

Two Algorithms for Nearest-Neighbor Search in High Dimensions

Jon M. Kleinberg*

February 7, 1997

Abstract

Representing data as points in a high-dimensional space, so as to use geometric methods for indexing, is an algorithmic technique with a wide array of uses. It is central to a number of areas such as information retrieval, pattern recognition, and statistical data analysis; many of the problems arising in these applications can involve several hundred or several thousand dimensions.

We consider the nearest-neighbor problem for d -dimensional Euclidean space: we wish to pre-process a database of n points so that given a query point, one can efficiently determine its nearest neighbors in the database. There is a large literature on algorithms for this problem, in both the exact and approximate cases. The more sophisticated algorithms typically achieve a query time that is logarithmic in n at the expense of an exponential dependence on the dimension d ; indeed, even the average-case analysis of heuristics such as k - d trees reveals an exponential dependence on d in the query time.

In this work, we develop a new approach to the nearest-neighbor problem, based on a method for combining randomly chosen one-dimensional projections of the underlying point set. From this, we obtain the following two results.

- (i) An algorithm for finding ε -approximate nearest neighbors with a query time of $O((d \log^2 d)(d + \log n))$.
- (ii) An ε -approximate nearest-neighbor algorithm with near-linear storage and a query time that improves asymptotically on linear search in all dimensions.

*IBM Almaden Research Center, San Jose CA 95120, on leave from Department of Computer Science, Cornell University, Ithaca NY 14853.

1 Introduction

The *nearest-neighbor problem* is central to a wide range of areas in which computational techniques are applied. Nearest-neighbor-based methods appear, for example, in algorithms for information retrieval [37, 38, 8, 15], pattern recognition [14, 18], statistics and data analysis [35, 16], data compression [27], and multimedia databases [36, 23, 40]. The pervasiveness of the problem arises in large part because of its effectiveness as a general-purpose means of indexing and comparing data: one represents objects as points in a high-dimensional metric space, and then uses “nearness” with respect to the underlying distance as a means of indexing and similarity-based comparison. While the design of theoretically efficient nearest-neighbor algorithms has been oriented mainly towards a small number of dimensions, many of the core applications of the nearest-neighbor problem take place in a very large number of dimensions. For example, the application of standard vector-space methods in information retrieval [37, 38, 8, 15] can result in problems with several thousand dimensions; much work has been devoted to “dimension-reduction” techniques such as principal component analysis [29] and latent semantic indexing [15], but these typically are not used to reduce the dimension below several hundred [15, 7]. Multimedia retrieval applications such as [23] can also involve several hundred dimensions.

In this paper, we consider one of the most commonly studied settings of the nearest-neighbor problem, both theoretically and in applications: namely, the case of points in \mathbf{R}^d with the Euclidean metric. That is, for points $x = (x_{(1)}, \dots, x_{(d)})$ and $y = (y_{(1)}, \dots, y_{(d)})$, we define the distance between x and y to be

$$d(x, y) = \left(\sum_{i=1}^d (x_{(i)} - y_{(i)})^2 \right)^{(1/2)}.$$

The nearest-neighbor problem in this setting is then the following. We are given a set P of n sites $\{p_1, \dots, p_n\}$; each site p_i is a point in \mathbf{R}^d . We must pre-process P so as to be able to efficiently answer queries of the following form: Given an arbitrary point $q \in \mathbf{R}^d$, return a site in P that is closest to q (with respect to $d(\cdot, \cdot)$).

There is a voluminous literature on algorithms for the Euclidean nearest-neighbor problem, and we do not attempt a comprehensive survey of it here. Establishing upper bounds on the time required to answer a nearest-neighbor query in \mathbf{R}^d appears to have been first undertaken by Dobkin and Lipton [17]; they provided an algorithm with query time $O(2^d \log n)$ and pre-processing $O(n^{2^{d+1}})$. (We use the term “pre-processing” to refer to the sum of the pre-processing time and storage required.) This was improved by Clarkson [11]; he gave an algorithm with query time $O(\exp(d) \cdot \log n)$ and pre-processing $O(n^{\lceil d/2 \rceil (1+\epsilon)})$, where $\exp(d)$ denotes a function that grows at least as quickly as 2^d . Most of the subsequent approaches and extensions (e.g. [42, 32, 2] and others) have required a query time of at least $\Omega(\exp(d) \cdot \log n)$. Indeed, even the average-case analysis of heuristics such as k-d trees [22, 6, 26, 39], for restricted cases such as input points distributed uniformly over a bounded subset of \mathbf{R}^d , reveals an exponential dependence on d in the query time. One exception to this phenomenon is a recent algorithm of Meiser [33] (designed, as are some of the above algorithms, for the more general problem of point location in an arrangement of hyperplanes); it obtains a query time of $O(d^5 \log n)$ with storage $O(n^{d+\epsilon})$.

It is natural to try improving the computational requirements by only looking for an *approximately nearest neighbor* of the query. We say that $p \in P$ is an ε -*approximate nearest neighbor* of q if for all $p' \in P$, we have $d(p, q) \leq (1 + \varepsilon)d(p', q)$. Settling for an approximate solution makes sense for a number of reasons – among other things, the use of fixed-precision arithmetic will typically mean that the above “exact” algorithms are in fact producing answers that are only approximate anyway; and perhaps more importantly, the methods used for mapping features to numerical coordinates in many of the applications cited above (e.g. [18, 37, 23, 36]) have been chosen on heuristic grounds, and so often an “exact” answer is no more valuable than a close approximation.

Finding ε -approximate nearest neighbors, for arbitrarily small $\varepsilon > 0$, has also been studied extensively. Arya, Mount, et al. [5, 4] obtain an algorithm with query time $O(\exp(d) \cdot \varepsilon^{-d} \log n)$ and pre-processing $O(n \log n)$. Clarkson [12] obtained a different algorithm which improves the dependence on ε to $\exp(d) \cdot \varepsilon^{-(d-1)/2}$. Again, these approaches result in query times that grow exponentially with d .

One can make several observations at this point. First, for algorithms with an exponential dependence on d in the query time, one has the following striking state of affairs: the “brute-force” algorithm — which simply computes the distance from the query to every point in P , in time $O(dn)$ — provides a faster query time even theoretically when $d \geq \log n$. Further, when one requires storage costs to be polynomial in n (for variable d), it appears that no algorithms are known with query times that improve on brute-force search once d is comparable to $\log n$. This reflects the well-known “curse of dimensionality” [12] that appears throughout computational geometry; it is particularly unfortunate in the present setting, since the dimension is quite large in many of the applications cited above. To quote from Arya, Mount, et al. [5], “. . . if the dimension is significantly larger than $\log n$ (as it for a number of practical instances), there are no approaches we know of that are significantly faster than brute-force search.”

In this work, we develop a new approach to the nearest-neighbor problem, based on a method for combining randomly chosen one-dimensional projections of the underlying point set P . From this we obtain the following two results.

- (i) A conceptually simple algorithm for finding ε -approximate nearest neighbors with storage $O(n \log d)^{2d}$ and a query time of $O((d \log^2 d)(d + \log n))$.
- (ii) An ε -approximate nearest-neighbor algorithm with near-linear storage and a query time that improves asymptotically on brute-force search in all dimensions.

We now turn to an overview of the algorithms.

Our Results

The First Algorithm. First, we provide an algorithm for ε -approximate nearest neighbors with a query time that is logarithmic in n and polynomial in d and ε^{-1} . Specifically, we show how to answer queries in time

$$O((d \log^2 d)(d + \log n))$$

with storage $O(n \log d)^{2d}$. (For the sake of exposition in this introduction, we use the $O()$ notation to suppress terms that are quadratic in ε^{-1} , but independent of d and n .) Hence the query time of this algorithm is better than that of brute-force search for all dimensions up to $o(n/\log^2 n)$.

The processing of a query is deterministic; however, the initial construction of the data structure is randomized and may fail with a probability that can be made arbitrarily small. (Increasing the pre-processing time by a factor of $\log \delta^{-1}$ reduces the failure probability to $1 - \delta$.) Every query made with the data structure will be correct provided the initial construction does not fail.

With essentially no modification to the data structure, we can return a set of k approximately nearest neighbors. We define this as follows. If $q \in \mathbf{R}^d$, and p_{i_1}, \dots, p_{i_n} is a listing of the points of P in order of increasing distance from q , then we say that the (ordered) set $S = \{s_1, \dots, s_k\} \subset P$ is an ε -approximate set of k nearest neighbors of q if for each $j = 1, \dots, k$, we have $d(s_j, q) \leq (1 + \varepsilon)d(p_{i_j}, q)$. Using the same pre-processing as before, we can return an ε -approximate set of k nearest neighbors with query time $O(k + (d \log^2 d)(d + \log n))$.

The Second Algorithm. Although our first algorithm provides a query time that is polynomial in d and logarithmic in n , its pre-processing requirements are prohibitively large — in particular, they contain d in the exponent. However, the same underlying data structure and search techniques can be used to obtain a second algorithm with storage that is near-linear in the size of P , and a query time that still asymptotically improves on brute-force search in all dimensions. This second algorithm uses randomization for processing a query, and with probability $1 - \delta$ returns an ε -approximate nearest neighbor. It has query time

$$O(n + d \log^3 n),$$

pre-processing time $O(d^2 n \log^2 n (\log^2 d + \log d \log \log n)) = O^*(d^2 n)$, and storage $O(dn \log^2 n (\log^2 d + \log d \log \log n)) = O^*(dn)$. (Here, the $O()$ notation suppresses terms that are quadratic in ε^{-1} and logarithmic in δ^{-1} ; the $O^*()$ notation also suppresses terms that are polynomial in $\log n$. The error probability is taken only over the choices made by the algorithm; not over any presumed distribution on query points.)

Thus, when $d = O(n/\log^3 n)$, we can answer a query with a net *constant* number of operations per site in P , rather than the d operations per site required by the brute-force algorithm. Moreover, in processing a query, this algorithm only computes $O(\log^3 n)$ distances in \mathbf{R}^d . Finally, the data structure underlying this algorithm can be made dynamic very easily; over any period of time in which the current size of P does not grow by more than a polynomial amount, the dynamic version supports insertions into P in time $O^*(d^2)$ and deletions from P in time $O^*(d)$. Provided that the construction of the data structure did not *fail* for the initial point set, it is guaranteed not to fail under an arbitrary sequence of insertions and deletions.

A General Proximity Problem. A number of classical proximity problems in computational geometry are expressed in terms of distance relations among a fixed set of n points, rather than in terms of a potentially infinite universe of queries. Some salient examples are

the *closest pair problem* — given n points, find the pair at minimum distance — and the *bichromatic closest pair problem* — given n red points and n blue points, find the red-blue pair at minimum distance. Such problems are well-solved in a small number of dimensions; but in high dimensions, Cohen and Lewis have recently observed [13] that even for approximate or average-case versions, the trivial $O(dn^2)$ algorithm is the best known if one does not resort to fast matrix multiplication.

We show that our second nearest-neighbor algorithm can be used directly to provide an algorithm for a general proximity problem that includes the closest pair and bichromatic closest pair problems as special cases. Our algorithm provides an ε -approximate answer with probability $1 - \delta$; its running time is $O((\varepsilon^{-2} \log \delta^{-1})(n^2 + nd \log n))$. Thus for all values of d up to $O(n/\log n)$, it provides a running time that is purely quadratic in the number of points, independent of the dimension.

Both nearest-neighbor algorithms are based on the same underlying technique, which we now describe. The basic ideas behind this technique appear to be quite general, and we do not believe that the present work fully exhausts their ramifications for high-dimensional proximity problems. We emphasize at the outset that the basic variants of our two algorithms are considerably simpler than nearly all the previous theoretical approaches described above; they do not require the use of complicated data structures, the manipulation of sets of hyperplanes in d dimensions, or the special handling of “degenerate” cases. As such, the algorithms appear to be simpler to implement even than the original method of Dobkin and Lipton [17].

Techniques and Basic Definitions

We build our data structures from the projections of the set P onto random lines through the origin in \mathbf{R}^d . The use of random projections onto lines has appeared in a number of contexts in recent high-dimensional geometric constructions and algorithms (e.g. [30, 24, 28, 31, 10]); thus it is worth our discussing what we gain from this approach in the setting of the nearest-neighbor problem. First of all, it is useful to note what we *do not* gain from this approach.

(i) Simple examples show that an ε -approximate nearest neighbor of the query need not lie close to it in the projection onto a random line; it can be the case that with high probability, $\Omega(n)$ other points of P will lie between the query and every ε -approximate nearest neighbor.

(ii) A standard application of random projections — the reduction of the dimension of the problem — appears to be of essentially no use in obtaining the bounds we are seeking here. In particular, the strongest result on distance-preserving projections of point sets is due to Frankl and Maehara [24], strengthening a bound of Johnson and Lindenstrauss [30]; they show that projecting P onto a random subspace of dimension roughly $9\varepsilon^{-2} \log n$ preserves all relative inter-point distances to within a factor of $1 + \varepsilon$, with high probability. But, for example, applying an algorithm with query time $\Omega(\exp(d))$ even in this reduced dimension results in a time bound of at least $n^{9\varepsilon^{-2}}$, which is of course much worse than the $O(dn)$ bound for the brute-force algorithm. (One encounters other problems as well; for example, once a random projection has been chosen, there will be a small fraction of queries that are

guaranteed to produce incorrect answers.)

Here, we make use of the following property of random projections onto a line: if x is closer to the query than y , then x has a probability greater than $\frac{1}{2}$ of also being closer to the query in the projection. As a result, with enough projections, we can begin making pairwise comparisons between points of P (with respect to a given query) with high probability. Using these pairwise comparisons, we can then arrive at an approximately nearest neighbor.

Since we are primarily concerned with case of small ε , we assume for the remainder of the paper that $\varepsilon \leq \frac{1}{2}$. The case of larger ε does lead to a performance improvement for the algorithms; the analysis requires only minor modifications from what is presented here.

Both of our algorithms germinate from the following geometric fact. If $x = (x_{(1)}, \dots, x_{(d)})$ and $y = (y_{(1)}, \dots, y_{(d)})$ are vectors in \mathbf{R}^d , we use $x \cdot y$ to denote the inner product of x and y : $x \cdot y = \sum_{i=1}^d x_{(i)}y_{(i)}$. Note that $(x - y) \cdot (x - y)$ is the square of the distance between x and y ; and $\|x\| = \sqrt{x \cdot x}$ is the *norm* of the vector x . Finally, let $S^{d-1} \subset \mathbf{R}^d$ denote the unit $(d - 1)$ -sphere $\{v \in \mathbf{R}^d : \|v\| = 1\}$. Our fact is the following: for $\gamma \leq \frac{1}{2}$, suppose that $(1 + \gamma)\|x\| \leq \|y\|$, and let v be a vector drawn uniformly at random from S^{d-1} . Then the probability that $|v \cdot x| < |v \cdot y|$ is at least $\frac{1}{2} + \frac{\gamma}{3}$. Let us actually be a little more concrete, and use $\mathcal{W}_{x,y}$ to denote the *set* of vectors $v \in S^{d-1}$ for which the event $|v \cdot x| < |v \cdot y|$ fails to happen; we will call this the *exceptional set* for (x, y) .

Suppose that, for points $x, y \in P$ and a query $q \in \mathbf{R}^d$, we want to compare $d(x, q) = \|x - q\|$ with $d(y, q) = \|y - q\|$ by computing their respective inner products with a set V of vectors $v_1, \dots, v_k \in S^{d-1}$, and then determining whether $(x - q)$ or $(y - q)$ has a smaller inner product with a majority of the vectors v_i . If $(1 + \varepsilon)d(x, q) \leq d(y, q)$, then the above experiment will fail only if a majority of the vectors in V lie in the exceptional set $\mathcal{W}_{(x-q), (y-q)}$. Since the measure of $\mathcal{W}_{(x-q), (y-q)}$ is at most $\frac{1}{2} - \frac{\varepsilon}{3}$, this in turn corresponds to a “non-uniformity” in the distribution of the finite set V on S^{d-1} .

Let us say that a finite set $V \subset S^{d-1}$ is ρ -*distinguishing*, for $\rho < \frac{1}{2}$, if strictly fewer than half the elements of V lie in any exceptional set of the form $\mathcal{W}_{x,y}$ that has measure at most ρ . As we have just observed, a ρ -distinguishing set can be used to correctly compare distances to within a factor of $1 + (\frac{2}{3} - 3\rho)$, by the above experiment. In section 2, we use a VC-dimension argument to show that with high probability, a random set of size $O(d\varepsilon^{-2} \log d \log(d\varepsilon^{-2}))$ will be $(\frac{1}{2} - \frac{\varepsilon}{3})$ -distinguishing, which is sufficient for our purposes.

Given this definition, the construction of our data structure *fails* only if a random set that we choose initially turns out not to be $(\frac{1}{2} - \frac{\varepsilon}{3})$ -distinguishing. Note that *failure* does not depend on the underlying point set P . Since the processing of a query is deterministic in our first algorithm, it follows that if one were to know of a $(\frac{1}{2} - \frac{\varepsilon}{3})$ -distinguishing set V in d dimensions, one could use it to process all future queries for any point set with a deterministic guarantee of correctness.

We describe the two algorithms themselves in Sections 3 and 4. The first algorithm uses a distinguishing set V to partition the set of queries into $O(n \log d)^{2d}$ equivalence classes with the following property: for each class, there is an ordering of the points of P so that for all k between 1 and n , the first k points in this ordering constitute an ε -approximate set of k nearest neighbors for all queries in the class. Moreover, the identity of the class containing a query can be determined by performing $O(d \log^2 d)$ inner products and binary searches.

The second algorithm uses a distinguishing set to build an “elimination tournament”

on the set of points; the tournament returns an ε -approximate nearest neighbor with high probability using a linear number of comparisons. We note that the analysis of tournaments with unreliable comparisons has been the subject of a number of previous papers (see e.g. [21, 1] and the references therein); however, the actual models considered in these papers are technically fairly distinct from the constraints imposed by our application here.

2 Some Geometric Lemmas

In this section, we prove some of the geometric lemmas required for the analysis of our algorithms. It is important to note that none of this is needed for the actual implementation of the algorithms, which will be described in a self-contained form in the following two sections.

We make use of the definitions from the previous section. First, we prove the basic geometric fact discussed above.

Lemma 2.1 *Let $x, y \in \mathbf{R}^d$, $\gamma \leq \frac{1}{2}$, and suppose that $(1 + \gamma)\|x\| \leq \|y\|$. Then for v uniformly distributed over S^{d-1} , we have $\Pr[|v \cdot x| \geq |v \cdot y|] \leq \frac{1}{2} - \frac{\gamma}{3}$.*

Proof. Let \mathcal{E} denote the event $|v \cdot x| \geq |v \cdot y|$. Let \mathcal{A} denote the (two-dimensional) affine span of the vectors x and y . For a vector $v \in S^{d-1}$, let $\pi_{\mathcal{A}}(v)$ denote the projection of v onto \mathcal{A} . Observe that

$$\begin{aligned} v \cdot x &= \pi_{\mathcal{A}}(v) \cdot x \\ v \cdot y &= \pi_{\mathcal{A}}(v) \cdot y. \end{aligned}$$

Thus, for given x and y , the outcome of the event \mathcal{E} depends only on the angle φ formed between $\pi_{\mathcal{A}}(v)$ and x (or, equivalently, on the angle formed between $\pi_{\mathcal{A}}(v)$ and y). We also note that the angle φ is uniformly distributed over $[0, 2\pi)$.

Now, if we let $\theta \in [0, \pi]$ denote the angle between x and y in \mathcal{A} and $r = \|x\|/\|y\| \leq \frac{1}{1+\gamma}$, the event \mathcal{E} occurs when $\cos^2(\theta - \varphi) \leq r^2 \cos^2 \varphi$; that is, when

$$\begin{aligned} \frac{\cos^2(\theta - \varphi)}{\cos^2 \varphi} &= \frac{(\cos^2 \theta)(\cos^2 \varphi) - (\sin 2\theta)(\sin \varphi)(\cos \varphi) + (\sin^2 \theta)(\sin^2 \varphi)}{\cos^2 \varphi} \\ &= \cos^2 \theta - (\sin 2\theta)(\tan \varphi) + (\sin^2 \theta)(\tan^2 \varphi) \\ &\leq r^2. \end{aligned}$$

Applying the quadratic formula to the last inequality, this occurs when

$$\tan \varphi \in \left[\frac{\sin 2\theta}{2 \sin^2 \theta} \pm \frac{r \sin \theta}{\sin^2 \theta} \right] = [\cot \theta \pm r \csc \theta].$$

Thus, for a given value of θ and r , we have

$$\Pr[\mathcal{E}] = \begin{cases} \frac{1}{\pi} (\tan^{-1} [\cot \theta + r \csc \theta] - \tan^{-1} [\cot \theta - r \csc \theta]), & \theta \in (0, \pi) \\ 0, & \theta = 0, \pi \end{cases}$$

For fixed r , one can therefore show that $\Pr[\mathcal{E}]$ is maximized over $[0, \pi]$ at $\theta = \frac{\pi}{2}$, since it is differentiable on the open interval $(0, \pi)$, and tedious calculations show that its derivative has precisely one zero in this interval, when $\theta = \frac{\pi}{2}$. Now, when $\theta = \frac{\pi}{2}$, we have

$$\Pr[\mathcal{E}] = \frac{2 \tan^{-1} r}{\pi}.$$

By Taylor's inequality, we have

$$\frac{2 \tan^{-1} r}{\pi} \leq \frac{2\frac{\pi}{4}r}{\pi} = \frac{r}{2} \leq \frac{1}{2} - \frac{\gamma}{3}$$

for $\gamma \leq \frac{1}{2}$. Thus,

$$\Pr[\mathcal{E}] \leq \frac{1}{2} - \frac{\gamma}{3}$$

as required. ■

Recall that we use $\mathcal{W}_{x,y}$ to denote the subset of S^{d-1} for which the event \mathcal{E} in the above lemma occurs, and refer to this as the *exceptional set* for (x, y) . We now want to consider the VC-dimension of the collection of exceptional sets. In order to make this exposition self-contained, we present the following definitions (see e.g. [41, 3] for more background; we work within the more general framework of the former paper). A *range space* is a pair $(\mathcal{P}, \mathcal{R})$, where $\mathcal{P} = (\Omega, \mathcal{F}, \mu)$ is a probability space and $\mathcal{R} \subset \mathcal{F}$ is a collection of measurable subsets of Ω . We say that a finite subset A of Ω is *shattered* by \mathcal{R} if for all $B \subset A$ there exists an $R \in \mathcal{R}$ such that $B = A \cap R$. The *VC-dimension* of $(\mathcal{P}, \mathcal{R})$ is the supremum of the cardinalities of finite subsets of Ω that can be shattered by \mathcal{R} ; hence we say that $(\mathcal{P}, \mathcal{R})$ has infinite VC-dimension if and only if arbitrarily large finite subsets of Ω can be shattered by \mathcal{R} .

We have been working with the sphere S^{d-1} and the uniform distribution; let $\mathcal{P}^{d-1} = (S^{d-1}, \mathcal{F}, \mu)$ denote the associated probability space, and let \mathcal{R}^ρ denote the collection of all $\mathcal{W}_{x,y}$ for which $\mu(\mathcal{W}_{x,y}) \leq \rho$.

Lemma 2.2 *Let $0 < \rho < 1$. The VC-dimension of the range space $(\mathcal{P}^{d-1}, \mathcal{R}^\rho)$ is at most $d' = 8(d+1) \log(4d+4)$.*

Proof. For a vector $u \in \mathbf{R}^d$, let H_u denote the *halfspace* $\{v \in S^{d-1} : u \cdot v \geq 0\}$. Now, for $\mathcal{W}_{x,y} \in \mathcal{R}^\rho$, we have $v \in \mathcal{W}_{x,y}$ if and only if $v \cdot x \geq v \cdot y \geq -v \cdot x$ or $-v \cdot x \geq v \cdot y \geq v \cdot x$. Thus,

$$\mathcal{W}_{x,y} = \left(H_{(x-y)} \cap H_{(x+y)} \right) \cup \left(H_{(-x-y)} \cap H_{(-x+y)} \right).$$

Radon's theorem (see e.g. [20]) implies that the VC-dimension of half-spaces in S^{d-1} is $d+1$. Each range in $(\mathcal{P}^{d-1}, \mathcal{R}^\rho)$ is a Boolean combination of half-spaces with four literals, and thus a theorem of Dudley [19, 3] implies that its VC-dimension is at most $d' = 8(d+1) \log(4d+4)$. ■

A finite subset A of Ω is said to be a γ -*sample* of the range space $(\mathcal{P}, \mathcal{R})$ if for all $R \in \mathcal{R}$, we have

$$\left| \frac{|R \cap A|}{|A|} - \mu(R) \right| \leq \gamma.$$

A theorem of Vapnik and Chervonenkis [41] says, subject to some technical conditions, that if $(\mathcal{P}, \mathcal{R})$ has VC-dimension at most k , for a natural number k , then a finite set A of size

$$\ell \geq \frac{16}{\gamma^2} \left(k \log \frac{16k}{\gamma^2} + \log \frac{4}{\delta} \right)$$

is a γ -sample of $(\mathcal{P}, \mathcal{R})$ with probability at least $1 - \delta$.

We use this theorem directly to obtain the following lemma. Let γ and δ be arbitrary positive constants.

Lemma 2.3 *There is a constant c_0 so that with probability at least $1 - \delta$, a set of $f(\gamma, \delta)$ vectors chosen uniformly at random from S^{d-1} is a $\frac{\gamma}{2}$ -sample of the range space $(\mathcal{P}^{d-1}, \mathcal{R}^\rho)$, where*

$$f(\gamma, \delta) = \frac{c_0}{\gamma^2} \left(d' \log \frac{d'}{\gamma^2} + \log \frac{1}{\delta} \right) = \Theta(d \log^2 d).$$

A consequence of this is the following.

Corollary 2.4 *Let $0 < \gamma < \frac{1}{2}$. With $f(\cdot, \cdot)$ as defined in Lemma 2.3, a set of $f(\gamma, \delta)$ vectors chosen uniformly at random from S^{d-1} is $(\frac{1}{2} - \gamma)$ -distinguishing with probability at least $1 - \delta$.*

3 The First Algorithm

We describe the first of our algorithms, which has query time $O(k + (d \log^2 d)(d + \log n))$ and pre-processing $O(n \log d)^{2d}$.

A Lemma on Digraphs. Before describing the algorithm itself, we require the following digression. By a *complete digraph*, we mean a directed graph with the property that for every pair of vertices i, j , exactly one of the directed edges (i, j) or (j, i) is present. We define an *apex* of a digraph to be a vertex with a directed path of length at most two to every other vertex. We define an *apex ordering* of a digraph to be an ordering i_1, \dots, i_n of the vertices with the property that i_k is an apex of the sub-digraph $G[i_k, i_{k+1}, \dots, i_n]$.

Lemma 3.1 *Every n -node complete digraph has an apex ordering, and such an ordering can be computed in time $O(n^2)$.*

Proof. First of all, it is well-known that every complete digraph G has an apex, and one can be found in time $O(n^2)$. As a proof, observe that a node of G with maximal out-degree is an apex.

To construct an apex ordering, one iteratively deletes an apex and updates the out-degrees of all other nodes. Using a Fibonacci heap [25], deletion can be performed in $O(\log n)$ time per operation, while decreasing the out-degree of each other node can be done in amortized constant time per operation. ■

Building the Data Structure. Recall that we are given a set $P = \{p_1, \dots, p_n\}$ of *sites*, with each $p_i \in \mathbf{R}^d$. For $1 \leq i, j \leq n$, let $p_{ij} = \frac{1}{2}(p_i + p_j)$. (So $p_{ii} = p_i$.) Define $\bar{\varepsilon} = \frac{1}{3}\varepsilon$, and let δ be an arbitrarily small positive *failure probability*. Let c_0 and $f(\cdot, \cdot)$ be as defined in Lemma 2.3, and define

$$L = f\left(\frac{\bar{\varepsilon}}{3}, \delta\right) = \Theta(d \log^2 d).$$

The data structure is built as follows.

- (i) Choose a set V of L vectors v_1, \dots, v_L uniformly at random from S^{d-1} .
- (ii) For each v_ℓ , sort the points p_{ij} according to the value of the inner product $v_\ell \cdot p_{ij}$. Let S_ℓ denote the resulting sorted list, and let Σ denote the set of all L sorted lists.
- (iii) Define an *interval* in list S_ℓ to be a pair of entries in S_ℓ (with dummy entries inserted at the beginning and end of S_ℓ), and a *primitive interval* to be an interval whose endpoints are adjacent in S_ℓ . Define a *trace* to be a sequence $\sigma = \sigma_1 \cdots \sigma_L$ of primitive intervals, one for each list in Σ . Note that there are at most $n^{2L} = n^{O(d \log^2 d)}$ traces.

Say that a trace $\sigma = \sigma_1 \cdots \sigma_L$ is *realizable* if there exists a point $q \in \mathbf{R}^d$ such that for each ℓ , $v_\ell \cdot q$ lies in the primitive interval σ_ℓ of S_ℓ . Below, we show that at most $O(n \log d)^{2d}$ traces are realizable, and we give a method for enumerating them.

- (iv) For each realizable trace $\sigma = \sigma_1 \cdots \sigma_L$, do the following.
 - (a) Say that p_i *σ -dominates* p_j in S_ℓ if the entry p_{ij} lies between the primitive interval σ_ℓ and the entry p_j in S_ℓ . Otherwise, p_j σ -dominates p_i . Build a complete digraph G_σ on the set $\{1, \dots, n\}$, with a directed edge from i to $j > i$ if p_i σ -dominates p_j in more than half the lists in Σ , and an edge from j to i otherwise.
 - (b) Construct an apex ordering S_σ^* of the nodes of G_σ . Store the pair (σ, S_σ^*) .

Processing a Query. Now we describe the algorithm for producing an ε -approximate set of nearest neighbors of cardinality k , for a given query point $q \in \mathbf{R}^d$.

- (i) For each ℓ , use binary search to insert the value $v_\ell \cdot q$ into the sorted list S_ℓ . Let σ_ℓ denote the primitive interval into which it falls. (If it falls on the boundary of two primitive intervals, arbitrarily assign it to one of them.)
- (ii) Set $\sigma = \sigma_1 \cdots \sigma_L$. Look up the pair (σ, S_σ^*) and return the first k entries in the sorted list S_σ^* .

Complexity. The running time for a query is the time for (1) L inner products in \mathbf{R}^d , (2) L binary searches, (3) a look-up in a table of size n^{2L} , and (4) the reading of the first k entries in a sorted list. Thus, the total query time is

$$O(Ld + L \log n + k) = O(k + (d \log^2 d)(d + \log n)).$$

The pre-processing and storage requirements are simply the following: for each trace σ that we store, we must compute the digraph G_σ in time $O(Ln^2)$, and we must store a

sorted list S_σ^* of length $O(n)$. Now, there are clearly $O(n^{2L})$ possible traces, and the cleanest approach conceptually would be to store them all. However, we now show that the number of *realizable traces* is significantly smaller.

Lemma 3.2 *The number of realizable traces is at most $\sum_{i=0}^d \binom{Ln^2}{i} = O(n \log d)^{2d}$.*

Proof. A realizable trace corresponds to a full-dimensional cell in the arrangement of hyperplanes

$$\{v_\ell \cdot x = v_\ell \cdot p_{ij} : 1 \leq \ell \leq L, 1 \leq i, j \leq n\}.$$

By a standard result on hyperplane arrangements (see e.g. [20, Thm 1.3]), the number of such full-dimensional cells is at most

$$\sum_{i=0}^d \binom{Ln^2}{i} < 2 \binom{Ln^2}{d} < 2 \left(\frac{eLn^2}{d} \right)^d = O(n \log d)^{2d}.$$

(One can obtain a somewhat tighter bound in the present case by making use of the fact that many of the pairs of hyperplanes are parallel; however, this improvement does not affect the overall asymptotic bound that we obtain.) ■

The set of realizable traces can be determined by constructing the arrangement in the proof of Lemma 3.2, and enumerating all its full-dimensional cells. For this one could use the algorithm given in [20], with a running time of $O(L^d n^{2d})$.

Correctness. Finally, let us show that the algorithm correctly answers approximate nearest-neighbor queries. We say that the construction of the data structure *succeeds* if the set $V = \{v_1, \dots, v_L\}$ is $(\frac{1}{2} - \frac{\varepsilon}{3})$ -distinguishing. By Corollary 2.4, this occurs with probability at least $1 - \delta$.

Theorem 3.3 *If the construction of the data structure succeeds, then for every $q \in \mathbf{R}^d$, the set returned in response to the query q is an ε -approximate set of nearest neighbors of q .*

Proof. Fix $q \in \mathbf{R}^d$, and suppose that the trace generated in processing the query q is σ . Let $S = s_1, \dots, s_k$ denote the first k sites in the sorted list S_σ^* . If S is not an ε -approximate set of nearest neighbors, it would follow that there exists an index m between 1 and k , and a set of points $P' \subset P$ of cardinality m , such that

$$(1 + \varepsilon) \max_{p \in P'} d(p, q) < d(s_m, q).$$

Hence, there would exist a point $p \in P$ such that $p \notin \{s_1, \dots, s_m\}$ and $(1 + \varepsilon)d(p, q) < d(s_m, q)$.

In the original labeling of the points of P , suppose that $s_m = p_i$ and $p = p_j$. Now, since S_σ^* is an apex ordering of G_σ , and since $s_m = p_i$ precedes $p = p_j$ in S_σ^* , it follows that there is a path of length at most two from i to j in G_σ ; hence the edge (i, j) is present in G_σ , or there is a path consisting of the edges $(i, \ell), (\ell, j)$ for a vertex $\ell \neq i, j$. We consider the latter case, the former being easier.

Consider the points p_i and p_ℓ . Since there is an edge (i, ℓ) in G_σ , we know that p_i σ -dominates p_ℓ in at least half the lists of Σ . Thus, for at least half the points $v_r \in V$, we have $|v_r \cdot q - v_r \cdot p_i| \leq |v_r \cdot q - v_r \cdot p_\ell|$; i.e. at least half the points in V lie in the set $\mathcal{W}_{(p_\ell - q), (p_i - q)}$. Now if it were the case that $(1 + \bar{\varepsilon})d(p_\ell, q) < d(p_i, q)$, then by Lemma 2.1 the exceptional set $\mathcal{W}_{(p_\ell - q), (p_i - q)}$ would have measure at most $\frac{1}{2} - \frac{\bar{\varepsilon}}{3}$, and this would contradict the assumption that V is $(\frac{1}{2} - \frac{\bar{\varepsilon}}{3})$ -distinguishing. Hence it follows that

$$d(p_i, q) \leq (1 + \bar{\varepsilon})d(p_\ell, q).$$

By the same reasoning, we also have

$$d(p_\ell, q) \leq (1 + \bar{\varepsilon})d(p_j, q).$$

Combining these two inequalities, we obtain $d(p_i, q) \leq (1 + \bar{\varepsilon})^2 d(p_j, q) \leq (1 + \varepsilon)d(p_j, q)$, which contradicts our initial assumption about p_i and p_j . ■

4 The Second Algorithm

We describe the second of our algorithms, which has query time $O(n + d \log^3 n)$, preprocessing time $O^*(d^2 n)$, and storage $O^*(dn)$. We use the constants defined in the previous section, as well as the following additional constants.

- γ_2 is chosen so that $e^{\gamma_2} \leq 1 + \varepsilon$.
- c_1 is chosen so that $e^{-\frac{1}{64}c_1\gamma_2^2} \leq \frac{1}{4}(\frac{1}{3}\delta)^{(1/\log n)}$.
- c_2 is chosen so that $e^{-\frac{1}{64}c_2\varepsilon^2} \leq \frac{1}{2}$.
- c'_2 is chosen so that $e^{-\frac{1}{64}c'_2\varepsilon^2} \leq \frac{1}{3}\delta$.
- γ_1 is chosen so that $\left(1 - \frac{c_1 \log^3 n}{n}\right)^{\frac{\gamma_1 n}{\log^3 n}} \geq 1 - \frac{1}{3}\delta$.
- c_3 is chosen so that $\left(1 - \frac{\gamma_1}{\log^3 n}\right)^{c_3 \log^3 n} \leq \delta$.

Building the Data Structure. We set $\varepsilon_0 = \frac{\gamma_2}{\log n}$ and define

$$L = f\left(\frac{1}{6}\varepsilon_0, \delta\right) = \Theta(d \log^2 n (\log^2 d + \log d \log \log n)).$$

As in the previous section we choose a set V of L vectors v_1, \dots, v_L . We will assume for the remainder of the section that V is a $\frac{1}{12}\varepsilon_0$ -sample for the range space $(\mathcal{P}^{d-1}, \mathcal{R}^{\frac{1}{2} - \frac{1}{6}\varepsilon_0})$, which by Lemma 2.3 happens with probability at least $1 - \delta$.

Our data structure is simply an $L \times n$ matrix M ; the entry $M[i, j]$ is set equal to $v_i \cdot p_j$.

By Lemma 2.1 and our assumption that V is a $\frac{1}{12}\varepsilon_0$ -sample, we have the following property of the data structure.

Lemma 4.1 *Let $q \in \mathbf{R}^d$, $p_i, p_j \in P$, $\varepsilon_0 \leq \gamma \leq \frac{1}{2}$, and suppose that $(1 + \gamma)d(p_i, q) \leq d(p_j, q)$. Let v_k be chosen uniformly at random from V . The probability that $|v_k \cdot (p_i - q)| \geq |v \cdot (p_j - q)|$ is at most $\frac{1}{2} - \frac{\gamma}{3} + \frac{\varepsilon_0}{12} \leq \frac{1}{2} - \frac{\gamma}{4}$.*

Processing a Query. Now we describe the algorithm for processing a query $q \in \mathbf{R}^d$. First, let p^* denote a point which minimizes $d(p, q)$ over all $p \in P$. Let Z denote the set

$$\{p_i \in P : d(p_i, q) \leq (1 + \varepsilon)d(p^*, q)\}.$$

So the goal of the algorithm is to output a point in Z with probability $1 - \delta$.

If $p_i, p_j \in P$ and $v_k \in V$, we say that p_i *dominates* p_j with respect to v_k (written $p_i \preceq_k p_j$) if

$$|v_k \cdot p_i - v_k \cdot q| < |v_k \cdot p_j - v_k \cdot q|.$$

Now, if $V' \subset V$, we say that p_i *dominates* p_j with respect to V' (written $p_i \preceq_{V'} p_j$) if $p_i \preceq_k p_j$ for strictly more than half the vectors in V' . The operation of deciding whether $p_i \preceq_{V'} p_j$ will be referred to as a V' -comparison of p_i and p_j ; if $p_i \preceq_{V'} p_j$ then we say that p_i *wins* the V' -comparison.

The algorithm is as follows. We assume for simplicity that n is a power of 2; that is, $n = 2^m$. Let T be a complete binary tree of depth m . Let T_h denote the set of nodes of T at height $h \leq m$; the leaves are at height 0.

Let $L_1 = c_1 \log^3 n$. We initially choose a multiset Γ of L_1 vectors from V , drawing each uniformly at random with replacement. We compute the inner product of q with each $v \in \Gamma$. (We note that if $L_1 \geq L$, then one can alternately use the entire set V in place of Γ ; however, the method given here leads to a cleaner analysis.) We assume for simplicity that L_1 is a power of 2 and write $b = \log L_1 = \Theta(\log \log n)$.

The algorithm consists of two phases:

Phase A

- To each node x at height $h \leq b$, we assign a randomly chosen sub-multiset Γ_x of Γ of size $c'_2 + c_2 h$.
- We randomly assign each point of P to a distinct leaf of T .
- Working from the leaves of T up to height b , we assign a point of P to $x \in T$ by Γ_x -comparing the points assigned to the two children of x and assigning the winner to x .
- Starting at height b , we perform the same procedure using Γ -comparisons.
- Let \tilde{p}_A denote the point that is assigned to the root of T .

Phase B

- We randomly choose a set $P' \subset P$ of size $c_3 \log^3 n$.
- We compute the distance from q to each $p \in P'$, and define \tilde{p}_B to be the point of P' whose distance to q is the smallest.

Finally, we determine which of \tilde{p}_A and \tilde{p}_B is closer to q , and return this point as our answer to the query.

Complexity. Each entry of the matrix M requires time $O(d)$ to compute, for a total time of $O(dLn) = O^*(d^2n)$. The space required to store M is simply $Ln = O^*(dn)$.

To process a query, we first compute $v \cdot q$ for each $v \in \Gamma$; this requires time $O(d \log^3 n)$. Phase B also takes time $O(d \log^3 n)$. Phase A consists simply of determining $p_i \preceq_k p_j$ for various indices i, j, k . Each comparison takes constant time, since the values of $v_k \cdot p_i$ and $v_k \cdot p_j$ are stored in M and the values of $v_k \cdot q$ are computed initially.¹ The total number of comparisons is

$$\begin{aligned} & \sum_{h=b+1}^m L_1 |T_h| + \sum_{h=0}^b (c'_2 + c_2 h) |T_h| \\ &= c_1 \log^3 n \sum_{h=b+1}^m \frac{n}{2^{h-b} c_1 \log^3 n} + c'_2 n \sum_{h=0}^b \frac{1}{2^h} + c_2 n \sum_{h=0}^b \frac{h}{2^h} \\ &\leq n + 2c'_2 n + 2c_2 n \\ &= O(n). \end{aligned}$$

Thus the total running time to process a query is $O(n + d \log^3 n)$.

Correctness. We will be making use of the following tail inequality (see e.g. [34]).

Lemma 4.2 *Let X_1, \dots, X_r be i.i.d. Bernoulli trials with success probability p , let $X = \sum_i X_i$, and let $\mu = rp$. Then*

$$\Pr[X < (1 - \gamma)\mu] < e^{-\frac{1}{2}\mu\gamma^2}.$$

A corollary of this is the following.

Corollary 4.3 *Let X_1, \dots, X_r be i.i.d. Bernoulli trials with success probability $\frac{1}{2} + \gamma$, where $0 < \gamma < \frac{1}{2}$. Let $X = \sum_i X_i$ and $\mu = rp$. Then*

$$\Pr\left[X \leq \frac{1}{2}r\right] < e^{-\frac{1}{2}\mu\gamma^2}.$$

Proof.

$$\Pr\left[X \leq \frac{1}{2}r\right] \leq \Pr\left[X < (1 - \gamma)\left(\frac{1}{2} + \gamma\right)r\right] = \Pr[X < (1 - \gamma)\mu] < e^{-\frac{1}{2}\mu\gamma^2}.$$

■

Combining this bound with Lemma 4.1, we have

¹We use the standard assumption (see e.g. Motwani and Raghavan's book [34]) of a RAM model in which one can access and randomly choose array indices of polynomial size in constant time.

Lemma 4.4 *Let $(1 + \gamma)d(p_i, q) \leq d(p_j, q)$. Let Γ' be a randomly chosen sub-multiset of Γ . The probability that $p_j \preceq_{\Gamma'} p_i$ is at most $e^{-\frac{1}{64}|\Gamma'|\gamma^2}$.*

In analyzing the algorithm, we consider two cases separately. The first is easier.

Case 1: $|Z| \geq \gamma_1 n / \log^3 n$. The probability that a member of Z enters the random set P' in Phase B is, by the definition of c_3 , at least $1 - \delta$. Hence we have

Lemma 4.5 *Given that we are in Case 1, the probability that an element of Z is returned is at least $1 - \delta$.*

Case 2: $|Z| \leq \gamma_1 n / \log^3 n$. Let \mathcal{E}_1 denote the event that there exist p_i and p_j such that $(1 + \gamma_2 / \log n)d(p_i, q) \leq d(p_j, q)$ and $p_j \preceq_{\Gamma} p_i$. A consequence of Lemma 4.4 and the definition of c_1 is the following.

Corollary 4.6 *The probability of event \mathcal{E}_1 is at most $\frac{1}{3}\delta$.*

Let P_1 denote the set of points that are assigned to nodes in T_b . Let \mathcal{E}_2 denote the event that $p^* \notin P_1$.

Lemma 4.7 *The probability of \mathcal{E}_2 is at most $\frac{2}{3}\delta$.*

Proof. Let y denote the node of T to which p^* is assigned, and let x denote the node of T_b that lies on the path from y to the root. Let T^x denote the subtree of T rooted at x , and \mathcal{E}_3 denote the event that some node of $Z \setminus \{p^*\}$ is assigned to a leaf of T^x . By the definition of the constant γ_1 , the probability of \mathcal{E}_3 is at most $\frac{1}{3}\delta$.

We now show that if \mathcal{E}_3 does not occur, then the probability that p^* is assigned to node x (and hence placed in P_1) is at least $1 - \frac{1}{3}\delta$. To see this, suppose that p^* has been assigned to a node z at height h in T^x , and let z' be the parent of z . The probability that x is not assigned to z' is, by Lemma 4.4, at most

$$\begin{aligned} e^{-\frac{1}{64}\varepsilon^2(c'_2 + c_2 h)} &= e^{-\frac{1}{64}\varepsilon^2 c'_2} \left(e^{-\frac{1}{64}\varepsilon^2 c_2} \right)^h \\ &\leq \frac{\frac{1}{3}\delta}{2^h}. \end{aligned}$$

Thus, summing over all the nodes on the path from x to y , we obtain the claimed bound. ■

Finally, we have

Lemma 4.8 *Given that we are in Case 2, the probability that an element of Z is returned is at least $1 - \delta$.*

Proof. We show that if events \mathcal{E}_1 and \mathcal{E}_2 do not occur, then an element of Z will be returned. To show this, we prove by induction on $h \geq b$ that there is a point p_{j_h} assigned to a node in T_h for which

$$\frac{d(p_{j_h}, q)}{d(p^*, q)} \leq \left(1 + \frac{\gamma_2}{\log n}\right)^{h-b}.$$

From this it will follow that the point p_{j_m} assigned to the root satisfies

$$\frac{d(p_{j_m}, q)}{d(p^*, q)} \leq \left(1 + \frac{\gamma_2}{\log n}\right)^{\log n} \leq e^{\gamma_2} \leq 1 + \varepsilon,$$

and hence $p_{j_m} \in Z$ as desired.

We start the induction by observing that we assumed \mathcal{E}_2 did not occur; hence when $h = b$, we can take $p_{j_h} = p^*$. Now suppose the induction hypothesis holds for a given value of h ; let us suppose that p_{j_h} is assigned to the node $x_h \in T_h$. Let x'_h denote the sibling of x_h , and y_{h+1} denote their parent. Let p'_h denote the point assigned to x'_h . Now if $d(p'_h, q)/d(p_{j_h}, q) > (1 + \gamma_2/\log n)$, then we use the assumption that event \mathcal{E}_1 did not occur to infer that p_{j_h} will be assigned to y_{h+1} . And if $d(p'_h, q)/d(p_{j_h}, q) \leq (1 + \gamma_2/\log n)$, then either of p_{j_h} or p'_h would satisfy the induction hypothesis for $h + 1$, and we know that one of them will be assigned to y_{h+1} . ■

Combining Lemmas 4.5 and 4.8, we have

Theorem 4.9 *The probability that an element of Z is returned by the algorithm is at least $1 - \delta$.*

5 A General Proximity Problem

In the introduction, we mentioned two classical proximity problems of computational geometry: *closest pair* and *bichromatic closest pair*. In this section, we consider a common generalization of the two, which we call the G -proximity problem. We are given a set of points $P = \{p_1, \dots, p_n\}$ in \mathbf{R}^d , and an undirected graph G on the vertex set P . The goal is the following: over all pairs of points in P that correspond to edges in G , find the pair at minimum distance. Thus the closest pair problem is the special case in which G is the complete graph; and the bichromatic closest pair problem is the special case in which G is a complete bipartite graph.

We show how the algorithm of the previous section can be used directly to obtain an ε -approximate algorithm for the G -proximity problem. The running time will be quadratic in the number of points and independent of d , provided $d = O(n/\log n)$; thus for high dimensions it provides an improvement on the brute-force $O(dn^2)$ algorithm.

Let $\bar{\varepsilon} = \frac{1}{3}\varepsilon$. The algorithm proceeds as follows.

(1) First, by a result of Frankl and Maehara [24], we can project P into a random subspace of dimension $O(\varepsilon^{-2} \log n)$ and preserve all relative inter-point distances to within a factor of $1 + \bar{\varepsilon}$, with high probability. The time to do this is $O(\varepsilon^{-2} nd \log n)$, after which we may assume that $d = O(\varepsilon^{-2} \log n)$.

(2) Next, we perform the pre-processing step of the algorithm from Section 4, with the approximation parameter set to $\bar{\varepsilon}$. From this we obtain an $L \times n$ matrix of inner products.

(3) For each $p_i \in P$, let $\Delta_i \subset P$ denote the set of neighbors of i in the graph G . We perform an approximate nearest-neighbor query using p_i as the query point and Δ_i as the underlying set of sites; in time $O((\varepsilon^{-2} \log \delta^{-1})n)$, we obtain an answer $q_i \in \Delta_i$.

(4) Finally, for each of the pairs (p_i, q_i) produced in the previous step, we compute $d(p_i, q_i)$ and return the pair at minimum distance.

We claim that with probability $1 - \delta$, the resulting pair is an ε -approximate answer to the G -proximity problem. To see this, suppose that (p_i, p_j) is a correct (exact) answer to the G -proximity problem. Then by the performance guarantee of the algorithm from Section 4, the pair (p_i, q_i) returned in step (3) satisfies $d(p_i, q_i) \leq (1 + \bar{\varepsilon})d(p_i, p_j)$ with probability $1 - \delta$. Finally, the pair (p^*, q^*) returned in step (4) satisfies $d(p^*, q^*) \leq d(p_i, q_i)$, from which the claim follows. Thus we have

Theorem 5.1 *The above algorithm has running time $O((\varepsilon^{-2} \log \delta^{-1})(n^2 + nd \log n))$, and with probability at least $1 - \delta$ it returns an ε -approximate answer to the G -proximity problem.*

Acknowledgements

We thank Eli Upfal for a number of discussions on the nearest-neighbor problem. We thank Jeremy Bem and Robert Kleinberg for discussions leading to the current analysis of the storage requirements of the first algorithm. Thanks also to Prabhakar Raghavan, Nimrod Megiddo, Monika Henzinger, and Pankaj Agarwal for their comments.

References

- [1] M. Adler, P. Gemmell, M. Harchol, R.M. Karp, C. Kenyon, “Selection in the presence of noise: the design of playoff systems,” *Proc. 5th ACM-SIAM SODA*, 1994.
- [2] P.K. Agarwal, J. Matousek, “Ray shooting and parametric search,” *Proc. 24th ACM STOC*, 1992.
- [3] N. Alon, J. Spencer, *The Probabilistic Method*, Wiley, 1992.
- [4] S. Arya, *Nearest Neighbor Searching and Applications*, PhD thesis, University of Maryland technical report CS-TR-3490, June 1995.
- [5] S. Arya, D. Mount, N. Netanyahu, R. Silverman, A. Wu, “An optimal algorithm for approximate nearest neighbor searching in fixed dimensions,” *Proc. 5th ACM-SIAM SODA*, 1994. Extended version appears as University of Maryland technical report CS-TR-3568, December 1995.
- [6] J.L. Bentley, “Multidimensional binary search trees used for associative searching,” *Comm. ACM*, 18(1975), pp. 509–517.
- [7] M.W. Berry, S.T. Dumais, A.T. Shippy, “A case study of latent semantic indexing,” U.T. Knoxville technical report CS-95-271, January 1995.
- [8] C. Buckley, A. Singhal, M. Mitra, and G. Salton, “New Retrieval Approaches Using SMART: TREC 4,” *Proceedings of the Fourth Text Retrieval Conference*, National Institute of Standards and Technology, 1995.

- [9] P.B. Callahan, S.R. Kosaraju, “A decomposition of multi-dimensional point sets with applications to k -nearest-neighbors and n -body potential fields,” *Proc. 24th ACM STOC*, 1992.
- [10] B. Chor, M. Sudan, “A geometric approach to betweenness,” *Proc. 3rd European Symposium on Algorithms*, 1995.
- [11] K. Clarkson, “A randomized algorithm for closest-point queries,” *SIAM J. Computing*, 17(1988), pp. 830–847.
- [12] K. Clarkson, “An algorithm for approximate closest-point queries,” *Proc. 10th ACM Symp. on Computational Geometry*, 1994.
- [13] E. Cohen, D. Lewis, “Approximating matrix multiplication for pattern recognition tasks,” *Proc. 8th ACM-SIAM SODA*, 1997.
- [14] T.M. Cover, P.E. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, 13(1967), pp. 21–27.
- [15] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, R. Harshman, “Indexing by latent semantic analysis,” *J. Soc. Info. Sci.*, 41(1990), pp. 391–407.
- [16] L. Devroye, T.J. Wagner, “Nearest neighbor methods in discrimination,” *Handbook of Statistics*, vol. 2, P.R. Krishnaiah, L.N. Kanal, eds., North-Holland, 1982.
- [17] D. Dobkin, R. Lipton, “Multidimensional search problems,” *SIAM J. Computing*, 5(1976), pp. 181–186.
- [18] R.O. Duda, P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley, 1973.
- [19] R.M. Dudley, “Central limit theorems for empirical measures,” *Annals of Prob.*, 6(1978), pp. 899–929.
- [20] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer, 1987.
- [21] U. Feige, D. Peleg, P. Raghavan, E. Upfal, “Computing with unreliable information,” *Proc. 22nd ACM STOC*, 1990.
- [22] R.A. Finkel, J.L. Bentley, “Quad trees – a data structure for retrieval on composite keys,” *Acta Inform.* 4(1974), pp. 1–9.
- [23] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, P. Yanker “Query by image and video content: the QBIC system,” *IEEE Computer* 28(1995), pp. 23–32.
- [24] P. Frankl, H. Maehara, “The Johnson-Lindenstrauss lemma and the sphericity of some graphs,” *J. Combinatorial Theory Ser. B*, 44(1988), pp. 355–362.
- [25] M.L. Fredman, R.E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the ACM* 34(1987), pp. 596–615.

- [26] J.K. Friedman, J.L. Bentley, R.A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” *ACM Trans. on Math. Software*, 3(1977), pp. 209–226.
- [27] A. Gersho, R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, 1991.
- [28] M. Goemans, D.P. Williamson, “Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming,” *Journal of the ACM*, 42(1995), pp. 1115–1145.
- [29] H. Hotelling, “Analysis of a complex of statistical variables into principal components,” *J. Educational Psychology*, 27(1933), pp. 417–441.
- [30] W.B. Johnson, J. Lindenstrauss, “Extensions of Lipschitz mappings into Hilbert space,” *Contemporary Mathematics* 26(1984), pp. 189–206.
- [31] D. Karger, R. Motwani and M. Sudan, “Approximate graph coloring by semidefinite programming,” *Proc. 35th IEEE Symposium on Foundations of Computer Science*, 1994, pp. 2–13.
- [32] J. Matousek, “Reporting points in halfspaces,” *Proc. 32nd IEEE FOCS*, 1991.
- [33] S. Meiser, “Point location in arrangements of hyperplanes,” *Information and Computation*, 106(1993), pp. 286–303.
- [34] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
- [35] Panel on Discriminant Analysis and Clustering, National Research Council, *Discriminant Analysis and Clustering*, National Academy Press, 1988.
- [36] A. Pentland, R.W. Picard, and S. Sclaroff, “Photobook: tools for content-based manipulation of image databases,” *Proceedings of the SPIE Conference on Storage and Retrieval of Image and Video Databases II*, 1994.
- [37] C.J. van Rijsbergen, *Information Retrieval*, Butterworths, 1979. Also at <http://dcs.glasgow.ac.uk/Keith/Preface.html>.
- [38] G. Salton. *Automatic Text Processing*. Addison-Wesley, Reading, MA, 1989.
- [39] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1989.
- [40] A.W.M. Smeulders and R. Jain, editors. *Image Databases and Multi-media Search*. Proceedings of the First International Workshop, IDB-MMS '96, Amsterdam. Amsterdam University Press, 1996.
- [41] V.N. Vapnik, A.Y. Chervonenkis, “On the uniform convergence of relative frequencies of events to their probabilities,” *Theory of Prob. App.*, 16(1971), pp. 264–280.
- [42] A.C. Yao, F.F. Yao, “A general approach to d -dimensional geometric queries,” *Proc. 17th ACM STOC*, 1985.