# Dimensionality Reduction Techniques for Proximity Problems

PIOTR INDYK [*]
STANFORD UNIVERSITY
indyk@cs.stanford.edu
(650-723-4532)

August 29, 1999

**Abstract**

In this paper we give approximation algorithms for several proximity problems in high dimensional spaces. In particular, we give the first Las Vegas data structure for $(1+\epsilon)$-nearest neighbor with polynomial space and query time polynomial in dimension $d$ and $\log n$, where $n$ is the database size. We also give a deterministic 3-approximation algorithm with similar bounds; this is the first deterministic constant factor approximation algorithm (with polynomial space) for any norm. For the closest pair problem we give a roughly $n^{1+\rho}$ time Las Vegas algorithm with approximation factor $O(1/\rho \log 1/\rho)$; this is the first Las Vegas algorithm for this problem. Finally, we show a general reduction from the furthest point problem to the nearest neighbor problem. As a corollary, we improve the running time for the $(1+\epsilon)$-approximate diameter problem from $n^{2-O(\epsilon^2)}$ to $n^{2-O(\epsilon)}$.

Our results are unified by the fact that their key component is a dimensionality reduction technique for Hamming spaces.

## 1 Introduction

The proximity problems is a class of geometric problems which involve the notion of a distance between points in a $d$-dimensional space. For example, the closest pair problem, furthest pair (or diameter) problem and nearest neighbor search all belong to this class. If the dimension $d$ is low, these problems have very efficient solutions [13, 3, 12]. However, the running time and/or space requirements of these algorithms grow exponentially with the dimension. This is unfortunate, since the high-dimensional versions of the above problems are of major and growing importance to a variety of applications, usually involving similarity search or clustering; some examples are information retrieval, image and video databases, vector quantization, data mining and pattern recognition. Therefore, a lot of recent research focused on *approximate* algorithms for these problems [4, 10, 7, 11, 6, 2]; this relaxation enables to overcome the "curse of dimensionality".

In this paper we present several new results for the aforementioned proximity problems. They are unified by the fact that their key component is a dimensionality reduction technique for Hamming spaces (which we describe in more detail at the end of this Section).

**Our results.** The first problem we address is the $c$-nearest neighbor problem. In this problem the goal is to construct (for a given set $P$ of $n$ points from $R^d$) a data structure, which given a

[*]Supported by Stanford Graduate Fellowship and NSF Grant IIS-9811904

query point $q$, finds any $c$-approximate nearest neighbor of $q$ in $P$; the latter is defined as any point $p'$ whose distance to $q$ is bounded by $c$ times the distance from $q$ to its nearest neighbor in $P$. Several algorithms have been proposed recently for this problem for the cases when the similarity between points is measured by a dot-product [4], $l_1$ or $l_2$ norm [10, 7, 11] and $l_\infty$ norm [6]. The intriguing common property of almost all of them [4, 10, 7, 11] is that they are randomized Monte Carlo, i.e. have small probability of returning an incorrect answer (the solution of [6] is deterministic, but unlike other solutions does not allow arbitrarily small approximation error). This situation stands in the contrast with the state of the art in low-dimensional Computation Geometry, where virtually almost all randomized algorithms are of Las Vegas type (i.e. whose correctness is always guaranteed). Thus one may ask if randomization, in particular of Monte Carlo type, is an inherent property of algorithms for high-dimensional proximity problems. We mention that from the practical perspective the Las Vegas algorithms offer a significant advantage over the Monte Carlo ones (if their running times is comparable) for the following reason: it was observed [1, 5] that the average error incurred by the algorithms is much lower than predicted by theoretical analysis. Therefore, one can significantly speed them up by setting artificially high upper bound for the error. However, a Monte Carlo algorithm *cannot* tell the difference between the usual case when the error is small and a rare case when the error is high (so another trial is needed). Having an algorithm which *always* output correct answer would help to resolve this problem.

Our first sequence of results addresses this issue. In particular, for the $(1+\epsilon)$-nearest neighbor problem in $l_1^d$, we show:

- a Las Vegas algorithm with polynomial (for fixed $\epsilon$) storage and query time poly$(\log n, d, 1/\epsilon)$, for any $\epsilon > 0$; this yields improvement over previous Monte Carlo algorithms of [7, 11] with similar parameters (although with smaller dependence of the storage on $\epsilon$)

- a deterministic algorithm with similar time/space bounds, for $\epsilon > 2$; this is the first deterministic constant factor approximation algorithm with polynomial storage, for any $l_p$ norm.

The results can be generalized to $l_2^d$; however, the approximation factor gets multiplied by $\sqrt{3}$.

We also give a subquadratic Las Vegas algorithm for $O(1)$-approximate dynamic NNS in Hamming space; this, in particular, gives a $O(1)$-approximation algorithm for the Closest Pair problem. We obtain it by combining the dimensionality reduction technique with the verification procedure for the Locality Sensitive Hashing algorithm [7]. Our algorithm runs in time $dn + n^{1+\rho}$ (for any $1 > \rho > 0$) and returns a $O(1/\rho(\log 1/\rho + 1))$-approximate solution. We mention that although one can obtain a subquadratic Las Vegas algorithm for closest pair by combining our Las Vegas nearest neighbor algorithm with the technique of [2], the resulting dependence of the approximation factor on $1/\rho$ would be exponential.

Our second type of results addresses the problem of approximating the *diameter* (i.e. furthest pair) of a set of points in Hamming space.[1] Recently, [2] gave a Monte Carlo $(1+\epsilon)$-approximate algorithm for this problem running in time $O(n^{2-O(\epsilon^2)})$. Here we improve it to $O(n^{2-O(\epsilon)})$ (see page 9 for the exact running time of the algorithm); moreover, our algorithm is much simpler. Our result is obtained via a general reduction from $\epsilon$-Furthest Neighbor Search Problem (FNS) to $\epsilon/c$-Nearest Neighbor Problem (NNS). The value of $c$ is upper bounded by 6 for $\epsilon \leq 2$ (for $\epsilon = 2$ we show a linear time algorithm in general metric spaces) and smaller than 6 for a large range of values of $\epsilon$. Thus any further improvement for the approximate NNS yields an improvement for

[1]By standard embedding techniques (e.g. see [7]) this implies algorithms for $l_1$ and $l_2$ norms with the running time multiplied by $\tilde{O}(d)$.

approximate FNS; in particular, a linear time algorithm for approximate closest pair problem (if one exists) would imply a linear time algorithm for approximate diameter.

**Our techniques.** The main technique used in all of our results is dimensionality reduction in Hamming spaces. Specifically, we use hashing to reduce the dimension to $O(\log n)$ and still preserve the gap between "small" and "large" distances. Although techniques has been used earlier ("traces" in [11] and Locality Sensitive Hashing in [7]), we need to proceed more carefully; in particular, some of our results require the hashing to be *deterministic.* It is interesting to note that the dimensionality reduction serves very different purposes depending on the application: for the Las Vegas NNS algorithm, it allows us to reduce the storage; for the Locality Sensitive Hashing, it allows us to reduce the error from $\log d$ to constant; for the Furthest Neighbor to Nearest Neighbor reduction, it allows us to relate the approximation factors for these two problems. Thus we believe that this technique will find further applications in designing efficient algorithms for high dimensional proximity problems.

Another technique introduced in this paper is "divide and conquer" on the dimensionality of the space. In particular, we show how (in certain situations) to reduce the original problem to several subproblems with smaller (and more tractable) dimensionality. Finally, we introduce a greedy set cover algorithm for the verification of correctness of Locality Sensitive Hashing data structure; the rationale is that, since we cannot verify the correctness for all $2^d$ queries, we apply approximation algorithm (which gives slightly worse bounds). It is interesting if there are other relations between NP-complete problems and proximity problems in high dimensions.

## 2 Preliminaries

**Notation:** We use the following notation. For a metric space $(X, d)$ and $r > 0$ we use $B(p, r)$ to denote the set of points in $X$ within distance $r$ from $p$. We use $d_H(x, y)$, for $x, y \in \Sigma^d$, to denote the Hamming distance between $x$ and $y$ (i.e. the number of positions on which $x$ and $y$ differ). We often skip the subscript $H$ if it is clear from the context which metric is used.

The formal definitions of the problems we address are as follows.

**Definition 1 ($c$-Nearest Neighbor Search Problem ($c$-NNS))** *Given a set $P = \{p_1, \ldots, p_n\}$ of points from some metric space $(X, d)$, devise a data structure which, given any $q \in X$, produces a point $p \in P$ such that $d(q, p) \leq c \min_{p' \in P} d(q, p')$.*

In [7] it was shown that $c$-NNS can be reduced to the (defined below) $(r, rc$-PLEB problem. The (binary-search type) reduction is deterministic and incurs only a *polylog(n)* overhead in the running time and storage requirements. Therefore, in this paper we focus on solving the latter problem, formally defined as follows.

**Definition 2 ($(r, R)$-Point Location in Equal Balls ($(r, R)$-PLEB))** *Given $n$ radius-$r$ balls centered at $P = \{p_1, \ldots, p_n\}$ in a metric $(X, d)$, devise a data structure which for any query point $q \in X$ does the following:*

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

**Definition 3 ($c$-Furthest Neighbor Search Problem ($c$-FNS))** *Given a set $P = \{p_1, \ldots, p_n\}$ of points from some metric space $(X, d)$, devise a data structure which, given any $q \in X$, produces a point $p \in P$ such that $d(q, p) \geq 1/c \max_{p' \in P} d(q, p')$.*

As is in case of $c$-NNS problem, $c$-FNS can be reduced to its decision question similar to $(r, R)$-PLEB, but with balls $B(p, r)$ replaced by their complement. Fortunately, the overhead of the reduction is very small due to the fact that (as we show in this paper) the 3-FNS problem can be solved in any metric space with $O(n)$ preprocessing/storage and constant query time. Therefore, a binary search reduction from $c$-FNS to its decision version incurs only constant overhead.

## 3 Deterministic algorithm for approximate nearest neighbor

In this section we describe a deterministic algorithm for the approximate nearest neighbor problem. The algorithm returns a point whose distance to the query point is at most $3+\epsilon$ times the distance to its nearest neighbor. The algorithm works for Hamming metric; later we show that similar results can be also obtained for $l_1$ norm and (with slightly worse approximation factor) for the $l_2$ norm.

It is clear that we can focus on solving the $(r, R)$-PLEB problem for $R \leq (2+O(\epsilon))r$, since approximate NNS can be reduced to the former problem by using binary search. Our solution consists of the following steps:

1. Hash the space $\{0,1\}^d$ into $\Sigma^{O(R)}$ for some (large) alphabet $\Sigma$
2. Encode each symbol from $\Sigma$ using a binary error-correcting code (with codeword length $O(\log n)$) obtaining a mapping into $\{0,1\}^{O(R \log n)}$
3. Divide the (binary) coordinates into groups of size $O(\log n)$ and solve the Nearest Neighbor problem within each group. During the query processing return the best (closest to the query) answer among all answers obtained from the subproblems.

Below we describe each step in detail.

**Hashing.** The goal of hashing is to obtain a mapping $f : \{0,1\}^d \to \Sigma^D$, such that

- $f$ is *non-expansive* (i.e. for any $x, y \in \{0,1\}^d$ we have $d(f(x), f(y)) \leq S d(x, y)$, where $S$ is a scaling factor
- $f$ is $(\epsilon, R)$-*contractive* (i.e. for any $x, y \in \{0,1\}^d$ such that $d(x, y) \geq R$ we have $d(f(x), f(y)) \geq SR(1-\epsilon)$)

Moreover, the value of $D$ should be close to $SR$.

The mapping is obtained as follows. Let $\mathcal{H}$ be a family of hash functions $h : [d] \to [v]$ such that for any $x, y \in [d]$ we have

$$\Pr[h(x) = h(y)] \leq \frac{1}{R/\epsilon}$$

We can assume that $P = R/\epsilon$ is a prime number (as otherwise we can always decrease $\epsilon$ by a constant factor so that this fact becomes true). In this case a family of functions $\mathcal{H}_P$ consisting of functions $h(x) = ax \bmod P$, $a \in [P]$, is known to satisfy the above condition. Having $\mathcal{H}$ we compute $f(x)$ as follows. For each $h \in \mathcal{H}$ we construct a mapping $f_h$ which for any $x \in \{0,1\}^d$:

- maps each bit $x_i$ into "bucket" $h(i)$
- sorts all bits $x_i$ within each bucket in ascending order of $i$'s

- concatenates all bits within each bucket into one symbol (empty buckets are represented by an additional special symbol) and outputs the resulting $P$-dimensional vector

The value of $f$ is now defined as concatenation of all $f_h(x)$ for $h \in \mathcal{H}_P$.

If we set the scaling factor $S$ to $|\mathcal{H}|$, then it is easy to see that the mapping $f$ is both non-expansive and $\epsilon$-contractive. The first property follows from the fact that each difference bit between $x$ and $y$ creates at most $S$ difference symbols between $f(x)$ and $f(y)$. On the other hand, the low probability of collision guaranteed by $\mathcal{H}$ implies the second property.

**Coding.** In this stage each element $a$ from $\Sigma$ (constructed implicitly by mapping the whole pointset using $f$) is replaced by a binary word $C(a)$ of length $\lambda$. The words have the property that for each pair of different symbols $a, b \in \Sigma$ we have $d(C(a), C(b)) \in [(1/2(1-\epsilon))\lambda, 1/2\lambda]$; from [14] we know that there exists (efficiently constructible) code with $\lambda = O(|\Sigma|(1/\epsilon)^2)$ having this property. Since the size of $\Sigma$ is bounded by $n\text{poly}(d, 1/\epsilon)$, it follows that $\lambda = O((\log n + \log d + \log 1/\epsilon)/\epsilon^2)$. By using $f$ and $C$, the resulting mapping (say $g$) is clearly non-expansive and $(1-(1-\epsilon)^2)$-contractive. Notice that the dimension $D'$ of the range of $g$ is $P^2 \cdot \lambda$ and the scaling factor $S' = P \cdot \lambda/2$. In particular it implies that for a pair $x, y$ s.t. $d(x, y) \geq R$ we have $d(g(x), g(y)) \geq R' = S'(1-\epsilon)^2 R$, which is $\Theta(D')$.

**Divide and conquer.** During this stage we partition the set $[D']$ of coordinates of $g$'s range into sets $S_1 \dots S_k$ of size $s = O(\log n)$. Ideally, we would like this partition to have the property that the function $g_{|S_i}$ (where $y_{|A}$ denotes the projection of $y$ on the coordinate set $A$) is non-expansive and $\beta$-contractive for some small $\beta$. Unfortunately, it is not difficult to see that it is impossible to achieve this goal for the whole domain of $g$ (which is too big). Therefore, we will only preserve this property for all pairs of points from the pointset $P$. Specifically, we will set $s = \frac{D'}{R'}t$ for $t = C/\epsilon^2 \log n$, in which case (for sufficient constant $C$) a random partition guarantees that for all $x, y \in P$ such that $d(x, y) \geq R$ we have $d(g(x)_{|S_i}, g(y)_{|S_i}) \geq t(1-\epsilon)$. The random choice can be easily derandomized by using the method of conditional probabilities as follows. Assume that the randomized algorithm first chooses $s$ elements of $S_1$, then $s$ elements for $S_2$ and so on. For each pair $x, y \in P$ let $E_{x,y}$ denote the event "$d(g(x)_{|S_i}, g(y)_{|S_i}) < t(1-\epsilon)$". It is not difficult to see that the probability of $E_{x,y}$ conditioned on a partial choice of partition has a hypergeometric distribution, and therefore is easily computable. This implies that the method of conditional probabilities yields polynomial time deterministic algorithm for constructing the required partition.

Having the partition, we proceed as follows. Let $P_i = g(P)_{|S_i}$; notice that $P_i$ contains points from $\{0,1\}^s$. For each $i$ we build a data structure $D_i$ solving a nearest neighbor problem for $P_i$. This is an easy task, since the dimensionality of the space is $s = O(\log n)$, and therefore we can precompute answers to all $2^s$ possible queries using polynomial (in $n$) space. The solution to the PLEB problem is now computed by querying all $D_i$'s and checking if the closest point found is close enough to the query point.

It remains to be shown that this approach yields a $3 + O(\epsilon)$-approximate answer. Clearly, it is sufficient to show that the last step (i.e. starting from $\{0,1\}^{D'}$ space) solves $(\frac{R'(1-\epsilon)}{3}, R')$-PLEB problem. Consider (an image of) a query point $q \in \{0,1\}^{D'}$ and assume that there are two points $p, p' \in \{0,1\}^{D'}$ such that $d(q, p) \leq \frac{R'(1-\epsilon)}{3}$ and $d(q, p') > R'$. By pigeonhole principle we know that there exists $S_i$ such that $d(q_{|S_i}, p_{|S_i}) \leq \frac{t(1-\epsilon)}{3}$. On the other hand, we know that $d(p_{|S_i}, p'_{|S_i}) > t(1-\epsilon)$. By triangle inequality

$$\begin{aligned} d(q_{|S_i}, p'_{|S_i}) &\geq d(p_{|S_i}, p'_{|S_i}) - d(q_{|S_i}, p_{|S_i}) \\ &\geq (d(p'_{|S_i}, q_{|S_i}) - d(p_{|S_i}, q_{|S_i})) - d(q_{|S_i}, p_{|S_i}) > \frac{t(1-\epsilon)}{3} \end{aligned}$$

Therefore, the algorithm never returns $p'$ if $p$ exists, and thus it solves the $(\frac{R'(1-\epsilon)}{2}, R')$-PLEB problem.

By the above discussion we prove the following Theorem.

**Theorem 1** *For any $\epsilon > 0$ there exists a deterministic data structure solving the $3+\epsilon$-NNS problem in Hamming metric with poly$(d, \log n, 1/\epsilon)$ query time and poly$(d, 1/\epsilon)n^{O(1/\epsilon^6)}$ space.*

**Other metrics.** The generalization to $l_1$ is immediate, as there exists a simple deterministic embedding of a finite subset $P$ of $l_1^d$ into $O(\Delta d/\epsilon)$-dimensional Hamming metric, where $\Delta$ is the diameter/closest pair ratio for $P$ and the mapping preserves the distances up to a multiplicative factor $1 \pm \epsilon$. Since the construction in [7] yields $\Delta = \text{poly}(d, \log n)$, the query time gets multiplied only by a small factor.

The result for the $l_2$ norm follows from the existence of non-expensive embedding of $l_2^d$ into $l_1^{O(d^2)}$ which contracts distances by at most a factor of $\sqrt{3}$. By applying this embedding and then applying the algorithm for $l_1$ we obtain a deterministic algorithm solving $3\sqrt{3}$-NNS.

**Las Vegas algorithm.** Unfortunately, we do not know if/how it is possible to obtain a deterministic $(1+\epsilon)$-approximate algorithm for NNS. However, by using the techniques described below, we *can* provide a randomized *Las Vegas* algorithm for this problem. More specifically, we present an algorithm which builds a data structure which for every query $q$ computes correctly its $(1+\epsilon)$-approximate nearest neighbor, but (with very small probability) exceeds the poly$(d, \log n, 1/\epsilon)$ time bound. Notice, that it is sufficient that the algorithm either returns correct answer or (with very small probability) returns a special don't know symbol "?" for some queries (in which case we can apply linear search). Thus (unlike the previous algorithms of [7] and [11]) each time the algorithm returns an answer, we are 100% certain about its correctness.

The data structure is built similarly as above. The only difference is that this time in the "divide and conquer" step the partition $S_1 \ldots S_k$ is chosen at random. From the previous discussion we know that for any query $q$ and $S_i$ as above there is a high probability that for any point $p' \in P$ such that $d(g(q), g(p')) \geq R'$ we have $d(g(q)_{|S_i}, g(p')_{|S_i}) \geq t(1-\epsilon)$. If the latter inequality indeed holds for all $p'$, the algorithm returns a correct answer. Otherwise, it may happen that the algorithm returns (incorrectly) $p'$ as an answer while it should have returned $p$. However, we can detect that case by checking if $d(g(q)_{|S_i}, g(p')_{|S_i})$ closely approximates $d(q, p')$ and returning "?" if it is not the case.

## 4 Furthest to nearest neighbor reduction

In this section we describe another application of dimensional hashing, namely a reduction from $(1+\epsilon)$-FNS to $(1+\epsilon/6)$-NNS (in Hamming spaces), for $\epsilon \in [0, 2]$. For $\epsilon > 2$ the furthest neighbor problem can be solved much more efficiently in general metric spaces (as we show at the end of this section) and therefore the case when $\epsilon > 2$ is not interesting.

The reduction is based on the following simple idea. Let $p, q \in \{0,1\}^d$; then $d(p,q) = d - d(p, \overline{q})$, where $\overline{q}$ denotes a complement of $q$. This implies the following fact: if $P$ is a set of points in $\{0,1\}^d$, $q \in \{0,1\}^d$ and $p \in P$ is the nearest neighbor of $q$ in $P$, then $y$ is also a furthest neighbor of $\overline{q}$ in $P$. Similarly, the furthest neighbor of a point is a nearest neighbor of its complement. Therefore, the *exact* versions of furthest and nearest neighbor are essentially equivalent.

Unfortunately, it is not difficult to observe that the above reduction does not necessarily preserve *approximation*. Assume that the furthest neighbor $p$ of $q$ in $P$ is within distance $R$ from $q$. If we find an $E$-approximate nearest neighbor of $\overline{q}$ in $P$, then we can obtain a point $p' \in P$ such that $d(\overline{q}, p') = Ed(\overline{q}, p) = E(d - R)$. Therefore

$$\frac{d(q,p)}{d(q,p')} \leq \frac{R}{d - E(d-R)}$$

Thus if we want to obtain an $E'$-approximate algorithm for FNS, we need to choose $E$ to make sure that $E' \geq \frac{R}{d-E(d-R)}$, or equivalently

$$E \leq \frac{d - R'}{d - R} = \frac{1-\rho'}{1-\rho}$$

where $R' = E'R$, $R = \rho d$ and $R' = \rho' d$.

It is easy to see that the relation between $E$ and $E'$ depends significantly on the ratio of $d$ to $R$: the smaller it is, the more efficient is the reduction. In order to reduce this ratio, we will apply the dimensional hashing technique similarly to section 3. However, in order to obtain better bounds, we will modify the first step (mapping $f$ from $\{0,1\}^d$ to $\Sigma^D$) and replace the deterministic hashing scheme by the following randomized procedure: for any $x \in \{0,1\}$ and any dimension $i = 1 \ldots D$, the $i$th coordinate of $f(x)$ is obtained by concatenating $k$ randomly chosen bits of $x$ (say from positions $i_1 \ldots i_k$); the indices $i_1 \ldots i_k$ are chosen independently and uniformly at random with replacement. It is not difficult to see that for two points $p, q$ such that $d(p,q) = \rho d$, the expected value of $d(f(p), f(q)$ is $(1-(1-\rho)^k)D$. By using Chernoff bound we can make sure than the actual value of $d(f(p), f(q)$ differs from its expectation by only a factor of $(1 \pm \alpha)$ when $D = \Omega(\log n/\alpha^2)$ is large enough; for simplicity, we skip $\alpha$ (i.e. assume it is 0) in the below calculations. Moreover, in the following we will assume that $\rho k = \Theta(1)$ (since it turns out that the optimal values of $\rho k$ belong to $[0.5, 1.5]$). Therefore, we know that for $\rho$ small enough the value of $1-(1-\rho)^k$ can be approximated by $(1-e^{-\rho k})(1 \pm \alpha')$, for $\alpha'$ arbitrarily close to 0 depending on the value of $\rho$. We can easily make $\rho$ arbitrarily small by adding arbitrary number of dummy coordinates, all set to zero; thus in the following we will assume $d(f(p), f(q)) = (1-e^{-\rho k})D$ (ignoring $\alpha'$) By applying error correcting codes this value goes down to $\frac{1-e^{-\rho k}}{2}D'$ (again ignoring the arbitrarily small error factor).

Denote $(1-e^{-\rho k})/2$ by $\gamma_k(\rho)$. By the above arguments we get that if we want to distinguish between $\rho$ and $\rho' = \rho/E'$ in the FNS problem, we need

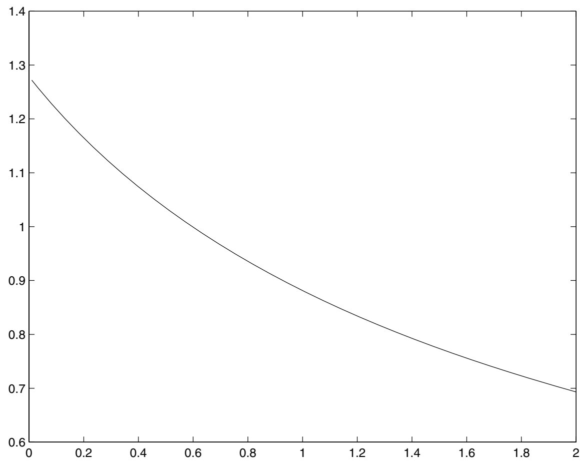
$$E \leq \frac{1-\gamma_k(\rho')}{1-\gamma_k(\rho' E')} \leq \frac{1+e^{-\rho' k}}{1+e^{-\rho' E' k}}$$

Let $\delta = \rho' k$. We will choose the value of $\delta = \delta(E')$ as a function of $E'$ in order to maximize the upper bound expression for $E$. Let $E = 1+\epsilon$ and $E' = 1+\epsilon'$. By numerical calculation we obtain the function $\delta$ as on Figure 1(a); this yields the bound for $\epsilon/\epsilon'$ as presented on Figure 1(b).

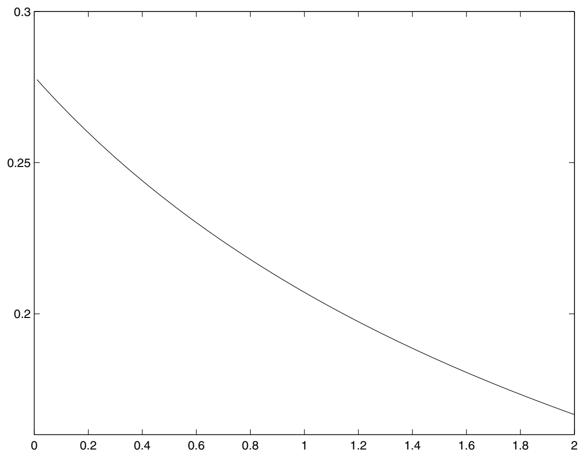
Incidentally, it turns out that for a "round" value of $\epsilon' = 2$, the optimal value of $\delta$ can be computed by hand. We spare the reader the details and just mention that it is equal to $\ln 2 \approx 0.693$ and yields $\epsilon/\epsilon' = 1/6 \approx 0.167$. By combining this observation with the apparent fact[2] that the ratio $\epsilon'/\epsilon$ is decreasing with $\epsilon'$, we obtain the following Theorem.

**Theorem 2** *There is a (randomized Monte Carlo) reduction of* $(1+\epsilon)$*-FNS problem for* $n$ *points in* $\{0,1\}^d$ *to* $(1+\epsilon/6)$*-NNS problem for* $n$ *points in* $\{0,1\}^{poly(d, \log n, 1/\epsilon)}$ *for* $\epsilon \in [0,2]$.

[2]See Figure 1; we defer the formal proof to the final version of this paper.



(a) The value of $\delta$ as a function of $\epsilon'$



(b) The ratio of $\epsilon/\epsilon'$ as a function of $\epsilon'$

Figure 1:

We mention that the reduction works also in the dynamic situation, i.e. when the points are added to and removed from the database. Also, it can be modified to yield *Las vegas* reduction.

It was shown in [7, 5] that there is a $(1+\epsilon)$-NNS algorithm for $n$ points in $\{0,1\}^d$ with preprocessing time/storage $dn + n^{1+1/(1+\epsilon)}$ and query time $dn^{1/(1+\epsilon)}$. By applying the above reduction (using the $\delta$ function from Figure 1) we obtain an algorithm for $(1+\epsilon)$-FNS with preprocessing $Dn + n^{1+\gamma(\epsilon)}$ and query time $Dn^{\gamma(\epsilon)}$ for $D = \text{poly}(d, \log n, 1/\epsilon)$, where $\gamma$ is depicted on Figure 2.

**3-FNS in general metric spaces.** The algorithm is very simple and proceeds as follows. During the preprocessing the algorithm chooses arbitrary point $p_1 \in P$ and computes $p_2 = \arg\max_{q \in P} d(p, q)$. Now, in order to answer a query $q$, the algorithm computes $i = \arg\max_{i=1,2} d(q, p_i)$ and returns $p_i$.

The proof of correctness is as follows. Let $q$ be query point and let $p$ be its furthest neighbor. Then

$$d(q,p) \leq d(p_1,q) + d(p_1,p) \leq d(p_1,q) + d(p_1,p_2) \leq d(p_i,q) + 2d(p_i,q) = 3d(p_i,q)$$

It is easy to see that the analysis is tight, even on a line metric.

## 5 Locality Sensitive Hashing

In this section we show how to obtain another Las Vegas algorithm for approximate NNS by applying the Locality Sensitive Hashing (LSH) technique of [7]. The approximation factor is bounded below by a constant (i.e. is not arbitrarily close to 1), however it uses much less storage than the algorithm from the previous section. We start by describing the idea of LSH and a corresponding Monte Carlo algorithm for NNS. Then we show that by using *greedy set cover* algorithm we can verify the correctness of the LSH data structure and therefore obtain a Las Vegas algorithm.

Intuitively, the main idea of LSH is to solve proximity problems by using hash functions such that the probability of collision is much higher for points close to each other than the ones which are far apart. The formal definition is as follows.

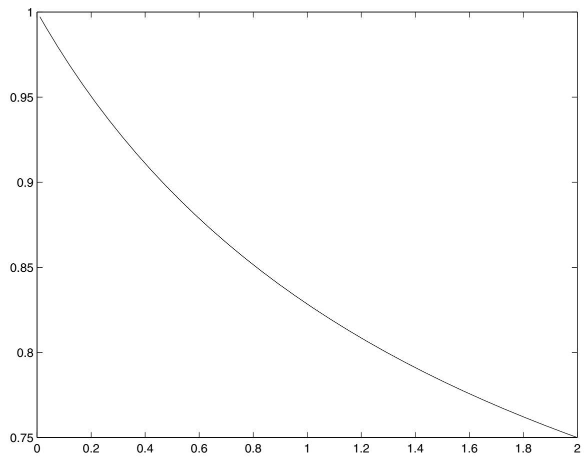


Figure 2: The exponent in the running time of $(1+\epsilon)$-FNS as a function of $\epsilon$.

**Definition 4 ([7])** *A family $\mathcal{H} = \{h : S \rightarrow U\}$ 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 $\Pr_{\mathcal{H}}[h(q) = h(p)] \leq p_2$.*

Although it seems unlikely that LSH families exist for any metric, they do exists for some specific ones. In particular, it is easy to observe the following Fact.

**Fact 1** *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.*

The benefit of LSH is that (as shown in [7]) their existence for a given metric immediately implies sublinear approximate nearest neighbor algorithm for that metric, as shown in the following Theorem.

**Fact 2 ([7])** *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 $\rho = -\frac{\ln p_1}{\ln p_1/p_2}$.*

It was shown in [7] that for the LSH function as in Fact 1 we can show $\rho \leq \frac{1}{\epsilon}$ if $\epsilon = 1 - \frac{r_2}{r_1} > 1$. Below we only describe the algorithm for PLEB using LSH, the proofs can be found in [7].

For $k$ specified later, define a function family $\mathcal{G} = \{g : S \rightarrow U^k\}$ such that $g(p) = (h_1(p), \ldots, h_k(p))$, where $h_i \in \mathcal{H}$. Next, for an integer $l$ we choose $l$ functions $g_1, \ldots, g_l$ from $\mathcal{G}$ independently and uniformly at random. For example, if we use functions from Fact 1, then essentially $g_i(p) = p_{|I_i}$, where $I_i$ is a set of $k$ coordinates sampled with replacement and $p_{|I}$ denotes a projection of $p$ on $I$. 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. If any bucket contains more than one element, we retain only one (chosen arbitrarily). To process a query $q$, we search all buckets $g_1(p), \ldots, g_l(p)$. 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.

Let $W_b(q) = P - B(q, b)$, and $p^*$ be the point in $P$ closest to $q$. The parameters $k$ and $l$ are chosen so as to ensure that with a constant probability there exists $g_j$ such that the following properties hold:

1. $g_j(p') \neq g_j(q)$, for all $p' \in W_{r_2}(q)$, and
2. if $p^* \in B(q, r_1)$ then $g_j(p^*) = g_j(q)$.

One can observe that if the properties (1) and (2) hold for some $g_j$, the search procedure works correctly.

The above algorithms are randomized and have constant probability of correct answer for any *fixed* query $q$. However, we can ensure than (with probability $1 - 2^{-d}$) the data structure is correct for *all* queries. This follows from the fact the total number of possible queries is $2^d$, so we can run $O(d)$ data structures in parallel and always return the best answer. This approach multiplies the storage and query time by $O(d)$ and guarantees that with high probability the data structure is correct for all $q$'s. In other words, with high probability the data structure has the property that for any $q$ such that there exists $p \in P$ with $d(q,p) \leq r_1$, there exists $I_i$ such that $p_{|I_i} = q_{|I_j}$ and for any $p' \in P$ such that $d(q, p') > r_2$ we have $p'_{|I_i} \neq q_{|I_j}$. Below we show how to verify that the data structure has this property for all $p \in P$ where $d(q,p) \leq r$ where $r \ln \frac{d}{r} \leq r_1$. Then we apply the dimensional hashing technique to reduce $d$ to $O(r)$ and in this way obtain a constant factor approximation.

The first step (i.e. verification) is done as follows. Consider any $p \in P$. We need to make sure that for any $q \in B(p, r)$ there exists $I_i$ s.t. $p$ occupies $g_{|I_i}(q)$. We will restate this problem in the *hitting-set* language as follows. For any set $\Delta$ of coordinates we say that $\Delta$ *hits* $I_i$ if $I_i \cap \Delta \neq \emptyset$. Let $A_i$ denote all indices $i$ which contain $p$. Notice that

there exists $\Delta$ with $|\Delta| \leq r$ which hits all $I_i$ for $i \in A_p$

if and only if

there exists $q \in B(p, r)$ such that no $g_i(q)$ contains $p$.

Therefore, it is sufficient to make sure that no small $\Delta$ hits all $I_p = \{I_i, i \in A_p\}$. To this end we apply the *greedy set-cover* algorithm, which constructs a cover for $I_p$ by picking elements hitting the largest number of $I_i$'s. By the result of [15] we know that if there exist a hitting set of size at most $r$, this procedure will find a set of size at most $r \ln \frac{d}{r}$. As we know (with very high probability) such a set does not exist, the greedy procedure will output a set larger than this bound, which proves that no small set $\Delta$ exists, which implies that the data structure is correct.

## 6 Open problems

Our research suggests several possibilities for improvements. In particular:

- give a deterministic $(1+\epsilon)$-NNS algorithm with polynomial storage and sublinear query time
- give a deterministic (or at least Las Vegas) subquadratic $(1+\epsilon)$-approximate algorithm for Closest Pair/Furthest Pair problem, for any $\epsilon > 0$
- construct an explicit embedding of $l_2^d$ into $l_1^{\text{poly}(d)}$ with distortion $1+\epsilon$ for arbitrarily small $\epsilon > 0$
- improve the reduction from FNS to NNS
- improve the upper bound for $c$-FNS in arbitrary metric spaces (or show $c = 3$ is tight)

## References

[1] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching", SODA'94, pp. 573-582.

[2] A. Borodin, R. Ostrovsky and Y. Rabani "Subquadratic Approximation Algorithms For clustering Problems in High Dimensional Spaces", STOC'99, to appear.

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

[4] E. Cohen, D. Lewis, "Approximating matrix multiplication for pattern recognition tasks", SODA'97, pp. 682-691.

[5] A. Gionis, P. Indyk and R. Motwani, "Approximate similarity search in high dimensions via hashing", VLDB'99, to appear.

[6] P. Indyk, "On Approximate Nearest Neighbors in Non-Euclidean Spaces", FOCS'98, pp. 148-155.

[7] P. Indyk, R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality", STOC'98, pp. 604-613.

[8] P. Indyk, R. Motwani, P. Raghavan, S. Vempala, "Locality-preserving hashing in multidimensional spaces", STOC'97, pp. 618-625.

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

[10] J. Kleinberg, "Two Algorithms for Nearest Neighbor Search in High Dimensions", STOC'97, pp. 599-608.

[11] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, " Efficient search for approximate nearest neighbor in high dimensional spaces", STOC'98, pp. 614-623.

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

[13] K. Mulmuley, "Computational geometry", Prentice Hall, 1993.

[14] J. Naor and M. Naor, "Small-bias probability spaces: efficient constructions and applications", *SIAM Journal on Computing*, vol.22, no.4, pp. 838-856.

[15] Rina Panigrahy, personal communication. See also S. Vishwanathan, SODA'96.