrink*(P) has at least $k+1$ descendents. Thus each cell in P has a "fertility" of at least $1+1/k$. If k is small, this value will be acceptably large. We will show that if k is large, then k must have had fertile ancestors. First, a technical lemma:

Lemma 3.8. Let P be a set of k quadtree cells, all the same size, and let S be a set of sites, with exactly one site for each cell of P. Each site is constrained to lie somewhere within the boundaries of its cell. Let $f(k)$ be the minimum total L_1 length of a set of arcs connecting the sites such that the result represents a connected graph. Then $f(k) \geq \lceil k/2^d - 1 \rceil$, measured by the side length of a cell in P, for any such set of cells and sites.

Proof. The proof is by induction. The cases $k \leq 2^d$ are trivially true. To prove the result for $k > 2^d$, it will be shown that any connected graph for P can be disconnected into two connected graphs on two sets of sites, with a total path length savings of at least 1. This implies $f(k) \geq f(m) + f(n) + 1$, for $m + n = k$, which implies the desired bound.

Given a connected graph on some $k > 2^d$ sites, with exactly one site inside each cell of P, there must be two cells at least distance 1 apart, that is, separated by a cell side length. This implies a situation like that shown below.

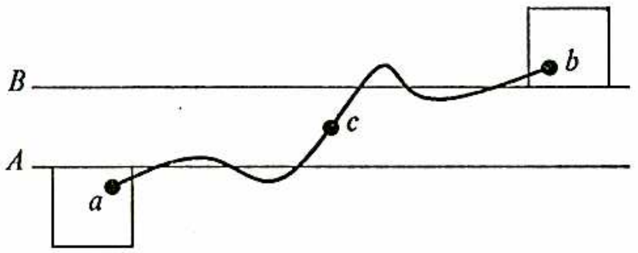

Figure 3.6. The inductive step for Lemma 3.8.

There must be a path connecting a and b, and that path must cross the region between hyperplanes A and B. Consider the portion of the path crossing this region. If there is no site on this portion, it can be deleted, saving at least 1 side length in distance.

If there is a site s along that portion, then do the following: Delete the arc connecting s to a's component, and project s to the B hyperplane. Also project all the sites in the path from s to the B hyperplane to that hyperplane. By so doing, a total of 1 side length in path length is saved, using the L_1 distance measure. Therefore, $f(k) \geq f(m) + f(n) + 1$, as desired. ∎

Lemma 3.9. Let P be a set of more than 2^d equal-sized quadtree cells. Let P' be the $2d$th quadtree descendents of P. If P' is neighbor-connected, then $|P'| \geq 2\,|P|$.

In other words, the ancestors of P' up to its $2d$th ancestors have average degree at least $2^{1/(2d)} > 1$. For example, if P has $2^d + 1$ cells, then two of those cells must be at least one side length away from each other. After $2d$ generations, with cells of side length only 2^{-2d} of that of the cells in P, there must be at least $2 \cdot 2^d + 1$ cells for those cells to be connected.

Proof. For any neighbor-connected set of descendents P', there is a set of sites and connecting arcs in P that is contained in P'. Since the total length of these arcs is $\lceil |P|/2^d - 1 \rceil$, and the maximum arc length contained in a cell of P' is $d/2^{2d}$, it follows that the number of cells in P' is at least $\lceil |P|/2^d - 1 \rceil 2^{2d}/d > 2\,|P|$. ∎

Note that the lemma does not say anything about what can happen when P has no more than 2^d cells. In this case, a group of 2^d neighbor-connected cells could be arranged around a single point in d-space, and even if they have only one child each, those children can still be connected. For the processing of ANN_c, however, the *Shrink* procedure guarantees the fertility of such small components.

Lemma 3.10. In the tree of cells examined by ANN_c, a connected component group P with k cells has at least $k(1 + 1/2^d)$ descendents after $2d + 1$ generations.

Proof. Let P' be a neighbor-connected group of $2d$th descendents of P. If $|P'| \leq 2^d$, then it will have $1 + 1/|P'| \geq 1 + 1/2^d$ children, yielding the desired number of descendents for its ancestors. If $|P'| > 2^d$ and P' has no more than 2^d ancestors, then the result follows. If P' has more than 2^d ancestors, then the fertility of those ancestors follows from Lemma 3.9. ∎

From Lemma 3.10 it follows that the tree of cells examined by ANN_c has $O(n)$ cells. Combining this fact and Lemmas 3.6 and 3.7, we get:

Theorem 3.11. Given the celltree for n sites, ANN_c can be used to solve the all nearest neighbors problem for those sites in $O(n)$ worst-case time.

Chapter 4.
Expected-Time Algorithms

§4.1 Introduction

In this chapter, algorithms will be described that are fast on the average, assuming that the input sites are random, that is, assuming that the input sites are independently, identically distributed random variables.

If the input sites are on a line (so that $d = 1$), then the ANN and NFN problems can be easily solved once the data are sorted. In this situation, it is possible to sort the data in linear expected time with bucketing methods that use the floor function. In such methods, addresses from 1 to n in random access memory are calculated for the input values by normalizing them to lie in $[1, n+1)$, and then truncating. The input values mapping to a given address, or "bucket," are then put in a list associated with that bucket, and these lists are sorted. If each bucket contains $O(1)$ input values, then the sorting operation requires linear time overall. When the data values are smoothly distributed in some bounded interval, this will be true on the average, and sorting takes linear expected time ([IS], [Knu], p. 105).

It may be, however, that the input data, although random, are not distributed smoothly, and a bucket contains more than $O(1)$ values on the average. For example, a discrete distribution that is nonzero only for a finite number of values results in buckets that are either empty, or contain $\Omega(n)$ values. Another bad case for bucketing occurs when the probability density function (PDF) for the input values has a "tail," and is nonzero on an unbounded region. In this case, normalization results in a large number of values in the small numbered buckets: Too much work is done for some large values, and not enough for small values.

Recently Devroye, and also Lueker ([Dev][Lue]), have shown that these difficulties can be at least partially overcome: By using a logarithmic transformation, the tail of an unbounded distribution can be mapped to a finite interval, allowing bucketing to work in $O(n)$ expected time when the PDF $f(x)$ is $O(1/x^{1+\alpha})$, for some $\alpha > 0$, as $x \rightarrow \infty$. Devroye has also shown that a finite number of poles may also be allowed and still have a linear time bound, where a pole is a value a such that $f(x) = O(1/(x-a)^{1-\beta})$, for some $\beta > 0$, as $x \to a$. (It is assumed that $f(x)$ is bounded elsewhere.)

In §4.2 below, it is shown that a celltree can be constructed in linear time, under conditions analogous to those of Devroye. By using algorithm ANN_c of §3.2, then, it is possible to solve the all nearest neighbors problem in linear expected time, under very broad conditions. In §4.3, it is shown that a MST supergraph can be found in linear worst case time for the planar L_1 case, when a celltree for the input sites is available.

A more direct use of bucketing methods for closest-point problems has been made by Bentley, Weide, and Yao [BWY]. In their work, sites are bucketed to equal-sized cells, and the technique of *spiral search* is employed to solve the ANN and NGN problems. In spiral searching, sites near a given site s_1 are found by looking at the cells near to that site, "spiraling" out from it so that the closest unexamined cell is examined next. If the sites are smoothly distributed, then $O(1)$ cells will need to be examined before a cell containing a site s_2 is found. Once this site is found, only $O(1)$ additional cells may contain a site closer than s_2 to s_1, resulting in a constant amount of additional work, since each cell will contain $O(1)$ sites on the average. This approach is heavily dependent on the smoothness of the distribution of the input, since not only should each cell have $O(1)$ sites on the average, but to limit the cost of spiraling out, each should have $\Omega(1)$ as well. Spiral searching has been guaranteed to take $O(n)$ time only when these conditions hold. On the other hand, this technique applies to any dimension and L_p norm.

When spiral searching is applied to the MST problem, the result is a linear-sized MST supergraph, from which the MST must be computed. Here bucket sorting provides a fast way to sort the GN graph edges prior to using Kruskal's algorithm. In §4.3, it is shown that linear expected time is required to bucket sort the edges resulting from a spiral searching algorithm, when the input sites are uniformly distributed. The difficulty here is that while the edge weights are random in this case, they are not independently distributed. It is therefore necessary to show that they are not very correlated, and will not cluster together in buckets.

§4.2 Building a Celltree in Linear Expected Time

In this section, we will prove the following theorem.

> **Theorem 4.1.** Suppose a set of sites is independently identically distributed. Further, suppose that the (unknown) probability density function $f(x)$ for the sites is $O(1/\|x\|^{d+1})$, as $\|x\| \to \infty$. Finally, suppose that the PDF has a finite number of poles, where for our purposes a pole is a point p and an associated neighborhood, with the property that $f(x) = O(1/\|x-p\|^{d-\beta})$, as $\|x-p\| \to 0$, for some $\beta > 0$. Then under these conditions a celltree for the sites may be constructed in linear expected time. It will be assumed that the floor and logarithm functions are available at unit cost, and the probabilistic algorithm of Chapter 3, for building a celltree, will be used.

To describe an algorithm with such a linear bound, a construction that is a "skeleton," or framework, for a celltree, will be indicated. This skeleton will have the following properties:

- ▶there are $O(n)$ leaves;
- ▶each leaf contains $O(1/n)$ probability mass, unless the leaf contains a pole;
- ▶the total mass contained in the tree is $1 - O(1/n)$;
- ▶it is possible in $O(1)$ time to find the leaf, if any, in the skeleton tree that contains a given site, using address calculation techniques requiring the floor function.

Using such a celltree skeleton, a celltree may be constructed in a two-phase process. First, each site is bucketed in $O(1)$ time to the skeleton leaf cell containing it. Next, a celltree is constructed for each leaf cell of the skeleton containing more than one site, using the probabilistic celltree construction procedure of the last chapter. Because of the properties above, the expected work for each skeleton leaf in building the celltree is $O(1)$, except for leaves near poles. For those leaves, the expected work is $O(n)$, as will be shown below.

To construct such a celltree skeleton, it will be convenient to assume for the moment that the PDF is dependent on the L_∞ distance to the origin, denoted r, and takes the form

$$f(r) = \begin{cases} 1/2^{d+1}, & r < 1; \\ 1/d(2r)^{d+1}, & r \geq 1. \end{cases}$$

By forming a celltree skeleton in which each leaf cell has $\Theta(1/n)$ probability mass for this distribution, it will guaranteed that there are $O(n)$ leaves and $O(1/n)$ probability mass per leaf cell when the PDF is $O(1/\|x\|^{d+1})$. Let $F(r)$ be the function indicating the probability mass between 0 and r for $f(r)$. Then $F(r)$ is simply $1 - 1/2r$, for $r \geq 1$, so that the mass outside of the region with $r \leq n/2$ is just $1/n$. Within this region, we consider each layer $n/2^{k+1} \leq r < n/2^k$, for integer $1 \leq k < \lfloor \lg n \rfloor$. Each such region contains mass $2^{k-1}/n$, and can be partitioned into

```
procedure Bucket_Celltree
begin
 for each site $s$ do
  $r \leftarrow \|s\|_\infty$;
  if $r \geq n/2$ then put $s$ into bucket 0
  else if $r \geq 2^{(\lg n) \bmod 1}$ then put $s$ into bucket $\lceil \lg(n/r) \rceil - 1$
  else put $s$ into bucket $\lfloor \lg n \rfloor$ fi fi;
 od;
 for Layer_Number $\leftarrow 1$ to $\lfloor \lg n \rfloor - 1$ do
  for each site $s$ in layer Layer_Number do find the cell of the $2^d(d+1)$ that contains $s$ od;
 od;
 for each large cell $C$ in each layer Layer_Number do
  $m \leftarrow \lfloor (Layer\_Number - d - 1)/d \rfloor$;
  Create a celltree with $C$ at root, with $m$ generations;
  Create a $d$-dimensional bucket array $B$, each index from 0 to $2^m - 1$;
  Traverse the celltree, mapping leaf cells to their buckets in $B$ and making a back pointer;
  for each site $s$ in $C$ do
   bucket $s$ into $B$, use backpointer to put $s$ in its leaf cell od;
  for each leaf cell of $C$ do
   build celltree for sites in that cell using probabilistic Build_Celltree od;
 od;
end;
```

Figure 4.1. Algorithm Bucket_Celltree

$2^d(2^d - 1)$ equal-sized hyper-cubes. Hence if each such cube is considered a celltree node, its children after no more than $\log_{2^d} 2^k/2^d(2^d-1)$ generations will have $\Theta(1/n)$ probability mass each. The cube of sites with $r \leq 2^{(\lg n) \bmod 1}$ should also be appropriately subdivided. The result is a celltree skeleton, and the verification of the four properties above is straightforward.

The resulting celltree construction algorithm is sketched in pseudocode in Figure 4.2. It is clear that $O(n)$ expected time is required for the algorithm, when no poles are present. When a pole is present, it is no longer true that each leaf of the skeleton tree contains $O(1/n)$ probability mass, and the work for some leaf cells will not be $O(1)$. However, we will show that if *Build_Celltree* is used to build a celltree within each leaf cell, then the time needed for the skeleton leaf cells around each pole will be $O(n)$ on the average.

To prove this time bound, it will be shown, for fixed $D > 0$, that expected $O(n)$ total work is done for skeleton leaf cells C that are L_∞ closer to a pole p than D. Let S be the collection of such cells in that neighborhood. Then we want to bound $\sum_{C \in S} E(T(|Contents(C)|))$, where $|Contents(C)|$ is the number of sites y that fall into cell C, and $T(|Contents(C)|)$ is $O(|Contents(C)| \log |Contents(C)|)$, the expected work done for $|Contents(C)|$ sites. Lueker [Lue] shows that the expectation can be pushed inside with $O(n)$ cost, so

$$\sum_{C \in S} E(T(|Contents(C)|)) = \sum_{C \in S} T(E(|Contents(C)|)) + O(n),$$

and we will bound the latter sum. Now divide the cells in S into layers, where the layer of cell C is $\min_{x \in C} \lfloor \|x - p\|_\infty / n^{1/d} \rfloor$. Note that p is fixed, so that as n

goes to infinity, the volume of every cell in S is $\Theta(1/n)$. Therefore, there will be $k = O(n^{1/d})$ layers, each with $O(k^{d-1})$ cells. The probability mass contained within L_∞ distance r of p is $O(r^\beta)$, and there are $O(1)$ cells in layers 0 and 1, so the work done for cells in those layers is

$$O\left(T\left(n\left(O(1)/n^{1/d}\right)^\beta\right)\right) = O(n^{1-\beta/d}\log n) = O(n).$$

Assume, without loss of generality, that the PDF $f(x)$ is monotonically decreasing in $\|x-p\|_\infty$, for sites $\|x-p\|_\infty < D$. Then the probability mass contained in a cell in layer j is $O(f(j/n^{1/d})/n)$, and the total work for cells in layers $j \geq 2$ is

$$\sum_{2\leq j\leq D'n^{1/d}} O(j^{d-1})T(nO(f(j/n^{1/d})/n)),$$

for some constant D'. This is

$$\begin{aligned}
&O\left(\sum_{2\leq j\leq D'n^{1/d}} j^{d-1}\left(\frac{j}{n^{1/d}}\right)^{\beta-d}\log\left(\frac{j}{n^{1/d}}\right)^{\beta-d}\right)\\
&= O\left(n^{1-1/d}\sum_{2\leq j\leq D'n^{1/d}}\left(\frac{j}{n^{1/d}}\right)^{\beta-1}\log\left(\frac{j}{n^{1/d}}\right)^{\beta-d}\right)\\
&= O\left(n^{1-1/d}n^{1/d}\int_{1/n^{1/d}}^{D'} z^{\beta-1}\log z^{\beta-d}dz\right)\\
&= O(n).
\end{aligned}$$

This completes the proof of Theorem 4.1. This proof follows that of Lueker of an analogous bound for sorting.

§4.3 Using Celltrees for MSTs

In this section it is shown that a celltree may be used to find an MST supergraph in $O(n)$ worst-case time, for the L_1 norm with $d = 2$.

The reduction in §2.3 from the MST to the NFN problem can be applied to this case. In that reduction, edges that will form an MST supergraph are found within each of the neighbor-connected components of the children of a neighbor-connected set of cells. Edges of the MST supergraph are then found between sites in different child components. With a celltree for the sites available, the *Shrink* processing of §3.2 can be applied to the components, so that by the reasoning of Lemma 3.10, only $O(n)$ quad-tree cells will be examined over the course of the reduction. Since a constant number of MST supergraph edges are found for each cell in a component, it follows that the MST supergraph found will have $O(n)$ edges.

It is also necessary, of course, to bound not only the number of edges generated, but also the total time solving NFN problems in finding those edges. In §4.3.1, a simple algorithm for solving the nearest foreign neighbor problem in the L_1 case is described, and an approach to bounding the total NFN solution time is indicated.

4.3.1 Finding NFN pairs using celltrees

As with the basic iterative NFN pair algorithm described in §2.2.1, we preprocess the celltree so that for each cell, the sites in the cell closest to each of the four corners of the cell are known. As with the basic algorithm, when applied to aligned cells A and B the celltree NFN algorithm first determines in constant time the NFN pairs for child cells that are diagonal from each other. The algorithm then determines the NFN pairs for those child cells of A and of B that are aligned. As these steps proceed, layers of pending cells will be maintained, for smaller and smaller cell sizes. The difference here is that the children of a cell in a celltree may be quite small relative to that cell, so that the pending layers will be of different sizes. In general a group of pending cells will be as in Figure 4.2, with one large cell limiting the amount by which the pending layer can narrow.

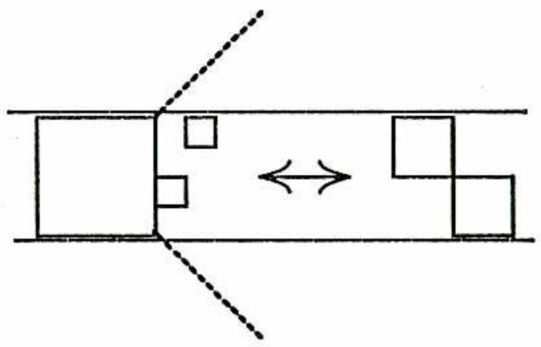

Figure 4.2. A row of pending cells.

However, only a constant amount of work is required to find the children of that cell and process the resulting diagonal pairs. This yields two sets of pending cells. Thus, the time $T(j,k)$ to compute the NFN is bounded by $T(a_1,b_1)+T(a_2,b_2)+O(1)$, where $j=a_1+a_2$ and $k=b_1+b_2$, hence is $O(j+k)$.

In order to show that the total time spent solving NFN pair problems is $O(n)$, the basic approach of §2.3.2 can be used, showing that the work per celltree node (not quadtree node) is constant. As in §2.3.2, some modifications will be necessary in the processing for finding the edges out of a component. To begin with, a simplification is possible: two equal-sized quadtree cells that are not adjacent need not have more than one MST edge between them. Therefore, the edge-cells of procedure *Find_MST_Edges* can be the cells of the child connected components.

Furthermore, any aligned row of cells in a child component need only have two edges out to neighboring components. For example, in Figure 4.3, only one edge between horizontally aligned H and H' is necessary in an MST supergraph. Suppose two edges, as shown, are necessary. Since the length of line segment $\overline{ce}$ is less than one (cell side length), and the length of $\overline{eb}$ is greater than one, the L_1 distance between a and c is less than that between a and b. Therefore, the edge between a and c is not longest in the 4-cycle shown, and by symmetry, the edge between b and d is not longest either. It follows that the longer of the two edges $\{a, b\}$ and $\{c, d\}$ is a longest edge in a 4-cycle. Therefore, only one MST supergraph edge is necessary between H and H', by Fact **1.3**.

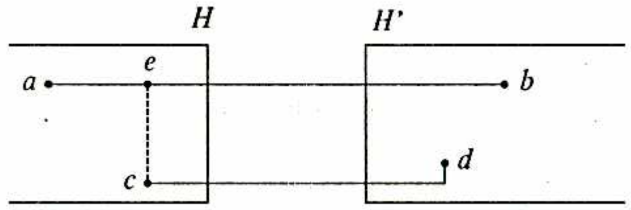

Figure 4.3. Only two edges are needed out of aligned rows.

It follows that when processing the edge-cell pairs required by *Find_MST_Edges* for a component, the cells in a child component may be organized into horizontally and vertically aligned groups, and two edges found out of each group. When processing H and H', the NFN algorithm could begin with the pending layer consisting of those two groups, and process them all jointly.

We need to show that these modifications eliminate the need for the pruning step of §2.3.2. That is, we must show that work done for each cell in the celltree is bounded by a constant. Consider the NFN algorithm step splitting one horizontal layer into two, as in Figure 4.2 shown above. Let the largest cell C in that layer be charged with the (constant) work of splitting the layer in two. No future calls in which C is examined will come from the other cells in the horizontal layer. Furthermore, no sites of the horizontal row containing A can be in the cone indicated with apex at C. If there were, such sites would be closer to the cells in C's layer than C is, and the current NFN estimate would exclude C from consideration. Also, NFN calls involving C cannot come from cells diagonal to A, as such calls can be answered in constant time. Therefore, as in §2.3.2, there is a region from C that cannot contain any points that will result in an NFN call examining C. Thus, the number of times that a cell can be charged over the course of the MST algorithm is bounded by a constant, by an argument similar to that suggested in §2.3.2.

§4.4 Finding MSTs Using Bucket Sorting

4.4.1 Sorting edge weights by bucketing

In the previous two sections, rather weak conditions on the PDF of the input sites allowed the construction of a celltree for those sites in linear expected time. In this section, we will make the much stronger assumption that the input sites are independently, uniformly distributed in the unit d-cube. As discussed in §4.1 above, under such conditions, an effective method of finding an MST supergraph is to find a geographic neighbor graph using spiral search. We will show that the resulting GN graph edges may be sorted by weight in linear expected time in this case, if the floor function is available. This will allow Kruskal's algorithm to find the MST from the GN graph in $O(n\alpha(m,n))$ time.

To simplify the analysis, we separately sort the edges resulting from each search direction (each cone defining the GN graph), and then merge the resulting lists in $O(n)$ time. To sort the edges for a cone C, put an edge with weight w into bucket number

$$k = \lceil n \exp\left(-w^d(n-1)V_C\right)\rceil.$$

Here V_C is the volume of the region consisting of those points of C that are closer than 1 to the origin. It will be shown that each bucket will contain $O(1)$ edges on the average, and that sorting the edges within each bucket will require $O(1)$ time, even if a quadratic-time sort is used. To see that the former statement is true, observe that all edges in bucket k will have $r_{k-1} > w \geq r_k$, where

$$r_k = \begin{cases} \left(\frac{\ln(n/k)}{(n-1)V_C}\right)^{1/d}, & k > 0; \\ \infty, & k = 0. \end{cases}$$

Therefore the probability that an edge for a site s falls into bucket k, so that the closest site to s is closer than r_{k-1} and farther than r_k, is

$$\text{Prob}\{\text{no site falls within } r_k\} - \text{Prob}\{\text{no site falls within } r_{k-1}\},$$

which is bounded by $(1 - V_C r_k^d)^{n-1} - (1 - V_C r_{k-1}^d)^{n-1}$, or

$$\left(e^{-(n-1)V_C r_k^d} - e^{-(n-1)V_C r_{k-1}^d}\right)\left(1 + O\left(\frac{\ln^2 n}{n}\right)\right),$$

which is

$$\left(\frac{k}{n} - \frac{k-1}{n}\right)\left(1 + O\left(\frac{\ln^2 n}{n}\right)\right) = \frac{1}{n}\left(1 + O\left(\frac{\ln^2 n}{n}\right)\right).$$

(Note that although the above applies only for $k > 1$, the $k = 1$ case follows similarly. Also, a site may be near a boundary of the containing cube, so that the corresponding region for bucket k is not contained in the cube. In this case, the above probability for an edge in bucket k is just an upper bound. Since less work is done for an edge that is not present, this condition will be ignored in the following.)

Thus $O(1)$ edges on the average will fall into bucket k, for each k. However, this does *not* imply that sorting the edge weights will require linear time, since the edge weight values are not independent random variables. In the next section, we show that despite this the sorting process takes linear time.

4.4.2 Analysis of the edge weight sort

As in §4.3.2, let y_k denote the number of edges falling into bucket k. The time required for sorting the edges by weight is $O(\sum_{1\leq k\leq n} y_k^2)$, and we want to show that for each k, the average sort time $\overline{y_k^2}$ is $O(1)$.

For fixed k and for site s_i, $1 \leq i \leq n$, let z_i be an indicator variable for bucket k. Let z_i take the value 1 if the edge weight for site s_i is in bucket k, and the value 0 otherwise. Then

$$\begin{aligned}\overline{y_k^2} &= \overline{\left(\sum_{1\leq i\leq n} z_i\right)^2}\\ &= \sum_{1\leq i\leq n} \overline{z_i^2} + \sum_{\substack{1\leq i,j\leq n\\ i\neq j}} \overline{z_i z_j}\\ &= n\phi + 2\binom{n}{2}\phi^*\\ &= O(1) + 2\binom{n}{2}\phi^*,\end{aligned}$$

where ϕ is the probability that $z_i = 1$, and ϕ^* is the probability that z_i and z_j are both 1. The former was shown to be $O(1/n)$ in the last subsection, but it remains to show that $\phi^* = O(1/n^2)$.

Before proving this, we need a bit of notation. Let C_i denote the translation of C to have apex at s_i. Let A_i denote the region in C_i with points closer than r_k to s_i. Let B_i denote the region in C_i of points farther than r_k and closer than r_{k-1}. Let C_j, A_j and B_j denote the corresponding regions for site s_j. Finally, let $F(R)$ denote the condition that region R contains a site, and $E(R)$ denote the condition that R does not contain a site. In this notation, for example, the chance that $z_i = 1$ is

$$\text{Prob}\{E(A_i)\}\text{Prob}\{F(B_i) \text{ given } E(A_i)\},$$

since this occurs when there are no sites in A_i and at least one site in B_i.

When proving $\phi^* = O(1/n^2)$, we may assume that site s_j is not in A_i, since then the edge for s_i cannot fall in bucket k. Also, the chance that s_j falls in B_i, with A_i empty, is less than the chance that any site does, and hence is $O(1/n)$. Since the chance that the edge weight for s_j falls in bucket k is in this case independent of where the edge for s_i falls, we have $\phi^* = O(1/n^2)$, if s_j falls into A_i or B_i. Similarly $\phi^* = O(1/n^2)$ if the edge for s_i falls into A_j or B_j.

To complete the proof that $\phi^* = O(1/n^2)$, we will show that $\phi_{j|i} = O(1/n)$, where $\phi_{j|i}$ is the probability that z_j is 1 given that z_i is 1. By the above reasoning, we can assume s_j is not in A_i or B_i, and vice versa.

To show that $\phi_{j|i} = O(1/n)$, we express it as a sum, based on the value of the distance D_{ij} between sites s_i and s_j:

$$\begin{aligned}\phi_{j|i} &= \mathrm{Prob}\{D \le D_{ij} < 2r_{k-1}\}\phi'_{j|i}\\ &\quad + \mathrm{Prob}\{D_{ij} \le D\}\phi''_{j|i}\\ &\quad + \mathrm{Prob}\{2r_{k-1} \le D_{ij}\}O(1/n)\\ &\le 2^d V_C r_{k-1}^d \phi'_{j|i} + C'D^d + O(1/n).\end{aligned}$$

Here D is a value depending on k that will be chosen to make the first two summands above $O(1/n)$, and C' is a constant dependent on d.

In order to make an appropriate choice for D, we must estimate the probability $\phi'_{j|i}$, which is the value of $\phi_{j|i}$ given that D_{ij} is between D and $2r_{k-1}$. If z_i is in fact 1, then the chance that $z_j = 1$ is increased, since it is guaranteed that $A_i \cap A_j$ contains no sites. We have

$$\phi'_{j|i} = \mathrm{Prob}\{E(A_j \setminus A_i)\}\mathrm{Prob}\{F(B_j) \text{ given } E(A_i \cup A_j), F(B_i)\}$$

which can be weakened to

$$\phi'_{j|i} \le \mathrm{Prob}\{E(A_j \setminus A_i)\}.$$

These probabilities are of course conditioned on $D \le D_{ij} \le 2r_{k-1}$. The chance that $z_j = 1$, without the condition that $z_i = 1$, can be expressed as

$$\mathrm{Prob}\{E(A_j \setminus A_i)\}\mathrm{Prob}\{E(A_i \cap A_j) \text{ given } E(A_j \setminus A_i)\}\mathrm{Prob}\{F(B_j) \text{ given } E(A_j)\}.$$

Since this value is $O(1/n)$, it follows that

$$\phi'_{j|i} \le \frac{O(1/n)}{\mathrm{Prob}\{E(A_i \cap A_j) \text{ given } E(A_j \setminus A_i)\}\mathrm{Prob}\{F(B_j) \text{ given } E(A_j)\}}.$$

Since $\mathrm{Prob}\{F(B_j) \text{ given } E(A_j)\} \ge \mathrm{Prob}\{F(B_j)\}$, we have

$$\phi'_{j|i} \le \frac{O(1/n)}{\mathrm{Prob}\{E(A_i \cap A_j) \text{ given } E(A_j \setminus A_i)\}\mathrm{Prob}\{F(B_j)\}}.$$

The probability that a site appears in region B_j is, for $k > 1$,

$$\begin{aligned}1 - \text{Prob}\{\text{no site appears in } B_j\} &= 1 - (1 - \text{Vol}(B_j))^{n-2} \\ &= 1 - \frac{k-1}{k}(1 + O(1/n)) \\ &= 1/k + O(1/n).\end{aligned}$$

Also, Prob$\{E(A_i \cap A_j)$ given $E(A_j \setminus A_i)\}$ is

$$\left(1 - \frac{\text{Vol}(A_i \cap A_j)}{(1 - \text{Vol}(A_j \setminus A_i))}\right)^{n-2} = \exp(-n\text{Vol}(A_i \cap A_j))(1 + O(\ln n/n)).$$

In §4.4.3 below, the following is proven.

Lemma 4.2. For sites s_i and s_j with $D_{ij} \geq D$,

$$\text{Vol}(A_i \cap A_j) \leq V_C r_k^d(1 - C_1 D/r_k + C_2(D/r_k)^2),$$

for appropriate positive constants C_1 and C_2.

Therefore, letting $R = D/r_k$,

$$\begin{aligned}\phi'_{j|i} &\leq \frac{1/n}{[\exp(-n\text{Vol}(A_i \cap A_j))(1 + O(\ln n/n))][1/k(1 + O(1/n))]} \\ &\leq \frac{k}{n}\exp(nV_C r_k^d(1 - C_1 R + C_2 R^2))(1 + O(\ln n/n)) \\ &= \exp(-C_1 R \ln(n/k) + C_2 R^2 \ln(n/k))(1 + O(\ln n/n)),\end{aligned}$$

so that

$$\begin{aligned}r_{k-1}^d \phi'_{j|i} &= \frac{\ln n/k}{nV_C}\exp(-C_1 R \ln(n/k) + C_2 R^2 \ln(n/k))(1 + O(\ln n/n)) \\ &= O(\frac{1}{n}\exp(\ln\ln(n/k) - C_1 R \ln(n/k) + C_2 R^2 \ln(n/k))(1 + O(\ln n/n)).\end{aligned}$$

Setting $D = 1/n^{1/d}$, so that $R = \Omega(1/(\ln(n/k))^{1/d})$, the above expression becomes $O(1/n)$, as does D^d. Therefore $\phi_{j|i}$ is bounded by $O(1/n)$, implying that each $\overline{y_k^2}$ is $O(1)$, so that the described sorting algorithm is indeed linear.

4.4.3 Proof of Lemma 4.2.

In this section, we will prove Lemma 4.2 concerning the volume of $A_i \cap A_j$. A few assumptions can be made concerning these cones: Each $\mathcal{C} \in F$ is convex, and has nonzero volume. Choose a coordinate system for proving the lemma for cone $\mathcal{C}$ such that site s_i is the origin, so A_i will be denoted simply A. Also choose the coordinate system so that one coordinate, denoted z, has a unit basis vector e_z with $e_z \in \mathcal{C}$. By symmetry, we may assume that site s_j has a positive z coordinate, and has the form $s_j = \alpha e_j + z_j e_z$, where e_j is a unit vector in the $z = 0$ hyperplane.

For region $\mathcal{E} \subset \Re^d$, and $t \in \Re$, let $t\mathcal{E} = \{tx \mid x \in \mathcal{E}\}$. For $p \in \Re^d$, let $\mathcal{E} + p = \{x + p \mid x \in \mathcal{E}\}$. Now since $\mathcal{C}$ is a convex cone, it may be represented as

$$\mathcal{C} = \bigcup_{t \geq 0} t(\mathcal{S} + te_z),$$

where $\mathcal{S}$ is a convex region in the $z = 0$ hyperplane. Let

$$t_{\max} = \max\{t \mid t(\mathcal{S} + te_z) \subset A\}.$$

Let $A' = \bigcup_{0 \leq t \leq t_{\max}} t(\mathcal{S} + te_z)$. Instead of bounding $A \cap A_j$, we will bound $A' \cap A'_j$, where A'_j denotes $A' + s_j$. The following lemma suggests that this is reasonable.

Lemma 4.3. There is a constant λ such that $\mathrm{Vol}(A') = \lambda \mathrm{Vol}(A)$.

Proof. Let $B_r = \{x \mid \|x\|_2 \leq r\}$. Then $A = B_{r_k} \cap \mathcal{C}$, but $B_{r_k} = r_k B_1$, and $r_k \mathcal{C} = \mathcal{C}$, so $A = r_k(B_1 \cap \mathcal{C})$. Therefore $t_{\max}$ is linearly proportional to r_k, so that

$$\mathrm{Vol}(A') = \int_0^{t_{\max}} \mathrm{Vol}(t\mathcal{S})dt = t_{\max}^d \mathrm{Vol}(\mathcal{S})/d = C_1 r_k^d \mathrm{Vol}(\mathcal{S}),$$

where C_1 is some constant. (Here $\mathrm{Vol}(t\mathcal{S})$ means volume of $\mathcal{S}$ in the $z = 0$ hyperplane.) Furthermore $\mathrm{Vol}(A) = r_k^d V_{\mathcal{C}}$, so the lemma follows. ∎

Lemma 4.4. $\mathrm{Vol}(A' \cap A'_j)$ is maximized for $z_j = 0$, that is, with s_j in the $z = 0$ hyperplane.

Proof. Note that $A'_j = \cup_{0 \leq t \leq t_{\max}} t(\mathcal{S} + \alpha e_j + (t + z_j)e_z)$. Furthermore, since $\mathcal{S}$ is convex and the origin 0 is in $\mathcal{S}$, it follows that $a\mathcal{S} \subset b\mathcal{S}$ for $a < b$. Therefore

$$(t - z_j)(\mathcal{S} + \alpha e_j + te_z) \subset t(\mathcal{S} + \alpha e_j + te_z)$$

for $t \geq z_j$. Using this fact, and the fact that

$$A' \cap A'_j = \bigcup_{z_j \leq t \leq t_{\max}} t(\mathcal{S} + te_z) \cap (t - z_j)(\mathcal{S} + \alpha e_j + te_z),$$

the lemma follows. ∎

In the following, we will assume $z_j = 0$, so s_j has the form $s_j = \alpha e_j$. Let $\mathcal{S}_{j,\alpha} = \mathcal{S} \cap (\mathcal{S} + \alpha e_j)$. Now $\mathrm{Vol}(A' \cap A'_j) = \int_0^{t_{\max}} \mathrm{Vol}(t\mathcal{S}_{j,\alpha})dt$, so we want to bound the volume of $\mathcal{S}_{j,\alpha}$..

Let H_j denote the $(d-2)$-dimensional hyperplane in the $z=0$ hyperplane that is perpendicular to e_j. Let T_j denote the projection of S on H_j, and let $T_{j,\alpha}$ denote the projection of $S_{j,\alpha}$ on H_j. Since S is convex, it may be written

$$S = \{x + ye_j \mid x \in T_j, y \in \Re, f_{\text{low}}(x) \le y \le f_{\text{high}}(x)\}.$$

for suitable f_{low} and f_{high} defined on T_j. Now let

$$I_1 = \{x + ye_j \mid x \in T_j, y \in \Re, f_{\text{high}}(x) < y \le f_{\text{high}}(x) + \alpha\},$$

and

$$I_2 = \{x + ye_j \mid x \in T_j, y \in \Re, f_{\text{high}}(x) < y < f_{\text{low}}(x) + \alpha\}.$$

Then $(S + s_j) \cup I_2 = S_{j,\alpha} \cup I_1$, and these are disjoint unions. Therefore

$$\text{Vol}(S_{j,\alpha}) = \text{Vol}(S + s_j) - \text{Vol}(I_1) + \text{Vol}(I_2),$$

and

$$\text{Vol}(tS_{j,\alpha}) = \text{Vol}(t(S + s_j)) - \text{Vol}(tI_1) + \text{Vol}(tI_2) \tag{4.1}$$

Lemma 4.5. $\text{Vol}(tI_1) = \alpha\beta_j\text{Vol}(tS)/t$, for some value β_j.

Proof. Plainly $\text{Vol}(tI_1) = \alpha\text{Vol}(tT_j)$, and since $\text{Vol}(tS)/\text{Vol}(tT_j) = \beta_j t$ for some β_j depending on e_j, the lemma follows. ∎

Lemma 4.6. $\text{Vol}(tI_2) = \alpha^2\gamma_j\text{Vol}(tS)/t^2$, for some value γ_j.

Proof. Let $g(x) = f_{\text{high}}(x) - f_{\text{low}}(x)$, for $x \in T_j$, and let $g^{-1}(y) = \{x \mid g(x) = y\}$. Then by elementary calculus,

$$\text{Vol}(I_2) = \int_0^\alpha (\alpha - w)\text{SA}(g^{-1}(w))dw,$$

where $\text{SA}(\mathcal{E})$ denotes the surface area in H_j of surface $\mathcal{E}$.

Now $g^{-1}(w)$ is a subset of the boundary of $T_{j,w}$. In particular, $g^{-1}(0)$ is a subset of the boundary of T_j, which can be proven as follows. If there is a point p in the interior of T_j with $g(p) = 0$, then $g(x) = 0$ for all $x \in T_j$, contradicting the assumption that C, and so S, has nonzero volume. Suppose $g(p) = 0$ for some point p in the interior of T_j, but $g(x) \ne 0$ for some $x \in T_j$. Then since p is an interior point, there is some point s such that p is a convex combination of $ax + bs$. Now since $g(x) \ne 0$, there are two points x' and x'' in S with the same projection x, but with different e_j components. There is also a point $s' \in S$ whose projection on H_j is s. Now $ax_1 + bs'$ and $ax_2 + bs'$ are both in S, and both project to p, yet they have distinct e_j components. This contradicts the assumption that $g(p) = 0$.

A similar argument shows that $g^{-1}(w)$ is a subset of the boundary of $T_{j,w}$. From this fact it follows that $\text{SA}(g^{-1}(w)) \le \text{SA}(T_{j,w})$. Now since $T_{j,w} \subseteq T_j$ for $w \ge 0$, and T_j and $T_{j,w}$ are convex, it follows that $\mathrm{SA}(T_{j,w}) \le \mathrm{SA}(T_j)$. Therefore, $\mathrm{SA}(g^{-1}(w)) \le \mathrm{SA}(T_j)$, so

$$\mathrm{Vol}(I_2) \le \int_0^\alpha (\alpha - w)\mathrm{SA}(T_j)dw \le \alpha^2 \mathrm{SA}(T_j).$$

But $\mathrm{Vol}(t\mathcal{S})/\mathrm{SA}(tT_j) = \gamma_j t^2$ for some constant γ_j dependent on e_j, so the lemma follows. ∎

Putting Lemmas 4.5 and 4.6 together with (4.1), we have

$$\mathrm{Vol}(t\mathcal{S}_{j,\alpha}) \le \mathrm{Vol}(t\mathcal{S}) - \frac{\alpha\beta_j \mathrm{Vol}(t\mathcal{S})}{t} + \frac{\alpha^2\gamma_j \mathrm{Vol}(t\mathcal{S})}{t^2}.$$

Therefore

$$\mathrm{Vol}(A' \cap A'_j) \le \mathrm{Vol}(A') - \frac{\alpha\beta_j \mathrm{Vol}(A')}{t_{\max}} + \frac{\alpha^2\gamma_j \mathrm{Vol}(A')}{t_{\max}^2}.$$

Now by an argument similar to that for Lemma 4.3, the minimum value

$$\alpha_{\min} = \min\{\alpha \mid s_j = \alpha e_j, s_j \notin B_D \cup C\}$$

is linear in D. As noted, $t_{\max}$ is linear in r_k, hence

$$\mathrm{Vol}(A' \cap A'_j) \le \mathrm{Vol}(A') - \frac{DC_3\beta_j \mathrm{Vol}(A')}{r_k} + \frac{D^2C_4\gamma_j \mathrm{Vol}(A')}{r_k^2},$$

for suitable constants C_3 and C_4. This bound depends on s_j, however. By minimizing β_j and maximizing γ_j with respect to e_j, we have

$$\mathrm{Vol}(A' \cap A'_j) \le \mathrm{Vol}(A') - \frac{DC_5 \mathrm{Vol}(A')}{r_k} + \frac{D^2C_6 \mathrm{Vol}(A')}{r_k^2},$$

for suitable positive constants C_5 and C_6. Since $A_j \setminus (A \cap A_j) \supset A'_j \setminus (A' \cap A'_j)$, it follows that

$$\mathrm{Vol}(A \cap A_j) \le \mathrm{Vol}(A_j) - \mathrm{Vol}(A'_j) + \mathrm{Vol}(A' \cap A'_j).$$

Lemma 4.2 follows from this fact and the above bound for $\mathrm{Vol}(A' \cap A'_j)$. ∎

CHAPTER 5.
SUMMARY AND FURTHER QUESTIONS

§5.1 New Ideas and New Applications of Old Ideas

While the asymptotically faster algorithms presented are of interest themselves, it is important to emphasize the main ideas underlying their development. The highlights of this work are the following:

▶The use of "scaling" approximation algorithms for the all nearest neighbors and nearest foreign neighbors problems.

▶The analysis of the total cost of solving the nearest foreign neighbor problems arising in solving the minimum spanning tree problem. This analysis involved "charging" quadtree cells for operations involving them, and bounding the total charge per quadtree cell using a geometric condition.

▶The use of random sampling in a divide-and-conquer algorithm. This idea is of course quite old, but this application of it, to split up a problem in many dimensions at once, is perhaps distinctive.

▶The use of bucketing to sort dependent data.

▶The use of the quadtree data structure for manipulating point data. Quadtrees have proven quite useful for many geometric problems (Samet, [Ros], p. 212) but more commonly for representing volumes or regions, rather than for point data. Furthermore, for many uses of quadtrees performance guarantees are limited, and few algorithms with better asymptotic performance have been devised using them.

▶The use of a new data structure, the celltree. This variant of the quadtree may well prove useful for other applications. Celltrees require linear space, unlike quadtrees, yet in a sense maintain geometrical information more accurately than k-d trees.

§5.2 Further Questions

Perhaps the most significant idea to emerge from this work is that of a "scaling" approximation algorithm. Such an approach should prove quite useful for the nearest foreign neighbor problem with other L_p norms. In particular, a $O(n \log^{O(1)}(1/\epsilon))$ algorithm for the Euclidean case surely exists. Closely related to the NFN problem is the diameter, or farthest pair, problem, in which a pair of points in a set is desired realizing the farthest distance apart for all such pairs. This problem, so similar in statement to a closest pair problem, seems to be much more difficult. Another class of problems for which a scaling approach is imaginable is that of range queries [Knu]. Still more difficult problems for which a scaling approach is conceivable are the post office problem and linear programming. While Khachian's algorithm ([Kha], [PS], p. 170) approximates a solution to a linear programming problem, it is not a scaling algorithm. Such an algorithm might, for example, use approximate versions of the input constraints and objective function, increasing accuracy in these approximations as a value closer to optimal is found. Gabow's results [Gab] on special cases of linear programming, such as network flows and matching, suggest that a scaling algorithm may exist. Still another area with possibilities is that of location problems, such as the smallest enclosing circle problem [Sha].

A very important question concerning these results is the behavior of these algorithms in practice. The scaling algorithms are simple enough that they are very likely to be useful in appropriate applications. The other algorithms, while more complex, may also be useful.

The chief impediment to the application of these new algorithms may be a problem plaguing many geometric algorithms, the "curse of dimensionality" ([GMW], p. 93). The algorithms all have a running time that is at least exponentially dependent on the dimension. In some cases, this is unavoidable. For example, the all nearest neighbors problem, as it is stated, can have an output exponentially large in the dimension. This might be alleviated by, for example, using only some of the coordinates to determine distance measures, choosing the coordinates perhaps on the basis of variability. Even the algorithms with good expected times, however, have run times exponentially increasing in the dimension.

The celltree data structure has been shown to be useful for closest-point problems. Is it possible to construct one without using the bitwise exclusive-or function? Clearly the numerical least common ancestor function can be implemented by using multiplication by powers of 2 and the floor function. Is there some more elegant way to perform this operation, or avoid the need for it? The celltree approach seems to be less attractive when the performance measure counts the number of bits examined, and not the number of operations on coordinates. Is this inevitable? Finally, the analysis of the algorithm for building a celltree can be tightened in a number of ways. Because of the repeated sampling at each step, the variance of the run time of this algorithm should be small, perhaps $O(n)$ as with quicksort [Knu].

The bucket sorting of dependent data in this situation leads to questions concerning other conditions for which bucket sorting might require linear time. A problem with bucketing methods is that the running time of the algorithms, though linear in the number of points, has a constant factor that is dependent on the distribution. Can this dependence be characterized, or ameliorated using "adaptive" methods that gather information from the input points, as in Weide's and in Chapiro's work ([Wei][Cha])?

REFERENCES

[Ben] J. L. Bentley. Multidimensional divide-and-conquer. *CACM,* Volume 23 (1980), Number 4, pp. 214–229.

[BFP] J. L. Bentley, M. G. Faust, and F. P. Preparata. Approximation Algorithms for Convex Hulls. *CACM,* Volume 25 (1982), Number 1, pp. 64–68.

[BGT] J. L. Bentley, H. Gabow, and R. E. Tarjan. Scaling and related techniques for geometry problems. *Proc. 16th ACM Symp. on Theory of Computing.* Washington, D.C., 1984, pp. 135–143.

[BWY] J. L. Bentley, B. Weide, and A. C. Yao. Optimal expected-time algorithms for closest-point problems. *ACM Trans. Math. Software,* Volume 6 (1982), Number 4, pp. 563–579.

[BP] G. Bilardi and F. P. Preparata. Probabilistic analysis of a new geometric searching technique. Manuscript, Dept. of EECS, Univ. Illinois, Urbana, 1982.

[Bro] K. Q. Brown. Geometric transformations for fast geometric algorithms. Tech. Report CMU-CS-80-101, Computer Science Dept., Carnegie-Mellon Univ., 1979.

[Cha] D. Chapiro. Sorting by recursive partitioning. Tech. Report STAN-CS-83-994, Computer Science Dept., Stanford Univ., 1983.

[Cla1] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. *Proc. 24th Symp. on Foundations of Computer Science,* Tucson, AZ, November 1983, pp. 226–232.

[Cla2] K. L. Clarkson. A nearly linear expected time algorithm for minimum spanning trees in coordinate spaces. Unpublished manuscript, 1983.

[Dev] L. Devroye. Average time behavior of distributive sorting algorithms. Tech. Rep. No. SOCS 79.4, School of Computer Science, McGill University, 1979.

[DoL] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing,* Volume 5 (1976), pp. 181–186.

[EGS] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. To appear in *SIAM Journal on Computing.*

[FT] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. To appear, *Proc. 25th Symp. on Foundations of Computer Science,* Singer Island, FL, 1984.

[Gab] H. N. Gabow. Scaling algorithms for network problems. *Proc. 24th Symp. on Foundations of Computer Science,* Tucson, AZ, November 1983, pp. 248–258.

[GJ] M. G. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness.* San Francisco, Calif.: W. H. Freeman, 1979.

[GMW] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* New York: Academic Press, 1981.

[GS] L. J. Guibas and J. Stolfi. On computing all north-east nearest neighbors in the L_1 metric. Unpublished manuscript.

[Hoa] C. A. R. Hoare. Quicksort. *Comp. J.* Volume 5 (1962), pp. 10–15.

[Hop] B. Hopkins and J. G. Skellam. A new method for determining the type of distribution of plant individuals. *Ann. Bot. Lond. N. S.* Volume 18 (1954), pp. 213–227.

[IS] E. J. Isaac and R. C. Singleton. Sorting by Address Calculation. *J. Assoc. Comp. Mach.,* Volume 3 (1956), pp. 169–174.

[Kha] L. G. Khachian. A polynomial algorithm for linear programming. *Doklady Akad. Nauk. USSR,* Volume 244 (1979), pp. 1093–1096.

[Kir] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing,* Volume 12 (1983), pp. 28–35.

[Knu] D. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Reading, Mass.: Addison Wesley, 1973.

[Kru] J. B. Kruskal, Jr. On the shortest spanning subgraph of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.,* Volume 7 (1956), pp. 48–50.

[Law] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rinehart, and Winston, 1976.

[LP] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *Proc. 8th ACM Symp. on Theory of Computing.* Hershey, 1976, pp. 231–235.

[LT] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing,* Volume 9 (1980), pp. 615–627.

[Lue] G. Lueker. A survey of address calculation techniques with unknown input distributions. Unpublished manuscript, 1983.

[LW] D. T. Lee and C.K. Wong. Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications. *SIAM Journal on Computing,* Volume 9 (1980), pp. 200–211.

[Mur] F. Murtaugh. Expected-time complexity results for hierarchic clustering algorithms which use cluster centres. *Inform. Proc. Let.,* Volume 16 (1983), pp. 237–241.

[P] F. P. Preparata. A new approach to planar point location. *SIAM Journal on Computing,* Volume 10 (1981), pp. 473–482.

[PS] C. Papadimitriou and K. Stieglitz.i *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1982.

[Ros] A. Z. Rosenfeld. *Multiresolution Image Processing and Analysis.* Springer-Verlag, 1984.

[Sha] M. I. Shamos. *Computational Geometry.* Ph.D. dissertation, Department of Computer Science, Yale University, 1977.

[Sha2] M. I. Shamos. Geometric Complexity. *Proc. 7th ACM Symp. on Theory of Computing,* 1975, pp. 224–233.

[SH] M. I. Shamos and D. Hoey. Closest-point problems. *Proc. 16th Symp. on Foundations of Computer Science,* 1975, pp. 151–162.

[Sup] K. J. Supowit. The relative neighborhood graph, with an application to minimum spanning trees. *J. Assoc. Comp. Mach.,* Volume 30 (1983), pp. 428–447.

[Tar1] R. E. Tarjan. Efficiency of a good but not linear disjoint set union algorithm. *J. Assoc. Comp. Mach.,* Volume 22 (1975), pp. 215–225.

[Tar2] R. E. Tarjan. *Data Structures and Network Algorithms.* Philadephia: SIAM, 1983.

[Tra] J. F. Traub. *Symposium on New Directions and Recent Results in Algorithms and Complexity.* New York: Academic Press, 1976.

[Wei] B. Weide. Statistical methods in algorithm design and analysis. Ph.D. Dissertation, Carnegie-Mellon University, Dept. of Computer Science, 1978.

[Yao] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing,* Volume 11 (1982), Number 4, pp. 721–736.

[Zol] J. Zolnowsky. Ph.D. Thesis, Stanford University, Dept. of Computer Science, 1978.

[Luc] G. Lueker. A survey of address calculation techniques with unknown input distributions. Unpublished manuscript, 1983.

[LW] D. T. Lee and C.K. Wong. Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications. *SIAM Journal on Computing,* Volume 9 (1980), pp. 200–211.

[Mur] F. Murtaugh. Expected-time complexity results for hierarchic clustering algorithms which use cluster centres. *Inform. Proc. Let.,* Volume 16 (1983), pp. 237–241.

[P] F. P. Preparata. A new approach to planar point location. *SIAM Journal on Computing,* Volume 10 (1981), pp. 473–482.

[PS] C. Papadimitriou and K. Stieglitz.i *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1982.

[Ros] A. Z. Rosenfeld. *Multiresolution Image Processing and Analysis.* Springer-Verlag, 1984.

[Sha] M. I. Shamos. *Computational Geometry.* Ph.D. dissertation, Department of Computer Science, Yale University, 1977.

[Sha2] M. I. Shamos. Geometric Complexity. *Proc. 7th ACM Symp. on Theory of Computing,* 1975, pp. 224–233.

[SH] M. I. Shamos and D. Hoey. Closest-point problems. *Proc. 16th Symp. on Foundations of Computer Science,* 1975, pp. 151–162.

[Sup] K. J. Supowit. The relative neighborhood graph, with an application to minimum spanning trees. *J. Assoc. Comp. Mach.*, Volume 30 (1983), pp. 428–447.

[Tar1] R. E. Tarjan. Efficiency of a good but not linear disjoint set union algorithm. *J. Assoc. Comp. Mach.*, Volume 22 (1975), pp. 215–225.

[Tar2] R. E. Tarjan. *Data Structures and Network Algorithms.* Philadephia: SIAM, 1983.

[Tra] J. F. Traub. *Symposium on New Directions and Recent Results in Algorithms and Complexity.* New York: Academic Press, 1976.

[Wei] B. Weide. Statistical methods in algorithm design and analysis. Ph.D. Dissertation, Carnegie-Mellon University, Dept. of Computer Science, 1978.

[Yao] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing,* Volume 11 (1982), Number 4, pp. 721–736.

[Zol] J. Zolnowsky. Ph.D. Thesis, Stanford University, Dept. of Computer Science, 1978.