# Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces*

Eyal Kushilevitz† Rafail Ostrovsky‡ Yuval Rabani§

**Abstract**

We address the problem of designing data structures that allow efficient search for approximate nearest neighbors. More specifically, given a database consisting of a set of vectors in some high dimensional Euclidean space, we want to construct a space-efficient data structure that would allow us to search, given a query vector, for the closest or nearly closest vector in the database. We also address this problem when distances are measured by the $L_1$ norm, and in the Hamming cube. Significantly improving and extending recent results of Kleinberg, we construct data structures whose size is polynomial in the size of the database, and search algorithms that run in time nearly linear or nearly quadratic in the dimension (depending on the case; the extra factors are polylogarithmic in the size of the database).

*A preliminary version of this paper appeared in STOC '98

†Computer Science Department, Technion — IIT, Haifa 32000, Israel. Email: `eyalk@cs.technion.ac.il`

‡Bell Communications Research, MCC-1C365B, 445 South Street, Morristown, NJ 07960-6438, USA. Email: `rafail@bellcore.com`

§Computer Science Department, Technion — IIT, Haifa 32000, Israel. Part of this work was done while visiting Bell Communications Research. Work at the Technion supported by BSF grant 96-00402, by a David and Ruth Moskowitz Academic Lecturship award, and by grants from the S. and N. Grand Research Fund, from the Smoler Research Fund, and from the Fund for the Promotion of Research at the Technion. Email: `rabani@cs.technion.ac.il`

# 1 Introduction

**Motivation.** Searching for a nearest neighbor among a specified database of points is a fundamental computational task that arises in a variety of application areas, including information retrieval [31, 32], data mining [19], pattern recognition [8, 14], machine learning [7], computer vision [4], data compression [17], and statistical data analysis [10]. In many of these applications the database points are represented as vectors in some high-dimensional space. For example, latent semantic indexing is a recently proposed method for textual information retrieval [9]. The semantic contents of documents, as well as the queries, are represented as vectors in $\mathbb{R}^d$, and proximity is measured by some distance function. Despite the use of dimension reduction techniques such as principal component analysis, vector spaces of several hundred dimensions are typical. Multimedia database systems, such as IBM's QBIC [15] or MIT's Photobook [30], represent features of images and queries similarly. In such applications, the mapping of attributes of objects to coordinates of vectors is heuristic, and so is the choice of metric. Therefore, an approximate search is just as good as an exact search, and is often used in practice.

**The problem.** Let $\mathcal{V}$ be some (finite or infinite) vector space of dimension $d$, and let $\|\cdot\|$ be some norm (Minkowsky distance function) for $\mathcal{V}$. Given a database consisting of $n$ vectors in $\mathcal{V}$, a slackness parameter $\epsilon > 0$, and a query vector $q$, a $(1+\epsilon)$-approximate nearest neighbor of $q$ is a database vector $a$ such that for any other database vector $b$, $\|q-a\| \leq (1+\epsilon)\|q-b\|$. We consider the following problem: Given such a database and $\epsilon > 0$, design a data structure $\mathcal{S}$ and a $(1+\epsilon)$-approximate nearest neighbor search algorithm (using $\mathcal{S}$).

We aim at efficient construction of $\mathcal{S}$, as well as quick lookup. Our performance requirements are: $(i)$ The algorithm for constructing $\mathcal{S}$ should run in time polynomial in $n$ and $d$ (and thus the size of $\mathcal{S}$ is polynomial in $n$ and $d$). $(ii)$ The search algorithm should improve significantly over the naïve (brute force) $O(dn)$ time exact algorithm. More precisely, we aim at search algorithms using (low) polynomial in $d$ and $\log n$ arithmetic operations.

**Our results.** We obtain results for the Hamming cube ($\{0,1\}^d$ with the $L_1$ norm), for Euclidean spaces ($\mathbb{R}^d$ with the $L_2$ norm), and for $\ell_1^d$ ($\mathbb{R}^d$ with the $L_1$ norm). Our results for the cube generalize to vector spaces over any finite field, thus we can handle a database of documents (strings) over any finite alphabet. Our results for Euclidean spaces imply similar bounds for distance functions used in latent semantic indexing (these are not metrics, but their square root is Euclidean).

Our data structures are of size $(dn)^{O(1)}$. For the $d$-dimensional Hamming cube, as well as for $\ell_1^d$, our search algorithm runs in time $O(d \operatorname{poly}\log(dn))$ (the logarithmic factors are different in each case). For $d$-dimensional Euclidean spaces, our search algorithm runs in time $O(d^2 \operatorname{poly}\log(dn))$. (The big-Oh notation hides factors polynomial in $1/\epsilon$.)

Our algorithms are probabilistic. They succeed with any desired probability (at the expense of time and space complexity). We have to make precise the claims for success: The algorithm that constructs $\mathcal{S}$ succeeds with high probability, and if successful, $\mathcal{S}$ is good for every possible query. If $\mathcal{S}$ has been constructed successfully, then given any query, the search algorithm succeeds to find an approximate nearest neighbor with high probability, and this probability can be increased as much as we like *without* modifying $\mathcal{S}$ (just by running the search algorithm several times). An alternative, weaker, guarantee is to construct a data structure that is good for most queries. Our algorithms provide the stronger guarantee. (This means that they can work when the queries are generated by an adversary that has access to the random bits used in the construction of the data structure.) Much of the difficulty arises from this requirement.

**Related work.** In computational geometry, there is a vast amount of literature on proximity problems in Euclidean spaces, including nearest neighbor search and the more general problem of point location in an arrangement of hyperplanes. We dare not attempt to survey but the most relevant papers to our work.

There are excellent solutions to nearest neighbor search in low (two or three) dimensions. For more information see, e.g., [29]. In high-dimensional space, the problem was first considered by Dobkin and Lipton [11]. They showed an exponential in $d$ search algorithm using (roughly) a double-exponential in $d$ (summing up time and space) data structure. This was improved and extended in subsequent work of Clarkson [5], Yao and Yao [34], Matoušek [27], Agarwal and Matoušek [1], and others, all requiring query time exponential in $d$. Recently, Meiser [28] obtained a polynomial in $d$ search algorithm using an exponential in $d$ size data structure.

For approximate nearest neighbor search, Arya et al. [3] gave an exponential in $d$ time search algorithm using a linear size data structure. Clarkson [6] gave a search algorithm with improved dependence on $\epsilon$. Recently, Kleinberg [23] gave two algorithms that seem to be the best results for large $d$ prior to this work. The first algorithm searches in time $O(d^2 \log^2 d + d \log^2 d \log n)$, but requires a data structure of size $O(n \log d)^{2d}$. The second algorithm uses a small data structure (nearly linear in $dn$) and takes $O(n + d \log^3 n)$ time (so it beats the brute force search algorithm).

Independently of our work, Indyk and Motwani [20] obtained several results on essentially the same problems as we discuss here. Their main result gives an $O(d \mathrm{poly} \log(dn))$ search algorithm using a data structure of size $O\left(n(1/\epsilon)^{O(d)} \mathrm{poly} \log(dn)\right)$ for Euclidean and other norms. They obtain this result using space partitions induced by spheres, and bucketing. In comparison with our work, they use exponential in $d$ (but not in $1/\epsilon$) storage in contrast with our polynomial in $d$ storage. Their search time is better than ours for the Euclidean case, and similar for the $L_1$ norm. They also point out that dimension reduction techniques, such as those based on random projections, can be used in conjunction with their other results to get polynomial size data structures which are good for most queries.

Also related to our work are constructions of hash functions that map "close" elements to "close" images. In particular, *non-expansive* hash functions guarantee that the distance between images is at most the distance between the original elements. However, such families are known only for one-dimensional points (Linial and Sasson [26]). For $d$-dimensional points, Indyk et al. [21] construct hash functions that increase the distance by some bounded amount ($d$ or $\sqrt{d}$ depending on the metric). These results are not useful for approximate nearest neighbor search as they can increase a very small distance $\delta$ to something which is much larger than $(1+\epsilon)\delta$. The constructions of Dolev et al. [13, 12] map all elements at distance at most $\ell$ to "close" images. This is also not useful for us, as the construction time is exponential in $\ell$.

**Our methods.** Our data structure and search algorithm for the hypercube is based on an inner product test. Similar ideas have been used in a cryptographic context by [18] and as a matter of folklore to design equality tests (see, e.g., [24]). Here, we refine the basic idea to be sensitive to distances. For Euclidean spaces, we reduce the problem essentially to a search in several hypercubes (along with random sampling to speed up the search). The reduction uses projections onto random lines through the origin. Kleinberg's algorithms are also based on a test using random projections.

His test relies on the relative positions of the projected points. In contrast, our test is based on the property that the projection of any vector maintains, in expectation, its (properly scaled) length. This property underlies methods of distance preserving embeddings into low-dimensional spaces, like the Johnson-Lindenstrauss Lemma [22] (see also Linial et al. [25]). The problem with these techniques is that when applied directly, they guarantee correct answers to most queries, but not to all possible queries. In order to overcome this difficulty, we resort to the theory of Vapnik-Chervonenkis (or VC-) dimension [33] to show the existence of a small finite sample of lines that closely imitate the entire distribution for any vector. A clustering argument, and another sampling argument allows us to use this sample to reduce our problem to the cube. Similar ideas work for the $L_1$ norm too (but we need to project onto the axes rather than onto random lines).

**Notation.** We denote by $n$ the number of database points, by $q$ the query point, and by $\epsilon$ the slackness (i.e., in reply to $q$ we must return a database point whose distance from $q$ is within a factor of $1+\epsilon$ of the minimum distance from $q$ to any database point).

**Metric spaces.** We consider the following metric spaces: The $d$-dimensional Hamming cube $Q_d$ is the set $\{0,1\}^d$ of cardinality $2^d$, endowed with the Hamming distance $H$. For $a,b \in Q_d$, $H(a,b) = \sum_i |a_i - b_i|$. For a finite field $F$, let $x, y \in F^T$. The *generalized Hamming distance* $\mathcal{H}(x,y)$ is the number of dimensions where $x$ differs from $y$. The Euclidean space $\ell_2^d$ is $\mathbb{R}^d$ endowed with the standard $L_2$ distance. The space $\ell_1^d$ is $\mathbb{R}^d$ endowed with the $L_1$ distance.

# 2 Approximate Nearest Neighbors in the Hypercube

In this section we present an approximate nearest neighbors algorithm for the $d$-dimensional cube. That is, all database points and query points are in $\{0,1\}^d$ and distances are measured by Hamming distance. The first idea behind our algorithm for the hypercube is to design a separate test for each distance $\ell$. Given a query $q$, such a test either returns a database vector at distance at most $(1+\epsilon)\ell$ from $q$, or informs that there is no database vector at distance $\ell$ or less from $q$. Given such a test, we can perform approximate nearest neighbor search by using binary search over $\ell \in \{1, 2, \dots, d\}$ (and also checking distance 0 — this can be done using any reasonable dictionary data structure).

We begin by defining a test, which we later use in the construction of our data structure. A $\beta$-test $\tau$ is defined as follows: We pick a subset $C$ of coordinates of the cube by choosing each element in $\{1, 2, \dots, d\}$ independently at random with probability $\beta$. For each of the chosen coordinates $i$ we pick independently and uniformly at random $r_i \in \{0,1\}$. For $v \in Q_d$, define the value of $\tau$ at $v$, denoted $\tau(v)$ as follows:

$$\tau(v) = \sum_{i \in C} r_i \cdot v_i \pmod 2.$$

Equivalently, the test can be viewed as picking a vector $\vec{R} \in \{0,1\}^d$ in a way that each entry gets the value 0 with "high" probability (i.e., $1 - \frac{\beta}{2}$) and the value 1 with "low" probability (i.e., $\beta/2$). With this view, the value of the test on $v \in Q_d$ is just its inner product with $\vec{R}$ modulo 2.[1]

[1]It is common to use inner product for "equality tests". However, these tests just distinguish equal $u, v$ from non-equal but they lose all information on the distance between $u$ and $v$. In our test, by appropriately choosing the value $\beta$, we can obtain some distance information.

Let $q$ be a query, and let $a, b$ be two database points with $H(q,a) \leq \ell$ and $H(q,b) > (1+\epsilon)\ell$. We claim that for $\beta = \frac{1}{2\ell}$ the above test distinguishes between $a$ and $b$ with constant probability. More formally, let $\beta = \frac{1}{2\ell}$, and let $\Delta(u,v) = \Pr_{\vec{R}}[\tau(u) \neq \tau(v)]$. Then, we have:

**Lemma 1** *There is an absolute constant $\delta_1 > 0$, such that for any $\epsilon > 0$ there is a constant $\delta_2 > \delta_1$ (depending on $\epsilon$ only), such that $\Delta(q,a) \leq \delta_1$ and $\Delta(q,b) \geq \delta_2$. (In what follows we denote by $\delta$ the constant $\delta_2 - \delta_1$.)*

**Proof :** For any $u, v \in Q_d$ with $H(u,v) = k$ we have $\Delta(u,v) = \frac{1}{2}(1 - (1 - \frac{1}{2\ell})^k)$ (if none of the $k$ coordinates where $u_i \neq v_i$ is chosen to be in $C$ then $\tau(u) = \tau(v)$; if at least one such coordinate, $j$, is in $C$ then, for every way of fixing all other choices, exactly one of the two choices for $r_j$ will give $\tau(u) \neq \tau(v)$). Note that $\Delta(u,v)$ is monotonically increasing with $k$. We set $\delta_1 = \frac{1}{2}(1 - (1 - \frac{1}{2\ell})^\ell)$ and $\delta_2 = \frac{1}{2}(1 - (1 - \frac{1}{2\ell})^{(1+\epsilon)\ell})$. Thus, $\delta_2 - \delta_1 = \frac{1}{2}\left[(1 - \frac{1}{2\ell})^\ell - (1 - \frac{1}{2\ell})^{(1+\epsilon)\ell}\right] = \Theta(1 - e^{-\epsilon/2})$. ■

The above lemma implies that, for $q, a$ and $b$ as above, a single test can get a small (constant) bias towards making the correct decision as to which point is closer to $q$. To amplify this bias we use several such tests as explained below (see Lemma 2).

**The data structure.** Our data structure $\mathcal{S}$ consists of $d$ substructures $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_d$ (one for each possible distance). Fix $\ell \in \{1, 2, \ldots, d\}$. We now describe $\mathcal{S}_\ell$: Let $M$ and $T$ be positive integers which we specify later. $\mathcal{S}_\ell$ consists of $M$ structures $\mathcal{T}_1, \ldots, \mathcal{T}_M$. So fix $i \in \{1, 2, \ldots, M\}$. Structure $\mathcal{T}_i$ consists of a list of $T$ $\frac{1}{2\ell}$-tests $\vec{R}_1, \vec{R}_2, \cdots, \vec{R}_T \in \{0,1\}^d$, and a table of $2^T$ entries (one entry for each possible outcome of the sequence of $T$ tests). Each entry of the table either contains a database point or is empty.

We construct the structure $\mathcal{T}_i$ as follows. We pick independently at random $T$ $\frac{1}{2\ell}$-tests $t_1, \ldots, t_T$ (defined by $\vec{R}_1, \vec{R}_2, \cdots, \vec{R}_T \in \{0,1\}^d$). For $v \in Q_d$, let its *trace* be the vector $t(v) = (t_1(v), \ldots, t_T(v)) \in \{0,1\}^T$. Let $\delta_1$ and $\delta$ be the constants from Lemma 1. An entry corresponding to $z \in \{0,1\}^T$ contains a database point $v$ with $H(t(v), z) \leq (\delta_1 + \frac{1}{3}\delta)T$, if such a point exists (any such point, if more than one exists), and otherwise the entry is empty. This completes the specification of $\mathcal{S}$ (up to the choice of $T$ and $M$). Notice that in a straightforward implementation, the size of $\mathcal{S}$ is $O(d \cdot M \cdot (dT + 2^T \log n))$ (we also need to keep the original set of points, this takes $dn$ space), and it takes $O(d \cdot M \cdot (dTn + 2^T n))$ time to construct $\mathcal{S}$.

**Lemma 2** *Let $q$ be a query, and let $a, b$ be two database points with $H(q,a) \leq \ell$ and $H(q,b) > (1+\epsilon)\ell$. Consider a structure $\mathcal{T}_i$ in $\mathcal{S}_\ell$, and let $\delta_1$, $\delta_2$, and $\delta$ be as in Lemma 1. Then*

- $\Pr[H(t(q), t(a)) > (\delta_1 + \frac{1}{3}\delta)T] \leq e^{-\frac{2}{9}\delta^2 T}$
- $\Pr[H(t(q), t(b)) < (\delta_2 - \frac{1}{3}\delta)T] \leq e^{-\frac{2}{9}\delta^2 T}$

**Proof :** Follows immediately by plugging in the success probabilities from Lemma 1 in the following Chernoff bounds: For a sequence of $m$ i.i.d. 0-1 random variables $X_1, X_2, \ldots, X_m$, $\Pr[\sum X_i > (p+\gamma)m] \leq e^{-2m\gamma^2}$, and $\Pr[\sum X_i < (p-\gamma)m] \leq e^{-2m\gamma^2}$, where $p = \Pr[X_i = 1]$ (see [2, Appendix A]). ■

Our goal is to show that we can answer "correctly" every possible query. This is formalized by the following definition.

**Definition 3** *For $q \in Q_d$, $\ell$, and $\mathcal{T}_i$ in $\mathcal{S}_\ell$, we say that $\mathcal{T}_i$* fails at $q$ *if there exists a database point $a$ with $H(q,a) \le \ell$ (or a database point $b$ with $H(q,b) > (1+\epsilon)\ell$), such that $H(t(q),t(a)) > (\delta_1 + \frac{1}{3}\delta)T$ (or $H(t(q),t(b)) < (\delta_2 - \frac{1}{3}\delta)T$, respectively). We say that $\mathcal{S}$* fails at $q$ *if there exists $\ell$, such that more than $\mu M/\log d$ structures $\mathcal{T}_i$ in $\mathcal{S}_\ell$ fail (where $\mu$ is a constant that affects the search algorithm). We say that $\mathcal{S}$* fails *if there exists $q \in Q_d$ such that $\mathcal{S}$ fails at $q$.*

The following theorem bounds the probability that $\mathcal{S}$ fails at any given query $q$.

**Theorem 4** *For every $\gamma > 0$, if we set $M = (d + \log d + \log \gamma^{-1}) \log d/\mu$ and $T = \frac{9}{2}\delta^{-2} \ln(2en \log d/\mu)$; then, for any query $q$, the probability that $\mathcal{S}$ fails at $q$ is at most $\gamma 2^{-d}$.*

**Proof :** For any $\ell$, $\mathcal{T}_i$ in $\mathcal{S}_\ell$, and database point $a$, the probability that $H(t(q),t(a))$ is not within the desired range (i.e., $\le (\delta_1 + \frac{1}{3}\delta)T$ if $H(q,a) \le \ell$ or $\ge (\delta_2 - \frac{1}{3}\delta)T$ if $H(q,a) > (1+\epsilon)\ell$ or anything otherwise) is at most $e^{-\frac{2}{9}\delta^2 T} = \frac{\mu}{2en \log d}$, by Lemma 2. Summing over the $n$ database points, the probability that $\mathcal{T}_i$ fails is at most $\frac{\mu}{2e \log d}$. Therefore, the expected number of $\mathcal{T}_i$-s that fail is at most $\frac{\mu M}{2e \log d}$. By standard Chernoff bounds (see [2, Appendix A]), the probability that more than $\frac{\mu M}{\log d}$ of the $\mathcal{T}_i$-s fail is at most $2^{-\mu M/\log d} = \gamma/d2^d$. Summing over all $d$ possible values of $\ell$ completes the proof. ■

We conclude

**Corollary 5** *The probability that $\mathcal{S}$ fails is at most $\gamma$.*

**Proof :** Sum the bound from the above theorem over $2^d$ possible queries $q$. ■

Notice that using the values from Theorem 4 for $M$ and $T$, and assuming that $\gamma$ and $\mu$ are absolute constants, the size of $\mathcal{S}$ is $O(\epsilon^{-2} d^3 \log d (\log n + \log\log d) + d^2 \log d (n \log d)^{O(\epsilon^{-2})})$, and the construction time is essentially this quantity times $n$.

## 2.1 The Search Algorithm

Our search algorithm assumes that the construction of $\mathcal{S}$ is successful. By Corollary 5, this happens with probability $1 - \gamma$. Given a query $q$, we do a binary search to determine (approximately) the minimum distance $\ell$ to a database point. A step in the binary search consists of picking one of the structures $\mathcal{T}_i$ in $\mathcal{S}_\ell$ uniformly at random, computing the trace $t(q)$ of the list of tests in $\mathcal{T}_i$, and checking the table entry labeled $t(q)$. The binary search step succeeds if this entry contains a database point, and otherwise it fails. If the step fails, we restrict the search to larger $\ell$s, otherwise we restrict the search to smaller $\ell$s. The search algorithm returns the database point contained in the last non-empty entry visited during the binary search.

**Lemma 6** *For any query $q$, the probability that the binary search uses a structure $\mathcal{T}_i$ that fails at $q$ is at most $\mu$.*

**Proof :** The binary search consists of $\log d$ steps, each examining a different value $\ell$. As we are assuming that $\mathcal{S}$ did not fail, the probability that for any given $\ell$, the random $\mathcal{T}_i$ in $\mathcal{S}_\ell$ that we pick fails is at most $\mu/\log d$. Summing over the $\log d$ steps completes the proof. ■

**Lemma 7** *If all the structures used by the binary search do not fail at $q$, then the distance from $q$ to the database point $a$ returned by the search algorithm is within a $1+\epsilon$ factor of the minimum distance from $q$ to any database point.*

**Proof :** Denote the minimum distance from $q$ to any database point by $\ell_{\min}$. If $\ell < \ell_{\min}/(1+\epsilon)$ then no database point is within distance $(1+\epsilon)\ell$ of $q$, and therefore, all the binary search steps that visit $\ell$ in this range fail. On the other hand, all the binary search steps that visit $\ell$ in the range $\ell \geq \ell_{\min}$ succeed. Therefore, the binary search ends with $\ell$ such that $\ell_{\min}/(1+\epsilon) \leq \ell \leq \ell_{\min}$. At that point, the database point $a$ returned has $H(t(q), t(a)) \leq (\delta_1 + \frac{1}{3}\delta)T$. Any database point $b$ with $H(q,b) > (1+\epsilon)\ell$ has $H(t(q), t(b)) > (\delta_2 - \frac{1}{3}\delta)T > (\delta_1 + \frac{1}{3}\delta)T$. Therefore, $H(q,a) \leq (1+\epsilon)\ell \leq (1+\epsilon)\ell_{\min}$. ■

Lemma 6 and 7 imply the main result of this section:

**Theorem 8** *If $\mathcal{S}$ does not fail, then for every query $q$ the search algorithm finds a $(1+\epsilon)$-approximate nearest neighbor with probability at least $1-\mu$ using $O(\epsilon^{-2}d(\log n + \log\log d + \log\frac{1}{\mu})\log d)$ arithmetic operations.*

**Proof :** The success probability claim is immediate from the above lemmas. The number of operations follows from the fact that we perform $\log d$ binary search steps. Each step requires computing the value of $T$ $\beta$-tests. Each $\beta$-test requires computing the sum of at most $d$ products of two elements. ■

**Remark:** Some improvements of the above implementation are possible. For example, note that the value of $M$ was chosen so as to guarantee that with constant probability no mistake is made throughout the binary search. Using results of [16], a binary search can still be made in $O(\log d)$ steps even if there is a constant probability of error at each step. This allows choosing $M$ which is smaller by a $O(\log d)$ factor and get the corresponding improvement in the size of $\mathcal{S}$ and the time required to construct it.

# 3 Approximate Nearest Neighbors in Euclidean Spaces

In this section we present our algorithm for Euclidean spaces. The main idea underlying the solution for Euclidean spaces is to reduce the problem to the problem on the cube solved in the previous section. In fact, we produce several cubes, and the search involves several cube searches.

**Notation.** Let $x \in \mathbb{R}^d$ and let $\ell > 0$. Denote by $\mathcal{B}(x,\ell)$ the closed ball around $x$ with radius $\ell$; i.e., the set $\{y \in \mathbb{R}^d \mid \|x-y\|_2 \leq \ell\}$. Denote by $\mathcal{D}(x,\ell)$ the set of database points contained in $\mathcal{B}(x,\ell)$.

Fix $x$ and $\ell$. Let $D, S$ and $L$ be parameters that we fix later. We map the points in $\mathcal{D}(x,\ell)$ into the $(D \cdot S)$-dimensional cube as follows: We pick a set of $D$ i.i.d. unit vectors $\{z_1, \ldots, z_D\}$ from the uniform (Haar) measure on the unit sphere, and project the points in $\mathcal{D}(x,\ell)$ onto each of these vectors. I.e., for every $a \in \mathcal{D}(x,\ell)$ and for every $z \in \{z_1, \ldots, z_D\}$ we compute the dot product $a \cdot z$. For every $z \in \{z_1, \ldots, z_D\}$ we place $S$ equally-spaced points in the interval $[x \cdot z - L, x \cdot z + L]$. We call these points *cutting points*. Each vector $z$ and cutting point $c$ determine a single coordinate of the cube. For any $a \in \mathcal{D}(x,\ell)$, the value of this coordinate is 0 if $a \cdot z \leq c$, and it is 1 otherwise.[2] Altogether, we get a mapping of $x$ into a point in the cube $\{0,1\}^{D \cdot S}$.

[2]Note that if the cutting points are $c_1 \leq c_2 \leq \ldots c_S$ then the $S$ coordinates obtained for a point $a$ by comparing $a \cdot z$ to the cutting points are always of the form: $j$ 0's followed by $S-j$ 1's. In other words, only $S+1$ out of the $2^S$ combinations of 0's and 1's are possible. This observation can be used to get certain improvements in the efficiency of our algorithm. See section 4 for details.

The following lemma analyzes the distribution of length when projecting a fixed (unit) vector on a random (unit) vector.

**Lemma 9** *Let $X$ be the length of the projection of a unit vector onto a random unit vector drawn from the uniform measure on the unit sphere. Then, for every $\delta > 0$ there exist $\alpha = \Theta(\delta)$ and $\alpha' = \Theta(\delta^{3/2})$ such that the following hold:*

1. $$\alpha_0 \triangleq \Pr\left[X < \sqrt{\frac{\delta}{d}}\right] < \alpha;$$

2. $$\alpha_\infty \triangleq \Pr\left[X > \sqrt{\frac{\log(1/\delta)}{d}}\right] < \alpha;$$

3. *For every $j \geq 1$ such that $(1+\delta)^j\sqrt{\delta/d} \leq \sqrt{\log(\delta)^{-1}/d}$,*
$$\alpha_j \triangleq \Pr\left[(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}} \leq X \leq (1+\delta)^j\sqrt{\frac{\delta}{d}}\right] \geq \alpha'.$$

**Proof :** Let $\mathcal{S}_d(r)$ denote the sphere of radius $r$ in $\mathbb{R}^d$ centered at the origin. Its surface area, $\mathcal{A}_d(r)$, is given by $\mathcal{A}_d(r) = 2\pi^{d/2}r^{d-1}/\Gamma(d/2) = \mathcal{A}_d(1)r^{d-1}$, where $\Gamma$ is the so-called Gamma function.[3] By "rotating" the space we can view the experiment of projecting a fixed vector on a random vector as if we project a random vector on the axis $x_1 = 1$. Therefore, the probabilities that we need to estimate are just "slices" of the sphere. In particular, consider the set of points $\{x \in \mathcal{S}_d(1) \mid x_1 \in (\tau - \omega, \tau)\}$ (with $\omega, \tau - \omega > 0$). The surface area of this set is lower bounded by $\omega \cdot \mathcal{A}_{d-1}(r)$, where $r = \sqrt{1-\tau^2}$. By symmetry, the same is true for $\{x \in \mathcal{S}_d(1) \mid x_i \in (-\tau, -\tau + \omega)\}$

To compute the probability of the desired event we compare the area of the slice with the area of the whole sphere. Note that $\mathcal{A}_{d-1}(1)/\mathcal{A}_d(1) = \Theta(\sqrt{d})$. Plug in $\tau = \tau(j) = (1+\delta)^j\sqrt{\delta/d}$ and $\omega = \omega(j) = \tau(j) - \tau(j-1) = \delta(1+\delta)^{j-1}\sqrt{\delta/d}$. Put $\xi = \xi(j) = \delta(1+\delta)^{2j}$, thus $r^2 = 1 - \tau^2 = 1 - \xi/d$ and $\omega = \delta\sqrt{\xi/d}/(1+\delta)$. We get that

$$\begin{aligned}
\Pr\left[(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}} \leq X \leq (1+\delta)^j\sqrt{\frac{\delta}{d}}\right] &\geq 2\omega\mathcal{A}_{d-1}(\sqrt{1-\tau^2})/\mathcal{A}_d(1) \\
&= 2\omega\mathcal{A}_{d-1}(1)\cdot(\sqrt{1-\tau^2})^{d-2}/\mathcal{A}_d(1) \\
&= \Theta\left(\frac{\delta}{1+\delta}\sqrt{\xi}\left(1-\frac{\xi}{d}\right)^{\frac{d-2}{2}}\right) \\
&= \Omega(\delta^{3/2}),
\end{aligned}$$

[3] For integer $d$ the Gamma function is given by

$$\Gamma(d/2) \triangleq \begin{cases} (\frac{d-2}{2})! & d \text{ even} \\ \frac{(d-2)(d-4)\cdots 1}{2^{(d-1)/2}}\sqrt{\pi} & d \text{ odd} \end{cases}$$

where the last equality follows from the fact that in the range of $j$ that interest us, $1 \le (1+\delta)^{j-1} < (1+\delta)^j \le \sqrt{\delta^{-1}\log(1/\delta)}$. This shows the third claim. Similar arguments show the first two claims. ■

**Corollary 10** *Using the above notation, there is an absolute constant $b$ such that*

$$\sum_{j=1}^{j_{\max}} \left(\alpha_j (1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\right) \le E[X] \le b\sqrt{\frac{\delta}{d}} + \sum_{j=0}^{j_{\max}} \left(\alpha_j (1+\delta)^{j}\sqrt{\frac{\delta}{d}}\right). \quad ■$$

In what follows, we denote by $b'$ the constant $b' = E[X]\sqrt{d}$.

The next lemma analyzes the lengths distribution with respect to a series of $D$ projections.

**Lemma 11** *Let $\delta, \alpha, \alpha'$ be as in Lemma 9. Let $\varphi, \beta > 0$. Set*

$$D = \frac{c}{\varphi^2}(8(d+1)\log(4(d+1))(\log(8(d+1)) + \log\log(4(d+1)) + \log\varphi^{-1}) + \log\beta^{-1}),$$

*for some absolute constant $c$. Let $z_1, \ldots, z_D$ be i.i.d. unit vectors from the uniform distribution on the unit sphere. Then, with probability at least $1-\beta$ the following holds: For every $x, y \in \mathbb{R}^d$ define*

$$\begin{aligned}
I_0 &= \{i \mid |(x-y)\cdot z_i| < \sqrt{\frac{\delta}{d}}\|x-y\|_2\}.\\
I_\infty &= \{i \mid |(x-y)\cdot z_i| > \sqrt{\frac{\log(1/\delta)}{d}}\|x-y\|_2\}.\\
I_j &= \{i \mid (1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x-y\|_2 \le |(x-y)\cdot z_i| \le (1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2\},
\end{aligned}$$

*where $j = 1, 2, \ldots, j_{\max}$, with $j_{\max}$ the largest possible such that $I_{j_{\max}} \cap I_\infty = \emptyset$.*[4] *Then*

1. *$|I_0|, |I_\infty| < (\alpha+\varphi)D$; and*
2. *for $j = 1, 2, \ldots, j_{\max}$, $\quad (\alpha_j - \varphi)D \le |I_j| \le (\alpha_j + \varphi)D$.*

**Proof :** Consider the following range space over the set of vectors in the unit sphere. Every pair of points $x, y \in \mathbb{R}^d$ defines several ranges: A range of vectors $z$ such that $|(x-y)\cdot z| < \sqrt{\delta/d}\|x-y\|_2$, a range such that $|(x-y)\cdot z_i| > \sqrt{\log(1/\delta)/d}\|x-y\|_2$, and ranges such that $(1+\delta)^{j-1}\sqrt{\delta/d}\|x-y\|_2 \le |(x-y)\cdot z_i| \le (1+\delta)^j\sqrt{\delta/d}\|x-y\|_2$, for $j = 1, 2, \ldots, j_{\max}$. Each of these ranges is a boolean combination of at most four (closed or open) half-spaces. Therefore, the VC-dimension of this range space is at most $8(d+1)\log(4(d+1))$ (see [2]). The lemma follows from the fact that a random subset of the unit sphere of size $D$ is a $\varphi$-sample with probability at least $1-\beta$. ■

**Lemma 12** *Let $L, \psi > 0$. Let $\sigma = [-L, L]$ be a segment of the real line. Set $S = \lceil \frac{1}{\psi} \rceil$. Let $-L = p_1 < p_2 < \ldots < p_S = L$ be equally-spaced points in $\sigma$. (I.e., $p_j = -L + 2L(j-1)/(S-1)$.) Then, every segment $[\sigma_1, \sigma_2) \subset \sigma$ contains at least $(-\psi + (\sigma_2 - \sigma_1)/2L)S$ and at most $(\psi + (\sigma_2 - \sigma_1)/2L)S$ points from the sample.*

---

[4] For simplicity we assume that these sets form a partition of the space; otherwise, there are minor changes in the constants.

**Proof :** The number of points in $[\sigma_1, \sigma_2)$ is approximately proportional to the measure of this segment (under the uniform measure on $\sigma$). It might be at worst one point below or one point above the exact proportion. $\blacksquare$

Let $w \in \mathbb{R}^d$, and let $\ell > 0$. Fix $L, \varphi, \beta, \psi$. (Thus, $D$ and $S$ are fixed.) Consider the following (random) embedding $\eta : \mathbb{R}^d \hookrightarrow Q_{DS}$: Let $z_1, \ldots, z_D$ be the random vectors in Lemma 11, and Let $p_1, \ldots, p_S$ be the points in Lemma 12. For $x \in \mathbb{R}^d$, $\eta(x) = \eta(x)_{11}\eta(x)_{12} \ldots \eta(x)_{ij} \ldots \eta(x)_{DS}$, where $\eta(x)_{ij} = 0$ if $(x - w) \cdot z_j \leq p_i$, and $\eta(x)_{ij} = 1$ otherwise. The following lemma summarizes some useful properties of this embedding.

**Lemma 13** *Let $\lambda_1 > 0$ be a sufficiently small constant, and let $\lambda_2 = \frac{1}{\delta}+1$. Set $L = \lambda_2 \ell \sqrt{\log(1/\delta)/d}$. Set $\varphi = \delta\alpha'$ and $\psi = \delta^2\lambda_1/2\lambda_2$.*

*Then, for $\eta$ the following holds with probability at least $1 - \beta$: For every $x \in B(w, \ell) \subseteq \mathbb{R}^d$ and $y \in \mathbb{R}^d$*

$$((1 - O(\delta))m - O(\delta))DS \leq H(\eta(x), \eta(y)) \leq ((1 + O(\delta))m' + O(\delta))DS,$$

*where*

$$m = \kappa \cdot \min\left\{\frac{\|x - y\|_2}{\ell}, \lambda_2 - 1\right\},$$

$$m' = \kappa \cdot \max\left\{\frac{\|x - y\|_2}{\ell}, \lambda_1\right\},$$

*and $\kappa = b'/2\lambda_2\sqrt{\log(1/\delta)}$.*

**Proof :** We prove the lemma for the case $\lambda_1 \ell \leq \|x - y\|_2 \leq (\lambda_2 - 1)\ell$. The proofs of the two extreme cases are similar. To analyze the distance in the cube, $H(\eta(x), \eta(y))$, we notice that this distance is influenced by the distribution of the projection lengths $|(x - y) \cdot z_1|, \ldots, |(x - y) \cdot z_D|$ among the sets $I_j$ (Lemma 11 guarantees that this distribution is "nice"); the error in estimating $|(x - y) \cdot z_i|$ for each set $I_j$, and, for each $z_i$, the error caused by discretizing the projection length with the $S$ cutting points (i.e., the value $\psi$ of Lemma 12). In what follows we assume that everything went well. That is, we avoided the probability $\beta$ that Lemma 11 fails.

First we prove the lower bound. Consider $I_j$ for $1 \leq j \leq j_{\max}$. By Lemma 11, at least $(\alpha_j - \varphi)D$ of the projections $|(x - y) \cdot z_i|$ are in the set $I_j$. For each such $z_i$, by the definition of $I_j$, we have that $|(x - y) \cdot z_i| \geq (1 + \delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x - y\|_2$. Every point $p_k$ $(1 \leq k \leq S)$ such that $p_k$ is between $(x - w) \cdot z_i$ and $(y - w) \cdot z_i$ contributes 1 to the Hamming distance. Lemma 12 shows that the number of such points is at least $(-\psi + ((1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x - y\|_2)/2L)S$, provided that both $(x - w) \cdot z_i$ and $(y - w) \cdot z_i$ are contained in the segment $[-L, L]$. As $x \in B(w, \ell)$ and by the triangle inequality $y \in B(w, (\lambda_2 + 1)\ell)$, the corresponding projections are not contained in the segment $[-L, L]$ only if they fall in the set $I_\infty$. For each vector this happens with probability at most $\alpha$, by Lemma 9. Thus the probability that both vectors fall in this segment is at least $1 - 2\alpha$.

For the lower bound we can ignore the bad events: the $i$-s for which $|(x - y) \cdot z_i|$ falls in $I_0$ and $I_\infty$, as well as the $i$-s for which $(x - w) \cdot z_i$ or $(y - w) \cdot z_i$ fall outside $[-L, L]$. These contribute non-negative terms to the distance. We get that $H(\eta(x), \eta(y))$ is at least

$$\sum_{j=1}^{j_{\max}} \left(-\psi + \frac{(1 + \delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x - y\|_2}{2L}\right) \cdot (\alpha_j - \varphi)DS - 2\alpha DS.$$

As $\varphi = \delta\alpha'$ and $\alpha_j > \alpha'$ we get that $\varphi < \delta\alpha_j$ and so $(\alpha_j - \varphi) > (1-\delta)\alpha_j$. Also, note that $\psi$ is at most $\delta$ times the other term: This term is minimized at $j = 1$ and $\|x - y\|_2 = \lambda_1\ell$, and in this case

$$\frac{(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x-y\|_2}{2L} = \frac{\lambda_1\ell\sqrt{\frac{\delta}{d}}}{2\lambda_2\ell\sqrt{\log(1/\delta)/d}} \geq \delta\lambda_1/2\lambda_2.$$

By Corollary 10,

$$\begin{aligned}\sum_{j=1}^{j_{\max}}\left(\alpha_j(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\right) &= \frac{1}{1+\delta}\sum_{j=1}^{j_{\max}}\left(\alpha_j(1+\delta)^{j}\sqrt{\frac{\delta}{d}}\right) \\ &\geq \frac{1}{1+\delta}\left(E[X] - (1+o(1))b\sqrt{\frac{\delta}{d}}\right) \\ &= (1-O(\delta))b'\sqrt{\frac{1}{d}}.\end{aligned}$$

Combining everything we get that the lower bound is at least

$$\left(\frac{(1-O(\delta))b'\sqrt{1/d}\|x-y\|_2}{2L} - 2\alpha\right)DS = \left((1-O(\delta))\frac{b'}{2\lambda_2\sqrt{\log(1/\delta)}}\cdot\frac{\|x-y\|_2}{\ell} - O(\delta)\right)DS.$$

Now we show the upper bound. By Lemma 12, at most $(\alpha_j+\varphi)D$ of the projections $|(x-y)\cdot z_i|$ are in the set $I_j$ for $1 \leq j \leq j_{\max}$, and at most $(\alpha+\varphi)D$ for $I_0$ and $I_\infty$. If $|(x-y)\cdot z_i|$ is in $I_j$, for $0 \leq j \leq j_{\max}$, then $|(x-y)\cdot z_i| \leq (1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2$. By Lemma 12, the contribution of $z_i$ to the Hamming distance is at most $(\psi + ((1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2)/2L)S$, provided (as before) that $(x-w)\cdot z_i$ and $(y-w)\cdot z_i$ are contained in the segment $[-L, L]$. The latter happens with probability at least $1-2\alpha$. With the remaining probability, the contribution of $z_i$ is no more than $S$.

If $z_i$ is in $I_\infty$ we have no bound on the distance between $x\cdot z_i$ and $y\cdot z_i$, but the contribution of $z_i$ to the Hamming distance is no more than $S$. Summing this up, we get an upper bound of at most

$$\sum_{j=0}^{j_{\max}}\left(\psi + \frac{(1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2}{2L}\right)\cdot(\alpha_j+\varphi)DS + 2\alpha DS + (\alpha_\infty+\varphi)DS.$$

As before, the choice of parameters implies that $(\alpha_j+\varphi) \leq (1+\delta)\alpha_j$ and $\psi \leq \delta\cdot\frac{(1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2}{2L}$. Using the lower bound in Corollary 10,

$$\begin{aligned}\sum_{j=0}^{j_{\max}}\left(\alpha_j(1+\delta)^{j}\sqrt{\frac{\delta}{d}}\right) &= \alpha_0\sqrt{\frac{\delta}{d}} + \sum_{j=1}^{j_{\max}}\left(\alpha_j(1+\delta)^{j}\sqrt{\frac{\delta}{d}}\right) \\ &\leq O(\delta)\sqrt{\frac{\delta}{d}} + E[X] \\ &= (1+O(\delta^{3/2}))b'\sqrt{\frac{1}{d}}.\end{aligned}$$

We get that the Hamming distance is at most

$$\left(\frac{(1+O(\delta))\sqrt{1/d}\|x-y\|_2}{2L} + (3+\delta)\alpha\right)DS = \left((1+O(\delta))\frac{b'}{2\lambda_2\sqrt{\log\delta^{-1}}}\cdot\frac{\|x-y\|_2}{\ell} + O(\delta)\right)DS. \qquad \blacksquare$$

**The data structure.** Our data structure $\mathcal{S}$ consists of a substructure $\mathcal{S}_a$ for every point $a$ in the database. Each $\mathcal{S}_a$ consists of a list of all the other database points, sorted by increasing distance from $a$, and a structure $\mathcal{S}_{a,b}$ for each database point $b \neq a$. Fix $a$ and $b$, and let $\ell = \|a-b\|_2$ be the distance from $a$ to $b$. (For simplicity, we'll assume that different $b$'s have different distances from $a$.) The structure $\mathcal{S}_{a,b}$ consists of (1) a list of unit vectors $z_1, z_2, \ldots, z_D$; (2) $D$ lists of points $P_1, P_2, \ldots, P_D$, each consists of $S$ points; (3) a Hamming cube data structure for dimension $D \cdot S$; and (4) the number of database points $|\mathcal{D}(a,\ell)|$.

We construct $\mathcal{S}_{a,b}$ as follows. Set $\delta = \epsilon/O(1)$. We pick $D$ i.i.d. unit vectors $z_1, z_2, \ldots, z_D$ from the uniform measure on the unit sphere, for $D = O((1/\delta^5)(d \log d(\log d + \log(1/\delta)) + \log(1/\beta)))$ For each set $P_i$, we pick $S$ equally-spaced points in $[z_i \cdot a - L, z_i \cdot a + L]$, where $S = O(1/\delta^3)$ We map the points in $\mathcal{D}(a,\ell)$ onto the $DS$-dimensional cube as explained above, and construct a data structure for approximate nearest neighbor search in this cube (where the database points are the images of the points in $\mathcal{D}(a,\ell)$, and the slackness parameter is $\delta$). This completes the specification of $\mathcal{S}$ (up to the choice of $\beta, \gamma, \lambda_1$, and some of the implicit constants).

**Definition 14** *We say that $\mathcal{S}_{a,b}$* fails *if the list of unit vectors $z_1, z_2, \ldots, z_D$ is not a $\varphi$-sample for the range space defined in the proof of Lemma 11, or if the construction of the cube data structure fails. We say that $\mathcal{S}$* fails *if there are $a, b$ such that $\mathcal{S}_{a,b}$ fails.*

**Lemma 15** *For every $\zeta > 0$, setting $\beta = \zeta/n^2$ and $\gamma = \zeta/n^2$ (where $\beta$ is the parameter of Lemma 11, and $\gamma$ is the parameter of Corollary 5) the probability that $\mathcal{S}$ fails is at most $\zeta$.*

**Proof :** Sum up the failure probabilities from Lemma 11 and Corollary 5 over all the structures we construct. ■

Our data structure $\mathcal{S}$ requires $O(n^2 \cdot \text{poly}(1/\epsilon) \cdot d^2 \cdot \text{poly}\log(dn/\epsilon) \cdot (n \log(d \log n/\epsilon))^{O(\epsilon^{-2})})$ space (we have $O(n^2)$ structures, and the dominant part of each is the $DS$-dimensional cube structure). The time to construct $\mathcal{S}$ is essentially its size times $n$ (again, the dominant part is constructing the cube structures).

## 3.1 The Search Algorithm

As with the cube, our search algorithm assumes that the construction of $\mathcal{S}$ succeeds. This happens with probability at least $1-\zeta$, according to Lemma 15. Given a query $q$, we search some of the structures $\mathcal{S}_{a,b}$ as follows. We begin with any structure $\mathcal{S}_{a_0,b_0}$, where $a_0$ is a database point, and $b_0$ is the farthest database point from $a_0$. Let $\ell_0 = \|a_0 - b_0\|_2$. Then $\mathcal{D}(a_0, \ell_0)$ contains the entire database. We proceed by searching $\mathcal{S}_{a_1,b_1}$, $\mathcal{S}_{a_2,b_2}$, ..., where $a_{j+1}, b_{j+1}$ are determined by the results of the search in $\mathcal{S}_{a_j,b_j}$.

So fix $j$. We describe the search in $\mathcal{S}_{a_j,b_j}$. Let $\ell_j = \|a_j - b_j\|_2$. Let $z_1, z_2, \ldots, z_D$ be the unit vectors, and let $P_1, P_2, \ldots, P_D$ be the sets of points for $\mathcal{S}_{a_j,b_j}$. We map $q$ to a node $\eta(q)$ of the $DS$-dimensional cube as explained above. We now search for a $(1+\delta)$-approximate nearest neighbor for $\eta(q)$ in the cube data structure of $\mathcal{S}_{a_j,b_j}$ (allowing failure probability $\mu$). Let the output of this search be (the image of) the database point $a'$. If $\|q - a'\|_2 > \frac{1}{10}\ell_{j-1}$, we stop and output $a'$. Otherwise, we pick $T$ database points uniformly at random from $\mathcal{D}(a_j, \ell_j)$, where $T$ is a constant. Let $a''$ be the closest among these points to $q$. Let $a_{j+1}$ be the closest to $q$ between $a_j, a'$ and $a''$, and let $b_{j+1}$ be the farthest from $a_{j+1}$ such that $\|a_{j+1} - b_{j+1}\|_2 \leq 2\|a_{j+1} - q\|_2$. (We find $b_{j+1}$ using binary search on the sorted list of database points in $\mathcal{S}_{a_{j+1}}$.) If no such point exists, we abort the search and we output $a_{j+1}$.

Before going into the formal proof of correctness and complexity, let us try to give the intuition behind the algorithm. Our test gives a good approximation for the distance in case that we have $\ell$ which is "close" to the actual distance (Lemma 13). This $\ell$ can be viewed as the precision with which we measure distances. If the precision is not accurate enough we cannot make the right decision. However, we are able to detect that $\ell$ is too large and to reduce it. This can guarantee only that if we start with $\ell_0$ and the nearest neighbor is at distance $\ell_{min}$ we will have about $O(\log \frac{\ell_0}{\ell_{min}})$ iterations, and this quantity may be enormous compared to $d$ and $\log n$. To overcome this obstacle (i.e., make sure that the number of iterations does not depend on the distances), we add the random sampling from the ball $\mathcal{D}(a_j, \ell_j)$. This allows us to make sure that not only the distances reduce but actually the number of points in the sets also decreases fast and this guarantees that the number of iterations is $O(\log n)$.

The following lemmas formulate the progress made by each step. For the analysis, let $a_{\min}$ be the closest point in the database to $q$ and let $\ell_{\min}$ be its distance.

**Lemma 16** *For every $j \geq 0$, $a_{\min} \in \mathcal{D}(a_j, \ell_j)$.*

**Proof :** $\mathcal{D}(a_j, \ell_j)$ contains all the database points whose distance from $a_j$ is at most $2\|q - a_j\|_2$. In particular (by triangle inequality), it contains all the database points whose distance from $q$ is at most $\|q - a_j\|_2 \geq \ell_{\min}$. Therefore, it contains $a_{\min}$. ■

**Lemma 17** *For every $j \geq 1$, if $\ell_j$ is such that $(\lambda_2 - 1)\ell_j < \ell_{\min}$ then for every $a \in \mathcal{D}(a_j, \ell_j)$ we have $\|q - a\|_2 \leq \ell_{\min}(1 + \delta)$.*

**Proof :** By the assumptions (and since we have $a_{\min} \in \mathcal{D}(a_j, \ell_j)$), the distance from $q$ to $a$ is at most $\ell_{\min} + \ell_j < \ell_{\min}(1 + \frac{1}{\lambda_2 - 1}) = \ell_{\min}(1 + \delta)$ (by the choice of $\lambda_2$; see Lemma 13). ■

**Lemma 18** *For every $j \geq 1$, $\|q - a'\|_2 \leq \max\{(1+\epsilon)\ell_{\min}, \frac{1}{10}\ell_{j-1}\}$ with probability at least $1 - \mu$.*

**Proof :** As $a_{\min} \in \mathcal{D}(a_{j-1}, \ell_{j-1})$, we claim that our search of $\mathcal{S}_{a_{j-1}, b_{j-1}}$ returns $a'$ whose distance from $q$ is at most $(1 + \epsilon)\max\{\ell_{\min}, \lambda_1 \ell_{j-1}\}$ with probability at least $1 - \mu$ (we set $\lambda_1$ such that $(1 + \epsilon)\lambda_1 \leq 1/10$). The reason is that, by Lemma 13,

$$H(\eta(q), \eta(a)) \leq ((1 + O(\delta))\kappa \max\{\ell_{\min}/\ell_{j-1}, \lambda_1\} + O(\delta))DS.$$

Therefore, the search algorithm returns a point $b_c$ such that

$$H(\eta(q), b_c) \leq (1 + \delta)((1 + O(\delta))\kappa \max\{\ell_{\min}/\ell_{j-1}, \lambda_1\} + O(\delta))DS.$$

Using Lemma 13 again, this point $b_c$ is the image of a point $a'$ whose distance from $q$ satisfies

$$\|q - a'\|_2 \leq (1 + O(\delta)) \max\{\ell_{\min}, \lambda_1 \ell_{j-1}\} + O(\delta)\ell_{j-1}.$$

■

**Lemma 19** *For every $j \geq 1$, $\mathcal{B}(q, \|q - a_j\|_2)$ contains at most $\frac{1}{2}|\mathcal{D}(a_{j-1}, \ell_{j-1})|$ points database with probability at least $1 - 2^{-T}$.*

**Proof :** First notice that $\mathcal{B}(q, \|q - a_j\|_2)$ contains database points from $\mathcal{D}(a_{j-1}, \ell_{j-1})$ only. Let $\xi$ be such that $\mathcal{B}(q, \xi)$ contains exactly half the points of $\mathcal{D}(a_{j-1}, \ell_{j-1})$. (For simplicity, we assume that the distances from $q$ to the database points are all distinct.) Each database point in the random sample we pick has probability $\frac{1}{2}$ to be in $\mathcal{B}(q, \xi)$. Therefore, the probability that $a'' \notin \mathcal{B}(q, \xi)$ is at most $2^{-T}$. ∎

**Lemma 20** *For all $j \geq 1$, $\mathcal{D}(a_j, \ell_j) \subset \mathcal{B}(q, \|q - a_{j-1}\|_2)$ with probability at least $1 - \mu$.*

**Proof :** Let $a \in \mathcal{D}(a_j, \ell_j)$. By the triangle inequality, $\|q - a\|_2 \leq \ell_j + \|q - a_j\|_2 \leq 3\|q - a_j\|_2$. By Lemma 18, $3\|q - a_j\|_2 \leq \frac{3}{10}\ell_{j-1}$. Since $\ell_{j-1} \leq 2\|q - a_{j-1}\|_2$, the lemma follows. ∎

**Corollary 21** *For every $j \geq 2$, $|\mathcal{D}(a_j, \ell_j)| \leq \frac{1}{2}|\mathcal{D}(a_{j-2}, \ell_{j-2})|$ with probability at least $1 - (\mu + 2^{-T})$.*

**Theorem 22** *If $\mathcal{S}$ does not fail, then for every query $q$ the search algorithm finds a $(1 + \epsilon)$-approximate nearest neighbor using expected $\mathrm{poly}(1/\epsilon) d^2 \mathrm{poly}\log(dn/\epsilon)$ arithmetic operations.*[5]

**Proof :** Corollary 21 says that within two iterations of the algorithm, with constant probability the number of database points in the current ball, $\mathcal{D}(a_j, \ell_j)$, is reduced by a factor of 2. Hence, within expected $O(\log n)$ iterations the search ends.

If the search ends because $\|q - a'\|_2 > \frac{1}{10}\ell_{j-1}$, then by Lemma 18 it must be that $\|q - a'\|_2 \leq (1 + \epsilon)\ell_{\min}$. Otherwise, the search ends because no database point $b \neq a_j$ satisfies: $\|a_j - b\|_2 \leq 2\|a_j - q\|_2$. In this case, $a_j = a_{\min}$, because $\|a_j - a_{\min}\|_2 \leq \|a_j - q\|_2 + \|a_{\min} - q\|_2 \leq 2\|a_j - q\|_2$. In either case, the search produces a $(1 + \epsilon)$-approximate nearest neighbor.

As for the search time, we have $O(\log n)$ iterations. In each iteration we perform $O(D(S + d)) = O(\mathrm{poly}(1/\epsilon) \cdot d^2 \cdot \mathrm{poly}\log(dn/\epsilon))$ operations to compute $q_c$; then, we execute a search in the $DS$-cube, which, by Theorem 8 takes $O(\mathrm{poly}(1/\epsilon) \cdot d \cdot \mathrm{poly}\log(dn/\epsilon))$ operations. ∎

# 4 Extensions

In what follows we discuss some other metrics for which our methods (with small variations) apply.

**Generalized Hamming metric:** Assume that we have a finite alphabet $\Sigma$ and consider the generalized cube $\Sigma^d$. For $x, y \in \Sigma^d$, the *generalized Hamming distance* $\mathcal{H}(x, y)$ is the number of dimensions where $x$ differs from $y$. The case $\Sigma = \{0, 1\}$ is the Hamming cube discussed in Section 2. Here we argue that the results in that section extend to the case of arbitrary $\Sigma$. For convenience, assume that $\Sigma = \{0, 1, \ldots, p-1\}$ for a prime $p$. (We can always map the elements of $\Sigma$ to such a set, perhaps somewhat larger, without changing the distances.) It suffices to show that a generalization of the basic test used in Section 2 has the same properties. The generalized test works as follows: pick each element in $\{1, 2, \ldots, d\}$ independently at random with probability $1/(2\ell)$. For each chosen coordinate $i$, pick independently and uniformly at random $r_i \in \Sigma$. (For every $i$ which is not chosen put $r_i = 0$.) The test is defined by $\tau(x) = \sum_{i=1}^{d} r_i x_i$, where multiplication and summation are done in $GF[p]$. As before, for any two vectors $x, y \in \Sigma^d$, let $\Delta(x, y) \stackrel{\triangle}{=} \Pr[\tau(x) \neq \tau(y)]$. If $\mathcal{H}(x, y) = k$ then $\Delta(x, y) = \frac{p-1}{p} \cdot (1 - (1 - \frac{1}{2\ell})^k)$. Therefore, the difference $\delta$ in the probabilities between the

[5] Alternatively, we can demand a deterministic bound on the number of operations, if we are willing to tolerate a vanishing probability error in the search.

case of vectors with Hamming distance at most $\ell$ and the case of vectors with Hamming distance at least $(1+\epsilon)\ell)$ is $\frac{p-1}{p} \cdot \left[(1-\frac{1}{2\ell})^{\ell} - (1-\frac{1}{2\ell})^{(1+\epsilon)\ell}\right]$. $\delta$ is minimized at $p=2$, so we do better with a larger alphabet. Notice that the number of possible traces here is $p^T$, so this construction gives a polynomial size data structure if and only if $p$ is a constant. In the next paragraph we mention how to handle non-constant $p$-s.

**$L_1$ norm for finite alphabet:** Consider, again, the case $\Sigma = \{0,1,\ldots,p-1\}$ and define the distance between $x$ and $y$ as $d(x,y) \triangleq \sum_{i=1}^{d} |x_i - y_i|$. The first observation is that this case is reducible to the case of the Hamming cube as follows: Map any $x \in \Sigma^d$ into $\hat{x} \in \{0,1\}^{d(p-1)}$ by replacing every coordinate $x_i \in \{0,1,\ldots,p-1\}$ of $x$ by $p-1$ coordinates of $\hat{x}$: These are $x_i$ ones followed by $(p-1-x_i)$ zeros. Observe that indeed $d(x,y) = H(\hat{x},\hat{y})$. Therefore, we can apply the cube construction and algorithm to $\hat{x}$. If $p$ is not a constant, this solution is not satisfactory, because it blows up not only the data structure size (by a factor polynomial in $p$), but also the search time (by a factor of $p$ at least). Our second observation is that we can restrict the blowup in search time to a factor of $O(\log p)$. This is because we do not really have to map $x$ into $\hat{x}$. The contribution of $x_i \in \{0,1,\ldots,p-1\}$ to $\tau(\hat{x})$ is just the sum modulo 2 of (at most $p-1$) $r_j$-s. As there are only $p$ possible sums (and *not* $2^p$) they can all be pre-computed and stored in a dictionary using $O(p)$ space. (Notice that this needs to be done for each test, for each coordinate.) To compute the value of the test on a query, we need to sum up the contributions of the coordinates. For each coordinate, it takes at most $O(\log p)$ time to retrieve the desired value (because operations on values in $\{0,1,\ldots,p-1\}$ take that much time). The same ideas can be used to handle the generalized Hamming metric for non-constant alphabets: Map each coordinate $x_i$ into $p$ binary coordinates, which are all zeros except for a one in position $x_i+1$. The Hamming distance in the $(dp)$-cube is twice the original distance. For a given test, there are only $p$ different values to consider for each original coordinate, so we can pre-compute them and retrieve them as before.

**$\ell_1^d$:** The construction and algorithm are similar to those for the Euclidean case. The only difference is in the embedding $\eta$ to the cube. Instead of projecting the points onto random unit vectors, we project them onto the original coordinates. Let $w, \delta, \lambda_1, \lambda_2, \ell$ be as in the Euclidean case. For the $i$-th coordinate, we place $S = 2d\lambda_2/\delta\lambda_1$ equally spaced points between $w_i - \lambda_2\ell$ and $w_i + \lambda_2\ell$. These partition the line into $S+1$ (finite or infinite) segments. We number them 0 through $S$ from left to right. Now for every $x \in \mathbb{R}^d$ we define $\eta(x)$ to be the vector with entries in $\{0,1,\ldots,S\}$, such that $\eta(x)_i$ is the number of the segment that contains $x_i$. Using Lemma 12, we show the equivalent of Lemma 13 in this case:

**Lemma 23** *For every $x$ such that $\|x-w\|_1 \le \ell$, for every $y \in \mathbb{R}^d$,*

$$(1-\delta)m'\frac{d}{\delta\lambda_1} \le \|\eta(x)-\eta(y)\|_1 \le (1+\delta)m\frac{d}{\delta\lambda_1},$$

*where $m = \min\left\{\frac{\|x-y\|_1}{\ell}, \lambda_2 - 1\right\}$, and $m' = \max\left\{\frac{\|x-y\|_1}{\ell}, \lambda_1\right\}$.*

**Proof :** We show the case $\lambda_1\ell \le \|x-y\|_1 \le (\lambda_2-1)\ell$. The other two cases are similar. Notice that in this case, for every $i$, $x_i, y_i \in [w_i - \lambda_2\ell, w_i + \lambda_2\ell]$. By Lemma 12, for every $i$,

$$-1 + \|x_i - y_i\|S/2\lambda_2\ell \le \|\eta(x)_i - \eta(y)_i\| \le 1 + \|x_i - y_i\|S/2\lambda_2\ell.$$

Thus, summing over the $d$ coordinates,

$$-d + \|x-y\|_1 S/2\lambda_2 \ell \leq \|\eta(x) - \eta(y)\|_1 \leq d + \|x-y\|_1 S/2\lambda_2 \ell.$$

As $d = \delta\lambda_1 S/2\lambda_2 \leq \delta\|x-y\|_1 S/2\lambda_2 \ell$, the lemma follows. ■

We use the construction for the finite alphabet $L_1$-norm to handle the embedded instance. The remainder of the argument is identical to the Euclidean case. Notice, however, that given a query $q$, computing $\eta(q)$ is more efficient than in the Euclidean case: In each coordinate, we can use binary search to determine the segment in $O(\log S) = O(\log(d/\epsilon))$ operations. Projecting is trivial and takes $O(1)$ operations per coordinate. In the Euclidean case, the search time is dominated by the calculation of $\eta(q)$. Thus, here it goes down to $O(d \mathrm{poly}\log(dn/\epsilon))$.

# 5 Acknowledgements

We thank Shai Ben-David, Nati Linial, and Avi Wigderson for helpful comments on earlier versions of this work.

# References

[1] P.K. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proc. of 24th STOC*, pp. 517–526, 1992.

[2] N. Alon and J. Spencer. *The Probabilistic Method.* Wiley, 1992.

[3] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *Proc. of 5th SODA*, pp. 573-582, 1994.

[4] J.S. Beis and D.G. Lowe. Shape indexing using approximate nearest-neighbor search in high-dimensional spaces. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 1000–1006, 1997.

[5] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830-847, 1988.

[6] K. Clarkson. An algorithm for approximate closest-point queries. In *Proc. of 10th SCG*, pp. 160–164, 1994.

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

[8] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE-IT*, 13:21–27, 1967.

[9] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J. Amer. Soc. Info. Sci.*, 41(6):391–407, 1990.

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

[11] D. Dobkin and R. Lipton. Multidimensional search problems. *SIAM J. Comput.*, 5:181-186, 1976.

[12] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas. Neighborhood preserving hashing and approximate queries. *Proc. of 5th SODA*, pp. 251–259, 1994.

[13] D. Dolev, Y. Harari, and M. Parnas. Finding the neighborhood of a query in a dictionary. *Proc. of 2nd ISTCS*, pp. 33–42, 1993.

[14] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

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

[16] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Computing with unreliable information. In *Proc. of 22nd STOC*, pp. 128–137, 1990.

[17] A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic, 1991.

[18] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st STOC*, pp. 25–32. ACM, 1989.

[19] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. In *1st International Conf. on Knowledge Discovery and Data Mining*, 1995.

[20] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In these proceedings.

[21] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-preserving hashing in multi-dimensional spaces. *Proc. of 29th STOC*, pp. 618–625, 1997.

[22] W.B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[23] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. of 29th STOC*, pp. 599–608, 1997.

[24] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[25] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[26] N. Linial and O. Sasson. Non-expansive hashing. *Proc. of 28th STOC*, pp. 509–518, 1996.

[27] J. Matoušek. Reporting points in halfspaces. In *Proc. of 32nd FOCS*, 1991.

[28] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

[29] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, 1993.

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

[31] G. Salton. *Automatic Text Processing.* Addison-Wesley, 1989.

[32] A.W.M. Smeulders and R. Jain (eds). *Proc. 1st Workshop on Image Databases and Multi-Media Search*, 1996.

[33] V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Prob. Appl.*, 16:264–280, 1971.

[34] A.C. Yao and F.F. Yao. A general approach to $d$-dimension geometric queries. In *Proc. of 17th STOC*, pp. 163–168, 1985.