Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality

(preliminary version)

Piotr Indyk* Rajeev Motwani†

Department of Computer Science

Stanford University

Stanford, CA 94305

{indyk,rajeev}@cs.stanford.edu

December 30, 1999

Abstract

The *nearest neighbor* problem is the following: Given a set of n points $P = \{p_1, \ldots, p_n\}$ in some metric space X, preprocess P so as to efficiently answer queries which require finding the point in P closest to a query point $q \in X$. We focus on the particularly interesting case of the d-dimensional Euclidean space where $X = \Re^d$ under some l_p norm. Despite decades of effort, the current solutions are far from satisfactory; in fact, for large d, in theory or in practice, they provide little improvement over the brute-force algorithm which compares the query point to each data point. Of late, there has been some interest in the *approximate nearest neighbors* problem, which is: Find a point $p \in P$ that is an ϵ-approximate nearest neighbor of the query q in that for all $p' \in P$, $d(p,q) \leq (1+\epsilon)d(p',q)$.

We present two algorithmic results for the approximate version that significantly improve the known bounds: (a) preprocessing cost polynomial in n and d, and a truly sublinear query time; and, (b) query time polynomial in $\log n$ and d, and only a mildly exponential preprocessing cost $\tilde{O}(n) \times O(1/\epsilon)^d$. Further, applying a classical geometric lemma on random projections (for which we give a simpler proof), we obtain the first known algorithm with polynomial preprocessing and query time polynomial in d and $\log n$. Unfortunately, for small ϵ, the latter is a purely theoretical result since the exponent depends on $1/\epsilon$. Experimental results indicate that our first algorithm offers orders of magnitude improvement on running times over real data sets. Its key ingredient is the notion of *locality-sensitive hashing* which may be of independent interest; here, we give applications to information retrieval, pattern recognition, dynamic closest-pairs, and fast clustering algorithms.

*Supported by a Stanford Graduate Fellowship and NSF Award CCR-9357849.

†Supported by a Sloan Fellowship, an IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR–9357849.

1 Introduction

The **nearest neighbor search (NNS)** problem is: Given a set of n points $P = \{p_1, \ldots, p_n\}$ in a metric space X with distance function d, preprocess P so as to efficiently answer queries for finding the point in P closest to a query point $q \in X$. We focus on the particularly interesting case of the d-dimensional Euclidean space where $X = \Re^d$ under some l_p norm. The low-dimensional case is well-solved [27], so the main issue is that of dealing with the "curse of dimensionality" [17]. The problem was originally posed in the 1960s by Minsky and Papert [54, pp. 222–225], and despite decades of effort the current solutions are far from satisfactory. In fact, for large d, in theory or in practice, they provide little improvement over a brute-force algorithm which compares a query q to each $p \in P$. The known algorithms are of two types: (a) low preprocessing cost but query time linear in n and d; and, (b) query time sublinear in n and polynomial in d, but with severely exponential preprocessing cost n^d. This unfortunate situation carries over to average-case analysis, and even to the **ϵ-approximate nearest neighbors (ϵ-NNS)** problem: Find a point $p \in P$ that is an ϵ-approximate nearest neighbor of the query q, in that for all $p' \in P$, $d(p, q) \leq (1 + \epsilon)d(p', q)$.

We present two algorithms for the approximate version that significantly improve the known bounds: (a) preprocessing cost polynomial in n and d, and a truly sublinear query time; and, (b) query time polynomial in $\log n$ and d, and only a mildly exponential preprocessing cost $\tilde{O}(n) \times O(1/\epsilon)^d$. Further, by applying a classical geometric lemma on random projections (for which we give a simpler proof), we obtain the first known algorithm with polynomial preprocessing and query time polynomial in d and $\log n$. Unfortunately, for small ϵ, this is a purely theoretical result as the exponent depends on $1/\epsilon$. Experimental results [37] indicate that the first algorithm offers orders of magnitude improvement on running times over real data sets. Its key ingredient is the notion of *locality-sensitive hashing* which may be of independent interest; we give applications to information retrieval, pattern recognition, dynamic closest-pairs, and fast clustering.

Motivation. The nearest neighbors problem is of major importance to a variety of applications, usually involving similarity searching. Some examples are: data compression [36]; databases and data mining [13, 39]; information retrieval [11, 21, 58]; image and video databases [29, 31, 56, 61]; machine learning [19]; pattern recognition [20, 26]; and, statistics and data analysis [22, 45]. Typically, the features of the objects of interest (documents, images, etc) are represented as points in $\Re^d$ and a distance metric is used to measure (dis)similarity of objects. The basic problem then is to perform indexing or similarity searching for query objects. The number of features (i.e., the dimensionality) ranges anywhere from tens to thousands. For example, in multimedia applications such as IBM's QBIC (Query by Image Content), the number of features could be several hundreds [29, 31]. In information retrieval for text documents, vector-space representations involve several thousands of dimensions, and it is considered to be a dramatic improvement that dimension-reduction techniques, such as LSI (latent semantic indexing) [9, 11, 21], principal components analysis [40] or the Karhunen-Loéve transform [44, 50], can reduce the dimensionality to a mere few hundreds!

Of late, there has been an increasing interest in avoiding the curse of dimensionality by resorting to *approximate* nearest neighbor searching. Since the selection of features and the use of a distance metric in the applications are rather heuristic and merely an attempt to make mathematically precise what is after all an essentially aesthetic notion of similarity, it seems like an overkill to insist on the absolute nearest neighbor; in fact, determining an ϵ-approximate nearest neighbor for a reasonable value of ϵ, say a small constant, should suffice for most practical purposes. Unfortunately, even this relaxation of goals has not removed the curse of dimensionality, although the recent results of Kleinberg [46] gives some improvements.

Previous Work. Samet [59] surveys a variety of data structures for nearest neighbors including variants of k-d trees, R-trees, and structures based on space-filling curves; more recent results are surveyed in [60]. While some perform well in 2-3 dimensions, in high-dimensional spaces they all exhibit poor behavior in the worst case and in typical cases as well (e.g., see Arya, Mount, and Narayan [4]). Dobkin and Lipton [23] were the first to provide an algorithm for nearest neighbors in $\Re^d$, with query time $O(2^d \log n)$ and preprocessing[1] cost $O(n^{2^{d+1}})$. Clarkson [16] reduced the preprocessing to $O(n^{\lceil d/2 \rceil (1+\delta)})$, while increasing the query time to $O(2^{O(d \log d)} \log n)$. Later results, e.g., Agarwal and Matoušek [1], Matoušek [51], and Yao and Yao [65], all suffer from a query time that is exponential in d. Meiser [52] obtained query time $O(d^5 \log n)$ but after $O(n^{d+\delta})$ preprocessing. The so-called "vantage point" technique [12, 13, 62, 63] is a recently popular heuristic, but we are not aware of any analysis for high-dimensional Euclidean spaces. In general, even the average-case analysis of heuristics for points distributed over regions in $\Re^d$ gives an exponential query time [7, 35, 59].

The situation is only slightly better for approximate nearest neighbors. Arya and Mount [3] gave an algorithm with query time $O(1/\epsilon)^d O(\log n)$ and preprocessing $O(1/\epsilon)^d O(n)$. The dependence on ϵ was later reduced by Clarkson [17] and Chan [15] to $\epsilon^{-(d-1)/2}$. Arya, Mount, Netanyahu, Silverman, and Wu [5] obtained optimal $O(n)$ preprocessing cost, but with query time growing as $O(d^d)$. Bern [8] and Chan [15] considered error ϵ polynomial in d and managed to avoid exponential dependence in that case. Recently, Kleinberg [46] gave an algorithm with $O(n \log d)^{2d}$ preprocessing and query time polynomial in d, ϵ, and $\log n$, and another algorithm with preprocessing polynomial in d, ϵ, and n but with query time $O(n + d \log^3 n)$. The latter improves the $O(dn)$ time bound of the brute-force algorithm.

For the Hamming space $\{0,1\}^d$, Dolev, Harari, and Parnas [25] and Dolev, Harari, Linial, Nisan, and Parnas [24] gave algorithms for retrieving *all* points within distance r of the query q. Unfortunately, for arbitrary r, these algorithms are exponential either in query time or preprocessing. Greene, Parnas, and Yao [38] present a scheme which, for binary data chosen *uniformly at random*, retrieves all points within distance r of q in time $O(dn^{r/d})$, using $O(dn^{1+r/d})$ preprocessing.

Very recently, Kushilevitz, Ostrovsky and Rabani [47] obtained a result similar to Proposition 3 below.

Overview of Results and Techniques. Our main results are algorithms[2] for ϵ-NNS described below.[3]

[1] Throughout, preprocessing cost refers to the space requirement; typically, the preprocessing time is roughly the same.

[2] Our algorithms are randomized and return an approximate nearest neighbor *with constant probability.* To reduce the error probability to α, we can use several data structures in parallel and return the best result, increasing complexity by a factor $O(\log \alpha)$,

[3] For the sake of clarity, the $\tilde{O}$ notation is used to hide terms that are poly-logarithmic in n.

Proposition 1 *For $\epsilon > 0$, there is an algorithm for ϵ-NNS in $\Re^d$ under the l_p norm for $p \in [1,2]$ which uses $\tilde{O}(n^{1+1/(1+\epsilon)} + dn)$ preprocessing and requires $\tilde{O}(dn^{1/(1+\epsilon)})$ query time.*

Proposition 2 *For $0 < \epsilon < 1$, there is an algorithm for ϵ-NNS in $\Re^d$ under any l_p norm which uses $\tilde{O}(n) \times O(1/\epsilon)^d$ preprocessing and requires $\tilde{O}(d)$ query time.*

Proposition 3 *For any $\epsilon > 0$, there is an algorithm for ϵ-NNS in $\Re^d$ under the l_p norm for $p \in [1,2]$ which uses $(nd)^{O(1)}$ preprocessing and requires $\tilde{O}(d)$ query time.*

We obtain these results by reducing ϵ-NNS to a new problem, viz., point location in equal balls. This is achieved by means of a novel data structure called *ring-cover trees*, described in Section 3. Our technique can be viewed as a variant of parametric search [53], in that they allow us to reduce an optimization problem to its decision version. The main difference is that in our case in answering a query we can only ask for a solution to a decision problem belonging to a prespecified set, since solving the decision problem (i.e., point location in equal balls) requires data structures created during preprocessing. We believe this technique will find further applications to problems where parametric search has been helpful.

In Section 4, we give two solutions to the point location problem. One is based on a method akin to the Elias bucketing algorithm [64] — we decompose each ball into a bounded number of cells and store them in a dictionary. This allows us to achieve $\tilde{O}(d)$ query time, while the preprocessing is exponential in d, implying Proposition 2. For the second solution, we introduce the technique of *locality-sensitive hashing*. The key idea is to use hash functions such that the probability of collision is much higher for objects that are close to each other than for those that are far apart. We prove that existence of such functions for any domain (not necessarily a metric space) implies the existence of fast ϵ-NNS algorithms for that domain, with preprocessing cost only linear in d and sublinear in n. We then present two families of such functions – one for a Hamming space and the other for a family of subsets of a set under the *resemblance* measure used by Broder et al [10] to cluster web documents. The algorithm based on the first family is used to obtain a nearest-neighbor algorithm for data sets from $\Re^d$, by embedding the points from $\Re^d$ onto a Hamming cube in a distance-preserving manner. The algorithm for the resemblance measure is shown to have several applications to information retrieval and pattern recognition. We also give additional applications of locality-sensitive hashing to dynamic closest-pair problem and fast clustering algorithms. All our algorithms based on this method are easy to implement and have other advantages — they exploit sparsity of data and the running times are much lower in practice [37] than predicted by theoretical analysis. We expect these results will have a significant practical impact.

An elegant technique for reducing complexity owing to dimensionality is to project the points into a random subspace of lower dimension, e.g., by projecting P onto a small collection of random lines through the origin. Specifically, we could employ the result of Frankl and Maehara [33], which improves upon the Johnson-Lindenstrauss Lemma [42], showing that a projection of P onto a subspace defined by roughly $9\epsilon^{-2} \ln n$ random lines preserves all inter-point distances to within a relative error of ϵ, with high probability. Applying this result to an algorithm with query time $O(1)^d$, we obtain an algorithm with query time $n^{9\epsilon^{-2}}$. Unfortunately, this would lead to a sublinear query time only for large values of ϵ. In Section A of the Appendix, we give a version of the random projection result using a much simpler proof than that of Frankl and Maehara. We also consider the extensions of the random projection approach to l_p norms for $p \neq 2$. Using random projections and Proposition 2, we obtain the algorithm described in Proposition 3. Unfortunately, the high preprocessing cost (its exponent grows with $1/\epsilon$) makes this algorithm impractical for small ϵ.

2 Preliminaries

We use l_p^d to denote the space $\Re^d$ under the l_p norm. For any point $v \in \Re^d$, we denote by $||\vec{v}||_p$ the l_p norm of the vector $\vec{v}$; we omit the subscript when $p = 2$. Also, $H^d = (\{0,1\}^d, d_H)$ will denote the *Hamming metric space* of dimension d. Let $\mathcal{M} = (X, d)$ be any metric space, $P \subset X$, and $p \in X$. We will employ the following notation: $d(p, P) = \min_{q \in P} d(p, q)$, and $\Delta(P) = \max_{p,q \in P} d(p, q)$ is the diameter of P,

Definition 1 *The* ball *of radius r centered at p is defined as $B(p, r) = \{q \in X \mid d(p, q) \leq r\}$. The* ring *$R(p, r_1, r_2)$ centered at p is defined as $R(p, r_1, r_2) = B(p, r_2) - B(p, r_1) = \{q \in X \mid r_1 \leq d(p, q) \leq r_2\}$.*

Let $V_p^d(r)$ denote the *volume* of a ball of radius r in l_p^d. The following fact is standard [57, page 11].

Fact 1 *Let $\Gamma(.)$ denote the gamma function. Then $V_p^d(r) = \dfrac{(2\Gamma(1+1/p))^d}{\Gamma(1+n/p)} r^d$ and $V_2^d(r) = \dfrac{2\pi^{d/2}}{d\,\Gamma(d/2)} r^d$.*

3 Reduction to Point Location in Equal Balls

The key idea is to reduce the ϵ-NNS to the following problems of point location in equal balls.

Definition 2 (Point Location in Equal Balls (PLEB)) *Given n radius-r balls centered at $C = \{c_1, \ldots, c_n\}$ in $\mathcal{M} = (X, d)$, devise a data structure which for any query point $q \in X$ does the following: if there exists $c_i \in C$ such that $q \in B(c_i, r)$ then return c_i, else return* NO.

Definition 3 (ϵ-Point Location in Equal Balls (ϵ-PLEB)) *Given n radius-r balls centered at $C = \{c_1, \ldots, c_n\}$ in $\mathcal{M} = (X, d)$, devise a data structure which for any query point $q \in X$ does the following:*

- *if there exists $c_i \in C$ with $q \in B(c_i, r)$ then return* YES *and a point c_i' such that $q \in B(c_i', (1+\epsilon)r)$,*
- *if $q \notin B(c_i, (1+\epsilon)r)$ for all $c_i \in C$ then return* NO,
- *if for the point c_i closest to q we have $r \leq d(q, c_i) \leq ((1+\epsilon)r)$ then return either* YES *or* NO.

Observe that PLEB (ϵ-PLEB) can be reduced to NNS (ϵ-NNS), with the same preprocessing and query costs, as follows: it suffices to find an exact (ϵ-approximate) nearest neighbor and then compare its distance from q with r. The main point of this section is to show that there is a reduction in reverse from ϵ-NNS to ϵ-PLEB, with only a small overhead in preprocessing and query costs. This reduction relies on a data structure called a *ring-cover tree*. This structure exploits the fact that for any point set P, we can either find a *ring-separator* or a *cover*. Either construct allows us to decompose P into smaller sets $S_1, \ldots, S_l$ such that for all i, $|S_i| \leq c|P|$ for some $c < 1$, and $\sum_i |S_i| \leq b|P|$ for $b < 1 + 1/\log^2 n$. This decomposition has the property that while searching P it is possible to quickly restrict the search to one of the sets S_i.

There is a simpler but much weaker reduction from ϵ-NN to ϵ-PLEB. Let R be the ratio of the smallest and the largest inter-point distances in P. For each $l \in \{1+\epsilon)^0, (1+\epsilon)^1, \ldots, R\}$, generate a sequence of balls $B^l = \{B^l_1, \ldots, B^l_n\}$ of radius l centered at $p_1, \ldots, p_n$. Each sequence B^l forms an instance of PLEB. Then, given query q, we find via binary search the minimal l for which there exists an i such that $q \in B^l_i$ and return p_i as an approximate nearest neighbor. The overall reduction parameters are: query time overhead factor $O(\log\log R)$ and space overhead factor $O(\log R)$. The simplicity of this reduction is very useful in practice. On the other hand, the $O(\log R)$ space overhead is unacceptable when R is large; in general, R may be unbounded. In the final version, we will show that by using a variation of this method, storage can be reduced to $O(n^2 \log n)$, which still does not give the desired $O(1/\epsilon)^d \tilde{O}(n)$ bound.

Definition 4 *A ring $R(p, r_1, r_2)$ is an $(\alpha_1, \alpha_2, \beta)$-***ring separator** *for P if $|P \cap B(p, r_1)| \geq \alpha_1 |P|$ and $|P \setminus B(p, r_2)| \geq \alpha_2 |P|$, where $r_2/r_1 = \beta > 1$, $\alpha_1, \alpha_2 > 0$.*

Definition 5 *A set $S \subset P$ is a (γ, δ)-***cluster** *for P if for every $p \in S$, $|P \cap B(p, \gamma\Delta(S))| \leq \delta|P|$, where $0 < \gamma, \delta < 1$.*

Definition 6 *A sequence $A_1, \ldots, A_l$ of sets $A_i \subset P$ is called a (b, c, d)-***cover** *for $S \subset P$, if there exists an $r \geq d\Delta(A)$ for $A = \cup_i A_i$ such that $S \subset A$ and for $i = 1, \ldots, l$,*

- $|P \cap (\cup_{p \in A_i} B(p, r))| \leq b|A_i|$,
- $|A_i| \leq c|P|$.

where $b > 1$, $0 < c < 1$, $d > 0$.

Theorem 1 *For any P, $0 < \alpha < 1$, and $\beta > 1$, one of the following two properties must hold:*

1. *P has an (α, α, β)-ring separator, or*
2. *P contains a $(\frac{1}{2\beta}, \alpha)$-cluster of size at least $(1 - 2\alpha)|P|$.*

Proof Sketch: First note that for $\alpha > 1/2$, property (1) must be false but then property (2) is trivially true. In general, assume that (1) does not hold. Then, for any point p and radius r define:

- $f^\infty_p(r) = |P - B(p, \beta r)|$,
- $f^0_p(r) = |P \cap B(p, r)|$.

Clearly, $f_p^{\infty}(0) = n$, $f_p^{\infty}(\infty) = 0$, $f_p^0(0) = 0$, and $f_p^0(\infty) = n$. Also, notice that $f_p^{\infty}(r)$ is monotonically decreasing and $f_p^0(r)$ is monotonically increasing. It follows that there must exist a choice of r (say r_p) such that $f_p^{\infty}(r_p) = f_p^0(r_p)$. Since (1) does not hold, for any value of r we must have $\min(f_p^{\infty}(r), f_p^0(r)) \leq \alpha n$, which implies that $f_p^{\infty}(r_p) = f_p^0(r_p) \leq \alpha n$.

Let q be a point such that r_q is minimal. Define $S = P \cap R(q, r_q, \beta r_q)$; it follows that $|S| \geq (1-2\alpha)n$. Also, notice that for any $s, s' \in S$, $d(s, s') \leq 2\beta r_q$, implying that $\Delta(S) \leq 2\beta r_q$. Finally, for any $s \in S$, $|P \cap B(s, r_q)| \leq |P \cap B(s, r_s)| \leq \alpha n$. ∎

Theorem 2 *Let S be a (γ, δ)-cluster for P. Then for any b, there is an algorithm which produces a sequence of sets $A_1, \ldots, A_k \subset P$ constituting a $(b, \delta, \frac{\gamma}{(1+\gamma)\log_b n})$-cover for S.*

Proof Sketch:

The algorithm below greedily computes a good cover for S.

Algorithm Cover: $S = P \cap R(q, r_q, \beta r_q)$;
$r \leftarrow \frac{\gamma \Delta(S)}{\log_b n}$; $j \leftarrow 0$;
repeat
  $j \leftarrow j+1$; choose some $p_j \in S$; $B_j^1 \leftarrow \{p_j\}$;
  $i \leftarrow 1$;
  while $|P \cap \cup_{q \in B_j^i} B(q, r)| > b|B_j^i|$ **do**
    $B_j^{i+1} \leftarrow P \cap \cup_{q \in B_j^i} B(q, r)$;
    $i \leftarrow i+1$
  endwhile;
  $A_j \leftarrow B_j^i$; $S \leftarrow S - A_j$; $P \leftarrow P - A_j$
until $S = \phi$;
$k \leftarrow j$.

In order to prove the correctness of the algorithm, it suffices to make the following four claims.

- $S \subset A = \cup_j A_j$ — Follows from the termination condition of the outer loop.
- for all $j \in \{1, \ldots, k\}$ and any $p \in S$, $|P \cap \cup_{q \in A_j} B(p, r)| \leq b|A_j|$ — Follows from the termination condition of the inner loop.
- for all $j \in \{1, \ldots, k\}$, $|A_j| \leq \delta |P|$ — Clearly, for any j, the inner loop is repeated at most $\log_b n$ times. Hence, $\max_{q \in A_j} d(p_j, q) \leq r \log_b n \leq \gamma \Delta(S)$. As S is a (γ, δ)-cluster, we have that $|B(p_j, \gamma \Delta(S)) \cap P| \leq \delta |P|$. Hence, $|A_j| \leq \delta |P|$.
- $r \leq \frac{\gamma \Delta(S)}{(1+\gamma)\log_b n}$ — Since $\Delta(A) \leq \Delta(S) + r \log_b n = \Delta(S) + \gamma \Delta(S) = (1+\gamma)\Delta(S)$.

∎

Corollary 1 *For any P, $0 < \alpha < 1$, $\beta > 1$, $b > 1$, one of the following properties must hold:*

1. *P has an (α, α, β)-ring separator $R(p, r, \beta r)$, or*
2. *There is a (b, α, d)-cover for some $S \subset P$ such that $|S| \geq (1-2\alpha)n$ and $d = \frac{1}{(2\beta+1)\log_b n}$.*

3.1 Constructing Ring-Cover Trees

The construction of a ring-cover tree is recursive. For any given P at the root, we use properties (1) and (2) in Corollary 1 to decompose P into some smaller sets $S_1, \ldots, S_l$; these sets are assigned to the children of the node for P. Note the base case case is when P is sufficiently small and we omit that in this abstract. We also store some additional information at the node for P which enables us to restrict the nearest neighbor search to one of the children of P, by using distance computations or point location queries. For simplicity, assume that we can invoke an exact PLEB (not ϵ-PLEB); the construction can be easily modified for approximate point location. There are two cases depending on which of the two properties (1) and (2) holds. Let $\beta = 2(1+\frac{1}{\epsilon})$, $b = 1 + \frac{1}{\log^2 n}$, and $\alpha = \frac{1-1/\log n}{2}$.

Case 1. In this case, we will call P a **ring node**. We define its children to be $S_1 = P \cap B(p, \beta r)$ and $S_2 = P - B(p, r)$. Also, we store the information about the ring separator R at the node P.

Case 2. Here, we call P a **cover node**. We define $S_i = P \cap \cup_{p \in A_i} B(p, r)$ and $S_0 = S - A$. The information stored at P is as follows. Let $r_0 = (1 + 1/\epsilon)\Delta(A)$ and let $r_i = r_0/(1+\epsilon)^i$ for $i \in \{1, \ldots, k\}$, where $k = \log_{1+\epsilon} \frac{(1+1/\epsilon)\log_b n}{\gamma} + 1$. Notice that $r_k = \frac{\gamma \Delta(A)}{\log_b n(1+\epsilon)} = \frac{r}{1+\epsilon}$. For each r_i, generate an instance of PLEB with balls $B(p, r_i)$ for $p \in A$; all instances are stored at P.

Theorem 3 *The ring-cover tree can be constructed in deterministic $\tilde{O}(n^2)$ time.*

Proof Sketch: The construction proceeds as follows. On each level, we determine if a node is a ring or cover node; then we compute the ring or the cover. As the number of levels is $\tilde{O}(1)$, it is sufficient to consider only the first (root) node P, which we construct as follows. Firstly, for each $p \in P$ we construct a list L_p containing all other points $q \in P$ sorted in increasing order of $d(p,q)$. This takes $O(n^2 \log n)$ time. Then, we apply the method from the proof of Theorem 1. More specifically, we compute the functions f_p^∞ and f_p^0 for each $p \in P$. Having the lists L_p it can be easily implemented in $O(n^2)$ time. Then we try to find a ring; if one is found we are done. In the opposite case we find a cluster and then apply the algorithm COVER to find a cover. It can be verified that this algorithm runs in time $\tilde{O}(n)$ time assuming the lists L_p are given. The PLEB generation adds only another $\tilde{O}(1)$ factor. Thus the total required time is $\tilde{O}(n^2)$. ■

We now describe how to efficiently search a ring-cover tree. It suffices to show that for any node P we can restrict the search to one of its children using a small number of tests. Let $\min_q(p, p')$ denote the point out of p and p' that is closer to q. The search procedure is as follows; we omit the obvious base case.

Procedure Search:

1. **if** P is a ring node with an (α, α, β)-ring separator $R(p, r, \beta r)$ **then**:
   (a) **if** $q \in B(p, r(1 + 1/\epsilon))$ **then** return **Search**(q, S_1);
   (b) **else** compute $p' =$ **Search**(q, S_2); return $\min_q(p, p')$.
2. **if** P is a cover node with a (b, c, d)-cover $A_1, \ldots, A_l$ of radius r for $S \subset P$ **then**:

(a) **if** $q \notin B(a, r_0)$ **then** compute $p = \textbf{Search}(q, P - A)$, choose any $a \in A$, and return $\min_q(p, a)$;

(b) **else if** $q \in B(a, r_0)$ for some $a \in A$ but $q \notin B(a', r_k)$ for all $a' \in A$ **then** using binary search on r_is, find an ϵ-NN p of q in A, compute $p' = \textbf{Search}(q, P - A)$, and return $\min_q(p, p')$;

(c) **else if** $q \in B(a, r_k)$ for some $a \in A_i$ **then** return $\textbf{Search}(q, S_i)$.

3.2 Analysis of Ring-Cover Trees

We begin the analysis of the ring-cover tree construction by establishing the validity of the search procedure.

Lemma 1 *Procedure Search(q, P) produces an ϵ-nearest neighbor for q in P.*

Proof Sketch: Consider the two cases:

1. P is a ring node.
   (a) Consider any $s \in P - S_1$. Then $d(s,p) \leq d(s,q) + d(q,p)$, implying that $d(s,q) \geq d(s,p) - d(q,p)$. Since $s \notin S_1$, we know that $d(s,p) \geq \beta r = 2(1+1/\epsilon)r$, while $d(p,q) \leq r(1+1/\epsilon)$. Then, $d(s,q) \geq (1+1/\epsilon)r \geq d(q,p)$.
   (b) For any $s \in B(p,r)$, $d(q,p) \leq d(q,s) + d(s,p)$, implying that $d(q,s) \geq d(q,p) - d(s,p) \geq d(q,p) - r$. It follows that $\frac{d(q,p)}{d(q,s)} \leq \frac{d(q,p)}{d(q,p)-r} = 1 + \frac{r}{d(q,p)-r} \leq 1 + \epsilon$.
2. P is a cover node.
   (a) Similar to Case 1(b),
   (b) Obvious.
   (c) For any $p \in P - S_i$, $d(p,a) \geq r$. Since $q \in B(a, r_k)$, we have $d(q,a) \leq r_k = \frac{r}{1+\epsilon} \leq \frac{d(p,q)}{1+\epsilon}$. ■

The proofs of Lemmas 2 and 3 are omitted.

Lemma 2 *The depth of a ring-cover tree is $O(\log_{1/2\alpha} n) = O(\log^2 n)$.*

Lemma 3 *Procedure Search requires $O(\log^2 n \times \log k)$ distance computations or PLEB queries.*

Lemma 4 *A ring-cover tree requires space at most $O(knb^{\log_{1/2\alpha} n}(1+2(1-2\alpha))^{\log_n}) = O(n \operatorname{polylog} n)$ not counting the additional non-data storage used by algorithms implementing PLEBs.*

Proof Sketch: Let $S(n)$ be an upper bound on the space requirement for a ring-cover tree for point-set P of size n. Then for a cover node:

$$S(n) \leq \max_{l} \max_{A_1 \ldots A_l,\ A_i \text{ disjoint},\ |A_i| \leq \alpha n,\ |A| \geq (1-2\alpha)n} [\sum_{i=1}^{l} S(b|A_i|)] + S(n - |A|) + |A|bk$$

For a ring node:

$$S(n) \leq 2S\left(\frac{n}{2}(1+2(1-2\alpha))\right)+1$$

The bound follows by solving this recurrence. ■

Corollary 2 *Given an algorithm for PLEB which uses $f(n)$ space on an instance of size n where $f(n)$ is convex, a ring-cover tree for an n-point set P requires total space $O(f(n \operatorname{polylog} n))$.*

Fact 2 *For any PLEB instance (C,r) generated by a ring-cover tree, $\dfrac{\Delta(C)}{r} = O\left(\dfrac{1+\epsilon}{\gamma}\log_b n\right)$.*

4 Point Location in Equal Balls

We present two techniques for solving the ϵ-PLEB problem. The first is based on a method similar to the Elias bucketing algorithm [64] and works for any l_p norm, establishing Proposition 2. The second uses *locality-sensitive hashing* and applies directly only to Hamming spaces (this bears some similarity to the indexing technique introduced by Greene, Parnas, and Yao [38] and the algorithm for all-pairs vector intersection of Karp, Waarts, and Zweig [48], although the technical development is very different). However, by exploiting Facts 2 and 4 (Appendix A), the instances of ϵ-PLEB generated while solving ϵ-NN for l_1^d can be reduced to ϵ-PLEB in H^m, where $m = d\log_b n \times \max(1/\epsilon, \epsilon)$. Also, by Fact 3 (Appendix A), we can reduce l_p^d to $l_1^{O(d)}$ for any $p \in [1,2]$. Hence, locality-sensitive hashing can be used for any l_p norm where $p \in [1,2]$, establishing Proposition 1. It can also be used for the *set resemblance* measure used by Broder et al [10] to cluster web documents. We assume, without loss of generality, that all balls are of radius 1.

4.1 The Bucketing Method

Assume for now that $p=2$. Impose a uniform grid of spacing $s = \epsilon/\sqrt{d}$ on $\Re^d$. Clearly, the distance between any two points belonging to one grid cuboid is at most ϵ. By Fact 2, each side of the smallest cuboid containing balls from C is of length at most $O(\sqrt{d}\log_b n \max(1/\epsilon,\epsilon))$ times the side-length of a grid cell. For each ball B_i, define $\overline{B}_i$ to be the set of grid cells intersecting B_i. Store all elements from $\cup_i \overline{B}_i$ in a hash table [34, 55], together with the information about the corresponding ball(s). (We can use hashing since by the preceding discussion the universe is of bounded size.) After preprocessing, to answer a query q it suffices to compute the cell which contains q and check if it is stored in the table.

We claim that for $0<\epsilon<1$, $|\overline{B}| = O(1/\epsilon)^d$. To see this, observe that $|\overline{B}|$ is bounded by the volume of a d-dimensional ball of radius $r = 2/\epsilon\sqrt{d}$, which by Fact 1 is $2^{O(d)}r^d/d^{d/2} \leq O(1/\epsilon)^d$. Hence, the total space required is $O(n)\times O(1/\epsilon)^d$. The query time is the time to compute the hash function. We use hash functions of the form:

$$h((x_1,\ldots,x_d)) = (a_1x_1+\ldots+a_dx_d \mod P) \mod M$$

where P is a prime, M is the hash table size, and $a_1,\ldots,a_d \in \mathcal{Z}_P^*$. This family gives a static dictionary with $O(1)$ access time [34]. The hash functions can be evaluated using $O(d)$ arithmetic operations. For general l_p norms, we modify s to $\epsilon/d^{1/p}$. The bound on $|\overline{B}|$ applies unchanged.

Theorem 4 *For $0 < \epsilon < 1$, there is an algorithm for ϵ-PLEB in l_p^d using $O(n) \times O(1/\epsilon)^d$ preprocessing and $O(1)$ evaluations of a hash function for each query.*

4.2 Locality-Sensitive Hashing

We introduce the notion of *locality-sensitive hashing* and apply it to sublinear-time similarity searching. The definition makes no assumptions about the object similarity measure. In fact, it is applicable to both *similarity* and *dissimilarity* measures; an example of the former is dot product, while any distance metric is an instance of the latter. To unify notation, we define a ball for a similarity measure D as $B(q,r) = \{p : D(q,p) \geq r\}$. We also generalize the notion of ϵ-PLEB to (r_1, r_2)-PLEB where for any query point q we require the answer to be YES if $P \cap B(q, r_2) \neq \emptyset$ and NO otherwise.

Definition 7 *A family $\mathcal{H} = \{h : S \to U\}$ is called* (r_1, r_2, p_1, p_2)-sensitive *for D if for any $q, p \in S$*

- *if $p \in B(q, r_1)$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \geq p_1$,*
- *if $p \notin B(q, r_2)$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \leq p_2$.*

In order for a locality-sensitive family to be useful, it has to satisfy inequalities $p_1 > p_2$ and $r_1 < r_2$ when D is a dissimilarity measure, or $p_1 > p_2$ and $r_1 > r_2$ when D is a similarity measure.

For k specified later, define a function family $\mathcal{G} = \{g : S \to U^k\}$ such that $g(p) = (h_1(p), \ldots, h_k(p))$, where $h_i \in \mathcal{H}$. The algorithm is as follows. For an integer l we choose l functions $g_1, \ldots, g_l$ from $\mathcal{G}$ independently and uniformly at random. During preprocessing, we store each $p \in P$ in the bucket $g_j(p)$, for $j = 1, \ldots, l$. Since the total number of buckets may be large, we retain only the non-empty buckets by resorting to hashing [34, 55]. To process a query q, we search all buckets $g_1(q), \ldots, g_l(q)$; as it is possible (though unlikely) that the total number of points stored in those bucket is large, we interrupt search after finding first $2l$ points (including duplicates). Let $p_1, \ldots, p_t$ be the points encountered therein. For each p_j, if $p_j \in B(q, r_2)$ then we return YES and p_j, else we return NO.

The parameters k and l are chosen so as to ensure that with a constant probability the following two properties hold:

1. if there exists $p^* \in B(q, r_1)$ then $g_j(p^*) = g_j(q)$ for some $j = 1 \ldots l$, and
2. the total number of collisions of q with points from $P - B(q, r_2)$ is less than $2l$, i.e.

$$\sum_{j=1}^{l} |(P - B(q, r_2)) \cap g_j^{-1}(g_j(q))| < 2l.$$

Observe that if (1) and (2) hold, then the algorithm is correct.

Theorem 5 *Suppose there is a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ for D. Then there exists an algorithm for (r_1, r_2)-PLEB under measure D which uses $O(dn + n^{1+\rho})$ space and $O(n^\rho)$ evaluations of the hash function for each query, where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$.*

Proof Sketch: It suffices to ensure that (1) holds with probability P_1 and (2) holds with probability P_2 such that both P_1 and P_2 are strictly greater than half. Assume that $p^* \in B(q, r_1)$; the proof is similar when $p^* \notin B(q, r_2)$. Set $k = \log_{1/p_2} n$, then the probability that $g(p') = g(q)$ for $p \in P - B(q, r_2)$ is at most $p_2^k = \frac{1}{n}$. Thus the expected number of elements from $P - B(q, r_2)$ colliding with q under fixed g_j is at most 1; the expected number of such collisions with *any* g_j is at most l. Thus by Markov inequality the probability that this number exceeds $2l$ is less than $1/2$; therefore the probability that the property (2) holds is $P_2 > 1/2$.

Consider now the probability of $g_j(p^*) = g_j(q)$. Clearly, it is bounded from below by

$$p_1^k = p_1^{\log_{1/p_2} n} = n^{-\frac{\log 1/p_1}{\log 1/p_2}} = n^{-\rho}.$$

Thus the probability that such a g_j exists is at least $P_1 = 1 - (1 - n^{-\rho})^l$. By setting $l = n^\rho$ we get $P_1 > 1 - 1/e > 1/2$. The theorem follows. ■

We apply Theorem 5 to two measures: the Hamming metric and *set resemblance* [10]; the latter is a similarity measure defined for any pair of sets A and B as $D(A, B) = \frac{|A \cap B|}{|A \cup B|}$. For the first measure, we apply a family of projections for fast hashing with AC^0 operations [6]. For the second measure, we use *sketch functions* used earlier [10] for estimation of the resemblance between given sets A and B.

Proposition 4 ([6]) *Let $S = \mathcal{H}^d$ and $D(p, q)$ be the Hamming metric for $p, q \in \mathcal{H}$. Then for any $r, \epsilon > 0$, the family $\mathcal{H} = \{h_i : h_i((b_1, \ldots b_d)) = b_i,\ i = 1 \ldots n\}$ is $\left(r, r(1+\epsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\epsilon)}{d}\right)$-sensitive.*

Corollary 3 *For any $\epsilon > 0$, there exists an algorithm for ϵ-PLEB in H^d (or, l_p^d for any $p \in [1, 2]$) using $O(dn + n^{1+1/(1+\epsilon)})$ space and $O(n^{1/(1+\epsilon)})$ hash function evaluations for each query. The hash function can be evaluated using $O(d)$ operations.*

Proof Sketch: We use Proposition 4 and Theorem 5. First, we need to estimate the value of $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$, where $p_1 = 1 - \frac{r}{d}$ and $p_2 = 1 - \frac{r(1+\epsilon)}{d}$. Without loss of generality, we assume that $r < \frac{d}{\ln n}$, since we can increase dimensionality by adding a sufficiently long string of 0s at the end of each point. Observe that

$$\rho = \frac{\ln 1/p_1}{\ln 1/p_2} < \frac{\ln \frac{1}{1-r/d}}{\ln \frac{1}{1-(1+\epsilon)r/d}} = \frac{\ln(1 - r/d)}{\ln(1 - (1+\epsilon)r/d)}$$

Multiplying both the numerator and the denominator by $\frac{d}{r}$ we obtain that:

$$\rho = \frac{\frac{d}{r}\ln(1 - r/d)}{\frac{d}{r}\ln(1 - (1+\epsilon)r/d)} = \frac{\ln(1 - r/d)^{d/r}}{\ln(1 - (1+\epsilon)r/d)^{d/r}} = \frac{U}{L}.$$

In order to upper bound ρ, we need to bound U from below and L from above; note that both U and L are negative. To this end we use the following inequalities [55]:

$$(1 - (1+\epsilon)r/d)^{d/r} < e^{-(1+\epsilon)} \quad \text{and} \quad (1 - r/d)^{d/r} > e^{-1}(1 - \frac{1}{d/r}).$$

Therefore,

$$\begin{aligned}
\frac{U}{L} &< \frac{\ln(e^{-1}(1-\frac{1}{d/r}))}{\ln e^{-(1+\epsilon)}} \\
&= \frac{-1+\ln(1-\frac{1}{d/r})}{-(1+\epsilon)} \\
&= 1/(1+\epsilon) - \frac{\ln(1-\frac{1}{d/r})}{1+\epsilon} \\
&< 1/(1+\epsilon) - \ln(1-1/\ln n)
\end{aligned}$$

where the last step uses the assumptions that $\epsilon > 0$ and $r < \frac{d}{\ln n}$. We conclude that

$$n^\rho < n^{1/(1+\epsilon)} n^{-\ln(1-1/\ln n)} = n^{1/(1+\epsilon)}(1-1/\ln n)^{-\ln n} = O(n^{1/(1+\epsilon)}).$$

■

The hash function evaluation can be made faster than $O(d)$ for sparse data, i.e., when the number of non-zero coordinates of a query point is small. It suffices to sample the bits from the non-zero entries of the vectors; a similar method works for the functions used to build a static dictionary. Moreover, our experience is that the preprocessing space and query time are much lower than the above bound indicates. In particular, we have implemented a variant of the above data structure for the case when data is stored on disk [37]. For a data set of 20,000 d-color histograms for images (with d ranging up to 64) only 3-9 disk accesses were required in order to achieve small average error.

Proposition 5 ([10]) *Let S be the set of all subsets of $X = \{1 \dots x\}$ and let D be the set resemblance measure. Then, for $1 > r_1 > r_2 > 0$, the following hash family is (r_1, r_2, r_1, r_2)-sensitive:*

$$\mathcal{H} = \{h_\pi : h_\pi(A) = \max_{a \in A} \pi(a),\ \pi \textit{ is a permutation of } X\}.$$

Corollary 4 *For $0 < \epsilon, r < 1$, there exists an algorithm for $(r, \epsilon r)$-PLEB under set resemblance measure D using $O(dn + n^{1+\rho})$ space and $O(n^\rho)$ evaluations of the hash function for each query, where $\rho = \frac{\ln r}{\ln \epsilon r}$.*

We now discuss further applications of the above corollary. For any pair of points $p, q \in \mathcal{H}^d$, consider the similarity measure $D(p,q)$ defined as the dot product $p \cdot q$. The dot product is a common measure used in information retrieval applications [32]; it is also of use in molecular clustering [14]. By using techniques by Indyk, Motwani, and Venkatasubramanian [41] it can also be used for solving the approximate largest common point set problem, which has many applications in image retrieval and pattern recognition. By a simple substitution of parameters, we can prove that for a set of binary vectors of approximately the same weight, PLEB under dot product measure (for queries of a fixed weight) can be reduced to PLEB under set resemblance measure. The fixed weight assumption can be easily satisfied by splitting the data points into $O(\log d)$ groups of approximately the same weight, and then making the same partition for weights of potential queries.

4.3 Further Applications of PLEB Algorithms

The PLEB procedures described above can also be used in cases where points are being inserted and deleted over time. In the randomized indexing method, insertion can be performed by adding the point to all indices, and deletion can be performed by deleting the point from all indices. In the bucketing method, insertion and deletion can be performed by adding or deleting all elements of $\overline{B}$ in the hash table. However, in order to apply these methods, we have to assume that the points have integer coordinates with absolute value bounded by, say, M. Let n be the maximum number of points present at any time.

Corollary 5 *There is a data structure for ϵ-PLEB in $\{1 \ldots M\}^d$ which performs insertions, deletions, and queries in time $O(1/\epsilon)^d poly(\log M, \log n)$ using storage $O(1/\epsilon)^d n$.*

Corollary 6 *There is a data structure for ϵ-PLEB in $\{1 \ldots M\}^d$ which performs insertions, deletions, and queries in time $\tilde{O}(dn^{1/(1+\epsilon)})$ using storage $O(dn + n^{1+1/(1+\epsilon)})$.*

The latter corollary follows from the fact that in order to compute $g(q)$ we do not need to keep the unary representation of p *explicitly*. Rather than that, it is sufficient for each coordinate to keep track of the breakpoint at which the sampled bits are changing values from 0 to 1; this clearly requires only constant memory words per coordinate.

By keeping several copies of PLEB as in the simple method described at the beginning of Section 3, we can answer approximate closest-pair queries. It is sufficient to check for every radius whether any cell (in the bucketing method) or any bucket (in the randomized indexing method) contains two different points; the smallest radius having this property gives an approximation to the closest-pair distance. The time bounds for all operations are as in the above corollaries, but multiplied by a factor $O(\log \log_{1+\epsilon} M)$. It is also easy to see that the *bichromatic pair problem* (in which the points are colored and we consider only pairs of different colors) or even *multichromatic pair problem* (for more than two colors) can be solved withing the same time bounds.

Combining both techniques, we obtain a method for dynamic estimation of closest pair. Eppstein [28] showed recently that dynamic closest-pair problem has many application to hierarchical agglomerative clustering, greedy matching and other problems, and provided a data structure making $\tilde{O}(n)$ distance computations per update operation. Our scheme gives an approximate answer in sublinear time. Moreover by an easy simulation of Kruskal's MST algorithm using dynamic multichromatic closest pair data structure, the *Approximate Minimum Spanning Tree* problem can be solved in time bounded by the cost of approximate bichromatic closest pair times $\tilde{O}(n)$. Thus we obtain the first algorithm solving this problem in subquadratic time for any d.

References

[1] P.K. Agarwal and J. Matoušek. Ray shooting and parametric search. In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, 1992, pp. 517–526.

[2] P.K. Agarwal, H. Edelsbrunner, O. Schwartzkopf and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In: *Discrete and Computational Geometry*, 6 (1991), no. 5, pp. 407-422.

[3] S. Arya and D. Mount. Approximate nearest neighbor searching. In: *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 271–280.

[4] S. Arya, D.M. Mount, and O. Narayan, Accounting for boundary effects in nearest-neighbor searching. *Discrete and Computational Geometry*, 16(1996):155–176.

[5] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In: *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 573–582.

[6] A. Andersson, P. B. Miltersen, S. Riis, M. Thorup. Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{\log n / \log\log n})$ is necessary and sufficient. In: *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, 1996, pp. 441–450.

[7] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(1975):509–517.

[8] M. Bern. Approximate closest-point queries in high dimensions. *Information Processing Letters*, 45(1993):95–99.

[9] M.W. Berry, S.T. Dumais, and A.T. Shippy. A case study of latent semantic indexing. U.T. Knoxville Technical Report CS-95-271, January 1995.

[10] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the Web. In: *Proceedings of the Sixth International World Wide Web Conference*, pp. 391-404, 1997.

[11] C. Buckley, A. Singhal, M. Mitra, and G. Salton. New Retrieval Approaches Using SMART: TREC 4. In: *Proceedings of the Fourth Text Retrieval Conference,* National Institute of Standards and Technology, 1995.

[12] W.A. Burkhard and R.M. Keller. Some approaches to Best-Match File Searching. *Communications of the ACM*, 16(1973):230–236.

[13] T. Bozkaya and M. Ozsoyoglu. Distance-Based Indexing for High-Dimensional Metric Spaces, In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1997.

[14] F. Cazals. Effective Nearest Neighbours Searching on the Hyper-Cube, with Applications to Molecular Clustering. In *Proceedings of the 14th Annual ACM Symposium on Computational Geometry, 1998.*

[15] T.M. Chan. Approximate Nearest Neighbor Queries Revisited. In: *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, 1997, pp. 352–358.

[16] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(1988):830–847.

[17] K. Clarkson. An algorithm for approximate closest-point queries. In: *Proceedings of the Tenth Annual ACM Symposium on Computational Geometry*, 1994, pp. 160–164.

[18] K. Clarkson. Nearest Neighbor Queries in Metric Spaces. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997, pp. 609–617.

[19] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10(1993):57–67.

[20] T.M. Cover and P.E. Hart, Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1967):21–27.

[21] S. Deerwester, S. T. Dumais, T.K. Landauer, G.W. Furnas, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the Society fo4 Information Sciences,* 41(1990):391–407.

[22] L. Devroye and T.J. Wagner, Nearest neighbor methods in discrimination. In: *Handbook of Statistics*, vol. 2, P.R. Krishnaiah and L.N. Kanal, eds., North-Holland, 1982.

[23] D. Dobkin and R. Lipton. Multidimensional search problems. *SIAM Journal on Computing*, 5(1976):181–186.

[24] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas. Neighborhood preserving hashing and approximate queries. In: *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 251–259.

[25] D. Dolev, Y. Harari, and M. Parnas. Finding the neighborhood of a query in a dictionary. In: *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems*, 1993, pp. 33–42.

[26] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, NY, 1973.

[27] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[28] D. Eppstein, Fast hierarchical clustering and other applications of dynamic closest pairs. In: *Proceedings of the Ninth ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[29] C. Faloutsos, R. Barber, M. Flickner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(1994):231–262.

[30] W. Feller. An Introduction to Probability Theory and its Applications. John Wiley & Sons, NY, 1991.

[31] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the QBIC system. *IEEE Computer*, 28(1995):23–32.

[32] W. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.

[33] P. Frankl and H. Maehara. The Johnson-Lindenstrauss Lemma and the Sphericity of Some Graphs. *Journal of Combinatorial Theory* B, 44(1988):355-362.

[34] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. *Journal of the ACM*, 31(1984):538–544.

[35] J.K. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(1977):209–226.

[36] A. Gersho and R.M. Gray. *Vector Quantization and Data Compression.* Kluwer, 1991.

[37] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. *Manuscript, 1997.*

[38] D. Greene, M. Parnas, and F. Yao. Multi-index hashing for information retrieval. In: *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994, pp. 722–731.

[39] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. In: *First International Conference on Knowledge Discovery & Data Mining*, 1995, pp. 142–149.

[40] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 27(1933):417–441.

[41] P. Indyk, R. Motwani, and S. Venkatasubramanian. Geometric Matching Under Noise - Combinatorial Bounds and Algorithms. *Manuscript, 1997.*

[42] W.B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26(1984):189–206.

[43] W.B. Johnson and G. Schechtman. Embedding l_p^m into l_1^n. *Acta Mathematica*, 149(1982):71–85.

[44] K. Karhunen. Über lineare Methoden in der Wahrscheinlichkeitsrechnung. *Ann. Acad. Sci. Fennicae*, Ser. A137, 1947.

[45] V. Koivune and S. Kassam. Nearest neighbor filters for multivariate data. *IEEE Workshop on Nonlinear Signal and Image Processing*, 1995.

[46] J. Kleinberg. Two Algorithms for Nearest-Neighbor Search in High Dimensions. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.

[47] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *These proceedings.*

[48] R.M. Karp, O. Waarts, and G. Zweig. The bit vector intersection problem. In: *Proceedings of 36th Annual IEEE Symposium on Foundations of Computer Science*, 1995, pp. 621–630.

[49] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In: *Proceedings of 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994, pp. 577–591.

[50] M. Loéve. Fonctions aleastoires de second ordere. *Processus Stochastiques et mouvement Brownian*, Hermann, Paris, 1948.

[51] J. Matoušek. Reporting points in halfspaces. In: *Computational Geometry: Theory and Applications*, 2(1992):169–186.

[52] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(1993):286–303.

[53] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM* 31(1983), pp. 852-865.

[54] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[55] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[56] A. Pentland, R.W. Picard, and S. Sclaroff. Photobook: tools for content-based manipulation of image databases. In *Proceedings of the SPIE Conference on Storage and Retrieval of Image and Video Databases II*, 1994.

[57] G. Pisier. *The volume of convex bodies and Banach space geometry.* Cambridge University Press, 1989.

[58] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Book Company, New York, NY, 1983.

[59] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1989.

[60] T. Sellis, N. Roussopoulos and C. Faloutsos. Multidimensional Access Methods: Trees Have Grown Everywhere. *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, 1997, pp. 13–15.

[61] A.W.M. Smeulders and R. Jain, eds. *Image Databases and Multi-media Search.* Proceedings of the First International Workshop, IDB-MMS '96, Amsterdam University Press, Amsterdam, 1996.

[62] J.K. Uhlmann. Satisfying General Proximity/Similarity Queries with Metric Trees. *Information Processing Letters*, 40(1991):175–179.

[63] P.N. Yiannilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In: *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 311–321.

[64] T. Welch. Bounds on the information retrieval efficiency of static file structures. Technical Report 88, MIT, June 1971.

[65] A.C. Yao and F.F. Yao, A general approach to d-dimensional geometric queries. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, 1985, pp. 163–168.

A The Dimension Reduction Technique

We first outline our proof for the random projections technique for dimension reduction. Combining this with Proposition 2, we obtain the result given in Proposition 3.

Definition 8 *Let $\mathcal{M} = (X, d)$ and $\mathcal{M}' = (X', d')$ be two metric spaces. The space $\mathcal{M}$ is said to have a c-***isometric embedding***, or simply a c-***embedding***, in $\mathcal{M}'$ if there exists a map $f : \mathcal{M} \to \mathcal{M}'$ such that*

$$(1-\epsilon)d(p,q) < d'(f(p), f(q)) < (1+\epsilon)d(p,q)$$

for all $p, q \in X$. We call c the **distortion** *of the embedding; if $c = 1$, we call the embedding* **isometric**.

Frankl and Maehara [33] gave the following improvement to the Johnson-Lindenstrauss Lemma [42] on $(1+\epsilon)$-embedding of any $S \subset l_2^d$ in $l_2^{O(\log|S|)}$.

Lemma 5 (Frankl-Maehara [33]) *For any $0 < \epsilon < \frac{1}{2}$, any (sufficiently large) set S of points in $\Re^d$, and $k = \lceil 9(\epsilon^2 - 2\epsilon^3/3)^{-1} \ln |S| \rceil + 1$, there exists a map $f : S \to \Re^k$ such that for all $u, v \in S$,*

$$(1-\epsilon)||u-v||^2 < ||f(u) - f(v)||^2 < (1+\epsilon)||u-v||^2.$$

The proof proceeds by showing that the square of the length of a projection of any unit vector v on a random k-dimensional hyperplane is sharply concentrated around $\frac{k}{d}$. Below we prove an analogous fact. However, thanks to the use of a different distribution, we are able to give a much simpler proof and also improve the constants. Note that the constants are important as they appear in the *exponent* of the time bounds of the resulting algorithm described in Proposition 3.

Lemma 6 *Let u be a unit vector in $\Re^d$. For any even positive integer k, let $U_1, \ldots, U_k$ be random vectors chosen independently from the d-dimensional Gaussian distribution*[4] *$N^d(0,1)$. For $X_i = u \cdot U_i$, define $W = W(u) = (X_1, \ldots, X_k)$ and $L = L(u) = ||W||^2$. Then, for any $\beta > 1$,*

1. *$E(L) = k$,*
2. *$\Pr[L \geq \beta k] < O(k) \times \exp(-\frac{k}{2}(\beta - (1 + \ln \beta)))$,*
3. *$\Pr[L \leq k/\beta] < O(k) \times \exp(-\frac{k}{2}(\beta^{-1} - (1 - \ln \beta)))$.*

Proof Sketch: By the spherical symmetry of $N^d(0,1)$ each X_i is distributed as $N(0,1)$ [30, page 77]. Define $Y_i = X_{2i-1}^2 + X_{2i}^2$, for $i = 1, \ldots, k/2$. Then, Y_i follows the Exponential distribution with parameter $\lambda = \frac{1}{2}$ (see [30, page 47]). Thus $E(L) = \sum_{i=1}^{k/2} E(Y_i) = (k/2) \times 2 = k$; also one can see that L follows the Gamma distribution with parameters $\alpha = \frac{1}{2}$ and $v = k/2$ (see [30, page 46]). Since this distribution is a dual of the Poisson distribution, we obtain that

$$\Pr[L \geq \beta k] = \Pr[P_{\beta k}^{1/2} \leq v - 1],$$

where P_t^α is a random variable following the Poisson distribution with parameter αt. Clearly

$$\Pr[P_t^\alpha \leq v - 1] = \sum_{i=0}^{v-1} e^{-\alpha t} \frac{(\alpha t)^i}{i!}$$

and therefore

$$\Pr[L \geq \beta k] = \sum_{i=0}^{v-1} e^{-\beta v} \frac{(\beta v)^i}{i!} \leq v e^{-\beta v} \frac{(\beta v)^v}{v!} \leq v e^{-\beta v} \frac{(\beta v)^v}{\frac{v^v}{e^v}} = v(e^{-\beta} \beta e)^v = v e^{-v(\beta - (1 + \ln \beta))},$$

which implies the desired result since $v = k/2$.

Finally, we claim the following bound, for some large constant $\gamma \gg 1$.

$$\Pr[L \leq k/\beta] = \sum_{i=v}^{\infty} e^{-v/\beta} \frac{(v/\beta)^i}{i!} \leq e^{-v/\beta} \sum_{i=v}^{\infty} \left(\frac{ev}{i\beta}\right)^i = e^{-v/\beta} \left[\sum_{i=v}^{\gamma ev/\beta} \left(\frac{ev}{i\beta}\right)^i + \sum_{i=\gamma ev/\beta + 1}^{\infty} \left(\frac{ev}{i\beta}\right)^i \right]$$

[4] Each component is chosen independently from the standard normal distribution $N(0,1)$.

The second sum is very small for $\gamma \gg 1$ and we bound only the first one. As the sequence $(\frac{ev}{i\beta})^i$ is decreasing for $i \geq v/\beta$, we can bound the first sum by

$$\left(\frac{\gamma e v}{\beta}\right) e^{-v/\beta} \left(\frac{e}{\beta}\right)^v = O(v) e^{-v(\beta^{-1}-(1-\ln\beta))}.$$

Since $v = k/2$, we obtain the desired result. ■

B Auxiliary facts

Fact 3 (Johnson-Schechtman [43]) *For any $1 \leq p < 2$ and $\epsilon > 0$, there exists a constant $\beta \geq 1$ such that for all $d \geq 1$, the space l_p^d has a $(1+\epsilon)$-embedding in $l_1^{\beta d}$.*

Fact 4 (Linial, London, and Rabinovich [49]) *For any $\epsilon > 0$ and every n-point metric space $\mathcal{M} = (X, d)$ induced by a set of n points in l_1^d, there exists m such that $\mathcal{M}$ has a $(1+\epsilon)$-embedding in H^m. If all points have coordinates from the set $\{1 \ldots R\}$, then $\mathcal{M}$ can be embedded isometrically in H^m for $m = Rd$.*