Journal of Computer and System Sciences **63**, 627–638 (2001)
doi:10.1006/jcss.2001.1781, available online at http://www.idealibrary.com on IDEAL®

On Approximate Nearest Neighbors under l_∞ *Norm*

Department of Computer Science, Stanford University, Stanford, California 94305
E-mail: indyk@cs.stanford.edu

Received March 30, 1999; revised August 1, 2000

The *nearest neighbor search (NNS)* problem is the following: Given a set of n points $P = \{p_1, ..., p_n\}$ in some metric space X, preprocess P so as to efficiently answer queries which require finding a point in P closest to a query point $q \in X$. The *approximate nearest neighbor search (c-NNS)* is a relaxation of NNS which allows to return any point within c times the distance to the nearest neighbor (called c-nearest neighbor). This problem is of major and growing importance to a variety of applications. In this paper, we give an algorithm for $(4\lceil \log_{1+\rho} \log 4d \rceil + 1)$-NNS algorithm in l_∞^d with $O(dn^{1+\rho} \log^{O(1)} n)$ storage and $O(d \log^{O(1)} n)$ query time. Moreover, we obtain an algorithm for 3-NNS for l_∞ with $n^{\log d+1}$ storage. The preprocessing time is close to linear in the size of the data structure. The algorithm can be also used (after simple modifications) to output the *exact* nearest neighbor in time bounded by $O(d \log^{O(1)} n)$ plus the number of $(4\lceil \log_{1+\rho} \log 4d \rceil + 1)$-nearest neighbors of the query point. Building on this result, we also obtain an approximation algorithm for a general class of *product* metrics. Finally, we show that for any $c < 3$ the c-NNS problem in l_∞ is provably as hard as the *subset query* problem (also called the *partial match* problem). This indicates that obtaining a sublinear query time and subexponential (in d) space for $c < 3$ might be hard. © 2001 Elsevier Science (USA)

1. INTRODUCTION

The *nearest neighbor search (NNS)* problem is the following: Given a set of n points $P = \{p_1, ..., p_n\}$ 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 $q \in X$. The nearest neighbors problem is of major and growing importance to a variety of applications, usually involving similarity search; some examples are information retrieval, image and video databases, protein databases, data mining and pattern

[1] Supported by a Stanford Graduate Fellowship and NSF Award CCR-9357849.

recognition. The problem was formally posed in 1969 by Minsky and Papert [MP69] and was a subject of extensive study since then. Many efficient algorithms were discovered for the case where X is a low dimensional Euclidean space [Cl88, Me93]. However, many (if not most) of recent applications require X to be either high-dimensional or non-Euclidean, or both. In these cases the known algorithms become inefficient due to the so-called "curse of dimensionality"—their query times and/or storage requirements grow exponentially with the dimension.

Because of this phenomenon, recent research has focused on the approximate version of the problem (called c-NNS), where the algorithm is allowed to return a point within c times the distance to the nearest neighbor (called a c-nearest neighbor). It is natural to expect that (as in the case for many NP-hard problems) solving approximate version of the problem will avoid exponential dependence on the dimension. Also, the relaxed version still remains of practical interest. This is due to the fact that distance measures are designed to reflect the inherent similarity between the objects, which is usually quite selective (in the sense that only a small fraction of objects is usually considered similar to the query point while the remaining ones are considered dissimilar). Thus if the distance function is chosen appropriately, small perturbations in the distance are unlikely to affect the quality of the output. Also, most of the existing approximate algorithms can be used (after simple modifications) to output the *exact* nearest neighbor with the query time multiplied by the number of c-nearest neighbors of the query point (we call this property *neighborhood sensitivity*). As the number of c-nearest neighbors is usually small for small values of c, the data structure enables fast access also in situations when the quality of the result cannot be compromised.

The approximate nearest neighbor problem was recently a subject of extensive research (see Section 1.1). The most recent results of [IM98] and [KOR98] give algorithms for approximate nearest neighbor in d-dimensional Euclidean space with polynomial storage and query time polynomial in $\log n$ and d. These algorithms are of mainly theoretical interest, as their storage requirements are quite large. Indyk and Motwani [IM98] also gave another algorithm with small polynomial storage and sublinear query time, with much better behavior in practice [GIM97]. Thus these results show that in Euclidean spaces the exponential dependence on the dimension can be avoided by resorting to approximation algorithms. Unfortunately, the techniques used to achieve these results heavily exploit properties of Euclidean norm and therefore do not seem applicable to other metric spaces (see Section 1.1 for more details).

In this paper, we investigate the complexity of approximate nearest neighbor search in non-Euclidean spaces. Our main result is an algorithm for the l_∞ norm (i.e., the norm of a vector is equal to the maximum absolute value of its coordinates). Our algorithm offers the following tradeoff: for any parameter $\rho > 0$ it uses $\tilde{O}(dn^{1+\rho})$-size[2] storage and finds a $c(d, \rho)$-nearest neighbor in $\tilde{O}(d)$ time for $c(d, \rho) = 4\lceil \log_{1+\rho} \log 4d \rceil + 1$; the preprocessing time is linear in the size of the data structure. Moreover, we give an algorithm for $c = 3$ for l_∞ with $n^{\log d + 1}$ storage.

[2] $\tilde{O}(f(n)) = O(f(n) \log^{O(1)} n)$.

Thus, we can guarantee a $O(\log \log d)$-nearest neighbor using a storage arbitrarily close to linear, or a 3-nearest neighbor with mildly superpolynomial storage. We note that the above bounds hold for worst-case scenarios and are likely to be much better for instances occurring in practice. Also, the algorithm is neighborhood sensitive.

The l_∞ norm is of significant practical interest (e.g., see [ALSS95]). Its advantage is that it enables to combine different similarity measures (color, texture, etc.) without any assumptions about their additivity [AGGM98]; it has also other attractive properties which make it useful for fuzzy information systems [Fagin96, Fagin98]. Moreover, it has important theoretical properties which enable us to further generalize the above result. One of them is that any finite metric space can be embedded in l_∞^d with finite d and no distortion. Moreover, if we allow a small multiplicative distortion t, then any n-point metric space can be embedded in l_∞^d with small dimension $d = \tilde{O}(n^{\frac{1}{\lfloor (t+1)/2 \rfloor}})$ [Mat96]. By exploiting this property we obtain a $tc(d, \rho)$-approximation algorithm for a *product* of k arbitrary finite metrics of size s with query time $\tilde{O}(ks^{\frac{1}{\lfloor (t+1)/2 \rfloor}})$ and storage $\tilde{O}(ks^{\frac{1}{\lfloor (t+1)/2 \rfloor}} n^{1+\rho})$, with polynomial preprocessing (a product of given metric spaces $M_1, ..., M_k$ with distance functions $D_1 ... D_k$ is a metric space over $M_1 \times \cdots \times M_k$ with the distance function D such that $D((p_1, ..., p_k), (q_1, ..., q_k)) = \max(D_1(p_1, q_1) \cdots D_k(p_k, q_k))$). Product metrics have been studied earlier by Agarwal, Gionis, Guibas and Motwani [AGGM98] with applications to similarity searching problem in large image databases, where the individual metric spaces represent various properties of images (color, texture, etc). Finally, Farach-Colton and Indyk recently gave several results on embedding of *Hausdorff* metrics into l_∞ [FI98].

Finally, we show that for any $c < 3$ the c-NNS problem in l_∞ is provably as hard as the *subset query* problem (also called the *partial match* problem). The latter problem has been investigated for last few decades (e.g., see [Riv74] or [Knu73], p. 557), but all solutions discovered so far have either $\Omega(n)$ query time or $2^{\Omega(d)}$ space. Thus, it is natural to assume the problem is hard (see [BOR] for some hardness results for this problem). Therefore, our reduction indicates that obtaining a sublinear query time and subexponential (in d) space for $c < 3$ is hard and therefore our 3-approximation upper bound is tight.

1.1. Related Work

In this section we review the work on approximate nearest neighbor in metric spaces. We note that for the exact case the best known result is due to Meiser [Me93], who obtained an algorithm with (roughly) $\tilde{O}(n^d)$ storage and query time $\tilde{O}(d^5)$; the query time can be improved to $O(d^3 \log n)$ ([AE], p. 46). The storage requirements can be significantly reduced when the underlying norm is l_∞—for this case Gabow *et al.* [GBT84] gave an algorithm with (roughly) $O(n \log^d n)$ storage and $O(\log^d n)$ query time.

Arbitrary norm. The first result for approximate nearest neighbor in $\Re^d$ was due to Bern [Bern93]. His result (after recent improvement by Chan [Chan97]) guarantees polynomial storage and query time polynomial in d and $\log n$, but with c

polynomial in d. Arya, Mount, Netanyahu, Silverman, and Wu [AMNSW94] obtained optimal $O(dn)$ storage but with query time $O((d/\varepsilon)^d \log n)$ (here and later we denote $\varepsilon = c-1$). ∎

Euclidean/Manhattan norm. The first approximate algorithm for any $c>1$ was discovered by Arya and Mount [AM93], who gave an algorithm with query time (roughly) $O(1/\varepsilon)^d \log^3 n$ and preprocessing (roughly) $O(1/\varepsilon)^d O(n)$. The dependence on ε was later reduced by Clarkson [Cl94] and Chan [Chan97] to $\varepsilon^{-(d-1)/2}$. Kleinberg [Kl97] 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 and n but with query time $O(n + d \log^3 n)$; all bounds also contain $1/\varepsilon^{O(1)}$ factors. Indyk and Motwani [IM98] and Kushilevitz *et al.* [KOR98] give algorithms with polynomial storage and query time polynomial in $\log n$ and d. Indyk and Motwani [IM98] also gave another algorithm with smaller storage requirements and sublinear query time. The algorithms by Kleinberg, Indyk *et al.* and Kushilevitz *et al.* are randomized Monte Carlo, i.e., are correct with high probability.

All of the approximate algorithms above with query time polynomial in d (i.e., [Kl97], [IM98] and [KOR98]) use techniques applicable only for Euclidean/Manhattan norms. The algorithm of [Kl97] is based random projections, which are not well defined for l_p norms with $p>2$. The first algorithm of [IM98] uses the Johnson–Lindenstrauss Lemma [JL84] to reduce the dimension to $O(\log n)$; this lemma provably does not hold for other norms (like l_∞ [Mat96, Mat]). The algorithm of [KOR98] and the second result of [IM98] solve the problem by embedding the original norm into the Hamming space; again it is known that such an embedding is not possible for the l_∞ norm. ∎

2. PRELIMINARIES

In this section we introduce the notions and notation used later in the paper.

Let l_p^d 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 $\vec{v}$. Also, for any set $S \subset \{1 \ldots d\}$ we use χ_S to denote the characteristic function of a set S, i.e., a vector $v \in \{0,1\}^d$ such that $v(a)=1$ iff $a \in S$.

Let (M, D) be any metric space, $P \subset X$, and $p \in X$. The *ball* of radius r centered at p is defined as $B(p, r) = \{q \in X \mid d(p, q) \leqslant r\}$. For a normed space X (whose norm we denote by $|\cdot|$) and $t \geqslant 0$ we say that a function $f\colon M \to X$ is a *t-embedding* (or an *embedding with distortion t*) if we have

$$\frac{1}{t} D(u, v) \leqslant |f(y) - f(v)| \leqslant D(u, v) \tag{1}$$

for any two points $u, v \in M$. We say that M t-embeds into X if there exists a t-embedding $f\colon M \to X$.

The following problem is a decision version of the c-NNS:

Definition 1 (c-Point Location in Equal Balls (c-PLEB). Given n unit balls centered at $P=\{p_1, ..., p_n\}$ in $\mathcal{M}=(X, d)$, devise a data structure which for any query point $q \in X$ does the following:

- if there exists $p \in P$ with $q \in B(p, 1)$ then return YES and a point p' such that $q \in B(p', c)$,
- if $q \notin B(p, c)$ for all $p \in P$ then return NO,
- if for the point p closest to q we have $1 \leqslant d(q, p) \leqslant c$ then return either YES or NO.

In [IM98] it was proved that given an algorithm for c-PLEB which uses $f(n)$ space on an instance of size n where f is convex, there is a data structure for $c-NNS$ problem requiring $O(f(n \operatorname{poly}(\log n, 1/(c-1))))$ space and using $O(\operatorname{poly}(\log n, 1/(c-1)))$ invocations to c-PLEB per query. Thus in this paper we will concentrate on solving the c-PLEB problem.

In addition, in this paper we assume that for all dimensions $i=1..d$ the ith coordinate of all points in the database are different. This assumption can be taken care of by using standard perturbation techniques without increasing the complexity of the algorithms.

All logarithms have base equal to 2.

3. SOLVING c-PLEB FOR l_∞

In this section we present an algorithm for solving the $(4\lceil \log_{1+\rho} \log 4d \rceil + 1)$-PLEB in l_∞ (for any parameter $\rho > 0$) using $O(dn^{1+\rho})$-size storage and having $O(d \log n)$ query time. The algorithm is based on the notion of a hyperplane separator, defined as follows.

Definition 2. For $\alpha, \beta, \gamma > 0$ such that $\alpha+\beta+\gamma=1$, a hyperplane H is a (α, β, γ)-separator for a point set P if:

- the set

$$P_L = \{p \in P \mid B(p, 1) \cap H = \varnothing \text{ and } p \text{ lies on the left side of } H\}$$

has cardinality $\alpha\, |P|$,

- the set

$$P_M = \{p \in P \mid B(p, 1) \cap H \neq \varnothing\}$$

has cardinality $\beta\, |P|$,

- the set

$$P_R = \{p \in P \mid B(p, 1) \cap H = \varnothing \text{ and } p \text{ lies on the right side of } H\}$$

has cardinality $\gamma\, |P|$.

The basic idea of the algorithm is as follows. We use hyperplanes $H_i(t)$ consisting of all points with the ith coordinate equal to t. We try to find hyperplane separator

H having the property that $|P_R|$ is not "much smaller" than $|P_M|$. If H exists, we divide P into $P_1 = P_L \cup P_M$ and $P_2 = P_R \cup P_M$ and apply algorithm recursively on P_1 and P_2; we prove that (due to the fact that $|P_R|$ is comparable to $|P_M|$) the increase in storage caused by duplicating P_M is moderate. If H does not exist, we prove that a large subset C of P has a small diameter. In such a case we pick any point from C as its representative and apply the algorithm recursively on $P-C$.

For any (α, β, γ)-separator, define the quantity

$$L(\beta, \gamma) = \log_{1/(\gamma+\beta)} \frac{\beta+\gamma}{\gamma} = \log_{1/(\gamma+\beta)} \frac{1}{\gamma} - 1.$$

The quantity $L(\beta, \gamma)$ will upper bound the exponent ρ in the space bound $n^{1+\rho}$; therefore, it is of interest to keep it low. In order to do this, that it is of crucial importance to keep $1-(\gamma+\beta)=\alpha$ fairly large (greater than a constant), since otherwise $L(\beta, \gamma) = \Omega(1/\alpha)$. On the other hand, large $1/\gamma$ is not really that much of a problem, since the dependence of L on $1/\gamma$ is only logarithmic. We will exploit this asymmetry below.

Also, note that $L(\beta, \gamma) > 0$ for any $\beta, \gamma > 0$ such that $\beta + \gamma < 1$. The following Lemma forms a basis of our algorithm.

Lemma 1. *For any set P and $\rho > 0$*

1. *Either there exists an (α, β, γ)-separator $H_j(t)$ for P such that $L(\beta, \gamma) \leqslant \rho$ and $\alpha, \gamma > \frac{1}{4d}$, or*

2. *There is a set $C \subset P$ of cardinality $|P|/2$ having diameter at most* $4\lceil \log_{1+\rho} \log 4d \rceil$.

Proof. We assume that for every $H_j(t)$ with $|P_R| > \frac{1}{4d}|P|$ the value of $L(\beta, \gamma)$ is at least ρ and show that it implies the existence of C as above. For any point $p \in P$ let $p_{|j}$ denote the jth coordinate of p; for simplicity in the following we assume $j=1$. For an integer i define $S_i = \{p \in P \mid p_{|1} \in [2i, 2i+2)\}$ and $s_i = |S_i|/n$. Note that the sets S_i are disjoint and therefore $\sum_i s_i = 1$. Define $t_i = \sum_{j \geqslant i} s_i$ and $t'_i = \sum_{j \leqslant i} s_i$. By translating the origin we can make sure that $t_0 = 1/2$ (note, that by our assumption, no two database points have the same value of jth coordinates). ▮

Claim 1. *For any $i \geqslant 0$ such that $t_{i+1} > \frac{1}{4d}$ we have $t_{i+1} < t_i^{1+\rho}$. Thus, $t_i \leqslant \frac{1}{2^{(1+\rho)^i}}$ for such i.*

Proof. Consider a plane $H_1(2i+1)$. Note that $t'_{i-1} \geqslant t_{i+1}$ for $i \geqslant 0$. We know that as long as $t_{i+1} > \frac{1}{4d}$ we have

$$L(s_i, t_{i+1}) > \rho.$$

or equivalently

$$\log_{\frac{1}{s_i+t_{i+1}}} \frac{s_i+t_{i+1}}{t_{i+1}} > \rho$$

or

$$\frac{s_i+t_{i+1}}{t_{i+1}} > \frac{1}{(s_i+t_{i+1})^\rho}.$$

As $t_{i+1}+s_i=t_i$ the above inequality implies

$$\frac{t_i}{t_{i+1}} > \frac{1}{t_i^\rho}$$

which implies the first part of the claim. The second part follows from the fact that $t_0=\frac{1}{2}$ and the first part. ▌

We now use Claim 1 to prove the theorem. Take $i=\lceil \log_{1+\rho} \log 4d \rceil$; from Claim 1 and monotonicity of t_i it follows that $t_i \leqslant \frac{1}{4d}$. By a symmetric argument, $t'_{-(i+1)} \leqslant \frac{1}{4d}$. Therefore, we have $\sum_{-i\leqslant j\leqslant i-1} s_i \geqslant 1-\frac{1}{2d}$. Thus if we delete from P all points from S_j for $i \notin [-i, i-1]$ we decrease the size of P by at most a factor of $1-\frac{1}{2d}$. The difference in the first coordinate for any pair of remaining points is at most $4i$. By repeating the above argument for each dimension we obtain a set (call it C) of size at least $\frac{n}{2}$ such that any pair of points differ on each coordinate by at most $R=4i=4\lceil \log_{1+\rho} \log 4d \rceil$; thus its diameter is bounded by R as well. ▌

Lemma 1 facilitates the following algorithm.

Preprocessing. For a given set P we build a tree data structure in the following way. The root of the tree corresponds to the set P; its subtrees are created as follows. If we find a separator with $L(\beta,\gamma) \leqslant \rho$ then we create two point sets $P_1=P_L \cup P_M$ and $P_2=P_M \cup P_R$ and build the data structures recursively on each of them. We also store the parameters of the separator at the root node; we call such a node a *separator* node. Otherwise, we know that P contains C of small diameter such that $|C| \geqslant n/2$. In this case we store the center of the box defining C, a point from C (representing C) and build recursively a data structure for $P-C$. We call such a node a *box* node. Thus each non-leaf node has at most two children. ▌

Searching. Given a query point q searching is performed recursively as follows. If the root is a separator node, we check if q is on the left or on the right side of the separating hyperplane; in the first case we move to node P_1, otherwise to P_2. On the other hand, if the root is a box node, we compute the distance t between q and the box. If the distance is less than 1, we output any point from the box (which is within at most distance $R+1$ from q). If $t>1$, we know that the point does not belong to a ball around any point from C, thus we move to the node $P-C$.

The correctness of the above procedures can be easily seen. It is thus sufficient to analyze the complexity of the procedure. ▌

LEMMA 2. *The search time is $O(d \log n)$.*

Proof. Consider the path from the root to the leaf traversed by the searching algorithm for given q. The number of box nodes on that path can be at most $\log n$, as a child of such a node contains at most half of the points of its parent. The time spent per each such node is $O(d)$, therefore the total number of operations performed at box nodes is $O(d \log n)$. On the other hand, the number of separator nodes can be at most $O(d \log n)$, as a child of such a node contains at most a fraction of $(1 - \frac{1}{4d})$ of the points of its parent. The time spent per separator is $O(1)$, thus the total incurred cost is $O(d \log n)$. ∎

In order to analyze the size of the data structure and the construction time, we will use the following lemma. Let T_P be a tree representing the data structure constructed for a set P. We define T'_P to be a (conceptual) binary tree obtained from T_P by modifying the box nodes in a following way. Instead of removing the points which fall inside the box, we create a balanced binary tree with the number of leaves equal to the number of inside points and attach it as the second child of the box node. The resulting data structure has still depth $O(d \log n)$ (since it could only increase by $\log n$); moreover the size of T_P is clearly upper bounded by the size of T'_P. In the following lemma we give a bound on the size of T'_P and then we use it to bound the space/time requirements of the preprocessing stage.

Lemma 3. *The tree T'_P has at most $O(n^{1+\rho})$ nodes.*

Proof. Let $N(n)$ be the maximum number of leaf nodes of T'_P for $|P| = n$. We will show inductively that $N(n) \leqslant n^{1+L}$ where $L = L(\beta, \gamma) \leqslant \rho$. Assume that $N(m) \leqslant m^{1+L}$ for all $m < n$. We analyze only separator nodes, since the inductive step holds trivially for box nodes. For each separator node we can write the inequality

$$N(n) \leqslant N((\alpha + \beta)\, n) + N((\beta + \gamma)\, n).$$

By the inductive assumption we get

$$\begin{aligned} N(n) &\leqslant ((\gamma + \beta)\, n)^{1+L} + ((\beta + \alpha)\, n)^{1+L} \\ &= n^{1+L}((\gamma + \beta)(\gamma + \beta)^L + (1 - \gamma)^{1+L}) \end{aligned}$$

Since $L(\beta, \gamma) = L$ we can bound the latter quantity by

$$\begin{aligned} &\leqslant n^{1+L}\left((\gamma + \beta) \frac{\gamma}{\gamma + \beta} + (1 - \gamma) \right) \\ &\leqslant n^{1+L}(\gamma + (1 - \gamma)) \\ &= n^{1+L}. \end{aligned}$$ ∎

Corollary 1. *The data structure T_P is of size at most $O(dn^{1+\rho})$ and can be constructed in time at most $O(d^2 n^{1+\rho} \log n)$.*

Proof. The first part follow from the upper bound for the size of T_P via T'_P and the fact that both separator and box nodes have description size $O(d)$. In order to bound the running time notice that the processing of a node (say v) with m_v points (i.e., computing s_i's and deciding whether the node is a separator node or a box node) can be done in dm_v time. Since T'_P is a binary tree, m_v can be bounded by the number of leaves of v in T'_P. Since the tree has depth $O(d \log n)$, each leaf node can be "charged" in this way only $O(d \log n)$ times. Therefore, the sum of all m_v's can be bounded by $O(d \log n \cdot n^{1+\rho})$, which gives the promised bound on the preprocessing time. ▮

In this way we proved the following theorem.

THEOREM 1. *For any $\rho > 0$, there exists a deterministic algorithm for the* $(4 \lceil \log_{1+\rho} \log 4d \rceil + 1)$-*PLEB using* $O(dn^{1+\rho})$ *storage with query time* $O(d \log n)$. *The data structure can be constructed in* $O(d^2 n^{1+\rho} \log n)$ *time.*

3-Approximation. The above result provides a 5-approximation algorithm with space $n^{\log d+1}$. Unfortunately the factor 5 is tight for the above algorithm, if we require $n^{O(\log d)}$ storage. To see this, consider a set of $2d$ points such that for each coordinate i data set with the following property for each coordinate i and value $v \in \{-1, 1\}$ there is exactly one point p such that $p_{|i} = -2v$. By replicating this data set $\frac{n}{2d}$ times we obtain n points such that:

- the fraction of points p s.t. $p_{|i} = -2$ is $\frac{1}{2d}$
- the fraction of points p s.t. $p_{|i} = 2$ is $\frac{1}{2d}$
- the fraction of points p s.t. $p_{|i} = 0$ is $1 - 1/d$

Now, one can verify that for this data set no good separator exists. However, the diameter of the data set is equal to 4. Therefore the approximation factor is equal to 5.[3]

In the following we present a modified version of the algorithm which achieves factor 3 within the space bound $n^{\log d+1}$. The basic idea is to enhance the search procedure for the clusters of diameter 4 contained in $[-2, 2]^d$ (call it C). Firstly, note that we can assume $q \in [-3, 3]^d$. Moreover, observe that if the query point q belongs to $[-1, 1]^d$, then any point from C is within distance 3 from q, so we are done. Therefore, we only have to take care of the situation when there is i such that $|q_{|i}| \in (1, 3]$. Note that in such a case the only points p from P which could be within distance 1 from q are those s.t. $p_{|i} > 0$; call this set P_i. If we knew i during preprocessing, we could build a separate search structure for P_i. Since $|P_i| \leqslant n/2$, this approach would be very efficient. Of course, i is not known during the preprocessing. However, it turns out that we can afford to build the structures for *all* $i = 1 \ldots d$ and still keep the $n^{O(\log d)}$ space bound. Let $L = \log d + 1$. We need to show that the $N(n) \leqslant n^{1+L}$, with the notation as in the proof of Theorem 1. To see this, observe that the number of nodes in the data structures built for a cluster C of size n is bounded by

$$d \cdot N(n/2) \leqslant d \cdot n^{1+L}/2^{1+L} = n^{1+L}/4$$

and thus $N(n) \leqslant n^{1+L}$. The search time can be verified to remain $O(d \log n)$. ▮

[3] One could observe that the algorithm is not completely doomed, since the data set in fact *does* contain a subset of size $n/2$ of diameter 2 (the set of points with all coordinates non-negative). It is possible, however, to construct a similar data set for which no small subset has diameter less than 5, while keeping the fraction of coordinates with values in $\{-2, 2\}$ small. Since this counterexample does not seem to be particularly informative, we omit the description.

Modifications and other applications. By modifying the above construction slightly one can obtain a data structure of size $O(dn+n^{1+\rho})$ but with query time $O(d(d+\log n))$. To this end notice that the size of a separator node is constant. The size of the leaf can be made constant as well, by storing a pointer to the structure containing the point-set. Finally, we observe that if a box node contains at least $2d$ points, then by deleting half of them and adding d-word description of the box we only decrease the total storage; therefore if all nodes contain at least $2d$ points then the cost of box nodes is negligible. We exploit this fact by creating a tree as above except for the fact that now each leaf contains $O(d)$ points; thus we have to add $O(d^2)$ to the search time (we perform linear search within each leaf node).

The aditional benefit of the above construction is that if the set P has description size M, then the size of the data structure is in fact $O(M+n^{1+\rho})$. A situation when $M=o(nd)$ occurs for example when P is defined as all substrings (of length d) of a given sequence of length n. In this case $M=O(n)$ and the above construction saves up to a factor of d in the storage requirements.

Finally, we mention applications to similarity searching in product metrics. By the embedding theorem mentioned in the introduction we can embed any finite metric (say of size s) in l_∞^d with distortion t, where $d=s^{\frac{1}{\lfloor (t+1)/2 \rfloor}}$. Since the product metric of k identical l_∞^d metrics is just an l_∞^{dk} metric, we obtain a $tc(d,\rho)$-approximation algorithm for a product of k arbitrary finite metrics with query time $\tilde{O}(kd)$ and storage $\tilde{O}(kdn^{1+\rho})$. Since by the result of [Amb98] each embedding can be done in deterministic $\tilde{O}(ds^2)$ time, the preprocessing time is bounded by $\tilde{O}(kds^2+(kd)^2\, n^{1+\rho})$. ▮

4. LOWER BOUND

In this section we provide an indication that getting improving the approximation factor below 3 while maintaining space subexponential in d could be impossible. We achieve by showing a reduction from the *superset query* to c-approximate PLEB in l_∞ for any $c<3$.

We define the superset query problem as follows.

DEFINITION 3 (SUPERSET QUERY (SQ)). Given n sets $S_1 \dots S_n$ such that $S_i \subset X=\{1 \dots d\}$, devise a data structure which for any query set $Q \subset X$, does the following: if there exists S_i such that $S_i \subset Q$ then return S_i, else return NO.

We refer to superset query problem with parameters n, d as the (n,d)-SQ problem. The problem has been investigated for last few decades (e.g., see [Riv74] or [Knu73], p. 557), but all solutions discovered so far have either $\Omega(n)$ query time or $2^{\Omega(d)}$ space. Thus, it is natural to assume the problem is hard (see [BOR] for some hardness results for this problem).

We establish the hardness of c-PLEB in the following lemma.

LEMMA 4. *For any $1 \leqslant c<3$ the (n,d)-SQ problem is reducible to the c-PLEB problem in l_∞^d.*

Proof. Define function g as

$$g(x) = \begin{cases} 0 & \text{if} \quad x = 0 \\ \frac{2}{3} & \text{if} \quad x = 1 \end{cases}$$

and a function f as

$$f(x) = \begin{cases} \frac{1}{3} & \text{if} \quad x = 0 \\ 1 & \text{if} \quad x = 1 \end{cases}$$

Now define functions F and G which map sets into points as $F(S) = f(\chi_S)$ and $G(Q) = g(\chi_Q)$, where $f((b_1 \ldots b_u))$ denotes $(f(b_1), \ldots, f(b_u))$ and χ_A denotes the characteristic vector of A. We reduce SQ to c-PLEB by mapping all sets in the database to points by means of function F and building the c-PLEB data structure for this set of points. Then, in order to find the answer for a query set Q, we perform the c-PLEB query with argument $G(Q)$.

The correctness of the reduction follows from the following claim.

CLAIM 2. *For $S, Q \subset X$ let $D(Q, S) = |G(Q) - F(S)|_\infty$. Then $D(Q, S) = \frac{1}{3}$ if $S \subset Q$ and is equal to 1 otherwise.* ▮

ACKNOWLEDGMENTS

The author thanks Ashish Goel and the anonymous referees for helpful comments on the preliminary version of this paper.

REFERENCES

[ALSS95] R. Agrawal, K. Lin, H. S. Sawhney, and K. Shim, Fast similarity search in the presence of noise, scaling, and translation in time-series databases, *in* "Proc. 21st International Conference on Very Large Databases," 1995.

[AE] P. K. Agarwal and J. Erickson, Geometric range searching and its relatives, *in* "Advances in Discrete and Computational Geometry" (B. Chazelle, E. Goodman, and R. Pollack, Eds.), Amer. Math. Soc., Providence, RI, 1998.

[Amb98] A. Ambainis, manuscript, 1998.

[AGGM98] P. K. Agarwal, A. Gionis, L. Guibas, and R. Motwani, Rank-based similarity searching, in preparation, 1998.

[AM93] S. Arya and D. Mount, Approximate nearest neighbor searching, *in* "Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms," pp. 271–280, 1993.

[AMNSW94] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu, An optimal algorithm for approximate nearest neighbor searching, *J. Assoc. Comput. Mach.* **45** (1998), 891–923.

[Bern93] M. Bern, Approximate closest-point queries in high dimensions, *Inform. Process. Lett.* **45** (1993), 95–99.

[BOR] A. Borodin, R. Ostrovsky, and Y. Rabani, Lower bounds for high dimensional nearest neighbor search and related problems, *in* "STOC'99."

[Bou85] J. Bourgain, On Lipschitz embedding of finite metric spaces in Hilbert space, *Israel J. Math.* **52** (1985), 46–52.

[Chan97] T. M. Chan, Approximate nearest neighbor queries revisited, *Discrete Comput. Geom.* **20** (1998), 359–373.

[Cl88] K. Clarkson, A randomized algorithm for closest-point queries, *SIAM J. Comput.* **17** (1988), 830–847.

[Cl94] K. Clarkson, An algorithm for approximate closest-point queries, *in* "Proceedings of the Tenth Annual ACM Symposium on Computational Geometry," pp. 160–164, 1994.

[Coh97] E. Cohen, Fast algorithms for t-spanners and stretch-t paths, *SIAM J. Comput.* (1997).

[Fagin96] R. Fagin, Combining fuzzy information from multiple systems, *in* "Proceedings of the ACM Symposium on Principles of Database Systems," pp. 216–227, 1996.

[Fagin98] R. Fagin, Fuzzy queries in multimedia database systems, *in* "Proceedings of the ACM Symposium on Principles of Database Systems," 1998 (invited paper).

[QBIC] C. Faloutsos, R. Barber, M. Flickner, W. Niblack, D. Petkovic, and W. Equitz, Efficient and effective querying by image content, *J. Intell. Inform. Systems* **3** (1994), 231–262.

[FI98] M. Farach-Colton and P. Indyk, Approximate nearest neighbor algorithms for Hausdorff metric via embeddings, manuscript, 1998.

[GBT84] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, Scaling and related techniques for computational geometry, *in* "Proceedings of the 16th Annual ACM Symposium on Theory of Computing," pp. 135–143, 1984.

[GIM97] A. Gionis, P. Indyk, and R. Motwani, Similarity search in high dimensions via hashing, manuscript, 1997.

[I97] P. Indyk, Deterministic superimposed coding with applications to pattern matching, *in* "Proceedings of the 38th Symposium on Foundations of Computer Science," 1997.

[IM98] P. Indyk and R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, *in* "Proceedings of the 30th Annual ACM Symposium on Theory of Computing," 1998.

[JL84] W. B. Johnson and J. Lindenstrauss, Extensions of Lipschitz mapping into Hilbert space, *Contemp. Math.* **26** (1984), 189–206.

[HKP97] J. Hellerstein, C. H. Papadimitriou, and E. Koutsoupias, On the analysis of indexing schemes, *in* "Symposium on Principles of Database Systems," 1997.

[Kl97] 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.

[Knu73] D. Knuth, "The Art of Computer Programming: Sorting and Searching," 1973.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, Efficient search for approximate nearest neighbor in high dimensional spaces, *in* "Proceedings of the 30th Annual ACM Symposium on Theory of Computing," 1998.

[Mat] J. Matoušek, On embedding expanders into l_p spaces, *Israel J. Math.*

[Mat96] J. Matoušek, On the distortion required for embedding finite metric spaces into normed spaces, *Israel J. Math.* **93** (1996), 333–344.

[Me93] S. Meiser, Point location in arrangements of hyperplanes, *Inform. Comput.* **106** (1993), 286–303.

[MP69] M. Minsky and S. Papert, "Perceptrons," MIT Press, Cambridge, MA, 1969.

[Over83] M. H. Overmars, "The Design of Dynamic Data Structures," Springer-Verlag, Berlin, 1983.

[Riv74] R. L. Rivest, "Analysis of Associative Retrieval Algorithms," Ph.D. thesis, Stanford University, 1974.