Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2005-073
AIM-2005-031

November 4, 2005

New LSH-based Algorithm for Approximate Nearest Neighbor

Alexandr Andoni, Piotr Indyk

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

New LSH-based Algorithm for Approximate Nearest Neighbor

Alexandr Andoni Piotr Indyk

November 3, 2005

Abstract

We present an algorithm for c-approximate nearest neighbor problem in a d-dimensional Euclidean space, achieving query time of $O(dn^{1/c^2+o(1)})$ and space $O(dn+n^{1+1/c^2+o(1)})$.

1 Introduction

A similarity search problem involves a collection of objects (documents, images, etc.) that are characterized by a collection of relevant features and are represented as points in a high-dimensional attribute space; given queries in the form of points in this space, we are required to find the nearest (most similar) object to the query. A particularly interesting and well-studied instance is d-dimensional Euclidean space. This problem is of major importance to a variety of applications; some examples are: data compression, databases and data mining, information retrieval, image and video databases, machine learning, pattern recognition, statistics and data analysis. Typically, the features of each object of interest (document, image, etc) are represented as a point in $\Re^d$ and a distance metric is used to measure 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.

The low-dimensional case (e.g., for $d = 2$) is well-solved, therefore the main issue is that of dealing with a large number of dimensions. Despite decades of intensive effort, the current solutions are not entirely satisfactory; in fact, for large enough d, in theory or in practice, they often provide little improvement over a linear algorithm that compares a query to each point from the database. This phenomenon is often called "the curse of dimensionality". In particular, it was shown in [WSB98] (both empirically and theoretically) that *all* current indexing techniques (based on space partitioning) degrade to linear search for sufficiently high dimensions.

In recent years, several researchers proposed to avoid the running time bottleneck by using *approximation* (e.g., [AMN+94, Kle97, IM98, KOR98, HP01, KL04, HPM04, DIIM04, Pan06], see also [DIS03]). This is due to the fact that, in many cases, approximate nearest neighbor is almost as good as the exact one; in particular, if the

distance measure accurately captures the notion of user quality, then small differences in the distance should not matter. Moreover, an efficient approximation algorithm can be used to solve the *exact* nearest neighbor problem, by enumerating *all* approximate nearest neighbors and choosing the closest point. For many data sets this approach results in very efficient algorithms (see e.g., [ADI+05]).

In [IM98, GIM99], the authors introduced an approximate high-dimensional similarity search scheme with provably sublinear dependence on the data size. It relied on a new method called *locality-sensitive hashing (LSH)*. The key idea is to hash the points using several hash functions so as to ensure that, for each function, the probability of collision is much higher for objects which are close to each other than for those which are far apart. Then, one can determine near neighbors by hashing the query point and retrieving elements stored in buckets containing that point. In [IM98, GIM99] the authors provided such locality-sensitive hash functions for the case when the points live in binary Hamming space[1] $\{0,1\}^d$. In a followup work [DIIM04], the authors introduced LSH functions that work directly in Euclidean space and result in a (slightly) faster running time. The latter algorithm forms the basis of E^{2}LSH package [AI04] for high-dimensional similarity search, which has been used in several applied scenarios. Further, [Pan06] proposed a somewhat different algorithm that exploits similar ideas.

The running times and space requirements achieved by those algorithms is depicted in the following table.

Paper	Metric	Space	Query time	Comments
[IM98, GIM99]	Hamming	$dn + n^{1+1/c}$	$dn^{1/c}$	
[DIIM04]	Euclidean	$dn + n^{1+\rho'(c)}$	$dn^{\rho'(c)} \log n$	$\rho'(c) < 1/c$
[Pan06]	Euclidean	dn	$dn^{\rho''(c)}$	$\rho''(c)/c \to 2.09$
this paper	Euclidean	$dn + n^{1+1/c^2+o(1)}$	$dn^{1/c^2+o(1)}$	

It was conjectured by the second author (e.g., in [Ind03]) that the exponent in the query time can be further improved, to $1/c^2$. The conjecture was motivated by the fact that an algorithm with such exponent exists for the closely related problem of finding the *furthest neighbor* [Ind03].

In this paper we essentially resolve this conjecture by providing an algorithm with query time $dn^{\rho(c)}$ using space $dn + n^{1+\rho(c)}$, where $\rho(c) = 1/c^2 + f(n)$, $f(n) = o(1)$. This significantly improves over the earlier running time of [DIIM04]. In particular, for $c = 2$, our exponent tends to 0.25, while the exponent in [DIIM04] was around 0.42.

Techniques. We obtain our result by carefully designing a family of locality-sensitive hash functions in l_2. The starting point of our construction is the method

[1]The algorithm can be extended to other norms, such as l_2, by using embeddings. However, this extension adds additional complexity to the algorithm.

of [DIIM04]. There, a point p was mapped into $\Re^1$ by using random projection. Then, the line $\Re^1$ was partitioned into equal-length intervals of length w, where w is a parameter. The hash function for p returned the index of the interval containing the projection of p.

An analysis in [DIIM04] showed that the query time exponent has an interesting dependence on the parameter w. If w tends to infinity, the exponent tends to $1/c$, which yields no improvement over [IM98, GIM99]. However, for small values of w, the exponent lies slightly below $1/c$. In fact, the unique minimum exists for each c.

In this paper we utilize a "multi-dimensional version" of the aforementioned approach. Specifically, we first perform random projection into $\Re^t$, where t is superconstant, but relatively small (i.e., $t = o(\log n)$). Then we partition the space $\Re^t$ into cells. The hash function function returns the index of the cell which contains projected point p.

The partitioning of the space $\Re^t$ is somewhat more involved than its one-dimensional counterpart. First, observe that the natural idea of partitioning using a grid does not work. This is because this process roughly corresponds to hashing using concatenation of several one-dimensional functions (as in [DIIM04]). Since the LSH algorithms performs such concatenation anyway (see Preliminaries), grid partitioning does not result in any improvement. Instead, we use the method of "partitioning by balls", introduced in [CCG+98] in the context of embeddings into tree metrics. Its idea is as follows. Create a sequence of balls $B_1, B_2 \ldots$, each of radius w, with centers chosen independently "at random". Each ball B_i then defines a cell, containing points $B_i - \cup_{j<i} B_j$.

In order to apply this method in our context, we need to take care of a few issues. First, we cannot use the method as given, since locating a cell containing a given point could take a long time. Instead, we show that one can simulate the above procedure by replacing each ball by a "grid of balls". It is not difficult then to observe that a finite (albeit exponential in t) number of such grids suffices to cover all points in $\Re^t$. Since t is sub-logarithmic in $\log n$, one can enumerate all grids in time $n^{o(1)}$.

The second and the main issue is the choice of w. Again, it turns out that for large w, the method yields only the exponent of $1/c$. However, a careful analysis shows that, as in the one-dimensional case, the minimum is achieved for finite w. In particular, for proper w, the exponent tends to $1/c^2$ as t tends to infinity.

2 Preliminaries

2.1 Notation

Through the paper, we work in the Euclidean space. For a point $p \in \Re^d$, we denote by $B(p, r)$ the ball centered at p with radius r. We also call $\mathrm{Vol}^d(r)$ the volume of a ball with radius r in $\Re^d$. $\mathrm{Vol}^d(r)$ is equal to $\frac{2\pi^{d/2}}{d\Gamma(d/2)} r^d$ (see, for example, [Pis89], page 11).

2.2 Problem definition

In this paper, we solve the c-approximate near neighbor in l_2, the Euclidean space.

Definition 2.1 (c-approximate near neighbor, or c-NN). *Given a set P of points in a d-dimensional Euclidean space $\Re^d$, and parameters $R > 0$, $\delta > 0$, construct a data structure which, given any* query *point q, does the following with probability $1 - \delta$: if there exists an R-near neighbor of q in P, it reports some cR-near neighbor of q in P.*

In the following, we will assume that δ is an absolute constant bounded away from 1. Note that the probability of success can be amplified by building and querying several instances of the data structure.

Formally, an R-near neighbor of q is a point p such that $||p - q||_2 \leq R$. Note that we can scale down the coordinates of all points by R, in which case we need only to solve the c-NN problem for $R = 1$. Thus, we will consider that $R = 1$ for the rest of the paper.

2.3 Locality-Sensitive Hashing

To solve the c-approximate near neighbor, we use the Locality-sensitive hashing scheme (LSH). LSH scheme was first proposed in [IM98] for c-NN in l_1. Since l_2 is efficiently embeddable into l_1, [IM98]'s algorithm also holds for l_2. Later, however, [DIIM04] found an improved LSH for l_2.

Below we describe the general LSH scheme, which we use for the algorithm presented in this paper. The LSH scheme relies on existence of *locality-sensitive hash functions*. Consider a family $\mathcal{H}$ of hash functions mapping $\Re^d$ to some universe U.

Definition 2.2 (Locality-sensitive hashing). *A family $\mathcal{H}$ is called* (R, cR, p_1, p_2)-sensitive *if for any $p, q \in \Re^d$*

- *if $\|p - q\| \leq R$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \geq p_1$,*
- *if $\|p - q\| \geq cR$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \leq p_2$.*

In order for an LSH family to be useful, it has to satisfy $p_1 > p_2$. Thus, if the point q is close to p, then q and p should likely fall in the same bucket. In contrast, if q is far from p, then q and p should be less likely to fall in the same bucket.

An LSH family can be utilized as follows. Given a family $\mathcal{H}$ of hash functions with parameters (R, cR, p_1, p_2) as in the definition above, we amplify the gap between the "high" probability p_1 and "low" probability p_2 by concatenating several functions. In particular, for k and L specified later, we choose L functions $g_j(q) = (h_{1,j}(q), \ldots, h_{k,j}(q))$, where $h_{t,j} (1 \leq t \leq k, 1 \leq j \leq L)$ are chosen independently and uniformly at random from $\mathcal{H}$. During preprocessing, we store each $p \in P$ (input point set) 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 standard hashing of the hash values $g_j(p)$.

To process a query q, the algorithm searches the buckets $g_1(q), \ldots, g_L(q)$. For each point v found in these buckets, the algorithm computes the distance from q to v and reports the point v iff $||v - q|| \leq cR$. If the buckets $g_1(q), \ldots, g_L(q)$ contain too many points (more than $3L$), the algorithm stops after checking $3L$ points and reports that no R-near neighbor was found. Query time is $O(Lk \cdot \tau)$, assuming that computing one LSH function $h_{t,j}$ takes $O(\tau)$ time.

We choose k and L to be $k = \log_{1/p_2} n$ and $L = n^\rho$, where $\rho = \frac{\log 1/p_1}{\log 1/p_2}$. Then, with constant probability, the algorithm will report a point $v \in B(q, cR)$ if there exists a point $v^* \in B(q, R)$. Furthermore, the query time becomes $O\left(n^\rho k \cdot \tau\right)$ and the preprocessing time becomes $O\left(n^{1+\rho} k \cdot \tau\right)$.

2.3.1 A family of locality-sensitive hash functions for Hamming metric

As seen above, the core of the LSH scheme is an LSH family satisfying the definition 2.2. For illustration purposes, we describe next the LSH family of [IM98], designed for the Hamming metric on $\{0,1\}^d$, yielding query time of $O(n^{1/c}d)$ and preprocessing time of $O(n^{1+1/c}d)$. Note that LSH family is the only part of the algorithm which explicitly depends on the metric.

We define a hash function $h \in \mathcal{H}$ as $h(x) = x_i$, where i is a random index in the range $[1, d]$ and x_i is the i^{th} coordinate of x. Given this family, we can compute the probability p_1 as $Pr[h(p) = h(q)]$ given that $||p - q||_H \leq R$; this yields $p_1 \geq 1 - R/d$. Similarly, we obtain $p_2 \leq 1 - cR/d$.

Now we can compute the query and preprocessing times. First, note that computing a hash function h takes $O(1)$ time, i.e., $\tau = O(1)$. Then, since $n^\rho = n^{\frac{\log p_1}{\log p_2}} = O(n^{1/c})$ ([IM98]), the query time becomes $O(n^\rho k \cdot \tau) = O(n^{1/c}d)$. Similarly, the preprocessing time becomes $O(n^{1+\rho} k \cdot \tau) = O(n^{1+1/c}d)$.

3 Main algorithm

Our new algorithm for c-NN uses a new family of LSH functions for l_2, while reusing the LSH scheme of section 2.3. This new family is presented below. Once we describe the new family of LSH functions, we prove that the query time is $O(n^{1/c^2+o(1)})$ by showing that $L = n^\rho = O(n^{1/c^2+o(1)})$, $k = O(\log n)$, and that $\tau = O(dn^{o(1)})$.

3.1 LSH Family for l_2

We first describe an "ideal" LSH family for l_2. Although this approach has some deficiencies, we show how to overcome them, and obtain a good family of LSH functions. The final description of the LSH family is presented in the figure 3.1.

Ideal LSH family. Construct a hash function as follows. Consider G^d, a regular infinite grid of balls in $\Re^d$: each ball has radius w and has the center at $4w \cdot \mathbb{Z}^d$. Let G_i^d, for i positive integer, be the grid G^d shifted uniformly at random; in other words, $G_i^d = G^d + s_i$, where $s_i \in [0, 4w]^d$. Now we choose as many G_i^d's as are needed to cover the entire space $\Re^d$ (i.e., until each point from $\Re^d$ belongs to at least one of the balls). Suppose we need U such grids to cover the entire space with high probability. In what follows, we formally denote by G_i^d the centers of the balls in the grid G_i^d, i.e., $G_i^d = \{(4w \cdot x_1, 4w \cdot x_2 \ldots 4w \cdot x_d) + s_i \mid x_1, x_2 \ldots x_d \in \mathbb{Z}\}$.

We define h on a point p as a tuple $(u, x_1, x_2, ...x_d)$, $u \in [1, U]$ and $(x_1, ...x_d) \in G_u^d$. The tuple $(u, x_1, x_2, ...x_d)$ specifies the ball which contains the point p: $p \in$

Initialization of a hash function $h \in \mathcal{H}$

1. For $i = 1$ to U, choose a random shift $s_i \in [0, 4w]^t$, which specifies the grid $G_i^t = G^t + s_i$ in the t-dimensional Euclidean space.
2. Choose a matrix $A \in M_{t,d}$, where each element A_{ij} is distributed according to the normal distribution $N(0,1)$ times a scaling factor, $\frac{1}{\sqrt{t}}$. The matrix A represents a random projection from $\Re^d$ to $\Re^t$.

Computing $h()$ **on a point** $p \in \Re^d$

1. Let $p' = Ap$ be the projection of the point p onto a t-dimensional subspace.
2. For each $u = 1, 2, \ldots U$
3. Check whether $p' \in B((x_1, x_2, \ldots x_t), w)$ for a $(x_1, x_2, \ldots x_t) \in G_u^t$.
4. Once we find such a ball, set $h(p) = (u, x_1, x_2, \ldots x_t)$, and stop.
5. Return 0^{t+1} if we do not find any such ball.

Figure 1: Algorithms for initializing a hash function h from the LSH hash family, and for computing $h(p)$ for a point $p \in \Re^d$.

$B((x_1, x_2, \ldots x_n), w)$. If there are several balls that contain p, then we take the one with the smallest value u.

Computing $h(p)$ can be done in $\tau = O(U)$ time as follows. Iterate through all $G_1^d, G_2^d, \ldots G_U^d$, and find the first G_u^d such that p is inside a ball with the center from G_u^d.

Intuitively, this family satisfies our locality-sensitive definition: the closer are the points p, q, the higher is the probability that p, q belong to the same ball. Indeed, if we choose a suitable radius $w \geq 1/2$, then we will get $L = n^\rho = O(n^{1/c^2 + o(1)})$.

However, the deficiency of this algorithm is that U might be very large, namely $\Omega(2^d)$ (see lemma 4.1). We show how to circumvent this deficiency next.

Actual LSH family. Our actual construction follows closely the "ideal" family described above, with the exception of one additional step that we use to reduce U, the number of grids covering the space.

To this end, we use the technique of dimensionality reduction. Namely, we initially project the entire space $\Re^d$ to a lower-dimensional space $\Re^t$. The parameter t is $o(\log n)$, and therefore factors exponential in t are still sub-linear in n. Subsequently, we choose the grids $G_1^t, G_2^t, \ldots G_U^t$ in the lower-dimensional space $\Re^t$. Moreover, to compute $h(p)$, we also project the point p to a lower dimensional space $\Re^t$, and then perform the computation as before. Note that τ becomes $\tau = O(dt) + O(U^t)$ corresponding to the projection and the bucket-computation stages respectively.

Figure 3.1 describes the resulting algorithm for selecting a hash function h, and the algorithm for computing $h(p)$ on any point p.

4 Analysis of the LSH family

We start by bounding the number of grids needed to cover the whole space $\Re^t$.

Lemma 4.1. *Consider a d-dimensional space $\Re^d$. Let G^d be a regular infinite grid of balls of radius w placed at coordinates $4w \cdot \mathbb{Z}^d$. Define G_i^d, for positive integer i, as $G_i^d = G^d + s_i$, where $s_i \in [0, 4w]^d$ is a random shift of the grid G^d. If $U_d = 2^{\Omega(d \log d)} \log n$, then, the grids $G_1^d, G_2^d, \ldots G_{U_d}^d$ cover the entire space $\Re^d$, w.h.p.*

Proof. First, observe that the entire space is covered if and only if the hypercube $[0, 4w]^d$ is covered by G_i^d's (due to the regularity of the grids).

To prove that $[0, 4w]^d$ is covered, we partition the hypercube $[0, 4w]^d$ into smaller hypercubes and prove that each of them is covered with a high enough probability. Specifically, we partition the hypercube $[0, 4w]^d$ into smaller hypercubes, each of size $\frac{w}{\sqrt{d}} \times \frac{w}{\sqrt{d}} \cdots \times \frac{w}{\sqrt{d}}$. There are in total $N = \frac{(4w)^d}{(w/\sqrt{d})^d} = (4\sqrt{d})^d$ such hypercubes. Let x_1 be the probability that a small hypercube is covered by one grid G_i^d. Then $x_1 \geq \frac{(w/\sqrt{d})^d}{(4w)^d} = 1/N$ because, for a small cube to be covered, it is sufficient that the center of the ball $B(0^d + s_i, w)$ falls inside the cube, which happens with probability $\frac{(w/\sqrt{d})^d}{(4w)^d}$. Furthermore, if x_n is the probability that a small hypercube is covered by any of the U_d grids G_i^d, then $x_n \geq 1 - (1 - x_1)^{U_d}$.

Thus, we can compute the probability that there exists at least one uncovered small hypercube, which is also the probability that the entire $[0, 4w]^d$ hypercube is uncovered:

$$Pr[\text{not covered}] \leq N(1 - x_n) \leq N\left(1 - x_1\right)^{U_d} \leq N(1 - 1/N)^{U_d}$$

Now, if we choose $U_d \geq a(N(\log n + \log N)$ for a suitable constant a, then we obtain

$$Pr[\text{not covered}] \leq N(1 - 1/N)^{aN(\log n + \log N)} \leq N 2^{-\log n - \log N} \leq 1/n$$

Concluding, w.h.p., we cover the entire space with the grids $G_1^d, \ldots G_{U_d}^d$, if we choose $U_d = \Omega(N(\log n + \log N)) = 2^{\Omega(d \log d)} \log n$. □

The next lemma states the main technical result of this paper.

Lemma 4.2. *Consider the hash function h described in the figure 3.1, and let p, q be some points in $\Re^d$. Let p_1 be the probability that $h(p) = h(q)$ given that $||p - q|| \leq 1$, and let p_2 be the probability that $h(p) = h(q)$ given that $||p - q|| \geq c$. Then, for $w = \Theta\left(\sqrt[3]{t}\right)$, we obtain $\rho = \frac{\log 1/p_1}{\log 1/p_2} = 1/c^2 + O\left(\frac{\log t}{t^{1/3}}\right)$.*

Proof. Our proof proceeds in three stages. First we show that the dimensionality reduction onto $\Re^t$ does not distort the distance $\|p - q\|$ too much. Second, we estimate the probabilities p_1, p_2, and, finally, we compute $\rho = \frac{\log 1/p_1}{\log 1/p_2}$.

We can estimate the probability that $\|p - q\|$ is not distorted much as follows. We consider the distortion to be "high" when either $\|p' - q'\| \leq (1 - \epsilon)\|p - q\|$ or $\|p' - q'\| \geq (1 + \epsilon)\|p - q\|$, where p', q' are the projections of p, q respectively, and $\epsilon = \Omega\left(\frac{1}{w}\right)$ is the distortion. Thus, if we call $\overrightarrow{\Delta} = p - q$, the probability of high

distortion, called f, can be bounded as (see [DG99])

$$\begin{aligned} f &= \Pr\left[\|p'-q'\| \leq (1-\epsilon)\|p-q\|\right] + \Pr\left[\|p'-q'\| \geq (1+\epsilon)\|p-q\|\right] \\ &= \Pr\left[\|A\overrightarrow{\Delta}\| \leq (1-\epsilon)\|\overrightarrow{\Delta}\|\right] + \Pr\left[\|A\overrightarrow{\Delta}\| \geq (1+\epsilon)\|\overrightarrow{\Delta}\|\right] \\ &\leq \exp\left[-O(t\epsilon^2)\right] + \exp\left[-O(t\epsilon^2)\right] \\ &= \exp\left[-O(t\epsilon^2)\right] \end{aligned} \tag{1}$$

Therefore, with probability at least $1-f$, the distortion is not "high", meaning that if $\|\overrightarrow{\Delta}\| \leq 1$, then $\|A\overrightarrow{\Delta}\| \leq (1+\epsilon)$ and if $\|\overrightarrow{\Delta}\| \geq c$, then $\|A\overrightarrow{\Delta}\| \geq (1-\epsilon)c$.

Next, we estimate the probabilities p_1 and p_2. For two points $p' = Ap, q' = Aq \in \Re^t$, at distance $\Delta' = \|p'-q'\|$, the probability of collision can be deduced as follows. Consider the sequence of grids $G_1^t, G_2^t, \ldots, G_U^t$, and let G_u^t be the first grid that contains the ball $B(x,w)$ such that either $p' \in B(x,w)$ or $q' \in B(x,w)$. Note that the position of this ball defines whether $h(p) = h(q)$ or not. In particular, if $p', q' \in B(x,q)$ then $h(p) = h(q)$ and, otherwise, if exactly one of p', q' is in $B(x,q)$ then $h(p) \neq h(q)$. Thus, we can conclude the probability of collision of points p, q that map to p', q' under dimensionality reduction to be

$$\begin{aligned} \Pr[h(p)=h(q)] &= \Pr[p' \in B(x,w) \wedge q' \in B(x,w) \mid p' \in B(x,w) \vee q' \in B(x,w)] \\ &= \tfrac{|B(p',w) \cap B(q',w)|}{|B(p',w) \cup B(q',w)|} \\ &= \tfrac{2C(\Delta',w)}{2\operatorname{Vol}^t(w) - 2C(\Delta',w)} \\ &= \tfrac{I(\Delta',w)}{1-I(\Delta',w)} \end{aligned} \tag{2}$$

where $C(\Delta', w) = \frac{|B(p',w) \cap B(q',w)|}{2}$ is the volume of the cap of a ball of radius w, with the cap being at distance $\Delta'/2$ from the center of the ball; and $I(\Delta', w) = C(\Delta', w)/\operatorname{Vol}^t(w)$ is the ratio of the volume of the cap to the volume of the entire ball.

From inequality (2), we can thus bound the probabilities p_1 and p_2. For points p, q at distance $\|p-q\| \leq 1$, we have

$$\begin{aligned} p_1 &= \Pr[h(p)=h(q)] \\ &\geq \Pr[h(p)=h(q) \mid \Delta' \leq (1+\epsilon)] \cdot \Pr[\Delta' \leq (1+\epsilon)] \\ &\geq \tfrac{I(1+\epsilon,w)}{1-I(1+\epsilon,w)} \cdot (1-f) \end{aligned} \tag{3}$$

For points p, q at distance $\|p-q\| \geq c$,

$$\begin{aligned} p_2 &= \Pr[h(p)=h(q)] \\ &\leq \Pr[h(p)=h(q) \mid \Delta' \geq (1-\epsilon)c] + \Pr[\Delta' \leq (1-\epsilon)c] \\ &\leq \tfrac{I((1-\epsilon)c,w)}{1-I((1-\epsilon)c,w)} + f \end{aligned} \tag{4}$$

Further on, computing $\rho = \frac{\log 1/p_1}{\log 1/p_2}$ is only a matter of estimating $I(\Delta', w)$ (the ratio of the volume of the cap to the volume of the entire ball), and putting it together with the inequalities (2), (3), and (4). This is precisely what we do next.

The ratio $I(\Delta', w)$, the ratio of the volume of the cap to the volume of the entire ball, has the following form

$$\begin{aligned} I(\Delta', w) &= C(\Delta', w) \cdot (\mathrm{Vol}^t(w))^{-1} \\ &= \frac{2\pi^{\frac{t-1}{2}}}{\Gamma(\frac{t-1}{2})(t-1)} \int_{\Delta'/2}^{w} (w^2 - y^2)^{\frac{t-1}{2}} dy \cdot \left(\frac{2\pi^{t/2}}{\Gamma(t/2)t} w^t\right)^{-1} \\ &= \frac{t}{\sqrt{\pi}(t-1)} \frac{\Gamma(t/2)}{\Gamma(\frac{t-1}{2})} \int_{\frac{\Delta'}{2w}}^{1} (1 - y^2)^{\frac{t-1}{2}} dy \qquad (5) \\ &= \Theta(\sqrt{t}) \cdot \int_{\frac{\Delta'}{2w}}^{1} (1 - y^2)^{\frac{t-1}{2}} dy \end{aligned}$$

where the last step follows from the fact $\frac{\Gamma(t/2)}{\Gamma(\frac{t-1}{2})} = \Theta(\sqrt{t})$.

An upper bound for this quantity is

$$\begin{aligned} I(\Delta', w) &\leq \Theta(\sqrt{t}) \int_{\frac{\Delta'}{2w}}^{1} e^{-\frac{t-1}{2}y^2} dy \\ &\leq \Theta(\sqrt{t}) \sqrt{\frac{2}{t-1}} \int_{\frac{\Delta'\sqrt{(t-1)/2}}{2w}}^{\infty} e^{-y^2} dy \qquad (6) \\ &\leq \Theta\left(\frac{2w}{\Delta'\sqrt{(t-1)/2}}\right) \exp\left[-\frac{t-1}{2}\frac{\Delta'^2}{4w^2}\right] \\ &\leq A_u \cdot o(1) \cdot \exp\left[-\frac{t-1}{2}\frac{\Delta'^2}{4w^2}\right] \end{aligned}$$

where A_u is some constant. Note that in the third step we used the bound for the tail of Gaussian distribution as in [Fel91], Chapter VII.1, Lemma 2.

Similarly, a lower bound is

$$I(\Delta', w) \geq A_l\sqrt{t} \int_{\frac{\Delta'}{2w}}^{\frac{\Delta'}{2w}+\beta} (1 - y^2)^{\frac{t-1}{2}} dy \geq \left[A_l\sqrt{t}\left(1 - \left(\frac{\Delta'}{2w} + \beta\right)^2\right)^{\frac{t-1}{2}} \cdot \beta\right] \qquad (7)$$

where A_l is a constant, and $\beta \in \left[0, 1 - \frac{\Delta'}{2w}\right]$ is chosen below.

We can bound p_1 further as follows. For $\epsilon = \Omega\left(\sqrt{\frac{1}{t}}\right)$, eqn. (1) implies $f \leq 1/2$. Then, for $\beta = \frac{2}{A_l t}$,

$$\begin{aligned} p_1 &\geq \frac{I(1+\epsilon, w)}{1 - I(1+\epsilon, w)}(1 - f) \\ &\geq I(1 + \epsilon, w) \cdot \tfrac{1}{2} \\ &\geq \frac{A_l\sqrt{t}\beta}{2} \cdot \left(1 - \left(\tfrac{1+\epsilon}{2w} + \beta\right)^2\right)^{\frac{t-1}{2}} \qquad (8) \\ &= \frac{1}{\sqrt{t}}\left(1 - \left(\tfrac{1+\epsilon}{2w} + \tfrac{2}{A_l t}\right)^2\right)^{\frac{t-1}{2}} \end{aligned}$$

Next, to bound p_2, note that

$$I((1 - \epsilon)c, w) \leq A_u \cdot o(1) \cdot \exp\left[-\frac{t-1}{2}\frac{\Delta'^2}{4w^2}\right] \leq 1/2$$

Furthermore, if we set $\epsilon = \Omega\left(\frac{1}{w}\right)$, we obtain that

$$f \leq \exp\left[-O(t\epsilon^2)\right] \leq \exp\left[-O(t/w^2)\right] \leq \left(1 - O(1/w^2)\right)^{\frac{t-1}{2}} \leq \frac{A_l\sqrt{t}}{t}(1-O(1/w^2+1/t))^{\frac{t-1}{2}} \leq I((1-\epsilon)c, w)$$

(the last inequality uses the lower bound (7)).

Using the above two bounds, we obtain

$$\begin{aligned} p_2 &\leq \frac{I((1-\epsilon)c,w)}{1-I((1-\epsilon)c,w)} + f \\ &\leq 3I((1-\epsilon)c, w) \\ &\leq A'_u \cdot o(1) \cdot \exp\left[-\frac{t-1}{2}\frac{((1-\epsilon)c)^2}{4w^2}\right] \end{aligned} \tag{9}$$

Finally, we can bound ρ using the formulas (8, 9):

$$\begin{aligned} \rho &= \frac{-\log p_1}{-\log p_2} \\ &\leq \frac{-\log\frac{1}{\sqrt{t}}\left(1-\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^2\right)^{\frac{t-1}{2}}}{-\log\left[A'_u\cdot o(1)\cdot\exp\left[-\frac{t-1}{2}\left(\frac{(1-\epsilon)c}{2w}\right)^2\right]\right]} \\ &\leq \frac{-\log\left(1-\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^2\right)+O\left(\frac{\log t}{t}\right)}{\left(\frac{(1-\epsilon)c}{2w}\right)^2} \\ &= \frac{\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^2+\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^4/2+\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^6/3+\ldots+O\left(\frac{\log t}{t}\right)}{\left(\frac{(1-\epsilon)c}{2w}\right)^2} \\ &= \frac{\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^2}{\left(\frac{(1-\epsilon)c}{2w}\right)^2}\left(1+\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^2/2+\left(\frac{1+\epsilon}{2w}+\frac{2}{A_l t}\right)^4/3+\ldots+O\left(\frac{w^2\log t}{t}\right)\right) \\ &\leq \frac{1}{c^2}\left(\frac{1+\epsilon}{1-\epsilon}+\frac{O(w^2)}{t}\right)\left(1+O\left(\frac{1}{w^2}\right)+O\left(\frac{1}{w^4}\right)+\ldots+O\left(\frac{w^2\log t}{t}\right)\right) \\ &\leq \frac{1}{c^2}\left(1+O(\epsilon)+O\left(\frac{1}{w^2}\right)+O\left(\frac{w^2\log t}{t}\right)\right) \end{aligned} \tag{10}$$

Replacing $\epsilon = \Omega(1/w)$ and $w = O\left(\sqrt[3]{t}\right)$ in the inequality from above, we obtain

$$\begin{aligned} \rho &\leq \frac{1}{c^2}\left[1+O\left(\frac{1}{w}+\frac{1}{w^2}+\frac{w^2\log t}{t}\right)\right] \\ &\leq \frac{1}{c^2}+O\left(\frac{\log t}{t^{1/3}}\right) \end{aligned} \tag{11}$$

□

Theorem 4.3. *There exists an algorithm solving c-NN problem in l_2^d that achieves $O(dn^{1/c^2+o(1)})$ query time and $O(dn^{1+1/c^2+o(1)})$ space and preprocessing.*

Proof. The result follows from lemmas 4.1 and 4.2 for $t = \log^{3/4} n$. By lemma 4.2, we have $\rho = 1/c^2 + O\left(\frac{\log\log n}{\log^{1/4} n}\right)$. Furthermore, k can be bound as (using eqn. (9))

$$k = \frac{\log n}{\log 1/p_2} \leq \frac{\log n}{-\log A'_u \cdot o(1) \cdot \exp\left[-\frac{t-1}{2}\frac{(c(1-\epsilon))^2}{4w^2}\right]} = O\left(\frac{\log n}{t/w^2 - \log A'_u \cdot o(1)}\right) = O(\log n)$$

Finally, $\tau = O(dt)+O(U^t) = O(dt+2^{t\log t}\log n) = O(dt)+2^{O(\log^{3/4} n \log\log n)}\log n = O(dn^{o(1)})$. The theorem follows. □

References

[ADI+05] A. Andoni, M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Nearest Neighbor Methods for Learning and Vision, Neural Processing Information Series, MIT Press*, 2005.

[AI04] A. Andoni and P. Indyk. E2lsh: Exact euclidean locality-sensitive hashing. *Implementation available at* `http://web.mit.edu/andoni/www/LSH/index.html`, 2004.

[AMN+94] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[CCG+98] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. *Annual Symposium on Foundations of Computer Science*, 1998.

[DG99] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstrauss lemma. *ICSI technical report TR-99-006, Berkeley, CA*, 1999.

[DIIM04] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Proceedings of the ACM Symposium on Computational Geometry*, 2004.

[DIS03] T. Darrell, P. Indyk, and G. Shakhnarovich. Nearest neighbor methods for learning and vision. *Neural Processing Information Series, MIT Press, to appear. See also NIPS Workshop at* `http://www.ai.mit.edu/projects/vip/nips03ann`, 2003.

[Fel91] W. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, NY, 1991.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, 1999.

[HP01] S. Har-Peled. A replacement for voronoi diagrams of near linear size. *Annual Symposium on Foundations of Computer Science*, 2001.

[HPM04] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-medians and their applications. *Annual ACM Symposium on Theory of Computing*, 2004.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing*, 1998.

[Ind03] P. Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[KL04] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[Kle97] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *Proceedings of the Thirtieth ACM Symposium on Theory of Computing*, pages 614–623, 1998.

[Pan06] R. Panigrahy. Entropy-based nearest neighbor algorithm in high dimensions. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2006.

[Pis89] G. Pisier. *The volume of convex bodies and Banach space geometry.* Cambridge University Press, 1989.

[WSB98] Roger Weber, Hans J. Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proceedings of the 24th Int. Conf. Very Large Data Bases (VLDB)*, 1998.