Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions*

Alexandr Andoni
MIT
andoni@mit.edu

Piotr Indyk
MIT
indyk@mit.edu

Abstract

We present an algorithm for the c-approximate nearest neighbor problem in a d-dimensional Euclidean space, achieving query time of $O(dn^{1/c^2+o(1)})$ and space $O(dn + n^{1+1/c^2+o(1)})$. This almost matches the lower bound for hashing-based algorithm recently obtained in [27]. We also obtain a space-efficient version of the algorithm, which uses $dn + n\log^{O(1)} n$ space, with a query time of $dn^{O(1/c^2)}$. Finally, we discuss practical variants of the algorithms that utilize fast bounded-distance decoders for the Leech Lattice.

1. Introduction

The *nearest neighbor* problem is defined as follows: given a collection of n points, build a data structure which, given any query point, reports the data point that is closest to the query. A particularly interesting and well-studied instance is where the data points live in a d-dimensional Euclidean space. This problem is of major importance in several areas; 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 the 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.

There are several efficient algorithms known for the case when the dimension d is "low" (see [30] for an overview). Therefore the main issue is that of dealing with a large number of dimensions. Despite decades of intensive effort, the current solutions suffer from either space or query time that is *exponential* in d. 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 recent years, several researchers proposed methods for overcoming the running time bottleneck by using *approximation* (e.g., [7, 23, 21, 25, 16, 24, 17, 12, 8, 28, 1], see also [31]). In that formulation, the algorithm is allowed to return a point, whose distance from the query is at most c times the distance from the query to its nearest points; $c = 1+\epsilon > 1$ is called the *approximation factor*. The appeal of this approach is that, in many cases, an 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., [4]).

In [25, 21, 8, 1], the authors constructed data structures for the $(1+\epsilon)$-approximate nearest neighbor problem which avoided the curse of dimensionality. Specifically, for any constant $\epsilon > 0$, the data structures support queries in time $O(d \log n)$, and use space which is polynomial in n. Unfortunately, the exponent in the space bounds is roughly C/ϵ^2 (for $\epsilon < 1$), where C is a "non-negligible" constant. Thus, even for, say, $\epsilon = 1$, the space used by the data structure is large enough so that the algorithm becomes impractical even for relatively small data sets. In fact, in a recent paper [6] we show that, in order to solve the *decision* version of the approximate nearest neighbor problem, the exponent in the space bound must be at least $\Omega(1/\epsilon^2)$, as long as the search algorithm performs only a constant number of "probes" to the data structure. The algorithms of [25, 21] use only *one* probe.

In [21, 15], the authors introduced an alternative approach, which uses much smaller space while preserving

*This work was supported in part by NSF CAREER award CCR-0133849, David and Lucille Packard Fellowship and Alfred P. Sloan Fellowship.

sub-linear query time (see Figure 1 for the running times). It relies on a concept of *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 [21, 15] 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 [12], 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 [5] for high-dimensional similarity search, which has been used in several applied scenarios. Recently, [28] proposed a different method of utilizing locality-sensitive hash functions, which results in near-linear space, at the cost of somewhat higher query time.

The natural question raised by this line of research is: what is the smallest exponent ρ achievable via the locality-sensitive hashing approach ? It was conjectured by the second author (e.g., in [19]) that $\rho \le 1/c^2$. The conjecture was motivated by the fact that an algorithm with such exponent exists for the closely related problem of finding the *furthest neighbor* [20].

Our results. In this paper we essentially resolve the issue by providing an algorithm with query time $dn^{\rho(c)}$ using space $dn + n^{1+\rho(c)}$, where

$$\rho(c) = 1/c^2 + O(\log\log n / \log^{1/3} n).$$

This significantly improves over the earlier running time of [12]. In particular, for $c = 2$, our exponent tends to 0.25, while the exponent in [12] was around 0.45. Moreover, a recent paper [27] shows that hashing-based algorithms (as described in Section 2.3) cannot achieve $\rho < 0.462/c^2$. Thus, the running time exponent of our algorithm is essentially optimal, up to a small constant factor.

Our result immediately implies improved algorithms for several approximate problems in high dimensional spaces. For example, it is known [21, 18] that the c-approximate minimum spanning tree (MST) problem for n points in l_2^d can be computed by using $O(n \log n)$ calls to the c-approximate near neighbor oracle for that space. Thus, our result implies a $dn^{1+1/c^2+o(1)}$-time algorithm for the c-approximate MST problem. Other problems for which similar improvement is obtained include dynamic closest pair and facility location [18].

Unfortunately, the convergence of the exponent to the $1/c^2$ limit is rather slow. To be more precise: the running time of the algorithm is bounded by the formula

$$t^{O(t)} n^{1/c^2 + O(\log t)/t^{1/2}}$$

where t is a parameter chosen to minimize the expression. The $t^{O(t)}$ factor appears due to the fact that our algorithm exploits certain configurations of points in a t-dimensional space; the "quality" of the configurations increases with t. One can observe that the parameter t needs to be somewhat large for the exponent to be competitive against the earlier bounds. But then the factor $t^{O(t)}$ becomes very large, erasing the speedup gained from the improved exponent (unless n is really large).

To overcome this difficulty, we modify the algorithm to make it efficient for more moderate values of n. Specifically, we replace the aforementioned configurations of points by known constructions of "nice" point-sets in specific dimensions. In particular, by utilizing Leech Lattice [26] in 24 dimensions, we obtain an algorithm with exponent $\rho(c)$ such that $\rho(2) \le 0.37$, while the leading term in the running time is reduced to only few hundred. Moreover, if the dimension d does not exceed 24, the exponent is reduced[2] further, and we achieve $\rho(2) \le 0.27$. The leading term in the running time remains the same.

Finally, we show that the LSH functions can be used as in [28] to design a data structure with nearly-linear space of $O(dn + n\log^{O(1)} n)$ and query time $dn^{O(1/c^2)}$. This improves over the earlier bound of $dn^{O(1/c)}$ due to [28].

Paper	Metric	Space	Query time	Comments
[21, 15]	Hamming	$n^{1+1/c}$	$dn^{1/c}$	
[12]	Euclidean	$n^{1+\rho'(c)}$	$dn^{\rho'(c)}$	$\rho'(c) < 1/c$
[28]	Euclidean	n	$dn^{\rho''(c)}$	$\frac{\rho''(c)}{c} \to 2.09$
here	Euclidean	$n^{1+1/c^2+o(1)}$	$dn^{1/c^2+o(1)}$	
	Euclidean	n	$dn^{O(1/c^2)}$	

Figure 1. Space and time bounds for LSH-based data structures. Factors polynomial in $\log n$ and $1/\epsilon$, as well as an additive term of dn in the storage bound, are omitted for clarity.

1.1 Techniques

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

[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.

[2] An astute reader will observe that if *both* the dimension d and approximation factor c are fixed constants, one can obtain a data structure with *constant* query time, essentially via table lookup. However, this approach leads to "big-Oh" constants that are exponential in the dimension, which defeats our goal of achieving a practical algorithm.

our construction is the method of [12]. 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 [12] 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 [21, 15]. 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 super-constant, 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 [12]). Since the LSH algorithms performs such concatenation anyway (see Preliminaries), grid partitioning does not result in any improvement. Instead, we use the method of "ball partitioning", introduced in [9] in the context of embeddings into tree metrics (a similar technique was also used in the SDP-based approximation algorithm for graph coloring [22]). 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 $\Re^t$.

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 $1/c$. Specifically, it was shown in [9] that for any two points $p, q \in \Re^t$, the probability that the partitioning separates p and q is at most $O\left(\sqrt{t} \cdot \|p - q\|/w\right)$. This formula can be showed to be tight for the range of w where it makes sense as a lower bound, that is, for $w = \Omega\left(\sqrt{t} \cdot \|p - q\|\right)$. However, as long as the separation probability depends linearly on the distance between p and q, the exponent ρ is still equal to $1/c$. Fortunately, a more careful analysis shows that, as in the one-dimensional case, the minimum is achieved for finite w. For that value of w, the exponent tends to $1/c^2$ as t tends to infinity.

Leech Lattice LSH. In order to obtain a more practical algorithm, we introduce a different partitioning method that avoids the $t^{O(t)}$ factor. Specifically, we use tessellations induced by (randomly shifted) Voronoi diagrams of *fixed* t-dimensional point constellations which have the following two nice properties:

- The closest constellation point to a given point can be found efficiently, and
- The exponent ρ induced by the constellation is as close to $1/c^2$ as possible.

The partitioning is then implemented by randomly projecting the points into $\Re^d$, and using the Voronoi diagram. We discovered that a constellation in 24 dimensions known as Leech Lattice [26] satisfies the above properties quite well. First, the nearest point in the lattice can be found by using a (bounded) decoder of [2] which perform only 519 floating point operations per decoded point. Second, the exponent $\rho(c)$ guaranteed by that decoder is quite attractive: for $c = 2$ the exponent $\rho(2)$ is less than 0.37. The intuitive reason for that is that Leech Lattice is a "very symmetric" constellation, and thus its Voronoi cells are very "round". Moreover, if the dimension d does not exceed 24, then we can skip the dimensionality reduction part. In that case we get $\rho(2) \leq 0.27$, while the leading term in the running time remains the same.

Near-linear space algorithm. This result is achieved by plugging our new LSH function into the algorithm of [28]. Unlike the algorithm of [21] (which used n^ρ independent hash tables), his algorithm uses only *one* hash table to store the data set P. The hash table is then probed by hashing not just the query point q (as in [21]) but by hashing several points chosen randomly from the neighborhood of q. The intuition behind this approach is as follows. Let $p^* \in P$ be a point within distance 1 from q. If a random LSH function causes collision between p^* and q with probability $1/n^\rho$, then it is plausible that, with constant probability[3], a random hash function causes collision between q and a "non-negligible" (say, $\approx 1/n^\rho$) fraction of the points in the unit ball around q. Since q and p^* are "close", it follows that, with constant probability, a random hash function causes collision between p^* and a "non-negligible" (although slightly smaller) fraction of the unit ball around q, which is exactly what the algorithm of [28] needs.

Converting this intuition into a formal proof is somewhat technical. This is mostly due to the fact that the new LSH functions are more complex than the ones from [12] (used in [28]), and thus we had to extend his framework to a more general setting. We defer the proofs to the full version of this paper.

[3] In the actual proof, the probability is $1/\log^{O(1)} n$.

2. Preliminaries

2.1. Notation

In this 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, and we call $\bar{B}(p, r)$ its surface. For a ball with radius r in $\Re^d$, we call its surface area $\mathrm{Sur}^d(r)$ and its volume $\mathrm{Vol}^d(r)$. We note that $\mathrm{Sur}^d = S_d \cdot r^{d-1}$ and $\mathrm{Vol}^d(r) = \frac{S_d \cdot r^d}{d}$, where S_d is the surface area of a ball of radius one (see, for example, [29], page 11).

We also need a (standard) bound on the volume of the cap of a ball $B(p, r)$. Let $C(u, r)$ be the volume of the cap at distance u from the center of the ball. Alternatively, $C(u, r)$ is the half of the volume of the intersection of two balls of radius r with centers at distance $2u$. Furthermore, let $I(u, r) = \frac{C(u,r)}{\mathrm{Vol}^d(r)}$ be the cap volume relative to the volume of the entire sphere. We can bound $I(u, r)$ as follows.

Lemma 2.1. *For any $d \geq 2$ and $0 \leq u \leq r$,*

$$\frac{A_l}{\sqrt{d}} \left(1 - \left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}} \leq I(u, r) \leq \left(1 - \left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}}$$

The proof is deferred to the appendix.

We also use the following standard facts about random projections in Euclidean spaces (for proofs see, e.g., [21]). Let $A \in M_{t,d}$ be a random projection from $\Re^d$ to $\Re^t$; specifically each element of A is chosen from normal distribution $N(0, 1)$, multiplied by a scaling factor $\frac{1}{\sqrt{t}}$.

Fact 2.2. *For any vector $v \in \Re^d$, the value $\|Av\|^2/\|v\|^2$ is distributed with probability density $tP_{\chi^2}(xt)$, where $P_{\chi^2}(x) = \frac{x^{t/2-1}e^{-x/2}}{\Gamma(t/2)2^{t/2}}$ is the chi-squared distribution with t degrees of freedom. The expectation of $\|Av\|^2/\|v\|^2$ is equal to 1.*

Fact 2.3. *For any vector $v \in \Re^d$, $\Pr_A[\|Av\| > 2\|v\|] \leq \exp[-\Omega(t)]$.*

Fact 2.4. *For any vector $v \in \Re^d$ and any constant $\alpha > 10$, $\Pr_A[\|Av\| > \alpha\|v\|] \leq \exp[-\Omega(t\sqrt{\alpha})]$.*

2.2. Problem definition

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

Definition 2.5 (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). Below we describe the general LSH scheme, as it was first proposed in [21]. In this paper we reuse the same LSH scheme, but we introduce a new family of locality-sensitive hash functions.

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 2.6 (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 $\Pr_{\mathcal{H}}[h(q) = h(p)] \leq p_2$.*

The LSH functions can be used to solve the c-NN problem, as per the following theorem of [21]. Let $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$.

Fact 2.7. *Given a family of $(1, c, p_1, p_2)$-sensitive hash functions for $\Re^d$, where each function can be evaluated in time τ, one can construct a data structure for c-NN with $O((d + \tau)n^\rho \log_{1/p_2} n)$ query time and space $O(dn + n^{1+\rho})$.*

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 2.

Ideal LSH family. Construct a hash function $\tilde{h}$ 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^d_u, for positive integer u, be the grid G^d shifted uniformly at random; in other words, $G^d_u = G^d + s_u$, where $s_u \in [0, 4w]^d$. Now we choose as many G^d_u'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.

We define $\tilde{h}$ on a point p as a tuple $(u, x_1, x_2, ...x_d)$, $u \in [1, U]$ and $(x_1, ...x_d) \in G^d_u$. The tuple $(u, x_1, x_2, ...x_d)$ specifies the ball which contains the point p: $p \in B((x_1, x_2, \ldots x_n), w)$. If there are several balls that contain p, then we take the one with the smallest value u. Computing $\tilde{h}(p)$ can be done in $\tau = O(U)$ time: we iterate through all $G^d_1, G^d_2, ...G^d_U$, and find the first G^d_u such that p is inside a ball with the center from G^d_u.

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 family is that the time to compute $\tilde{h}(p)$ might be too large if $d = \Omega(\log n)$ since we need to set $U = \Omega(2^d)$ (see lemma 3.1). We show how to circumvent this deficiency next.

Actual LSH family. Our actual construction utilizes the "ideal" family described above, while introducing an additional step, necessary to reduce U, the number of grids covering the space. The algorithm is given in Figure 2.

To reduce U, we project $\Re^d$ to a lower-dimensional space $\Re^t$ via a random dimensionality reduction. The parameter t is $o(\log n)$, such that factors exponential in t are $o(n)$. After performing the projection, we choose the grids $G^t_1, G^t_2, ...G^t_U$ in the lower-dimensional space $\Re^t$. Now, to compute $h(p)$, we compute the projection of p onto the lower dimensional space $\Re^t$, and process the projected point as described earlier. In short, the actual hash function is $h(p) = \tilde{h}(Ap)$, where A is a random matrix representing the dimensionality reduction mapping, and $\tilde{h}$ works in the t-dimensional space. Note that τ becomes $\tau = O(dt) + O(U^t)$ corresponding to the projection and the bucket-computation stages respectively.

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

1. For $u = 1$ to U, choose a random shift $s_u \in [0, 4w]^t$, which specifies the grid $G^t_u = G^t + s_u$ 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 the t-dimensional subspace given by A.
2. For each $u = 1, 2, \ldots U$
   3. Check whether $B(p', w) \cap G^t_u \neq \emptyset$, i.e., whether there exist some $(x_1, x_2, \ldots x_t) \in G^t_u$ such that $p \in B((x_1, x_2, \ldots x_t), w)$.
   4. Once we find such $(x_1, x_2, \ldots x_t)$, 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 2. Algorithms for initializing a hash function h from the LSH hash family, and for computing $h(p)$ for a point $p \in \Re^d$.

3.2. Analysis of the LSH family

We start by bounding the number of grids G^d needed to cover the entire space $\Re^d$, for any dimension d.

Lemma 3.1. *Consider a d-dimensional space $\Re^d$. Let G^d be a regular infinite grid of balls of radius w placed at coordinates $\sigma w \cdot \mathbb{Z}^d$, where $2 \leq \sigma \leq d^{O(1)}$. Define G^d_u, for positive integer u, as $G^d_u = G^d + s_u$, where $s_u \in [0, \sigma w]^d$ is a random shift of the grid G^d. If $U_d = 2^{O(d \log d)} \log n$, then, the grids $G^d_1, G^d_2, \ldots G^d_{U_d}$ cover the entire space $\Re^d$, w.h.p.*

Proof. First, observe that the entire space is covered if and only if the hypercube $[0, \sigma w]^d$ is covered by grids G^d_u (due to the regularity of the grids).

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

Thus, we can compute the probability that there exists at least one uncovered micro-cube, which is also the probability that the entire $[0, \sigma w]^d$ hypercube is uncovered. Set $U_d = aN(\log n + \log N)$ for a suitable constant a. Using union bound, we obtain that the probability that the entire hypercube is not covered is at most

$$N\,(1-x)^{U_d} \leq N(1-1/N)^{U_d} \leq$$
$$\leq N(1-1/N)^{aN(\log n+\log N)} \leq N2^{-\log n-\log N} \leq 1/n$$

Concluding: with probability at least $1-1/n$ we cover the entire space with the grids $G_1^d, \ldots G_{U_d}^d$, if we choose $U_d = O(N(\log n+\log N)) = 2^{O(d\log d)}\log n$. □

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

Lemma 3.2. *Consider the hash function h described in the figure 2, 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 $w = \Theta\left(\sqrt[4]{t}\right)$, we obtain $\rho = \frac{\log 1/p_1}{\log 1/p_2} = 1/c^2 + O\left(\frac{\log t}{t^{1/2}}\right)$.*

Proof. The proof proceeds in three stages. Fix some points $p, q \in \Re^d$ at distance Δ. First we compute the probability of collision of p and q given that their distance after the projection is equal to some fixed value Δ'. Next, for each of the cases when $\Delta \leq 1$ and $\Delta \geq c$, we compute the collision probabilities (p_1 and p_2, resp.) by integrating over the range of the possible (distorted) distances Δ'. Finally, given the bounds on p_1 and p_2, we compute the value of $\rho = \frac{\log 1/p_1}{\log 1/p_2}$.

Suppose points p and q are projected under the dimensionality reduction into points $p' = Ap$ and $q' = Aq$, $p', q' \in \Re^t$, with $\Delta' = \|p'-q'\|$; the probability of collision of p and q 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 such that p' or q' are inside a ball $B(x,w)$ with center in G_u^t. Note that the position of this ball defines whether $h(p) = h(q)$ or not. In particular, if $p', q' \in B(x,w)$ then $h(p) = h(q)$ and, otherwise, if exactly one of p', q' is inside $B(x,w)$ then $h(p) \neq h(q)$. Thus, we can conclude that the probability of collision of points p, q is

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

where $C(\Delta'/2, w)$ and $I(\Delta'/2, w)$ are respectively the cap volume and the relative cap volume, as defined in the preliminaries.

In the next step we bound p_2. This is done by integrating the collision probability over all possible values of Δ', i.e., over all distortions of $\|p-q\|$. As noted in fact 2.2, the distortion of $\|p-q\|^2$ is distributed with probability density P_{χ^2}.

$$\begin{aligned}
p_2 &= \int_0^\infty \Pr[h(p)=h(q) \mid \|p'-q'\| = \sqrt{\tfrac{x}{t}}c] \cdot P_{\chi^2}(x)dx \\
&\leq \int_{0\leq\sqrt{\frac{x}{t}}c\leq 2w} P_{\chi^2}(x) \cdot \frac{I\left(\frac12\sqrt{\frac{x}{t}}c, w\right)}{1-I\left(\frac12\sqrt{\frac{x}{t}}c, w\right)}dx \\
&\leq \int_0^\infty P_{\chi^2}(x) \cdot 2I\left(\tfrac12\sqrt{\tfrac{x}{t}}c, w\right)dx \\
&\leq 2\int_0^\infty P_{\chi^2}(x) \cdot \left(1-\left(\frac{\frac12\sqrt{\frac{x}{t}}c}{w}\right)^2\right)^{\frac{t}{2}} dx \\
&\leq 2\int_0^\infty P_{\chi^2}(x) \cdot \exp\left[-\tfrac{t}{2}\cdot x\tfrac{c^2}{4w^2t}\right]dx \\
&= 2\int_0^\infty P_{\chi^2}(x) \cdot \exp\left[-\tfrac12 \cdot x\tfrac{c^2}{4w^2}\right]dx
\end{aligned}$$

where, for the third inequality, we used lemma 2.1. Setting $\epsilon = \frac{1}{4w^2}$, and replacing the expression for P_{χ^2}, we obtain

$$\begin{aligned}
p_2 &\leq 2\int_0^\infty \tfrac{x^{t/2-1}}{\Gamma(t/2)2^{t/2}}\exp\left[-\tfrac{x}{2}\right]\cdot\exp\left[-\tfrac{x\cdot c^2\epsilon}{2}\right]dx \\
&= 2\int_0^\infty \tfrac{(x(1+c^2\epsilon))^{t/2-1}}{(1+c^2\epsilon)^{t/2-1}\Gamma(t/2)2^{t/2}}\cdot\exp\left[-\tfrac{x(1+c^2\epsilon)}{2}\right]dx \\
&= \tfrac{2}{(1+c^2\epsilon)^{t/2}}\int_0^\infty P_{\chi^2}(x(1+c^2\epsilon))d(x(1+c^2\epsilon)) \\
&= \tfrac{2}{(1+c^2\epsilon)^{t/2}}
\end{aligned} \tag{2}$$

We bound p_1 from below in a similar way

$$\begin{aligned}
p_1 &\geq \int_{0\leq\sqrt{\frac{x}{t}}\leq 2w} P_{\chi^2}(x)\cdot\frac{I(\frac12\sqrt{\frac{x}{t}},w)}{1-I(\frac12\sqrt{\frac{x}{t}},w)}dx \\
&\geq \int_0^{4w^2t} P_{\chi^2}(x)\cdot I(\tfrac12\sqrt{\tfrac{x}{t}},w)dx \\
&\geq \int_0^{t/\epsilon} P_{\chi^2}(x)\cdot\frac{A_l}{\sqrt{t}}\left(1-\left(\frac{\frac12\sqrt{\frac{x}{t}}}{w}\right)^2\right)^{\frac{t}{2}}dx \\
&= \frac{A_l}{\sqrt{t}}\int_0^{t/\epsilon} P_{\chi^2}(x)\cdot\left(1-\tfrac{x\epsilon}{t}\right)^{\frac{t}{2}}dx \\
&= \frac{A_l}{\sqrt{t}}\int_0^{t/\epsilon} P_{\chi^2}(x)\cdot\left(\frac{1}{1+\frac{x\epsilon/t}{1-x\epsilon/t}}\right)^{\frac{t}{2}}dx \\
&\geq \frac{A_l}{\sqrt{t}}\int_0^{t/\epsilon} P_{\chi^2}(x)\cdot\exp\left[-\tfrac{t}{2}\cdot\tfrac{x\epsilon/t}{1-x\epsilon/t}\right]dx \\
&\geq \frac{A_l}{\sqrt{t}}\int_0^{4t} P_{\chi^2}(x)\cdot\exp\left[-\tfrac{t}{2}\cdot\tfrac{x\epsilon}{t}(1+8\epsilon)\right]dx \\
&\geq \frac{A_l}{\sqrt{t}}\left(\int_0^\infty P_{\chi^2}(x)\cdot\exp\left[-\tfrac{x\epsilon}{2}(1+8\epsilon)\right]dx - \int_{4t}^\infty P_{\chi^2}(x)dx\right)
\end{aligned} \tag{3}$$

Note that the term $\int_{4t}^\infty P_{\chi^2}(x)dx$ represents the probability of expansion by more than a factor of 2, which is at most $\exp[-\Omega(t)]$ by fact 2.3. Furthermore, replacing the expression for P_{χ^2}, we obtain

$$\begin{aligned} p_1 &\geq \frac{A_l}{\sqrt{t}}\left(\int_0^\infty P_{\chi^2}(x)\cdot\exp\left[-\frac{x\epsilon}{2}(1+8\epsilon)\right]dx - e^{-\Omega(t)}\right) \\ &\geq \frac{A_l}{\sqrt{t}}\int_0^\infty \frac{x^{\frac{t}{2}-1}}{\Gamma(\frac{t}{2})2^{\frac{t}{2}}}e^{-x/2}\cdot\exp\left[-\frac{x\epsilon}{2}(1+8\epsilon)\right]dx - e^{-\Omega(t)} \\ &= \frac{A_l}{\sqrt{t}}\int_0^\infty \frac{(x(1+(1+8\epsilon)\epsilon))^{\frac{t}{2}-1}}{(1+(1+8\epsilon)\epsilon)^{\frac{t}{2}-1}\Gamma(\frac{t}{2})2^{\frac{t}{2}}}\exp\left[-\frac{x(1+(1+8\epsilon)\epsilon)}{2}\right]dx \\ &\quad -e^{-\Omega(t)} \\ &= \frac{A_l}{\sqrt{t}}\cdot\frac{1}{(1+(1+8\epsilon)\epsilon)^{t/2}} - e^{-\Omega(t)} \end{aligned} \tag{4}$$

For $\epsilon = 1/4w^2 = o(1)$, we obtain $p_1 \geq \frac{A_l}{2\sqrt{t}}\cdot\frac{1}{(1+\epsilon+8\epsilon^2)^{t/2}}$.

Finally, we can bound ρ as follows:

$$\begin{aligned} \rho &= \frac{\log 1/p_1}{\log 1/p_2} \leq \frac{\log\frac{2\sqrt{t}}{A_l}\cdot(1+(1+8\epsilon)\epsilon)^{t/2}}{\log\frac{1}{2}\cdot(1+c^2\epsilon)^{t/2}} \\ &= \frac{\log(1+(1+8\epsilon)\epsilon)^{t/2}+\log\frac{2\sqrt{t}}{A_l}}{\log(1+c^2\epsilon)^{t/2}-\log 2} \\ &= \frac{\log(1+(1+8\epsilon)\epsilon)+\frac{2\log 2\sqrt{t}/A_l}{t}}{\log(1+c^2\epsilon)-\frac{2\log 2}{t}} \\ &\leq \frac{\log(1+(1+8\epsilon)\epsilon)}{\log(1+c^2\epsilon)}\cdot\left(1+\frac{2\log 2\sqrt{t}/A_l}{t\log(1+(1+8\epsilon)\epsilon)}\right) \\ &\quad \cdot\left(1+O\left(\frac{2\log 2}{t\log(1+c^2\epsilon)}\right)\right) \\ &\leq \frac{(1+8\epsilon)\epsilon}{c^2\epsilon-(c^2\epsilon)^2/2}\cdot\left(1+O\left(\frac{\log t}{t\log(1+\epsilon)}\right)\right) \\ &\leq \frac{1}{c^2}(1+8\epsilon)\cdot\left(1+O(c^2\epsilon/2)\right)\cdot\left(1+O\left(\frac{\log t}{t\epsilon}\right)\right) \\ &\leq \frac{1}{c^2}\cdot\left(1+O\left(\epsilon+\frac{w^2\log t}{t}\right)\right) \\ &\leq \frac{1}{c^2}\cdot\left(1+O\left(\frac{\log t}{t^{1/2}}\right)\right) \end{aligned} \tag{5}$$

for $w = \sqrt[4]{t}$, which also implies $\epsilon = O(t^{-1/2}) = o(1)$. □

Theorem 3.3. *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 by using the LSH family in figure 2 with the general LSH scheme described in section 2.3. By lemma 3.2, for $t = \log^{2/3} n$, we have $\rho = 1/c^2 + O\left(\frac{\log\log n}{\log^{1/3} n}\right)$. Furthermore, k can be bounded as (using eqn. (2))

$$k = \frac{\log n}{\log 1/p_2} \leq \frac{\log n}{\log\left(1+c^2/4w^2\right)^{t/2}/2} =$$

$$O\left(\frac{\log n}{t/w^2 - O(1)}\right) = O(\log n)$$

Finally, by lemma 3.1 for $\sigma = 4$, $\tau = O(dt) + O(U^t) = O(dt) + O\left(2^{t\log t}\log n\right) = O(dt) + 2^{O(\log^{2/3} n\log\log n)}\log n = O(dn^{o(1)})$. The theorem follows. □

4. Acknowledgments

The authors would like to thank Assaf Naor for stimulating discussions about the ball partitioning method. Also, they thank Ofer Amrani for supplying the bounded distance decoder code, and to Alex Vardy and Erik Agrell for answering numerous questions about Leech Lattice decoders.

References

[1] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. *Proceedings of the Symposium on Theory of Computing*, 2006.

[2] O. Amrani and Y. Be'ery. Efficient bounded-distance decoding of the hexacode and associated decoders for the leech lattice and the golay code. *IEEE Transactions on Communications*, 44:534–537, May 1996.

[3] O. Amrani, Y. Be'ery, A. Vardy, F.-W. Sun, and H. C. A. van Tilborg. The leech lattice and the golay code: Bounded-distance decoding and multilevel constructions. *IEEE Transactions on Information Theory*, 40:1030–1043, July 1994.

[4] 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.

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

[6] A. Andoni, P. Indyk, and M. Pătraşcu. On the optimality of the dimensionality reduction method. *Manuscript*, 2006.

[7] S. Arya, D. Mount, N. 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.

[8] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bounds for approximate nearest neighbor searching. *Proceedings of the Symposium on Foundations of Computer Science*, 2004.

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

[10] J. H. Conway and J. A. Sloane. Soft decoding techniques for codes and lattices, including the golay code and the leech lattice. *IEEE Trans. Inf. Theor.*, 32(1):41–50, 1986.

[11] J. H. Conway and J. A. Sloane. *Sphere Packings, Lattices, and Groups*. Springer-Verlag, New York, 1993.

[12] 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.

[13] U. Feige and G. Schechtman. On the optimality of the random hyperplane rounding technique for max cut. *Random Struct. Algorithms*, 20(3):403–440, 2002.

[14] G. Forney and A. Vardy. Generalized minimum distance decoding of euclidean-space codes and lattices. *IEEE Transactions on Information Theory*, 42:1992–2026, November 1996.
[15] 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.
[16] S. Har-Peled. A replacement for voronoi diagrams of near linear size. *Proceedings of the Symposium on Foundations of Computer Science*, 2001.
[17] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-medians and their applications. *Proceedings of the Symposium on Theory of Computing*, 2004.
[18] P. Indyk. *High-dimensional computational geometry*. Department of Computer Science, Stanford University, 2001.
[19] P. Indyk. Approximate algorithms for high-dimensional geometric problems. *Invited talk at DIMACS Workshop on Computational Geometry. Available at* `http://theory.csail.mit.edu/~indyk/high.ps`, 2002.
[20] P. Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2003.
[21] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing*, 1998.
[22] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1994.
[23] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.
[24] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2004.
[25] 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.
[26] J. Leech. Notes on sphere packings. *Canadian Journal of Mathematics*, 1967.
[27] R. Motwani, A. Naor, and R. Panigrahy. Lower bounds on locality sensitive hashing. *Proceedings of the ACM Symposium on Computational Geometry*, 2006.
[28] R. Panigrahy. Entropy-based nearest neighbor algorithm in high dimensions. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2006.
[29] G. Pisier. *The volume of convex bodies and Banach space geometry*. Cambridge University Press, 1989.
[30] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Elsevier, 2006.
[31] G. Shakhnarovich, T. Darrell, and P. Indyk, editors. *Nearest Neighbor Methods in Learning and Vision*. Neural Processing Information Series, MIT Press, 2006.
[32] A. Vardy and Y. Be'ery. Maximum-likelihood decoding of the leech lattice. *IEEE Transactions on Information Theory*, 39:1435–1444, July 1993.

A. Cap of the sphere

Proof. The result follows immediately from the lemma 9 of [13], which gives bounds on the ratio of the surface areas of the cap to that of the ball. Specifically, note that $I(u,r)$ has the following form

$$\begin{aligned} I(u,r) &= C(u,r)\cdot(\mathrm{Vol}^d(r))^{-1} \\ &= \int_u^r \tfrac{S_{d-1}}{d-1}(r^2-y^2)^{\frac{d-1}{2}}dy\cdot\left(\tfrac{S_d}{d}r^d\right)^{-1} \\ &= \tfrac{d}{d-1}\cdot\left(\int_u^r S_{d-1}(r^2-y^2)^{\frac{d-1}{2}}dy\cdot\left(S_d r^d\right)^{-1}\right) \end{aligned}$$

The quantity $\int_u^r S_{d-1}(r^2-y^2)^{\frac{d-1}{2}}dy\cdot\left(S_d r^d\right)^{-1}$ represents precisely the ratio of the surface area of the cap $C(u,r)$ (excluding the base) to the surface area of a ball of radius r in the $(d+1)$-dimensional space. This ratio is bounded [13] as

$$\frac{A_l}{\sqrt{d+1}}\left(1-\left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}} \le \int_u^r S_{d-1}(r^2-y^2)^{\frac{d-1}{2}}dy\cdot\left(S_d r^t\right)^{-1} \le \frac{1}{2}\left(1-\left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}}$$

Thus, multiplying the above bounds by $\frac{d}{d-1}$, we obtain that

$$\frac{d}{d-1}\cdot\frac{A_l}{\sqrt{d+1}}\left(1-\left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}} \le I(u,r) \le \frac{d}{d-1}\cdot\frac{1}{2}\left(1-\left(\frac{u}{r}\right)^2\right)^{\frac{d}{2}}$$

which implies the lemma. □

B. Lattice-based LSH family

In this section we describe a practical variant of the LSH family based on lattices in Euclidean spaces. Although, theoretically, these families are not asymptotically better than the ones described earlier, they are likely to perform better in practice, due to much lower "big-Oh" constants.

We start by presenting the general lattice-based approach. Then, we give an algorithm based on a concrete 24-dimensional lattice, called Leech Lattice [26]. For the Leech lattice-based algorithm, we include the actual values of the resulting exponent ρ, the main indicator of the performance.

B.1. Lattices in arbitrary dimension

The algorithm in this section uses an arbitrary lattice in some t-dimensional space. An example of a t-dimensional lattice is the regular grid of points in $\Re^t$, although, as mentioned in the introduction, it does not serve well our purposes. For a given lattice, we need an efficient lattice decoding function, to which we refer as $\textsc{LatticeDecode}(x)$. The function $\textsc{LatticeDecode}(x)$ takes as input a point $x \in \Re^t$ and returns the lattice point that is the closest to x.

Given a specific lattice with a decoding function $\textsc{LatticeDecode}(x)$, an LSH function is constructed as follows (formally presented in figure B.1). First, if $d > t$, we choose a random *projection* from d-dimensional space to t-dimensional space, which we represent as a matrix A of dimension $t \times d$. If $d \le t$, then, instead, we choose a random *rotation* in the t-dimensional space, which we also represent as a matrix A of dimension $t \times d$ (here, A is equal to the first d columns of an random ortonormal matrix of dimension $t \times t$). Finally, we choose a random *translation* in the t-dimensional space, which we represent as a vector T of dimension $t \times 1$. The values of A, T identify an LSH function.

For an LSH function h specified by values of A and T, we define $h(p)$ as being $h(p) = \textsc{LatticeDecode}(A \cdot p + T)$. Or, in words, for $p \in \Re^d$, we first project p into $\Re^t$ using A (or rotate it if $d \le t$); then, we translate the projection using T; and, finally, we find the closest point in lattice using LatticeDecode. The output of LatticeDecode gives the value of $h(p)$.

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

1. If $d > t$, choose a random projection from d-dimensional space to t-dimensional space. The projection is represented by 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, $\frac{1}{\sqrt{t}}$.
2. If $d \le t$, choose a random rotation in the t-dimensional space. The rotation is represented by the matrix A, which is equal to the first d coordinates of an $t \times t$ ortonormal matrix.
3. Choose a random translation in the t-dimensional space. The translation is represented by a vector $T \in M_{t,1}$.

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

1. Let $x = A \cdot p + T$.
2. Return $\textsc{LatticeDecode}(x)$.

Figure 3. Algorithms for initializing an LSH function h and for computing $h(p)$ for a point $p \in \Re^d$.

The performance of the resulting LSH scheme depends heavily on the choice of the lattice. Intuitively, we would like a lattice that lives in $\Re^t$ for high t, is "dense"[4], and has a fast decoding function LatticeDecode. With a higher t, the dimensionality reduction is more accurate. A "denser" lattice gives a sharper difference in collision probabilities of close and far points.

[4] A measure of "density" is, for example, the density of hypersphere packing induced by the lattice. The density of hypersphere packing is the percent of the space that is covered by non-overlapping balls centered at lattice points.

B.2. Leech Lattice

In this section, we focus on a particular lattice in 24 dimensional space, the Leech Lattice [26]. We give numerical values for the ρ when we use the Leech Lattice in the algorithm B.1 with a specific decoder described below.

The Leech Lattice has been studied extensively (see, e.g., [11, 10, 2, 32, 14]) and is known to be the lattice that gives the densest (lattice) hypersphere packing in 24 dimensions. Below, we denote the Leech Lattice by λ_{24} and call the corresponding decoding function $\textsc{LatticeDecode}_{\lambda_{24}}(x)$. Several efficient decoders for the Leech Lattice are known; the best of them [32] requires 3595 floating point operations to decode one point. However, even faster decoders are known (e.g.,see [3, 2, 14]) for the *bounded-distance* decoding problem. A bounded-distance decoder guarantees to return the correct result only when the query point x is sufficiently close to one of the lattice points; otherwise the decoder gives no guarantees. Note that a bounded-distance decoder yields an LSH function, albeit not necessarily as good as the perfect decoder.

We have investigated the bounded-distance decoder of [2], which we call $\textsc{LatticeDecode}^B_{\lambda_{24}}(x)$. Their implementation uses at most 519 real operations per decoded point. For that decoder, we computed the values of the resulting collision probabilities (for the case $d > 24$). The results are depicted in Table 1. The probabilities are computed using Monte-Carlo simulation with 10^7 trials. Specifically, in a trial, we generate a random point p and some other point q, such that $p - q$ is drawn from a 24-dimensional Gaussian distribution, scaled by $\frac{1}{\sqrt{24}}$ times the radius. The points p and q collide iff $\textsc{LatticeDecode}^B_{\lambda_{24}}(p) = \textsc{LatticeDecode}^B_{\lambda_{24}}(q)$. Table 1 summarizes the estimated probabilities of collision for different values of radii (the confidence intervals are computed with 95% accuracy). These probabilities yield values for ρ that are summarized in Table 2. The table shows maximum likelihood ρ and conservative ρ. The max likehood ρ is the ratio of corresponding max likehood values of p_1 and p_2 (from the middle column). The conservative ρ is the ratio of lowest estimate of p_1 from the confidence interval to the highest estimate of p_2 in the confidence interval.

For the case when $d \le 24$, the collision probabilities are summarized in table 3. The method for computing the probabilities is as before, except for the generation of the point q. In this case, the vector $q - p$ is a random vector of *fixed* length. The resulting values of ρ are summarized in Table 4.

Radius	Est. collision prob.	Confidence interval
0.7	0.0853465	[0.0853409, 0.0853521]
0.8	0.0525858	[0.0525813, 0.0525903]
0.9	0.0311720	[0.0311685, 0.0311755]
1.0	0.0177896	[0.0177869, 0.0177923]
1.1	0.0097459	[0.0097439, 0.0097479]
1.2	0.0051508	[0.0051493, 0.0051523]
1.3	0.0026622	[0.0026611, 0.0026633]
1.4	0.0013332	[0.0013324, 0.0013340]
1.5	0.0006675	[0.0006670, 0.0006681]
1.6	0.0003269	[0.0003265, 0.0003273]
1.7	0.0001550	[0.0001547, 0.0001553]
1.8	0.0000771	[0.0000769, 0.0000773]
1.9	0.0000368	[0.0000366, 0.0000370]
2.0	0.0000156	[0.0000155, 0.0000157]

Table 1. Probabilities of collision of two points, for $d > 24$, under the hash function described in figure B.1 with bounded-distance Leech Lattice decoder. The values were obtained through Monte-Carlo simulation for 10^7 trials. Confidence interval corresponds to 95% accuracy.

Radius	Est. collision prob.	Confidence interval
0.7	0.0744600	[0.0744548, 0.0744653]
0.8	0.0424745	[0.0424705, 0.0424786]
0.9	0.0223114	[0.0223084, 0.0223144]
1.0	0.0107606	[0.0107585, 0.0107627]
1.1	0.0046653	[0.0046639, 0.0046667]
1.2	0.0017847	[0.0017838, 0.0017856]
1.3	0.0005885	[0.0005880, 0.0005890]
1.4	0.0001602	[0.0001599, 0.0001605]
1.5	0.0000338	[0.0000337, 0.0000340]
1.6	0.0000073	[0.0000072, 0.0000074]
1.7	0.0000009	[0.0000008, 0.0000010]
1.8	0.0000000	[0.0000000, 0.0000001]

Table 3. Probabilities of collision of two points, for $d \leq 24$, under the hash function described in figure B.1 with bounded-distance Leech decoder. The values were obtained through Monte-Carlo simulation for 10^7 trials. Confidence interval corresponds to 95% accuracy.

c	Max likelihood of ρ	Conservative ρ	Radius R
1.5	0.5563	0.5565	1.2
2.0	0.3641	0.3643	1.0

Table 2. The values of $\rho = \frac{\log p_1}{\log p_2}$ corresponding to the collision probabilities in Table 1 ($d > 24$). Probabilities p_1 and p_2 are the collision probabilities corresponding to radii R and cR, respectively.

c	Max likelihood of ρ	Conservative ρ	Radius R
1.5	0.4402	0.4405	1
2.0	0.2671	0.2674	0.8

Table 4. The values of $\rho = \frac{\log p_1}{\log p_2}$ corresponding to the collision probabilities in table 1 ($d \leq 24$). Probabilities p_1 and p_2 are the collision probabilities corresponding to radii R and cR respectively.