Discrete Comput Geom 4:101-115 (1989)

Discrete & Computational
Geometry

An $O(n \log n)$ Algorithm for the All-Nearest-Neighbors Problem*

Pravin M. Vaidya

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Communicated by Herbert Edelsbrunner

Abstract. Given a set V of n points in k-dimensional space, and an L_q-metric (Minkowski metric), the all-nearest-neighbors problem is defined as follows: for each point p in V, find all those points in $V-\{p\}$ that are closest to p under the distance metric L_q. We give an $O(n \log n)$ algorithm for the all-nearest-neighbors problem, for fixed dimension k and fixed metric L_q. Since there is an $\Omega(n \log n)$ lower bound, in the algebraic decision-tree model of computation, on the time complexity of any algorithm that solves the all-nearest-neighbors problem (for $k = 1$), the running time of our algorithm is optimal up to a constant factor.

1. Introduction

The all-nearest-neighbors problem is one of the very well-studied proximity problems in computational geometry [2], [4], [5], [7], [9]. We are given a set V of n points in k-dimensional space and a distance metric L_q. Each point x is given as a k-tuple of real numbers $(x_1, x_2, \ldots, x_k)$. The L_q distance between a pair of points x, y is given by $(\sum_i |x_i - y_i|^q)^{1/q}$. (Note that the L_∞ distance between x and y is given by $\max_i |x_i - y_i|$.) The nearest (closest) neighbors of a point $p \in V$ are all those points in V that are closest to p under the distance metric L_q. The all-nearest-neighbors problem is defined as follows: for each point p in V, find the nearest (closest) neighbors of p. We assume that the dimension k and the

* This research was supported by a fellowship from the Shell Foundation. The author is currently at AT&T Bell Laboratories, Murray Hill, New Jersey, USA.

distance metric L_q are fixed. We use distance for L_q distance, and $d(x, y)$ to denote the distance between x and y.

As far as the model of computation is concerned we assume that all arithmetic, comparison, and memory-access operations require constant time.

The simplest algorithm for the all-nearest-neighbors problem may be phrased as follows: for each point p in V, explicitly test if every point p' in $V-\{p\}$ is a closest neighbor of p. This algorithm runs in time $\Theta(n^2)$. However, it is possible to obtain algorithms that require $o(n^2)$ time. Bentley [2] utilizes multidimensional divide and conquer to develop an $O(n(\log n)^{k-1})$ algorithm for the all-nearest-neighbors problem. An $O(n \log \delta)$ algorithm is presented by Clarkson [5] and by Gabow *et al.* [6], where δ is the ratio of the maximum to the minimum distance between a pair of points in V. Clarkson [5] gives a randomized algorithm, with an expected running time of $O(n \log n)$ for any set of input points.

We give a deterministic algorithm for the all-nearest-neighbors problem, with a worst-case running time of $O(n \log n)$. In the algebraic decision-tree model of computation, there is a lower bound of $\Omega(n \log n)$ on the time complexity of any algorithm that solves the all-nearest-neighbors problem for dimension $k = 1$ [1], [7]. So the running time of our algorithm is optimal up to a constant factor. We note however that the multiplicative constant in the running time of our algorithm depends on the dimension k, and is of the order of $(ck)^k$ where c is a constant that does not depend on the dimension k. The space requirement of our algorithm is $O(n)$.

Our algorithm is based on the following idea (technique). The given points are maintained in cubical boxes together with some neighborhood information for each box. The boxes are split into smaller boxes, it is determined which of the given points are in the smaller boxes, and the neighborhood information is refined. This is repeated till each box contains a fixed number of points, and then the nearest neighbors for the points in each box are obtained from the neighborhood information for the box. This technique has been proposed and used by several authors [5], [6], [10]. Similar ideas appear in [3] and [11]. Clarkson [5] and Gabow *et al.* [6] use the above idea to develop algorithms for the all-nearest-neighbors problem. So our algorithm and the all-nearest-neighbor algorithms in [5] and [6] are similar in this respect. But there are critical differences. The scheme for splitting a box into smaller boxes and for splitting the set of input points in a box when the box is split, and the order in which the boxes are split, are both quite different in our algorithm. We utilize a scheme that relies on a combinatorial lemma for trees (Section 4) for splitting the set of input points in a box, and split the boxes in decreasing order of volume; this gives us a deterministic algorithm whose worst-case running time is $O(n \log n)$.

At this point we note that the expected running time of Clarkson's randomized algorithm [5] is $O((c')^k n \log n)$, where c' is a constant independent of dimension k; whereas the worst-case running time of our algorithm is $O((ck)^k n \log n)$, where c is a constant that does not depend on k. Thus the expected running time of Clarkson's randomized algorithm has a better dependence on the dimension than the worst-case running time of our algorithm. On the other hand, Clarkson's

randomized algorithm requires the use of floor and bitwise exclusive-or operations, whereas our algorithm does not.

A generalization of the all-nearest-neighbors problem is defined as follows. For a point $p \in V$, let $D(p)$ be the multiset of distances defined by $D(p) = \{z: z = d(p, p'), p' \in V, z \neq 0\}$, and let $d_1(p) \leqq d_2(p) \leq \cdots \leq d_m(p)$ be the m smallest distances in $D(p)$. Then the m-nearest-neighbors of p are all those points in V whose distance from p is at most $d_m(p)$. The all-m-nearest-neighbors problem is as follows: for every point p in V, find the m-nearest-neighbors of p. A slight modification of our algorithm for the all-nearest-neighbors problem leads to an $O(mn \log n)$ algorithm for the all-m-nearest-neighbors problem.

We note that if the metric in the given problem is positive definite or semidefinite rather than one of the standard L_q metrics, the problem can be transformed in $O(n)$ time to a problem with the L_2 (Euclidean) metric by applying a linear transformation and a projection to the given set of points. The transformation does not increase the dimension k.

We define a box b to be the product $J_1 \times J_2 \times \cdots \times J_k$ of k intervals (either closed, semiclosed, or open) or, equivalently, the set of those points $x = (x_1, x_2, \ldots, x_k)$ such that x_i lies in the interval J_i, for $i = 1, \ldots, k$. A box is a cubical box iff all the k intervals defining the box are of identical length. The centre $\alpha(b)$ of a box b is defined to be the point $(\alpha_1(b), \ldots, \alpha_k(b))$ where $\alpha_i(b)$ is the centre of the ith interval defining the box, for $i = 1, \ldots, k$. For a hyperplane $h = \{x: \beta^T x = \gamma\}$, we define $L(h)$ to be the left open halfspace $\{x: \beta^T x < \gamma\}$, and $R(h)$ to be the right closed halfspace $\{x: \beta^T x \geqq \gamma\}$.

2. An Overview

In the algorithm we maintain a collection B of disjoint closed cubical boxes which contain all the n points in the given set V. Each box has been shrunk as much as possible so that further shrinkage would either destroy the cubical shape of the box or push out of the box a point in V that was originally located in the box. With each box $b \in B$, there are associated two subsets of B, *Neighbors*(b) and *Attractors*(b) such that:

(1) For each point $p \in b \cap V$, every nearest neighbor of p is located in box b itself or in some box in *Neighbors*(b).
(2) *Attractors*$(b) = \{b': b \in$ *Neighbors*$(b')\}$. Thus if $p \in b \cap V$ is the nearest neighbor of some point $p' \in V$ then p' must be located in box b itself or in some box in *Attractors*(b).

In the beginning, B contains a single box which is a smallest cubical box containing all the points in V. At the end, every box $b \in B$ is degenerate and consists of a single point in V, i.e., $|b| = |b \cap V| = 1$, and *Neighbors*(b) is exactly the set of nearest neighbors of the unique point in $b \cap V$.

Let $d_{\max}(b)$ denote the greatest distance between a pair of points in b. Let $d_{\max}(b, b')$ denote the maximum distance between a point in b and a point in b',

and let $d_{\min}(b, b')$ denote the minimum distance between a point in b and a point in b'. For each box $b \in B$, we maintain a parameter *Estimate*(b) given by

$$Estimate(b) = \begin{cases} d_{\max}(b) & \text{if } |b \cap V| \geq 2, \\ \min\limits_{b' \in Neighbors(b)} \{d_{\max}(b, b')\}, & \text{otherwise.} \end{cases}$$

For each point $p \in b \cap V$, *Estimate*(b) serves as an upper bound on the distance between p and a nearest neighbor of p. In order to eliminate unnecessary boxes from *Neighbors*(b), we ensure that *Neighbors*(b) satisfies the invariant

$$\forall b' \in Neighbors(b), \qquad d_{\min}(b, b') \leq Estimate(b).$$

The algorithm proceeds in stages. At the beginning of each stage we choose a box b in B that has the largest volume (size) among all the boxes in B for splitting. The chosen box b is split into 2^k cubical boxes $b_1, b_2, \ldots, b_{2^k}$, by k mutually orthogonal hyperplanes passing through its center $\alpha(b)$, each hyperplane being perpendicular to one of the coordinate axes. Simultaneously, the set of points $b \cap V$ is split into the sets of points $b_1 \cap V, b_2 \cap V, \ldots, b_{2^k} \cap V$ using a scheme based on a combinatorial lemma for trees (Section 4). Out of these 2^k boxes we discard those which do not contain a point in V, and each of the remaining boxes is shrunk as much as possible to obtain the set *Successors*(b). The shrinking process ensures that when b is split the set of data points $b \cap V$ is also split, or, equivalently, shrinking guarantees that there are at least two boxes in *Successors*(b). We then remove the split box b from B and add to B all the boxes in *Successors*(b). Finally, we create the estimates, and the neighbor and attractor sets, for the boxes in *Successors*(b); update the estimates and the neighbor sets for the boxes in *Attractors*(b); and update the attractor sets for the required boxes in B. That concludes a stage. For each $b \in B$, at the end of a stage *Neighbors*(b) satisfies the above-described invariant.

Throughout the algorithm *Estimate*(b) serves as an approximation to the distance between a point $p \in b \cap V$ and a nearest neighbor of p, and the volume (size) of b is a measure of the error in this approximation. By splitting a box that has the largest volume, an estimate that has the largest error is refined.

We note that the algorithms for the all-nearest-neighbors problem described in [5] and [6] also maintain the given points in cubical boxes together with estimates for the nearest-neighbor distances of points in each box and a set of boxes similar to *Neighbors*(b) for each box b. So our algorithm is similar to the algorithms in [5] and [6] in this respect. But the scheme for splitting a box together with the set of input points in the box, and the order in which the boxes are split, are totally different in our algorithm; the scheme and the order that we use lead to a worst-case running time of $O(n \log n)$.

3. The Algorithm

Let $h_i(b)$ be the hyperplane defined by $h_i(b) = \{x: x_i = \alpha_i(b)\}$, where $\alpha(b) = (\alpha_1(b), \alpha_2(b), \ldots, \alpha_k(b))$ is the centre of box b. Let $L(h_i(b))$ be the left open

halfspace $\{x: x_i < \alpha_i(b)\}$ and $R(h_i(b))$ be the right closed halfspace $\{x: x_i \geq \alpha_i(b)\}$. Let $Immediate-Successors(b)$ be the set of boxes defined by

$$Immediate-Successors(b) = \{b': b' = b \cap f_1 \cap \cdots \cap f_k, \text{ where } f_i = L(h_i(b)) \text{ or } f_i = R(h_i(b)), 1 \leq i \leq k\}.$$

Corresponding to a cubical box b, let $shrunk(b)$ be a cubical box such that:

(1) If $|b \cap V| \leq 1$ then $shrunk(b) = b \cap V$.
(2) If $|b \cap V| \geq 2$ then (i) $shrunk(b) \subseteq b$, (ii) $shrunk(b) \cap V = b \cap V$, and (iii) the maximum L_∞ distance between a pair of points in $shrunk(b) \cap V$ equals the size of $shrunk(b)$.

We now give the algorithm for the all-nearest-neighbors problem.

Algorithm All-Nearest-Neighbors

1. $Neighbors(b_0) := \emptyset$, $Attractors(b_0) := \emptyset$, $B := \{b_0\}$, where b_0 is a smallest cubical box containing all the n points in V.
2. Repeat Steps 2.1–2.6 until each box $\hat{b} \in B$ satisfies $|\hat{b}| = |\hat{b} \cap V| = 1$, i.e., until each box $\hat{b} \in B$ is degenerate and consists of a single point in V.
   2.1. Choose, for splitting, a box $b \in B$ that has the largest volume among all the boxes in B.
   2.2. Split the set of points $b \cap V$ into the sets of points $b_1 \cap V$, $b_2 \cap V, \ldots, b_{2^k} \cap V$, where $b_1, b_2, \ldots, b_{2^k}$ are the boxes in $Immediate-Successors(b)$.
   2.3. $Successors(b) := \{b': b' = shrunk(b''), b'' \in Immediate-Successors(b), |b'' \cap V| \geq 1\}$.
   2.4. For each box $b' \in Successors(b)$.
   create $Neighbors(b')$ and $Estimate(b')$.
   For each box $b' \in Attractors(b)$,
   update $Neighbors(b')$ and $Estimate(b')$.
   2.5. For each box $b' \in Successors(b)$,
   create $Attractors(b')$.
   For each box $b' \in (B - Successors(b))$,
   update $Attractors(b')$.
   2.6. $B := B - \{b\} \cup Successors(b)$.

end All-Nearest-Neighbors

Let B^* be the set of all the boxes generated during the execution of the algorithm. The boxes in B^* form a tree in which the boxes are the vertices, the children of b are the boxes in $Successors(b)$, and the leaves are all the points in the set V. Every box in this tree has at least two and at most 2^k children. Also the total number of boxes in B^* is at most $2n$.

The running time of the algorithm may be divided into:

(i) Time to select a largest box at each stage summed over all stages.

(ii) Total time required to split the set of data points $b \cap V$, and shrink box b, for all boxes $b \in B^*$.

(iii) Total time required to maintain *Estimate*(b), *Neighbors*(b), and *Attractors*(b) for all boxes $b \in B^*$.

We maintain a heap [8] for the boxes in B. This allows us to pick in constant time a box in B that has the largest volume. Since $|B^*| \leq 2n$ the total time for heap maintenance is $O(n \log n)$. Then (i) is $O(n \log n)$.

In Section 4 we show that (ii) is $O(n \log n)$. To split the set of data points $b \cap V$ efficiently we utilize k ordered lists $List_i(b)$, $1 \leq i \leq k$, the ith list containing the points in $b \cap V$ ordered on the ith coordinate. The list $List_i(b)$ is embedded in the leaves of a complete binary search tree $T_i(b)$. When b is split, from the k ordered lists for b we obtain similar lists for all the boxes in *Successors*(b). In Section 4 we show how to split the set of data points in a box so that the total time to split $b \cap V$ for all boxes $b \in B^*$ is $O(n \log n)$. Once the k ordered lists corresponding to a box are available the box can be shrunk in constant time. Thus the total time for shrinking all the boxes in B^* is $O(n)$.

In Section 5 we show that (iii) is $O(n \log n)$. Since at each stage we split a box that has the largest volume, and force the neighbor set of each box $b \in B$ to satisfy the invariant $\forall b' \in Neighbors(b)$, $d_{\min}(b, b') \leq Estimate(b)$, we can bound the sizes of the attractor sets of all the boxes in B, and the sizes of the neighbor sets of nondegenerate boxes in B, by some constant dependent on k. This in turn enables us to obtain a bound of $O(n)$ on the total number of additions to (insertions into) the neighbor and attractor sets of all the boxes in B^*. Since the size of the attractor set of each box in B is bounded by a constant dependent on k, we get that the time for maintaining attractor sets is $O(n)$. For a box b, we implement *Neighbors*(b) by a data structure which allows insertions and deletions to be performed in $O(\log n)$ time, and allows access to a box b', with the greatest value of the parameter $d_{\min}(b, b')$, in $O(\log n)$ time. Then the total time, for maintaining neighbor sets of all the boxes in B^*, can be shown to be $O(n \log n)$. Finally, only addition of a box to *Neighbors*(b) can change the parameter *Estimate*(b), and the change due to the addition of a single box can be computed in constant time. So the total time for maintaining the estimates is $O(n)$. This gives a bound of $O(n \log n)$ on (iii).

We note that since the total number of insertions into the neighbor and attractor sets is $O(n)$, the space required by our algorithm is $O(n)$.

4. Splitting the Set of Data Points in a Box

In this section we describe how to split the set of data points $b \cap V$ in a box b efficiently, so that the total time required to split $b \cap V$ for all the boxes generated during the algorithm is $O(n \log n)$.

We make the following definitions. Let T be a rooted tree such that each nonleaf vertex has at least two children. For a vertex v in T, we define $m(v)$, $v_{\max}$, $s(v)$, and $lsa(v)$. Let $m(v)$ be the number of leaves in the subtree rooted at v, and let the largest son of v, denoted by $v_{\max}$, be a child of v such that

for any child v' of v, $m(v_{\max}) \geq m(v')$. For a nonleaf vertex v let $s(v) = m(v) - m(v_{\max})$, and for a leaf vertex v let $s(v) = 0$. We define the lowest smaller ancestor of v, denoted by $lsa(v)$, as follows. If each vertex on the path from the root to v, other than the root, is the largest son of its father then $lsa(v)$ is the root; otherwise $lsa(v)$ is the lowest vertex v' on the path from the root to v such that v' is not the largest son of its father.

In the case of the tree of boxes generated during the algorithm, for a box $b \in B^*$, $m(b)$ is identical to the number of points in $b \cap V$, $b_{\max}$ is a box in $Successors(b)$ that contains the largest number of points in V among all the boxes in $Successors(b)$, and $s(b)$ equals the number of points in $(b - b_{\max}) \cap V$.

In order to bound the total amount of work for splitting sets of data points in boxes we require a lemma about weightings on tree vertices.

Tree-Weighting Lemma. *Let T be a rooted tree with t vertices such that each nonleaf vertex has at least two and at most r children. Define the weight $w(v)$ of a nonleaf vertex v by $w(v) = s(v)(1 + \log_2 m(lsa(v)) - \log_2 s(v))$ and define the weight of a leaf to be 0. Then $\sum_{v \in T} w(v) \leq 4rt \log_2 t$.*

Proof. The lemma is proved by induction on the number of leaves in T. The base case is when T consists of a single leaf vertex v and then $w(v) = 0$. So let the number of leaves in T be at least two, then $t \geq 2$. Let P be a sequence of vertices in T as follows. The first vertex in P is the root of T, and, for $j > 1$, the jth vertex in P is the largest son of the $(j-1)$st vertex in P. The sequence terminates in a leaf. Let Q be the set of all vertices v such that v is the son of some vertex in P and v is not the largest son of its father. For a vertex v, let $T(v)$ denote the subtree rooted at v. We inductively assume that,

$$\forall v \in Q, \qquad \sum_{u \in T(v)} w(u) \leq 4rm(v) \log_2 m(v).$$

We note that, for each vertex v in P, $lsa(v) = \text{root}$, and so,

$$\forall v \in P, \qquad w(v) = s(v)(1 + \log_2 t - \log_2 s(v)).$$

As a vertex v has at most r children we have $s(v) \leq (1 - 1/r)m(v)$, and so,

$$\forall v \in P, \qquad \log_2 t - \log_2 s(v) \geq \log_2 m(v) - \log_2 s(v) \geq -\log_2(1 - 1/r).$$

Thus,

$$\forall v \in P, \qquad w(v) \leq 4rs(v)(\log_2 t - \log_2 s(v)).$$

Then

$$\begin{aligned} \sum_{v \in T} w(v) &= \sum_{v \in P} w(v) + \sum_{v \in Q} \sum_{u \in T(v)} w(u) \\ &\leq \sum_{v \in P} 4rs(v)(\log_2 t - \log_2 s(v)) + \sum_{v \in Q} 4rm(v) \log_2 m(v). \end{aligned}$$

Now since $\sum_{v \in P} s(v) \leq t$, and $\sum_{v \in Q} m(v) \log_2 m(v) \leq \sum_{v \in P} s(v) \log_2 s(v)$, we get that $\sum_{v \in T} w(v) \leq 4rt \log_2 t$. □

A proper ancestor of a leaf in a rooted tree is a nonleaf vertex on the path from the root to the leaf. We also require an upper bound on the number of proper ancestors of a set of leaves in a complete binary tree, and such a bound is provided by the following lemma.

Ancestors Lemma. *Let T be a complete binary tree of height $h(T)$ containing $2^{h(T)}-1$ vertices and let L be a set of leaves in T. Let Ancestors(L, T) be the set of all the proper ancestors in T of the leaves in L. Then $|Ancestors(L, T)| \leq |L|(h(T) - \log|L|) - 1$.*

Proof. The bound holds for $h(T) = 2$. Let us assume that the upper bound holds for all complete binary trees containing at most $2^{h(T)-1}-1$ vertices. Let T_1 and T_2 be the subtrees rooted at the two sons of the root of T, and let L_1, L_2, be the sets of leaves in L that are in T_1, T_2, respectively. Then

$$|Ancestors(L_1, T_1)| \leq |L_1|(h(T) - 1 - \log|L_1|) - 1,$$
$$|Ancestors(L_2, T_2)| \leq |L_2|(h(T) - 1 - \log|L_2|) - 1,$$

and

$$|Ancestors(L, T)| = |Ancestors(L_1, T_1)| + |Ancestors(L_2, T_2)| + 1.$$

After some algebraic manipulation it is seen that

$$|Ancestors(L, T)| \leq |L|(h(T) - \log|L|) - 1 - |L_1| - |L_2| + |L_2| \log\left(\frac{|L_1|+|L_2|}{|L_2|}\right) + |L_1| \log\left(\frac{|L_1|+|L_2|}{|L_1|}\right).$$

The proof of the lemma follows from the observation that $\log((|L_1|+|L_2|)/|L_1|) \leq |L_2|/|L_1|$ and $\log((|L_1|+|L_2|)/|L_2|) \leq |L_1|/|L_2|$. □

We now give a scheme for splitting the set of input points in a box when the box is split. The boxes in B^* (i.e., the set of all the boxes generated during the algorithm) form a tree in which the vertices are the boxes in B^*, the children of every box b are the boxes in *Successors*(b), and the leaves are the points in V. In addition, each nonleaf box in this tree has at least two and at most 2^k children, and the number of vertices in this tree is at most $2n$. Suppose we can split the set of data points $b \cap V$ in time proportional to $s(b)(1+\log_2 m(lsa(b)) - \log_2 s(b))$. Then from the Tree-Weighting Lemma it follows that the total time required to split $b \cap V$ for all $b \in B^*$ is $O(n \log n)$. This is summarized in the following lemmas.

Splitting Lemma 1. *The total time required to split the set of data points $b \cap V$ for all boxes b generated during the algorithm is $O(k4^k n \log n)$.*

Proof. By the Tree-Weighting Lemma and Splitting Lemma 2 below. □

Splitting Lemma 2. *The set of points $b \cap V$ can be split in $O(k2^k s(b)(1+\log_2 m(lsa(b)) - \log_2 s(b)))$ time.*

We now show that $b \cap V$ can be split in $O(s(b)(1+\log_2 m(lsa(b)) - \log_2 s(b)))$ time. Corresponding to each box b we have k ordered lists $List_i(b)$, $1 \le i \le k$, the ith list containing the points in $b \cap V$ ordered on the ith coordinate. From an entry for a point p in each of these lists there are pointers to the entries for p in all the remaining $k-1$ lists. For each, i, $1 \le i \le k$, $List_i(b)$ is embedded in the leaves of a corresponding complete binary search tree $T_i(b)$ of height $\lceil \log_2 m(lsa(b)) \rceil$. The leaves of $T_i(b)$ are the points in $lsa(b) \cap V$ ordered on the ith coordinate, however, only the leaves that are points in $b \cap V$ are linked together to form the ordered list $List_i(b)$. In the process of splitting $b \cap V$, we split the k ordered lists $List_i(b)$, $i = 1, \ldots, k$, and, for each box obtain b' in $Successors(b)$, we obtain the k ordered lists $List_i(b')$, $i = 1, \ldots, k$. For each $b' \in (Successors(b) - \{b_{\max}\})$, $List_i(b')$ will be embedded in the leaves of a complete binary search tree $T_i(b')$ of height $\lceil \log_2 m(b') \rceil$.

To split $b \cap V$ we first obtain the points in $(b - b_{\max}) \cap V$ in time proportional to $s(b) = |(b - b_{\max}) \cap V|$. As before let $h_i(b)$ be a hyperplane orthogonal to the ith coordinate axis and passing through the center of b, and let $L(h_i(b))$ and $R(h_i(b))$ be the corresponding left open and right closed halfspaces. Among the two boxes $b \cap L(h_i(b))$ and $b \cap R(h_i(b))$, let b_i be the one that contains the smaller number of points in V. We can obtain the set of points $b_i \cap V$ in time proportional to $|b_i \cap V|$ by searching $List_i(b)$ simultaneously from both ends and stopping the first time the hyperplane $h_i(b)$ is crossed. We have two cases depending on $|(\bigcup_{i=1}^k b_i) \cap V|$. If $|(\bigcup_{i=1}^k b_i) \cap V| < 2^{-k} m(b)$ then $(\bigcup_{i=1}^k b_i) \cap b_{\max} = \emptyset$ and $(b - b_{\max}) = \bigcup_{i=1}^k b_i$, and in this case we spend time proportional to $k|(\bigcup_{i=1}^k b_i) \cap V| = ks(b)$ in obtaining the set of points $(b - b_{\max}) \cap V$. On the other hand, if $|(\bigcup_{i=1}^k b_i) \cap V| \ge 2^{-k} m(b)$, then we can afford to spend $O(m(b))$ time in isolating the points in $(b - b_{\max}) \cap V$. Thus the points in $(b - b_{\max}) \cap V$ can be obtained in $(k2^k s(b))$ time.

Once we have the set of points $(b - b_{\max}) \cap V$, we get the corresponding k sorted lists as follows. To obtain a list containing the set of points $(b - b_{\max}) \cap V$ sorted on the ith coordinate, we first label all the points in $List_i(b)$ that are located in $b - b_{\max}$. We then label all the vertices in $T_i(b)$ that are proper ancestors of the labeled points in $List_i(b)$. The labeling of the proper ancestors can be performed in time proportional to the number of proper ancestors plus the number of points in $(b - b_{\max}) \cap V$. Then from the Ancestors Lemma we can bound the time required for labeling by $O(s(b)(1+\log_2 m(lsa(b)) - \log_2 s(b)))$. Next, we traverse the labeled vertices of $T_i(b)$ in order and thereby obtain the labeled points in $List_i(b)$, i.e., the points in $(b - b_{\max}) \cap V$ in sorted order on the ith coordinate. The in order traversal may be performed in time proportional to the number of labeled vertices. Finally, we unlabel all the labeled vertices.

After the k ordered lists for the set $(b - b_{\max}) \cap V$ have been obtained, all the points in $(b - b_{\max}) \cap V$ are deleted from each of the lists $List_i(b)$, $i = 1, \ldots, k$.

For each box b' in $Successors(b) - \{b_{max}\}$, we can obtain the k ordered lists $List_i(b')$ from the k ordered lists for $(b - b_{max}) \cap V$, in $O(k|(b - b_{max}) \cap V|) = O(ks(b))$ time. In addition, for each b' in $Successors(b) - \{b_{max}\}$, $List_i(b')$ may be embedded in a complete binary search tree $T_i(b')$ of height $\lceil \log_2 m(b') \rceil$ in $O(m(b'))$ time.

Finally, we note that deleting the points in $(b - b_{max}) \cap V$ from $List_i(b)$, $i = 1, \ldots, k$, leaves behind the points in $b_{max} \cap V$. So the k ordered lists for the points in $b_{max} \cap V$ are now available. Since $lsa(b_{max}) = lsa(b)$, a suitable embedding of these lists in a complete binary search tree of height $\lceil \log_2 m(lsa(b_{max})) \rceil$ is also available.

Thus the entire process of splitting $b \cap V$ may be accomplished in $O(k2^k s(b)(1 + \log_2 m(lsa(b)) - \log_2 s(b)))$ time. We have thus proved Splitting Lemma 2.

5. Maintaining Neighbor and Attractor Sets, and Estimates

In this section we describe how the neighbor and attractor sets, and the estimates, may be maintained in $O(n \log n)$ time. We first observe that:

(1) A box b' is added to or deleted from $Attractors(b)$ whenever b is added to or deleted from $Neighbors(b')$.
(2) $Estimate(b)$ changes only when a box is added to $Neighbors(b)$.
(3) When b is split, there can be additions only to the neighbor sets of boxes in $Successors(b) \cup Attractors(b)$, only boxes in $Successors(b)$ can possibly get added to the neighbor set of a box in $Attractors(b)$, and for all $b' \in Successors(b)$, $Neighbors(b') \subseteq Neighbors(b) \cup Successors(b)$.

The above observations lead to the following procedure for modifying the neighbor and attractor sets, and the estimates, during the stage when b is split.

Procedure Modify-sets-estimates

1. $\forall b' \in Successors(b)$,
   $Neighbors(b') := Neighbors(b) \cup Successors(b) - \{b'\}$,
   $Attractors(b') := Attractors(b) \cup Successors(b) - \{b'\}$,
   If $|b' \cap V| \geq 2$ then $Estimate(b') := d_{max}(b')$
   else $Estimate(b') := \min_{b'' \in Neighbors(b) \cup Successors(b)} \{d_{max}(b', b'')\}$.
2. $\forall b' \in Attractors(b)$,
   $Neighbors(b') := Neighbors(b') \cup Successors(b) - \{b\}$,
   $Estimate(b') := \min\{Estimate(b'), \min_{b'' \in Successors(b)} \{d_{max}(b', b'')\}\}$.
3. $\forall b' \in Neighbors(b)$,
   $Attractors(b') := Attractors(b') \cup Successors(b) - \{b\}$.
4. $\forall b' \in Successors(b) \cup Attractors(b)$,
   delete from $Neighbors(b')$ every box b'' which satisfies $d_{min}(b', b'') > Estimate(b')$, and if b'' is deleted from $Neighbors(b')$ then also delete b' from $Attractors(b'')$.

end Modify-sets-estimates

As before let B^* be the set of all the boxes generated during the execution of the all-nearest-neighbors algorithm. We shall bound the total number of boxes that are added to *Neighbors*(b) and *Attractors*(b) for all $b \in B^*$ during the entire execution of the all-nearest-neighbors algorithm. During the stage when b is split, there are at most

$$2|\mathit{Successors}(b)|(|\mathit{Successors}(b)|+|\mathit{Neighbors}(b)|+|\mathit{Attractors}(b)|)$$

additions to the neighbor and attractor sets. So to bound the total number of additions to the neighbor and attractor sets we must bound the size of *Neighbors*(b) and *Attractors*(b) when b is split. Such bounds are provided by the Packing Lemmas that follow.

We let b_L denote a box in the current set of boxes B such that b_L has the largest volume among all the boxes in B.

Packing Lemma 1. *Let r be a positive integer. At the beginning of a stage, if $b \in B$ then the number of boxes b' in B such that $d_{\min}(b, b') \le r d_{\max}(b_L)$ is at most $2^k(2rk+3)^k$.*

Proof. Let *size*(b') denote the length of each of the k intervals defining b'. Let

$$C(b) = \{b' : b' \in B,\ d_{\min}(b, b') \le r d_{\max}(b_L)\}$$

and

$$A(b) = \{b' : b' \in B^*,\ \mathit{Successors}(b') \cap C(b) \neq \emptyset\}.$$

Since boxes are split in nonincreasing order of size, $\forall b' \in A(b)$, $\mathit{size}(b') \ge \mathit{size}(b_L)$. So each box b' in $A(b)$ may be shrunk to a box b'' such that $d_{\min}(b, b'') = d_{\min}(b, b') \le r d_{\max}(b_L)$, and $\mathit{size}(b'') = \mathit{size}(b_L)$. Let $\hat{A}(b)$ be the set of boxes obtained by shrinking all the boxes in $A(b)$ in this manner. We note that the L_q distance between two points in k-dimensional space is bounded by k times the L_∞ distance between the points. It then follows that there exists a box b^* such that $\mathit{size}(b^*) = (2rk+3)\ \mathit{size}(b_L)$, b^* and b have the same center, and b^* is a superset of every box $\hat{b}$ in $\hat{A}(b)$. As the boxes in $\hat{A}(b)$ are disjoint we get $|\hat{A}(b)| \le (2rk+3)^k$. Finally, as $|\mathit{Successors}(b)| \le 2^k$, we have

$$|C(b)| \le 2^k |A(b)| \le 2^k |\hat{A}(b)| \le 2^k (2rk+3)^k. \qquad \square$$

Packing Lemma 2. *At the beginning of each stage, if $b \in B$ and $|b \cap V| \ge 2$ then $|\mathit{Neighbors}(b)| \le c_1(k)$ where $c_1(k) = 2^k(2k+3)^k$.*

Proof. Since $|b \cap V| \ge 2$, we have $\mathit{Estimate}(b) = d_{\max}(b) \le d_{\max}(b_L)$, and hence, $\forall b' \in \mathit{Neighbors}(b)$, $d_{\min}(b, b') \le \mathit{Estimate}(b) \le d_{\max}(b_L)$. So from Packing Lemma 1 we get $|\mathit{Neighbors}(b)| \le 2^k(2k+3)^k$. $\square$

Packing Lemma 3. *At the beginning of each stage, if* $b \in B$ *then* $|Attractors(b)| \leq c_2(k)$ *where* $c_2(k) = 12^k + 2^k(8k+3)^k$.

Proof. Without loss of generality, let $\alpha(b)$, the center of box b, be the origin. Let

$$A_F(b) = \{b' : b' \in Attractors(b),\ d_{\min}(b, b') \geq 4d_{\max}(b_L)\}$$

and let

$$A_S(b) = Attractors(b) - A_F(b).$$

From Packing Lemma 1,

$$|A_S(b)| \leq 2^k(8k+3)^k.$$

We first observe that there cannot be a pair of boxes b_1, b_2, in $A_F(b)$ such that the centers $\alpha(b) = 0$, $\alpha(b_1)$, $\alpha(b_2)$ are collinear. Assume that there exist such boxes b_1 and b_2 in $A_F(b)$, and let

$$d(0, \alpha(b_2)) = d(0, \alpha(b_1)) + d(\alpha(b_1), \alpha(b_2)).$$

Then as

$$d_{\min}(b, b_2) + d_{\max}(b_L) \geq d(0, \alpha(b_2))$$

and

$$d_{\max}(b_1, b_2) \leq d(\alpha(b_1), \alpha(b_2)) + d_{\max}(b_L)$$

we get

$$d_{\min}(b, b_2) \geq d_{\max}(b_1, b_2) + d(0, \alpha(b_1)) - 2d_{\max}(b_L) > d_{\max}(b_1, b_2).$$

Then because of the invariant forced on the neighbor sets at each stage, we must have that $b \notin Neighbors(b_2)$ and $b_2 \notin Attractors(b)$ which is a contradiction.

By Packing Lemma 4 below, there also cannot be a pair of boxes b_1, b_2 in $A_F(b)$ such that

$$d\left(\frac{\alpha(b_1)}{d(0, \alpha(b_1))}, \frac{\alpha(b_2)}{d(0, \alpha(b_2))}\right) \leq \tfrac{1}{3}.$$

Let $\hat{A}_F(b) = \{p : p = \alpha(b')/d(0, \alpha(b')),\ b' \in A_F(b)\}$. Then $|\hat{A}_F(b)| = |A_F(b)|$, each point in $\hat{A}_F(b)$ is at a distance of 1 from the origin, and the distance between any two points in $\hat{A}_F(b)$ is at least $\frac{1}{3}$. Around each point in $\hat{A}_F(b)$ draw a ball of radius $\frac{1}{6}$. No two of these balls can intersect and the intersection of each such ball with the unit ball around the origin completely contains a ball of radius $\frac{1}{12}$. So the number of points in $\hat{A}_F(b)$ and hence the number of boxes in $A_F(b)$ is at most 12^k. Thus

$$|Attractors(b)| = |A_S(b)| + |A_F(b)| \leq 2^k(8k+3)^k + 12^k. \qquad \square$$

Packing Lemma 4. *At the beginning of a stage let b be a box in B and let*

$$A_F(b) = \{b' : b' \in \mathit{Attractors}(b),\ d_{\min}(b, b') \geq 4d_{\max}(b_L)\}.$$

Then there cannot be a pair of boxes b_1, b_2 in $A_F(b)$ such that

$$d\left(\frac{\alpha(b_1)}{d(0, \alpha(b_1))}, \frac{\alpha(b_2)}{d(0, \alpha(b_2))}\right) \leq \tfrac{1}{3}.$$

Proof. Assume that there is such a pair of boxes b_1, b_2, in $A_F(b)$. Let $d(0, \alpha(b_1)) \leq d(0, \alpha(b_2))$ and $\lambda = d(0, \alpha(b_1))/d(0, \alpha(b_2))$. We have

$$d(\alpha(b_1), \lambda\alpha(b_2)) \leq \tfrac{1}{3}d(0, \alpha(b_1)) = \tfrac{1}{3}d(0, \lambda\alpha(b_2)).$$

This gives

$$\begin{aligned} d(0, \alpha(b_2)) &= d(\alpha(b_1), \lambda\alpha(b_2)) + d(\lambda\alpha(b_2), \alpha(b_2)) \\ &\quad + d(0, \lambda\alpha(b_2)) - d(\alpha(b_1), \lambda\alpha(b_2)) \\ &\geq d(\alpha(b_1), \alpha(b_2)) + \tfrac{2}{3}d(0, \alpha(b_1)) \\ &\geq d(\alpha(b_1), \alpha(b_2)) + \tfrac{8}{3}d_{\max}(b_L). \end{aligned}$$

This would imply that $d_{\min}(b, b_2) > d_{\max}(b_1, b_2)$ and this cannot happen because of the invariant forced at each stage on the neighbor sets. □

From the packing lemmas and the procedure for maintaining the neighbor and attractor sets, it follows that during each stage there are at most a constant number of additions to neighbor and attractor sets. Since there are at most $2n$ stages it follows that $O(n)$ boxes are added to the neighbor and attractor sets during the entire execution of the all-nearest-neighbors algorithm. The size of the attractor set of any box in B never grows beyond a constant because at the beginning of a stage the size of the attractor set is bounded by a constant and during a stage there can be only a constant number of additions to an attractor set. So the total time for maintaining attractor sets is $O(n)$. The time to maintain the estimates is also $O(n)$ because the change in $\mathit{Estimate}(b)$ due to the addition of a box to $\mathit{Neighbors}(b)$ can be computed in constant time. We implement $\mathit{Neighbors}(b)$ by a data structure which allows insertions and deletions in $O(\log n)$ time, and allows access to a box b' that has the largest value for the parameter $d_{\min}(b, b')$ in $O(\log n)$ time. A heap or a 2-3 tree suffices [8]. Then the total work to maintain the neighbors sets is $O(n \log n)$. Explicitly evaluating the constants gives a bound of $O(8^k(8k+3)^k)n \log n)$ on the time for maintaining the neighbor and attractor sets.

6. All-m-Nearest-Neighbors

To get an algorithm for the all-m-nearest-neighbors problem, the all-nearest-neighbors algorithm is modified as follows. Any box $b' \in$ *Immediate-Successors*(b) such that $|b' \cap V| < m+1$ is immediately split into $|b' \cap V|$ boxes each containing exactly one point in V. Let $r_1(b) \leq r_2(b) \leq \cdots \leq r_m(b) \leq \cdots$ be a nondecreasing sequence of the distances in the multiset $\{r: r = d_{\max}(b, b'), b' \in \textit{Neighbors}(b)\}$. Then

$$\textit{Estimate}(b) = \begin{cases} d_{\max}(b) & \text{if } |b \cap V| \geq m+1, \\ r_m(b), & \text{otherwise.} \end{cases}$$

The sizes of the attractor sets of all the boxes in B can be bounded by $c_2(k)m$, and the sizes of the neighbor sets of nondegenerate boxes in B can be bounded by $c_1(k)$, where $c_1(k)$, $c_2(k)$ are constants dependent on k. During each stage there are $O(m^2)$ additions to the neighbor and attractor sets, and the number of stages is $O(n/m)$. Then the running time may be shown to be $O(mn \log n)$.

7. Conclusion

We have presented an $O(n \log n)$ algorithm for the all-nearest-neighbors problem. The running time of the algorithm is optimal up to a constant factor in the algebraic decision-tree model of computation. If the metric in the given problem is positive definite or semidefinite rather than one of the standard L_q metrics, the problem can be transformed in linear time to a problem with the L_2 (Euclidean) metric without increasing the dimension. A slight modification of the all-nearest-neighbors algorithm gives an $O(mn \log n)$ algorithm for the all-m-nearest-neighbors problem.

One question that naturally comes up is how does the algorithm behave if the points are maintained in a collection of regions other than disjoint cubical boxes? The algorithm will run with a similar time bound if the regions satisfy the following three conditions. First, the regions are easy to compute with so that splitting and shrinking a region can be quickly accomplished. Second, the volume of the intersection of any two regions is at most a small fraction of the volume of either region. Third, the ratio of the size of the smallest cube that contains a region to the size of the largest cube that is contained in the region is bounded by a fixed constant. The second and the third conditions would allow us to prove packing lemmas similar to the ones in the paper. Thus hyperrectangles which are balanced, i.e., hyperrectangles which do not deviate too far from the shape of a cube, will suffice. But hyperrectangles that are unbalanced, i.e., flat in some direction and quite long in some other direction will not suffice, since the size of the attractor sets will grow as the hyperrectangles get more and more unbalanced. A scheme for splitting the regions which forced the set of input points in a region to split evenly could lead to unbalanced regions, and so such a scheme (e.g., the splitting scheme used in building a k-dimensional tree) would not be adequate.

Acknowledgments

The author would like to thank Professor Herbert Edelsbrunner for helpful discussions. The author is grateful to Professor Alan Cline for pointing out that the algorithm works in the same asymptotic time complexity for any positive semidefinite metric.

References

1. M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th Annual ACM Symp. Theory Comput.*, 1983, pp. 80–86.
2. J. L. Bentley, Multidimensional divide-and-conquer, *Comm.* ACM **23** (1980), 214–229.
3. J. L. Bentley, D. F. Stanat, and E. H. Williams, Jr., The complexity of finding fixed radius nearest neighbors, *Inform. Process. Lett.* **6** (1977), 209–212.
4. J. L. Bentley, B. Weide, and A. C. Yao, Optimal expected-time algorithms for closest-point problems, *ACM Tran. Math. Software* **6** (1982), 563–579.
5. K. Clarkson, Fast algorithms for the all-nearest-neighbors problem, *Proc. 24th Annual Symp. Found. Comput. Sci.*, 1983, pp. 226–232.
6. H. N. Gabow, J. L. Bentley, and R. E. Tarjan, Scaling and related techniques for geometry problems, *Proc. 16th Annual ACM Symp. Theory Comput.*, 1984, pp. 135–143.
7. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985,
8. E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice Hall, Englewood Cliffs, NJ, 1977.
9. M. I. Shamos, Computational Geometry, Ph.D. dissertation, Yale University, New Haven, CT, 1978.
10. M. O. Rabin, Probabilistic algorithms, in *Algorithms and Complexity* (J. F. Traub, ed.), Academic Press, New York, 1976, pp. 21–30.
11. F. Yuval, Finding nearest neighbors, *Inform. Process. Lett.* **3** (1975), 113–114.

Received July 21, 1986, and in revised form December 23, 1987.