SIAM J. COMPUT.
Vol. 39, No. 1, pp. 302–322

THE FAST JOHNSON–LINDENSTRAUSS TRANSFORM AND APPROXIMATE NEAREST NEIGHBORS*

NIR AILON† AND BERNARD CHAZELLE‡

Abstract. We introduce a new low-distortion embedding of ℓ_2^d into $\ell_p^{O(\log n)}$ $(p = 1, 2)$ called the *fast Johnson–Lindenstrauss transform* (FJLT). The FJLT is faster than standard random projections and just as easy to implement. It is based upon the preconditioning of a sparse projection matrix with a randomized Fourier transform. Sparse random projections are unsuitable for low-distortion embeddings. We overcome this handicap by exploiting the "Heisenberg principle" of the Fourier transform, i.e., its local-global duality. The FJLT can be used to speed up search algorithms based on low-distortion embeddings in ℓ_1 and ℓ_2. We consider the case of approximate nearest neighbors in ℓ_2^d. We provide a faster algorithm using classical projections, which we then speed up further by plugging in the FJLT. We also give a faster algorithm for searching over the hypercube.

Key words. dimension reduction, random matrices, approximate nearest neighbors

AMS subject classification. 68Q01

DOI. 10.1137/060673096

1. Introduction.

Metric embeddings have been proven to be very useful algorithmic tools. In the most general setting, one wishes to map a finite metric space (U, d_U) to another metric space (V, d_V) belonging to some restricted family, without distorting the pairwise distances too much. If we denote the mapping by Φ, we usually care about minimizing the relative distortion, which we can define here as

$$\max_{x_1,x_2\in U} \frac{|d_U(x_1,x_2) - d_V(\Phi(x_1),\Phi(x_2))|}{d_U(x_1,x_2)}. \tag{1}$$

The metric space V usually belongs to a restricted family of metric spaces, which is easier to work with. Much of the computer science literature uses the expression $\|\Phi\|_{Lip} \cdot \|\Phi^{-1}\|_{Lip}$ as a standard measure of distortion, where $\|\cdot\|_{Lip}$ is the *Lipschitz norm* of a function between metric spaces. In our work, we will be interested in mappings Φ for which (1) is bounded by a small ε, which is equivalent to $\|\Phi\|_{Lip} \cdot \|\Phi^{-1}\|_{Lip} \le 1+O(\varepsilon)$. Hence, our definition is essentially the standard one. Also, much work related to this subject makes the technical assumption that Φ is 1-Lipschitz, namely, it does not stretch pairwise distances, but we do not assume that here.

Linial, London, and Rabinovich in their seminal paper [34] first considered the algorithmic applications of metric embeddings. Embedding into ℓ_1 metrics is especially important in optimization of hard cut problems in graphs. Embedding into tree metrics is useful in optimization of hard network design and clustering problems.

In many cases, we are interested in embedding high-dimensional normed metrics spaces into low-dimensional normed spaces. Papadimitriou et al. [39] use low-dimensional embeddings to speed up computation of low-rank approximation of ma-

*Received by the editors October 24, 2006; accepted for publication (in revised form) January 9, 2009; published electronically May 28, 2009.
http://www.siam.org/journals/sicomp/39-1/67309.html

†Google Research, New York, NY 10011 (nailon@gmail.com). Most of this author's work was done while a student in the Department of Computer Science, Princeton University.

‡Department of Computer Science, Princeton University, Princeton, NJ 08544 (chazelle@cs.princeton.edu). This author's work was supported in part by NSF grant CCF-0634958 and NSF CCF 0832797.

trices. Schulman [41] used them for efficiently approximating certain metric clustering problems and Indyk [26] used them for data streaming applications. Indyk and Motwani [29] and Kushilevitz, Ostrovsky, and Rabani [33] used low-dimensional embeddings to speed up approximate nearest neighbor searching. The latter example is the main motivation for this work, though we hope to see other applications as well.

Our main result is a new low-distortion embedding of ℓ_2^d into $\ell_p^{O(\log n)}$ $(p = 1, 2)$, where n is the number of points. The possibility of obtaining such an embedding with distortion $(1+\varepsilon)$ for any $\varepsilon > 0$ using the Johnson–Lindenstrauss (JL) transform has long been known. In this work we consider the complexity of implementing this embedding, and show a nontrivial way to obtain a significant speedup. We call our new embedding the *fast Johnson–Lindenstrauss transform* (FJLT), and present it in this section. Naive implementations of JL incur only a polynomial cost, which is enough for some applications. However, for other applications, such as approximate nearest neighbor searching, sublinear algorithms are sought. Applying JL is the bottleneck of some of the fastest algorithms. We apply the $p = 1$ case to improve them.

1.1. History of the Johnson–Lindenstrauss transform. By the JL lemma [28, 30, 35], n points in Euclidean space can be projected down to $k = c\varepsilon^{-2} \log n$ dimensions while incurring a distortion of at most $1 + \varepsilon$, where $c > 0$ is some global constant. More precisely, for any set V of n points in d dimensions, taking $k = c\varepsilon^{-2} \log n$ (for some global constant c) suffices to ensure that with constant probability, for all $x, y \in V$,

$$\sqrt{\frac{k}{d}}\|x - y\|_2(1 - \varepsilon) \le \|\Phi x - \Phi y\|_2 \le \sqrt{\frac{k}{d}}\|x - y\|_2(1 + \varepsilon).$$

The original JL transform was defined in Johnson and Lindenstrauss's seminal paper [30]. It is, in fact, no more than a projection Φ from d dimensions onto a randomly chosen k-dimensional subspace. This is useful provided $k < d$, which will be implied by the assumption

$$n = 2^{O(\varepsilon^2 d)}. \tag{2}$$

Following [30], researchers (Frankl and Maehara [20] and later Dasgupta and Gupta [15]) suggested variants and simplifications of their design and/or proof together with improvements on the constant c. The two papers, though simplifying and improving the original JL result, do not depart from it in the sense that the random projection they assumed is equivalent to the one assumed in [30]. If we represent this projection by a $k \times d$ real matrix Φ, then the rows of the matrix can be computed as follows: The first row is a random unit vector uniformly chosen[1] from S^{d-1} (the $(d-1)$-dimensional unit sphere in $\mathbb{R}^d$). The second row is a random unit vector from the space orthogonal to the first row, the third is a random unit vector from the space orthogonal to the first two rows, and so on. (In order to choose a random unit vector in $\mathbb{R}^m$, one can choose m independently and identically distributed (i.i.d.) normally distributed random variables, and normalize the resulting vector to achieve norm 1.) The resulting matrix is a projection onto a random k-dimensional subspace of $\mathbb{R}^n$ (together with a choice of an orthogonal basis, having no effect on the ℓ_2-norm). This choice of Φ satisfies the following three properties (which, in fact, completely characterize it):

[1] By this we mean the Haar measure on the sphere.

- Spherical symmetry: For any orthogonal matrix $A \in O(d)$, ΦA and Φ have the same distribution.
- Orthogonality: The rows of Φ are orthogonal to each other.
- Normality: The rows of Φ are unit-length vectors.

The first to depart from the above distribution on Φ were Indyk and Motwani [29], who dropped the orthogonality and the normality conditions. They noticed that in order to obtain the JL guarantee, one can independently choose every entry of Φ using the distribution $N(0, 1/d)$. The norm of each row of Φ becomes a random variable. Furthermore, the rows are no longer necessarily orthogonal. Normality and orthogonality are satisfied only on expectation. Indeed, the squared ℓ_2 norm of every row of Φ is 1 on expectation, and the inner product of every two rows is 0 on expectation. The independence of the entries in Φ make the proof much simpler. Note that the distribution on Φ remains spherically symmetric.

The next bold and ingenious step was taken by Achlioptas [1], who dropped the spherical symmetry condition. He noticed that the only property of Φ needed for the transformation to work is that $(\Phi_i \cdot x)^2$ is tightly concentrated around mean $1/d$ for all unit vectors x, where Φ_i is the ith row of Φ. The distribution he proposed is very simple: Choose each entry of Φ uniformly from $\{+d^{-1/2}, -d^{-1/2}\}$ (note that the normality condition is restored). The motivation in [1] was to make random projections easier to use in practice. Indeed, flipping coins is much easier than choosing Gaussian-distributed numbers. More surprisingly, he shows that if the entries of Φ are chosen independently according to the following distribution:

$$\begin{cases} +(d/3)^{-1/2} & \text{with probability } 1/6, \\ 0 & 2/3, \\ -(d/3)^{-1/2} & 1/6, \end{cases} \tag{3}$$

then the JL guarantee holds (note that normality is now dropped). The nice property of this distribution is that it is *relatively sparse*: On expectation, a (2/3)-fraction of Φ is 0. Assuming we want to apply Φ on many points in $\mathbb{R}^d$ in a real-time setting, by keeping a linked list of all the nonzeros of Φ during preprocessing we get a 3-fold speedup in the transformation computation, compared to a naive matrix-vector multiplication.

In all the aforementioned methods, projecting a point requires multiplication by a dense k-by-d matrix, and so mapping each point takes $O(d \log n)$ time (for fixed ε). Is that optimal?

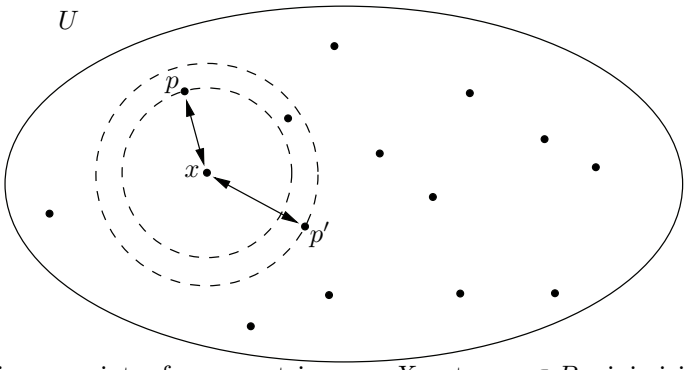
The attempt to sparsify Φ and obtain a superconstant speedup is the first point of departure for this work. A lower bound of Alon [3] dashes any hope of reducing the number of rows of Φ by more than a factor of $O(\log(1/\varepsilon))$, so the obvious question is whether the matrix can be made sparse. Bingham and Mannila [8] considered sparse projection heuristics for dimension-reduction based algorithms, and noticed that in practice they seem to give a considerable speedup with little compromise in quality. Achlioptas's method [1] can be used to reduce the density of Φ by only a constant factor, but one simply cannot go much further because a sparse matrix will typically distort a sparse vector. To prevent this, we use a central concept from harmonic analysis known as the *Heisenberg principle* (so named because it is the key idea behind the uncertainty principle): A signal and its spectrum cannot both be concentrated. With this in mind, we precondition the random projection with a Fourier transform (via an FFT) in order to isometrically enlarge the support of any sparse vector. To

prevent the inverse effect, i.e., the sparsification of dense vectors, we randomize the Fourier transform. The resulting FJLT shares the low-distortion characteristics of a random projection but with lower complexity. As stated above, it embeds ℓ_2 into ℓ_p for $p = 1, 2$. The running time of the FJLT is $O(d \log d + \min\{d\varepsilon^{-2} \log n, \varepsilon^{p-4} \log^{p+1} n\})$, which outperforms the $O(d\varepsilon^{-2} \log n)$ complexity of its predecessors. Note the simpler form of $O(d \log d + \varepsilon^{-3} \log^2 n)$ for ℓ_1 and $n = 2^{O(\varepsilon d)}$ (subsumed by 2).

1.2. Approximate nearest neighbor searching. For a fixed metric space (U, d_U) and a finite subset (database) $P \subseteq U$, *ε-approximate nearest neighbor* (ε-ANN) searching (Figure 1) is the problem of preprocessing P so that given a query $x \in U$, a point $p \in P$ satisfying

$$d_U(x, p) \leq (1 + \varepsilon) d_U(x, p') \quad \forall p' \in P$$

can be efficiently returned. In other words, we are interested in a point p further from x by a factor of no more than $(1 + \varepsilon)$ compared with the closest point to x in P. We will be interested in the Euclidean $(\mathbb{R}^d, \ell_2)$ and the Hamming cube $(\{0, 1\}^d, \ell_1 \equiv (\ell_2)^2)$ cases.



Given a point x from a metric space X, return $p \in P$ minimizing $d(x, p)$, or any point p' satisfying $d(x, p') \leq (1 + \varepsilon) d(x, p)$.

FIG. 1. *ε-approximate nearest neighbor searching.*

This problem has received considerable attention lately. There are two good reasons for this: (i) ANN boasts more applications than virtually any other geometric problem [27]; (ii) allowing a small error ε makes it possible to break the curse of dimensionality.

There is abundant literature on (approximate) nearest neighbor searching [5, 6, 7, 9, 12, 13, 14, 18, 24, 29, 23, 27, 32, 33, 42, 43, 37]. The early solutions typically suffered from the curse of dimensionality, but the last decade has witnessed a flurry of new algorithms that "break the curse" (see [27] for a recent survey). A few milestones deserve special mention in the context of this work. Kleinberg [32] gave two algorithms for ANN in ℓ_2^d. The first runs in $O((d \log^2 d)(d + \log n))$ time but requires storage exponential in d. The second runs in $O(n + d \log^3 n)$ time, improving on the trivial $O(nd)$ algorithm, and requires only $O(dn \operatorname{polylog} n)$ space. Both algorithms are based on the key idea of *projection onto random lines*, used in subsequent results as well as in this work.

The first algorithms with query times of $\operatorname{poly}(d, \log n)$ and polynomial storage (for fixed ε) were those of Indyk and Motwani [29] and Kushilevitz, Ostrovsky, and

Rabani [33]. The first reference describes a reduction from ε-ANN to *approximate point location in equal balls* (ε-PLEB) for ℓ_2^d (as well as other metric spaces). The ε-PLEB problem is that of outputting a point $p \in P$ such that $\|x-p\|_2 \leq r(1+\varepsilon)$ if there exists a point for which $\|x-p\|_2 \leq r$, null if all points p satisfy $\|x-p\|_2 > (1+\varepsilon)r$, and either of the two answers otherwise. Based on a new space decomposition (of independent interest), Indyk and Motwani showed how to reduce an ε-ANN query to $O(\log^2 n)$ queries to an ε-PLEB oracle, a reduction later improved to $O(\log(n/\varepsilon))$ by Har-Peled [21]. The PLEB reduction can be thought of as a way of performing a binary search on the (unknown) distance to the nearest neighbor while overcoming the possible unboundedness of the search space (we overcome this problem by using a different, simpler technique). One approach to PLEB is to use LSH (locality-sensitive hashing [29]), which requires $O(n^{1/(1+\varepsilon)})$ query time and near-quadratic (for small ε) storage. Another approach is to use dimension-reduction techniques: using the methods of [21, 29, 25], this provides a query time of $O(\varepsilon^{-2} d \log n)$ with $n^{O(\varepsilon^{-2})}$ storage. We mention here that the dimension reduction overwhelms the running time of the algorithm: to remedy this was, in fact, another point of departure for our work on FJLT. Kushilevitz, Ostrovsky, and Rabani [33] described an ingenious (but intricate) reduction from ℓ_2^d to the Hamming cube, which results in $O(\varepsilon^{-2} d^2 \operatorname{polylog} n)$ query time and polynomial storage (again, for fixed ε).

ANN searching over the Hamming cube does not suffer from the "unbounded binary search" problem. Kushilevitz, Ostrovsky, and Rabani [33] gave a random-projection-based algorithm with a query time of $O((d \log d)\varepsilon^{-2} \log n)$—an extra $\log\log d$ factor can be shaved off [11]. We improve the running time of their algorithm to $O((d+\varepsilon^{-2}\log n)\log d)$, which is the best to date (using polynomial storage[2]). Again, we show how to optimize the dimension-reduction step in their algorithm, but this time over GF (2).

We present two new, faster ANN algorithms. Note that both of them contain additional improvements, independent of FJLT.

- One works for the d-dimensional Euclidean space. It stores n points in $\mathbb{R}^d$ and answers any ε-ANN query in $O(d \log d + \varepsilon^{-3} \log^2 n)$ time while using $n^{O(\varepsilon^{-2})}$ storage. The solution is faster than its predecessors (at least for subexponential n) and considerably simpler. This algorithm uses the FJLT and is presented in section 3.
- The other works for the d-dimensional Hamming cube. It stores n points in the d-dimensional Hamming cube and answers any ε-ANN query in $O((d + \varepsilon^{-2}\log n)\log d)$ time, while using $d^2 n^{O(\varepsilon^{-2})}$ storage. This improves on the best previous query time of $O((d \log d)\varepsilon^{-2} \log n)$ [33]. This algorithm does not use the FJLT and can be found in section 4. It can be read independently of the preceding sections.

2. The fast Johnson–Lindenstrauss transform. The FJLT is a random distribution of linear mappings from $\mathbb{R}^d$ to $\mathbb{R}^k$, where the embedding dimension k is set to be $c\varepsilon^{-2}\log n$ for some large enough constant $c = c(p)$. Recall that $p \in \{1, 2\}$ refers to the type of embedding we seek: $\ell_2^d \mapsto \ell_p^k$.

We may assume without loss of generality (w.l.o.g.) that $d = 2^m > k$. We will also assume that $n \geq d$ and $d = \Omega(\varepsilon^{-1/2})$ (otherwise the dimension of the reduced space is linear in the original dimension).

A random embedding $\Phi \sim \mathrm{FJLT}(n, d, \varepsilon, p)$ can be obtained as a product of three

[2] Assuming constant ε.

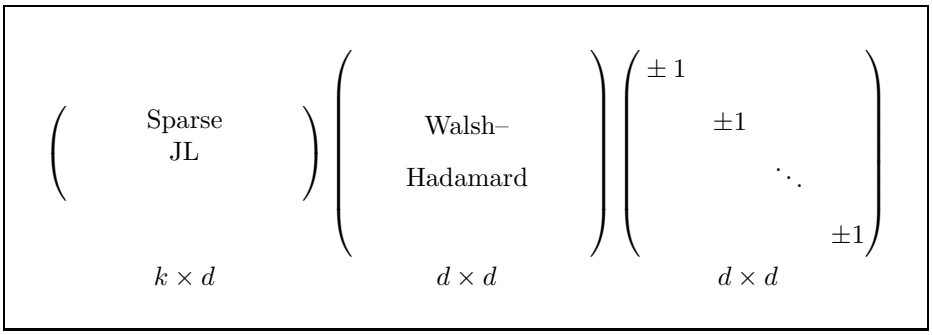


FIG. 2. *The FJLT transform.*

real-valued matrices (Figure 2): $\Phi = PHD$. The matrices P and D are random and H is deterministic:

- $\boldsymbol{P}$ is a k-by-d matrix whose elements are independently distributed as follows. With probability $1-q$ set P_{ij} as 0, and otherwise (with the remaining probability q) draw P_{ij} from a normal distribution of expectation 0 and variance q^{-1}. The sparsity constant q is given as
$$q = \min\left\{\Theta\left(\frac{\varepsilon^{p-2}\log^p n}{d}\right), 1\right\}.$$
- $\boldsymbol{H}$ is a d-by-d normalized Walsh–Hadamard matrix:
$$H_{ij} = d^{-1/2}(-1)^{\langle i-1, j-1\rangle},$$
where $\langle i, j\rangle$ is the dot-product (modulo 2) of the m-bit vectors i, j expressed in binary.
- $\boldsymbol{D}$ is a d-by-d diagonal matrix, where each D_{ii} is drawn independently from $\{-1, 1\}$ with probability $1/2$.

The Walsh–Hadamard matrix encodes the discrete Fourier transform over the additive group GF $(2)^d$: its FFT is particularly simple and requires $O(d \log d)$ time. It follows that, with high probability (which we make precise in Lemma 1), the mapping Φx of *any* vector $x \in \mathbb{R}^d$ can be computed in time $O(d \log d + qd\varepsilon^{-2} \log n)$ (all running times are expected over the random bits of the algorithm). We now make our statement on the FJLT precise.

Before stating the main lemma on the merits of Φ, we provide some intuition for the construction. The matrix P is sparse in the sense that, on expectation, only a q-fraction of its elements are nonzero. It cannot be used as a fast JL transform by itself because the variance of the estimator $\|Px\|_p$ is too high for certain *bad* inputs x. It turns out that these bad inputs are sparse vectors: Intuitively, the extreme case is when x is a vector consisting of exactly one nonzero coordinate. Only the nonzeros of P which are aligned with this unique coordinate contribute to the estimator $\|Px\|_p$. These nonzeros are too rare to ensure a sufficiently strong measure concentration. However, if x is smooth (in a sense that will be made precise in the proof of Lemma 1), then $\|Px\|_p$ does have good concentration properties. The mapping HD ensures that with overwhelming probability, the vector HDx *is* smooth. Since HD is orthogonal, the Euclidean norm remains invariant. Hence, HD can be used to precondition the input x before applying P.

LEMMA 1 (the FJLT lemma). *Fix a set X of n vectors in $\mathbb{R}^d$, $\varepsilon < 1$, and $p \in \{1, 2\}$. Let $\Phi \sim$ FJLT. With probability at least $2/3$, the following two events occur:*

1. *For all $x \in X$,*

$$(1-\varepsilon)\alpha_p \|x\|_2 \le \|\Phi x\|_p \le (1+\varepsilon)\alpha_p \|x\|_2,$$

where $\alpha_1 = k\sqrt{2\pi^{-1}}$ and $\alpha_2 = k$ (recall that $k = c\varepsilon^{-2}\log n$ for some global c).

2. *The mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^k$ requires*

$$O\left(d \log d + \min\{d\varepsilon^{-2}\log n, \varepsilon^{p-4}\log^{p+1} n\}\right)$$

operations.

Remark. By repeating the construction $O(\log(1/\delta))$ times we can amplify the success probability to $1-\delta$ for any $\delta > 0$. If we know X, it is possible to test for success. In the ANN example presented in the next section, however, we do not know X. In that application, X is the set of differences between the query point x and the database points $p \in P$:

$$X = \{x - p : \ p \in P\}.$$

Although P is known, the query point x is information we obtain online after the preprocessing. However, one can amplify the probability of success of ANN by repeating the construction $O(\log(1/\delta))$ times, running the nearest neighbor algorithm w.r.t. each construction and taking the nearest among the outputs.

Proof. W.l.o.g., we can assume that $\varepsilon < \varepsilon_0$ for some suitably small ε_0. Fix some $x \in X$, and define the random variable $u = HDx = (u_1, \ldots, u_d)^T$. Assume w.l.o.g. that $\|x\|_2 = 1$. Note that u_1 is of the form $\sum_{i=1}^d a_i x_i$, where each $a_i = \pm d^{-1/2}$ is chosen independently and uniformly. A Chernoff-type argument shows that, with probability at least $1 - 1/20$,

$$\max_{x \in X} \|HDx\|_\infty = O(d^{-1/2}\sqrt{\log n}). \tag{4}$$

Indeed,

$$\mathbf{E}\left[e^{tdu_1}\right] = \prod_i \mathbf{E}\left[e^{tda_i x_i}\right] = \prod_i \cosh(t\sqrt{d}\, x_i) \le e^{t^2 d\|x\|_2^2/2}. \tag{5}$$

Hence, for any $s > 0$, by Markov's inequality (plugging $t = sd$ into (5)),

$$\begin{aligned}
\mathbf{Pr}[|u_1| \ge s] &= 2\mathbf{Pr}[e^{sdu_1} \ge e^{s^2 d}] \\
&\le 2\mathbf{E}\left[e^{sdu_1}\right]/e^{s^2 d} \\
&\le 2e^{s^2 d\|x\|_2^2/2 - s^2 d} \\
&= 2e^{-s^2 d/2} \le 1/(20nd)
\end{aligned}$$

for $s = \Theta(d^{-1/2}\sqrt{\log n})$, from which (4) follows by a union bound over all $nd \le n^2$ coordinates of the vectors $\{HDx : \ x \in X\}$.

Assume from now on that (4) holds, i.e., $\|u\|_\infty \le s$ (where $u = HDx$ for any $x \in X$, which we keep fixed). It is convenient (and harmless) to assume that $m \stackrel{\text{def}}{=} s^{-2}$ is integral.

Note that $\|u\|_2 = \|x\|_2$ because both H and D are isometries. Define

$$y = (y_1, \ldots, y_k)^T = Pu = \Phi x.$$

By the definition of the FJLT, y_1 is obtained as follows: Pick random i.i.d. indicator variables $b_1, \ldots, b_d$, where each b_j equals 1 with probability q, and random i.i.d. variables $r_1, \ldots, r_d$ distributed $N(0, q^{-1})$. Then define y_1 as

$$y_1 = \sum_{j=1}^{d} r_j b_j u_j.$$

We also define the r.v. $Z = \sum_{j=1}^{d} b_j u_j^2$. It is well known (by the 2-stability of the normal distribution) that for any d real numbers $\alpha_1, \ldots, \alpha_d$, $\sum_{j=1}^{d} r_j \alpha_j \sim N(0, \sum \alpha_j^2)$. Hence,

$$(y_1 | Z = z) \sim N(0, q^{-1} z).$$

Note that all of $y_1, \ldots, y_k$ are i.i.d. (given u), and we can similarly define corresponding random i.i.d. variables $Z_1 (= Z), Z_2, \ldots, Z_k$. The expectation of these random variables is

$$\mathbf{E}[Z] = \sum_{j=1}^{d} u_j^2 \mathbf{E}[b_j] = q. \tag{6}$$

Let u^2 formally denote $(u_1^2, \ldots, u_d^2) \in (\mathbb{R}^+)^d$. By our assumption, u^2 lives in the following d-dimensional polytope:

$$\mathcal{P} = \left\{ (a_1, \ldots, a_d) : 0 \le a_j \le 1/m \, \forall j \text{ and } \sum_{j=1}^{d} a_j = 1 \right\}.$$

Let $u^* \in \mathbb{R}^d$ denote a vector such that u^{*2} is a vertex of $\mathcal{P}$. By symmetry, there will be no loss of generality in what follows if we fix

$$u^* = (\underbrace{m^{-1/2}, \ldots, m^{-1/2}}_{m}, \underbrace{0, \ldots, 0}_{d-m}).$$

The point u^* will be convenient for identifying extremal cases in the analysis of Z. We will use Z^* to denote the random variable Z corresponding to the case $u = u^*$. Immediately we identify that $Z^* \sim B(m, q)/m$ (in other words, the binomial distribution with parameters m, q divided by m). Consequently,

$$\operatorname{var}(Z^*) = q(1-q)/m. \tag{7}$$

The ℓ_1 case. We choose

$$q = \min\{1/(\varepsilon m), 1\} = \min\left\{ \Theta\left(\frac{\varepsilon^{-1} \log n}{d} \right), 1 \right\}.$$

We now bound certain moments of Z (over the random b_i's).

LEMMA 2. *For any* $t > 1$, $\mathbf{E}[Z^t] = O(qt)^t$ *and*

$$(1-\varepsilon)\sqrt{q} \le \mathbf{E}[\sqrt{Z}] \le \sqrt{q}.$$

Proof. For the case $q = 1$ the claim is trivial because then Z is constant 1. So we assume $q = 1/(\varepsilon m) < 1$.

It is easy to verify that $\mathbf{E}[Z^t]$ is a convex function of u^2 and hence achieves its maximum at a vertex of $\mathcal{P}$, namely, when $u = v$. So it suffices to prove the moment upper bounds for Z^*, which conveniently behave like a (scaled) binomial. By standard bounds on the binomial moments,

$$\mathbf{E}[Z^{*t}] = O(m^{-t}(mqt)^t) = O(qt)^t,$$

proving the first part of the lemma.

By Jensen's inequality and (6),

$$\mathbf{E}[\sqrt{Z}] \le \sqrt{\mathbf{E}[Z]} = \sqrt{q}.$$

This proves the upper-bound side of the second part of the lemma. To prove the lower-bound side, we notice that $\mathbf{E}[\sqrt{Z}]$ is a concave function of u^2, and hence achieves its minimum when $u = v$. So it suffices to prove the desired lower bound for $\mathbf{E}[\sqrt{Z^*}]$. Since $\sqrt{x} \ge 1 + \frac{1}{2}(x-1) - (x-1)^2$ for all $x \ge 0$,

$$\begin{aligned}\mathbf{E}[\sqrt{Z^*}] &= \sqrt{q}\,\mathbf{E}[\sqrt{Z^*/q}] \\ &\ge \sqrt{q}\left(1 + \frac{1}{2}\mathbf{E}[Z^*/q - 1] - \mathbf{E}[(Z^*/q-1)^2]\right).\end{aligned} \tag{8}$$

By (6) $\mathbf{E}[Z^*/q - 1] = 0$ and using (7),

$$\begin{aligned}\mathbf{E}[(Z^*/q-1)^2] &= \operatorname{var}(Z^*/q) = (1-q)/(qm) \\ &\le 1/(qm) = \varepsilon.\end{aligned}$$

Plugging this into (8) gives $\mathbf{E}[\sqrt{Z^*}] \ge \sqrt{q}\,(1-\varepsilon)$, as required. ∎

Since the expectation of the absolute value of $N(0,1)$ is $\sqrt{2\pi^{-1}}$, by taking conditional expectations

$$\mathbf{E}[|y_1|] = \sqrt{2/q\pi}\,\mathbf{E}[\sqrt{Z}],$$

and by Lemma 2,

$$(1-\varepsilon)\sqrt{2\pi^{-1}} \le \mathbf{E}[|y_1|] \le \sqrt{2\pi^{-1}}. \tag{9}$$

We now show that $\|y\|_1$ is sharply concentrated around its mean $\mathbf{E}[\|y\|_1] = k\mathbf{E}[|y_1|]$. To do this, first we bound the moments of $|y_1| = |\sum_j b_j r_j u_j|$. For any integer $t \ge 0$, by taking conditional expectation,

$$\mathbf{E}[|y_1|^t] = \mathbf{E}[(q^{-1}Z)^{t/2}]\,\mathbf{E}[|U|^t],$$

where $U \sim N(0,1)$. It is well known that $\mathbf{E}[|U|^t] = O(t)^{t/2}$; therefore, by Lemma 2,

$$\mathbf{E}[|y_1|^t] = O(t)^t.$$

It follows that the moment generating function

$$\mathbf{E}\left[e^{\lambda|y_1|}\right] = 1 + \lambda\mathbf{E}\left[|y_1|\right] + \sum_{t>1}\mathbf{E}\left[|y_1|^t\right]\lambda^t/t!$$

$$\leq 1 + \lambda\mathbf{E}\left[|y_1|\right] + \sum_{t>1}O(t)^t\lambda^t/t!$$

converges for any $0 \leq \lambda < \lambda_0$, where λ_0 is an absolute constant, and

$$\mathbf{E}\left[e^{\lambda|y_1|}\right] = 1 + \lambda\mathbf{E}\left[|y_1|\right] + O(\lambda^2) = e^{\lambda\mathbf{E}\left[|y_1|\right]+O(\lambda^2)}.$$

By independence,

$$\mathbf{E}\left[e^{\lambda\|y\|_1}\right] = \left(\mathbf{E}\left[e^{\lambda|y_1|}\right]\right)^k = e^{\lambda\mathbf{E}\left[\|y\|_1\right]+O(\lambda^2 k)}.$$

By Markov's inequality and (9),

$$\mathbf{Pr}[\|y\|_1 \geq (1+\varepsilon)\mathbf{E}\left[\|y\|_1\right] \leq \mathbf{E}\left[e^{\lambda\|y\|_1}\right]/e^{\lambda(1+\varepsilon)\mathbf{E}\left[\|y\|_1\right]}]$$

$$\leq e^{-\lambda\varepsilon\mathbf{E}\left[\|y\|_1\right]+O(\lambda^2 k)}$$

$$\leq e^{-\Omega(\varepsilon^2 k)}$$

for some $\lambda = \Theta(\varepsilon)$. The constraint $\lambda < \lambda_0$ entails that ε be smaller than the same absolute constant. A similar argument leads to a similar lower tail estimate. Our choice of k ensures that, for any $x \in X$, $\|\Phi x\|_1 = \|y\|_1$ deviates from its mean by at most ε with probability at least $1 - 1/20$. By (9), this mean, $k\mathbf{E}\left[|y_1|\right]$, is itself concentrated (up to a relative error of ε) around $\alpha_1 = k\sqrt{2\pi^{-1}}$; rescaling ε by a constant factor and ensuring (4) proves the ℓ_1 claim of the first part of the FJLT lemma.

The ℓ_2 case. We now choose

$$q = \min\left\{\frac{c_1\log^2 n}{d}, 1\right\}$$

for a large enough constant c_1.

LEMMA 3. *With probability at least* $1 - 1/20n$,

1. $q/2 \leq Z_i \leq 2q$ *for all* $i = 1, \ldots, k$, *and*
2. $kq(1-\varepsilon) \leq \sum_{i=1}^k Z_i \leq kq(1+\varepsilon)$.

Proof. If $q = 1$, then Z is the constant q and the claim is trivial. Otherwise, $q = c_1 d^{-1}\log^2 n < 1$. For any real λ, the function

$$f_\lambda(u_1^2, \ldots, u_d^2) = \mathbf{E}\left[e^{\lambda Z}\right]$$

is convex. Therefore, it achieves its maximum on the vertices of the polytope $\mathcal{P}$ (as in the proof of Lemma 2). Hence, as argued before, $\mathbf{E}\left[e^{\lambda Z}\right] \leq \mathbf{E}\left[e^{\lambda Z^*}\right]$. We conclude the proof of the first part by using standard tail estimates on the scaled binomial Z^*, derived from bounds on its moment generating function $\mathbf{E}\left[e^{\lambda Z^*}\right]$ (e.g., [4]), and union bounding.

For the second part, let $S = \sum_{i=1}^k Z_i$. Again, the moment generating function of S is bounded above by that of $S^* \sim B(mk, q)/m$ (all Z_i's are distributed as Z^*), for which it is immediate to check the required concentration bound. ∎

Henceforth we will assume that the premise of Lemma 3 holds with respect to all choices of $x \in X$. Using the union bound, this happens with probability at least $1 - 1/20$. For each $i = 1, \ldots, k$ the random variable $y_i^2 q / Z_i$ is distributed χ^2 with 1 degree of freedom. It follows that, conditioned on Z_i, $\mathbf{E}[y_i^2] = Z_i/q$ and the moment generating function of y_i^2 is

$$\mathbf{E}[e^{\lambda y_i^2}] = (1 - 2\lambda Z_i/q)^{-1/2}.$$

Given any $\lambda > 0$ less than some fixed λ_0, and for large enough ξ, the moment generating function converges and equals

$$\mathbf{E}[e^{\lambda y_i^2}] \le e^{\lambda Z_i/q + \xi \lambda^2 (Z_i/q)^2}$$

(we used the fact that $Z_i/q = O(1)$ from the first part of Lemma 3). Therefore, by independence,

$$\mathbf{E}[e^{\lambda \sum_{i=1}^k y_i^2}] \le e^{\lambda \sum_{i=1}^k (Z_i/q + \xi \lambda^2 \sum_{i=1}^k (Z_i/q)^2)},$$

and hence

$$\begin{aligned}
&\mathbf{Pr}\left[\sum_{i=1}^k y_i^2 > (1+\varepsilon)\sum_{i=1}^k Z_i/q\right] \\
&\qquad = \mathbf{Pr}\left[e^{\lambda}\sum_{i=1}^k y_i^2 > e^{(1+\varepsilon)\lambda}\sum_{i=1}^k Z_i/q\right] \\
&\qquad \le \mathbf{E}\left[e^{\lambda}\sum_{i=1}^k y_i^2\right] / e^{(1+\varepsilon)\lambda}\sum_{i=1}^k Z_i/q \\
&\qquad \le e^{-\varepsilon\lambda}\sum_{i=1}^k Z_i/q + \xi\lambda^2 \sum_{i=1}^k (Z_i/q)^2.
\end{aligned} \tag{10}$$

If we plug

$$\lambda = \frac{\varepsilon \sum_{i=1}^k (Z_i/q)}{2\xi \sum_{i=1}^k (Z_i/q)^2}$$

into (10) and assume that ε is smaller than some global ε_0, we avoid convergence problems (we used the assumption $\sum_{i=1}^k (Z_i/q) / \sum_{i=1}^k (Z_i/q)^2 \le 5$ following the premise of Lemma 3). We now conclude that

$$\mathbf{Pr}\left[\sum_{i=1}^k y_i^2 > (1+\varepsilon)k\right] \le e^{-\Omega(\varepsilon^2 k)}.$$

A similar technique can be used to bound the left tail estimate. By choosing $k = c\varepsilon^{-2} \log n$ for some large enough c, using the union bound, and possibly rescaling ε, we conclude the ℓ_2 case of the first part of the FJLT lemma.

Running time. Computing Dx takes $O(d)$ time, because D is a diagonal matrix. Computing $H(Dx)$ takes $O(d \log d)$ time using the Walsh–Hadamard transform. Finally, computing $P(HDx)$ takes $O(|P|)$ time, where $|P|$ is the number of nonzeros in P. This number is distributed $B(nk, q)$. It is now immediate to verify that

$$\mathbf{E}[|P|] = O(\varepsilon^{p-4} \log^{p+1} n).$$

Using a Markov bound, we conclude the proof for the running time guarantee of the FJLT lemma.

This concludes the proof of the FJLT lemma, up to possible rescaling of ε. ∎

3. ANN searching in Euclidean space. We give a new ANN algorithm for ℓ_2^d that differs (and improves upon) its predecessors in its use of *handles*: a bootstrapping device that allows for fast binary searching on the distance to the nearest neighbor. Plugging in the FJLT automatically provides additional improvement. Let P be a set of n points in $\mathbb{R}^d$. Given $x \in \mathbb{R}^d$, let $p_{\min}$ be its nearest neighbor in P. Recall that the answer to an ε-ANN for x is any point $p \in P$ such that

$$\|x - p\|_2 \leq (1+\varepsilon)\|x - p_{\min}\|_2.$$

The algorithm has two stages. First (part I), we compute an answer q to an $O(n)$-ANN query for x. This opens the way for a second stage, where we answer the ε-ANN query for x within the (smaller) set $P_x = P \cap \mathcal{B}_2(q, 2\|x-q\|_2)$, where $\mathcal{B}_2(q,r)$ denotes the ℓ_2-ball of radius r centered at q. The key property of P_x is that it contains $p_{\min}$ and the distance from x to its furthest neighbor in P_x is only $O(n\|x - p_{\min}\|_2)$. This sets the stage for a binary search over a bounded domain (part II). Instead of reducing the problem to an ANN query over the Hamming cube (as in [33]), we embed P directly into a low-dimensional ℓ_1-space, which we then discretize. We get a two-fold benefit from our use of handles and of the FJLT. The entire scheme is outlined as pseudocode in Figure 3.

3.1. Part I: Linear-factor approximation. To find an $O(n)$-ANN for query x, we choose a random direction $v \in \mathbb{R}^d$ and return the *exact* nearest neighbor with respect to the pseudometric D_v, defined as

$$D_v(x,p) = |v^T x - v^T p|.$$

This can be done in $O(d + \log n)$ time by preprocessing all the $(v^T p)$'s (Figure 4). As we shall see in Lemma 4, this returns an $O(n)$-ANN of x (with respect to the full d-dimensional Euclidean space) with constant probability. By repeating the procedure $O(\log \delta^{-1})$ times (with independently drawn vectors v) and keeping the output point nearest to x, we increase the probability of success to the $O(n)$-ANN query to $1-\delta$ for any arbitrarily small $\delta > 0$.

Let $p_{\min}$ denote the exact nearest neighbor of x in the d-dimensional Euclidean space, and let $p^v_{\min}$ denote the (random) nearest neighbor of x with respect to D_v.

LEMMA 4.

$$\mathbf{E}\left[\|x - p^v_{\min}\|_2\right] = O(n\|x - p_{\min}\|_2).$$

Proof. Let $\chi(p)$ be the indicator variable of the event $D_v(x,p) \leq D_v(x, p_{\min})$. Elementary trigonometry (Figure 5) shows that

$$\mathbf{E}\left[\chi(p)\right] = O(\|x - p_{\min}\|_2 / \|x - p\|_2).$$

Clearly, $\|x - p^v_{\min}\|_2 \leq \sum_{p \in P} \chi(p)\|x - p\|_2$; therefore, by linearity of expectation,

$$\mathbf{E}\left[\|x - p^v_{\min}\|_2\right] \leq \sum_{p \in P} \|x - p\|_2 \,\mathbf{E}\left[\chi(p)\right] = O(n\|x - p_{\min}\|_2). \qquad \blacksquare$$

```
ε-ANN

  PREPROCESS(P)
    preprocess for O(n)-ANN
    set Φ ← random (ℓ2 → ℓ1) FJLT matrix
    precompute Φp for all p ∈ P
    for all O(n^2) possible P_x
      for all possible handles p ∈ P_x
        for all possible O(log(n/ε)) binary search radii l
          precompute random discretization T : R^k → Z^k
          precompute table S : Z^k → P_x ∪ {exception}

  QUERY(x)
    set q ← O(n)-ANN of x
    set P_x ← {p ∈ P : ||p − q|| ≤ 2||x − q||}  (O(log n) time using preprocessing)
    compute reduction Φx

    set l ← R(P_x)  (radius)
    set p ← q  (handle)
    for t = 1, 2, ..., t_max(= O(log(n/ε)))  (binary search loop)
      identify precomputed S, T corresponding to P_x, p, l
      set p' ← S[T(Φx)]
      if p' = exception
        break for-loop  (with possible warning)
      else if D_T(x, p') > (1 + 1/2)k  (failure)
        set l ← l + 2^{−t}
      else  (success)
        set l ← l − 2^{−t}
        set p ← p'

    return p
```

At iteration t in the binary search loop. Variables p, l in the program correspond to p_t, l_t in section 3.2.

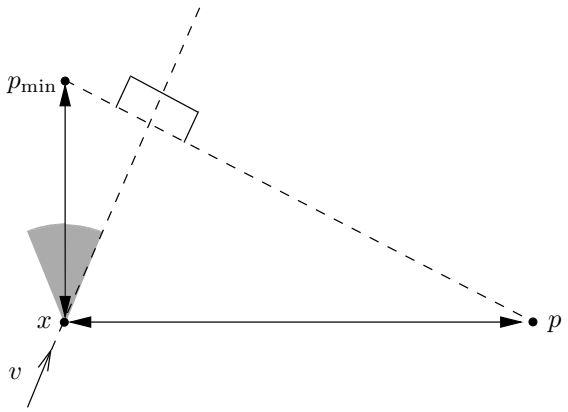
FIG. 3. *Pseudocode for* ε-ANN *in Euclidean space.*

```
O(n)-ANN

  PREPROCESS(P)
    set v ← random direction in R^d
    precompute v^T p for all p ∈ P
    build binary search tree for exact 1-dimensional nearest neighbor
      with respect to D_v(x, p) = |v^T x − v^T p|

  QUERY(x)
    return exact nearest neighbor of x with respect to D_v
```

FIG. 4. *Pseudocode for* $O(n)$-ANN *in Euclidean space.*



E $[\chi(p)]$ is maximized when $x-p$ is orthogonal to $x-p_{\min}$. In this case, if the random direction v is in the gray area, $\chi(p)=1$. This event happens with probability $\frac{2}{\pi}\arctan(\|x-p_{\min}\|_2/\|x-p\|_2) = O(\|x-p_{\min}\|_2/\|x-p\|_2)$.

FIG. 5. *Why the $O(n)$-ANN algorithm works.*

3.2. Part II: Binary search with handles. Assume that the previous step succeeds in returning an $O(n)$-ANN q for x. From now on, we can confine our search within the working set P_x. Note that there are only $O(n^2)$ distinct sets P_x and, given x, P_x can be found by binary search in $O(d+\log n)$ time. If the diameter of P_x is small enough, say $\Delta(P_x) \le \frac{1}{2}\varepsilon\|x-q\|_2$, then q is a satisfactory answer to the ε-ANN query for x. By precomputing all diameters, we can test this in constant time and be done if the outcome is positive. So, assume now that $\Delta(P_x) > \frac{1}{2}\varepsilon\|x-q\|_2$. The distance from x to any point of P_x lies in the interval $I(P_x) = [L(P_x), R(P_x)]$, where

$$\begin{aligned} L(P_x) &= \Omega(\Delta(P_x)/n), \\ R(P_x) &= O(\varepsilon^{-1}\Delta(P_x)). \end{aligned} \tag{11}$$

Each step t in the binary search is associated with two items, l_t and p_t:

1. A *search radius* $l_t \in I$. With high probability, the tth step in the binary search finds out whether there is a point in P_x at distance at most $(1+\varepsilon)l_t$ to x (*success*) or whether all points are at distance at least l_t to x (*failure*). For initialization, we set l_1 to the high endpoint $R(P_x)$ of the search interval. For $t>1$, we follow the standard binary search scheme by updating
$$l_t = \begin{cases} l_{t-1} - 2^{-t+1}l_1 & \textit{success}, \\ l_{t-1} + 2^{-t+1}l_1 & \textit{failure}. \end{cases}$$
2. A *handle* $p_t \in P_x$ such that $\|x-p_t\|_2 \le 2l_t$. We set $p_1 = q$. In case of *success* in step $t-1$, p_t is updated as the witness point of distance at most $(1+\varepsilon)l_1$ to x. In case of *failure* there is no update: $p_t = p_{t-1}$.

The number $t_{\max}$ of binary search steps is designed to be the smallest integer a such that the search interval size of $\Theta(R(P_x)2^{-a})$ is sensitive enough to detect a relative error of ε with respect to $L(P_x)$. More precisely, we need

$$R(P_x)2^{-t_{\max}} = O(\varepsilon L(P_x)),$$

which solves to $t_{\max} = O(\log(n/\varepsilon))$.

Determining whether the step is a *success* (together with a witness) or a *failure* will be done in a transformed space. The transformation will be a composition of two steps. The first step will be a random normalized ($\ell_2 \to \ell_1$) FJLT matrix Φ applied to the points in P and to x. By the FJLT lemma, with high probability, for any point $p \in P$,

$$(1-\varepsilon)\|x-p\|_2 \le \|\Phi x - \Phi p\|_1 \le (1+\varepsilon)\|x-p\|_2.$$

As usual, $k = O(\varepsilon^{-2}\log n)$ denotes the embedding dimension.

The second step will be a random discretization of $\mathbb{R}^k$ that will allow table lookup. The discretization could be done be selecting a randomly shifted regular grid in each of the k dimensions (with carefully chosen cell size). We choose a slightly different discretization that makes the analysis easier.

3.3. Poisson discretization. At step t, for $i = 1, \ldots, k$, consider a random two-sided infinite Poisson process Ψ_i with rate k/l_t. This implies that the number of random points in every interval $[a, a+\tau)$ obeys a Poisson distribution with expectation $\tau k/l_t$. Given $u \in \mathbb{R}^k$, define the quantization $T(u)$ of u to be the k-dimensional integer vector, whose ith coordinate is the signed count of Poisson events between 0 and u_i, i.e.,

$$T(u)_i = \begin{cases} |\Psi_i \cap [0, u_i)| & \text{if } u_i \ge 0, \\ -|\Psi_i \cap [u_i, 0)| & \text{otherwise.} \end{cases}$$

We use T to define a pseudometric D_T over $\mathbb{R}^d$,

$$D_T(x,y) = \|T(\Phi x) - T(\Phi y)\|_1.$$

By well-known properties of Poisson random variables, for fixed $x, y \in \mathbb{R}^d$, $D_T(x,y)$ is a Poisson variable with rate $k\|\Phi(x-y)\|_1/l_t$.[3] The point process might fail its purpose, so we say that $p \in P_x$ is *reliable at step* t if either

1. $\|x-p\|_2 \le l_t$ and $D_T(x,p) \le (1+\frac{1}{2}\varepsilon)k$, or
2. $\|x-p\|_2 > l_t$ and $\frac{1}{(1+\frac{1}{2}\varepsilon)}k\|x-p\|_2/l_t \le D_T(x,p) \le (1+\frac{1}{2}\varepsilon)k\|x-p\|_2/l_t$.

If all points in P_x are reliable with respect to x and l_t, we say that step t is reliable. By our choice of k, concentration bounds [4] for the Poisson distribution show that all steps in the binary search are reliable.

3.4. A pruned data structure. Step reliability ensures that all the required information for the binary search is contained in the vector $T(\Phi x)$ (up to possible rescaling of ε), and so we can use that discrete vector to index a lookup table $S : \mathbb{Z}^k \to P_x$. The entry $S[v]$ stores a point $p \in P_x$ such that $T(\Phi(p))$ is the nearest ℓ_1-neighbor of v among all the transformed points $\{\, T(\Phi p') : p' \in P_x \,\}$. In particular, $S[T(\Phi x)]$ is the exact nearest neighbor of x in P_x with respect to D_T. For simplicity we defined the lookup table to require infinite memory. We will show how to prune the data structure shortly.

The idea is to navigate through the binary search process at the current step based on $S[T(\Phi(x))] \in P_x$. Let p' denote this point. If $D_T(x,p') \le (1+\frac{1}{2}\varepsilon)k$, then

[3] We could have replaced the Poisson process by using a randomly shifted regular grid on the real line, but that would have made the analysis more difficult because the distribution of the number of grid points in an interval is not closed under addition.

we conclude that $\|x - p'\|_2 \le l_t(1+\varepsilon)$, and the step is a *success* with witness p'. If $D_T(x,p') > (1+\frac{1}{2}\varepsilon)k$, then we conclude that all points $p \in P_x$ satisfy $\|x-p\|_2 > l_t$, and the step is a *failure*.

We now show how to prune S. We use the reliability of step t and the invariant of the binary search handle p_t to assert that

$$D_T(x,p_t) \le 2k(1+2\varepsilon); \tag{12}$$

therefore, only the points x in the D_T-metric ball specified by (12) need to be stored (say, in a pruned k-way tree). For fixed p_t and l_t, the number of vectors $T(\Phi x)$ that satisfy (12) is exactly the number of integral points $(n_1,\dots,n_k) \in \mathbb{Z}^k$ such that

$$|n_1| + \cdots + |n_k| \le 2k(1+2\varepsilon).$$

This puts the storage requirement at $\binom{3k}{k-1}2^{O(k)} = 2^{O(k)} = n^{O(\varepsilon^{-2})}$. If we implement the table as a weight-balanced radix tree, the lookup time is only $O(k)$. The complete binary search takes $O(k\log(n/\varepsilon)) = O(\varepsilon^{-2}\log^2(n/\varepsilon))$ time. Obviously, we may assume $\varepsilon > n^{-O(1)}$ (otherwise the naive algorithm is faster), so the binary search time is $O(\varepsilon^{-2}\log^2 n)$.

In preprocessing, we compute discretization maps T and tables S for all possible working sets P_x, search radii l, and handles $p \in P_x$. In case (12) does not hold, $S[T(\Phi x)]$ will be defined to return the value *exception* (and the search fails).

THEOREM 1. *Given a set P of n points in ℓ_2^d, for any $\varepsilon > 0$, there is a randomized data structure of size $n^{O(\varepsilon^{-2})}$ that can answer any ε-*ANN *query in time $O(d\log d + \varepsilon^{-3}\log^2 n)$ with high probability (over the preprocessing).*

4. ANN searching over the Hamming cube. Kushilevitz, Ostrovsky, and Rabani [33] gave an algorithm for ANN queries over $\{0,1\}^d$. Its bottleneck is the repeated multiplication of the query point by various random matrices. Our improvement is based on the observation that, although most of these matrices are dense, by using some algebra over GF (2), one can decompose them into a sparse part together with a dense part that is of low complexity.

As usual, $P \subseteq \{0,1\}^d$ will denote the fixed database of n points in the d-dimensional Hamming cube. Given a query $x \in \{0,1\}^d$, $p_{\min} \in P$ is the exact nearest neighbor. We let $\mathcal{B}_1(z,r)$ denote the Hamming ball of radius r (inclusive) around z.

The ANN data structure of [33] supports binary search on the unknown distance $D(x,p_{\min}) = \|x-p_{\min}\|_1$, using d separate preprocessed substructures, $\mathcal{S}_l$ $(1 \le l \le d)$. Each of these structures is meant to handle queries whose targeted nearest neighbors are at the distance l. To supply enough randomness so that *every* query succeeds with high probability (over preprocessing), each $\mathcal{S}_l$ itself is a collection of σ similarly built data structures $\mathcal{S}_{l,j}$. For any $j = 1,\dots,\sigma$, $\mathcal{S}_{l,j}$ consists of the following:

- a random k-by-d matrix $R^{l,j}$ whose elements are chosen independently in $\{0,1\}$, with the probability of a 1 being $1/2l$;
- a table $T_{l,j} : \{0,1\}^k \to P$, initialized as follows:
  - Set all entries to ∞; then,

– for each $p \in P$ in turn, set $T_{l,j}[\mathcal{B}_1(z, k(\mu(l) + \frac{1}{3}\varepsilon'))]$ to p, where

$$z \stackrel{\text{def}}{=} R^{l,j}p \pmod 2,$$

$$\mu(l) \stackrel{\text{def}}{=} \frac{1}{2}\left(1 - \left(1 - \frac{1}{2l}\right)^l\right),$$

$$\varepsilon' \stackrel{\text{def}}{=} \Theta(1 - e^{-\varepsilon/2}).$$

LEMMA 5 (see [33]). *Assume that $n \geq \log d$, and set $\sigma = cd \log d$ and $k = c\varepsilon^{-2} \log n$ for a large enough constant c. With constant probability, the following statement holds true simultaneously for all queries $x \in \{0,1\}^d$ and all $1 \leq l \leq d$: Given a random $j \in \{1, \ldots, \sigma\}$, with probability $1 - O(1/\log d)$:* (i) *The point $T_{l,j}[R^{l,j}x]$, if finite, is at distance at most $(1+\varepsilon)l$ from x, and* (ii) *if x's nearest neighbor is at distance at most l, then the point in question, indeed, is finite.*

For a random j, we say that a query point x *passes the l-test* if the point $T_{l,j}[R^{l,j}x]$ is finite. (Note that passing is not an intrinsic property of x but a random variable.) The test is called *reliable* if both (i) and (ii) in the lemma hold. Assuming reliability, failure of the test means that x's nearest neighbor lies at distance greater than l, while success yields a neighbor of x at distance at most $(1+\varepsilon)l$.

This immediately suggests [33] an ANN algorithm. Beginning with $l = \lceil d/2 \rceil$, run an l-test on x and repeat for $l/2$ if it passes and $3l/2$ if it fails; then, proceed in standard binary search fashion. Suppose for a moment that all the l-tests are reliable. Then, the binary search terminates with the discovery of an index l and a point $p \in P$ that is at most $(1+\varepsilon)l$ away from x, together with the certainty that the distance from x to its nearest neighbor exceeds l. Obviously, the point p is an acceptable answer to the ε-ANN query.

Using a union bound on the number of steps required in the binary search, we can count on the reliability of every test used in the binary search. In [33] it is claimed that the memory and preprocessing time requirements could be improved by a $\Theta(\log d)$ factor if the reliability probability is reduced from $1 - O(1/\log d)$ to a constant bounded away from $1/2$ and 1, using fault-tolerant techniques of [19]. A sufficient requirement for these techniques to work is that, given finite $v = T_{l,j}[R^{l,j}x]$, it is possible to efficiently test whether one of the previous tests was unreliable. This requirement remains true while using our improvements in section 4.1 below.

Note also that we may assume from now on that $n \geq \log d$: Indeed, having fewer than $\log d$ points gives us a naive (exact) algorithm with $O(d \log d)$ query time. The storage is $d^2 n^{O(\varepsilon^{-2})}$ and the query time is $O(d(\log d)\varepsilon^{-2} \log n)$. To improve this time bound, we seek to exploit the sparsity of the random matrices. That alone cuts down the query time to $O(d\varepsilon^{-2} \log n)$ in a trivial manner, the worst case being a query x that itself belongs to P.

4.1. Improvement using linear algebra. Linear algebra allows room for further improvement. For expository purposes, it is convenient to start the binary search with a d-test, so that the first test in the search is always successful. In general, consider the case where an l-step is to be performed, and let l' be the last previous successful test in the search. Note that $l \geq l'/2$. The algorithm is now in possession of a *handle*, namely, a point $p \in P$ at distance at most $(1+\varepsilon)l'$ from x. The cost of the current l-test is that of computing $y = R^{l,j}x$ for a random j.

The main idea is to evaluate y as $R^{l,j}x = R^{l,j}(x+2p) = R^{l,j}(x+p) + R^{l,j}p$ over GF (2). Here is the benefit of this decomposition: The point $x+p$ has at most $(1+\varepsilon)l'$ ones, and obviously only the corresponding columns of $R^{l,j}$ are relevant in computing $R^{l,j}(x+p)$. Assuming that the 1's within each column of $R^{l,j}$ are linked together in a list, the time for computing $R^{l,j}(x+p)$ is proportional to $d+k$ plus the number N of 1's within the relevant columns of $R^{l,j}$. The d comes from identifying the 1's of $x+p$, the k comes from initializing the result vector in $\mathrm{GF}(2)^k$ as zero, before scanning through the linked lists of 1's in the relevant columns. By construction, the expected value of N is at most $k(1+\varepsilon)l'(1/2l) \le 2k$ (over the randomness of the matrix). By precomputing all the points $\{ R^{l,j}q \mid q \in P \}$ in preprocessing (which adds only a factor of n to the storage), we can retrieve $R^{l,j}p$ in $O(k)$ time. In short, we can complete this binary search step in $O(d+k)$ expected time instead of the previous $O(dk)$ bound.

4.2. No query left behind. There is only one problem: The expectation of the query time is defined over the randomness of *both* the query algorithm and the preprocessing. To remove this dependency on the preprocessing, we must ensure that, for *any* query x, the expected running time of any binary search step is $O(d+k)$ over the random choices of the index j during query answering: We call this the *NQLB policy* (for "no query left behind").

It suffices to show that, for any l and any subset $V \subseteq \{1,\dots,d\}$ of column indices, the total number of 1's within all the columns (indexed by V) of all the matrices $R^{l,j}$ $(1 \le j \le \sigma)$ is $O(\sigma k|V|/l)$. This number is a random variable $Y = \sum_{1 \le i \le \sigma k|V|} y_i$, where each y_i is chosen independently in $\{0,1\}$ with a probability $1/2l$ of being 1. A Chernoff bound shows that $Y = O(\sigma k|V|/l)$ with probability at least $1 - 2^{-\Omega(\sigma k|V|/l)}$. Summing over all l, V, we find that the probability of violating the NQLB policy is at most

$$\sum_{l=1}^{d} \sum_{v=1}^{d} \binom{d}{v} 2^{-\Omega(\sigma k v/l)},$$

which is arbitrarily small.

THEOREM 2. *Given a set P of n points in the d-dimensional Hamming cube and any $0 < \varepsilon < 1$, there exists a random data structure of size $d^2 n^{O(\varepsilon^{-2})}$ that can answer any ε-ANN query in time $O((d + \varepsilon^{-2}\log n)\log d)$ in the sense that with high probability over its construction, uniformly for all possible queries x,*

1. *with high probability over the choice of $j \in \{1,\dots,\sigma\}$ a correct ε-ANN is returned* [33]*, and*
2. *the expected running time over the choice of $j \in \{1,\dots,\sigma\}$ is*

$$O((d + \varepsilon^{-2}\log n)\log d).$$

5. Concluding remarks.

Applications and improvements. Ailon and Liberty [2] have recently obtained an improvement to FJLT using tools from error correcting codes and probability in Banach spaces.

The FJLT can potentially improve other proximity-related problems such as closest pair, furthest neighbor, and clustering. Sarlós [40] recently discovered that the FJLT can be used to improve a result by Drineas, Mahoney, and Muthukrishnan [17] on fast approximate ℓ_2-regression.

A natural question is whether one can combine the FJLT with Achlioptas's [1] approach of using ± 1 matrices. More precisely, we would like to select each element of the sparse matrix Φ as 0 with probability $1-q$ and uniformly ± 1 (instead of a normal distribution) with probability $1-q$. Matousek [36] recently showed that this is indeed possible without extra cost for the $(\ell_2 \to \ell_2)$ embedding case and with a multiplicative cost of an additional ε^{-1} for the $(\ell_1 \to \ell_1)$ case (also affecting the ε-ANN application).

The ANN application presented here suffers from the $n^{O(1/\varepsilon^2)}$-space requirement, an almost insurmountable implementation bottleneck for small ε. It is natural to ask whether the space and time could be traded off so that an algorithm with running time $O(\varepsilon^{-2} d \log n)$ (comparable to [21, 29] and [33]) uses significantly less space.

The Kac random walk. We propose an alternative FJLT transform which we conjecture to be at least as good as the one described in this paper, yet much more elegant. This transform is based on the following random walk on the orthogonal group on $\mathbb{R}^{d\times d}$, defined by Kac [31]. At time $t=0$, the random walk is at the identity matrix: $U_0 = Id$. At time $t>0$, we choose two random coordinates $1 \le i_t < j_t \le d$ and a random angle $\theta_t \in [0, 2\pi)$, and set $U_{t+1} = R_{i_t,j_t,\theta_t} U_t$, where $R_{i,j,\theta}$ is a rotation of the (i,j)-plane by angle θ. Clearly U_t is an orthogonal matrix for all $t \ge 0$. The walk has the Haar measure on the group of orthogonal matrices as its unique stationary distribution. For any fixed $x \in \mathbb{R}^d$, computation of $U_T x$ is extremely efficient: for $t = 1, \ldots, T$, replace x_{i_t} (resp., x_{j_t}) with $x_{i_t} \cos\theta_t + x_{j_t} \sin\theta_t$ (resp., $-x_{i_t} \sin\theta_t + x_{j_t} \cos\theta_t$). The Kac version of FJLT is defined as follows: Compute $U_T x$ for all vectors $x \in X \subseteq \mathbb{R}^d$, and return the projection onto the first $O(\varepsilon^{-2} \log |X|)$ coordinates of the resulting vectors. How small can T be in order to ensure the same guarantee as the original JL dimension-reduction technique?

Kac defined this walk in the context of statistical physics in an attempt to simplify and understand Boltzmann's equation. Since then, much attention has been given to it from the viewpoints of pure and applied mathematics. For example, it can be used to efficiently estimate high-dimensional spherical integrals [22]. Its spectral properties are by now well understood [16, 10, 38]. We conjecture that $O(d \log d + poly(\log n, \varepsilon^{-1}))$ steps suffice, and propose this as an interesting problem.

Lower bounds for FJLT. It is natural to ask what is the fastest randomized linear mapping with the Johnson–Lindenstrauss guarantee. More precisely, we have the following question.

QUESTION 3. *What is the lower bound on the expected depth of a randomized linear circuit $C_{n,d} : \mathbb{R}^d \mapsto \mathbb{R}^{O(\varepsilon^{-2}\log n)}$ such that given any set $X \subseteq \mathbb{R}^d$ of n vectors, with probability at least $2/3$, $\alpha\|x\|_2(1-\varepsilon) \le \|C_{n,d}(x)\|_p \le \alpha\|x\|_2(1+\varepsilon)$ for all $x \in X$ for some $\varepsilon > 0$, $p \in \{1,2\}$, and α?*

Any nontrivial lower bound would imply a nontrivial lower bound for computation of Fourier transform, a well known open problem.

NQLB for ε-ANN in Euclidean space. Can we achieve a *no query left behind* guarantee for ANN in Euclidean space as we did for the Hamming cube case in section 4? A union bound over the finite number of queries was the main ingredient used for making sure that, with high probability, the preprocessing construction worked for all queries simultaneously. An adversary is powerless even if he knew the random bits used in the preprocessing. In the Euclidean space, however, an adversary with access to the preprocessing random bits may be able to choose difficult queries. Perhaps a bounded VC-dimension argument could be used to argue that such an attack is not possible by the adversary.

REFERENCES

[1] D. Achlioptas, *Database-friendly random projections: Johnson-Lindenstrauss with binary coins*, J. Comput. System Sci., 66 (2003), pp. 671–687.
[2] N. Ailon and E. Liberty, *Fast dimension reduction using rademacher series on dual bch codes*, Discrete Comput. Geom., to appear.
[3] N. Alon, *Problems and results in extremal combinatorics*—I, Discrete Math., 273 (2003), pp. 31–53.
[4] N. Alon and J. Spencer, *The Probabilistic Method*, 2nd ed., John Wiley and Sons, New York, 2000.
[5] S. Arya and D. M. Mount, *Approximate nearest neighbor queries in fixed dimensions*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Austin, TX, ACM, New York, 1993, pp. 271–280.
[6] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, *An optimal algorithm for approximate nearest neighbor searching fixed dimensions*, J. ACM, 45 (1998), pp. 891–923.
[7] M. W. Bern, *Approximate closest-point queries in high dimensions*, Inform. Process. Lett., 45 (1993), pp. 95–99.
[8] E. Bingham and H. Mannila, *Random projection in dimensionality reduction: Applications to image and text data*, in Knowledge Discovery and Data Mining, ACM, New York, 2001, pp. 245–250.
[9] A. Borodin, R. Ostrovsky, and Y. Rabani, *Lower bounds for high dimensional nearest neighbor search and related problems*, in Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC), ACM, New York, 1999, pp. 312–321.
[10] E. A. Carlen, M. C. Carvalho, and M. Loss, *Determination of the spectral gap for Kac's master equation and related stochastic evolution*, Acta Math., 191 (2003), pp. 1–54.
[11] A. Chakrabarti and O. Regev, *An optimal randomised cell probe lower bound for approximate nearest neighbour searching*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, Washington, D.C., 2004, pp. 473–482.
[12] T. M. Chan, *Approximate nearest neighbor queries revisited*, Discrete Comput. Geom., 20 (1998), pp. 359–373.
[13] K. L. Clarkson, *An algorithm for approximate closest-point queries*, in Proceedings of the 10th Annual ACM Symposium on Computational Geometry (SoCG), ACM, New York, 1994, pp. 160–164.
[14] K. L. Clarkson, *Nearest neighbor queries in metric spaces*, Discrete Comput. Geom., 22 (1999), pp. 63–93.
[15] S. Dasgupta and A. Gupta, *An Elementary Proof of the Johnson-Lindenstrauss Lemma*, Technical report 99-006, UC Berkeley, Berkeley, CA, 1999.
[16] P. Diaconis and L. Saloff-Coste, *Bounds for Kac's master equation*, Comm. Math. Phys., 209 (2000), pp. 729–755.
[17] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, *Sampling algorithms for ℓ_2 regression and applications*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, 2006, pp. 1127–1136.
[18] M. Farach-Colton and P. Indyk, *Approximate nearest neighbor algorithms for Hausdorff metrics via embeddings*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Comput. Soc. Press, Los Alamitos, CA, 1999, pp. 171–179.
[19] U. Feige, D. Peleg, P. Raghavan, and E. Upfal, *Computing with unreliable information*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1990, pp. 128–137.
[20] P. Frankl and H. Maehara, *The Johnson-Lindenstrauss lemma and the sphericity of some graphs*, J. Combin. Theory Ser. B, 44 (1988), pp. 355–362.
[21] S. Har-Peled, *A replacement for Voronoi diagrams of near linear size*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), Las Vegas, NV, IEEE Comput. Soc. Press, Los Alamitos, CA, 2001, pp. 94–103.
[22] W. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrica, 57 (1970), pp. 95–109.
[23] P. Indyk, *On approximate nearest neighbors in non-Euclidean spaces*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 148–155.
[24] P. Indyk, *Dimensionality reduction techniques for proximity problems*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New

York, 2000, pp. 371–378.
[25] P. Indyk, *High-Dimensional Computational Geometry*, thesis, Stanford University, Palo Alto, CA, 2000.
[26] P. Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Comput. Soc. Press, Los Alamitos, CA, 2000, pp. 189–197.
[27] P. Indyk, *Nearest Neighbors in High-Dimensional Spaces*, in Handbook of Discrete and Computational Geometry, CRC Press, Boca Raton, FL, 2004.
[28] P. Indyk and J. Matousek, *Low-Distortion Embeddings of Finite Metric Spaces*, in Handbook of Discrete and Computational Geometry, CRC Press, Boca Raton, FL, 2004.
[29] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1998, pp. 604–613.
[30] W. B. Johnson and J. Lindenstrauss, *Extensions of Lipschitz mappings into a Hilbert space*, Contemp. Math., 26 (1984), pp. 189–206.
[31] M. Kac, *Probability and Related Topics in Physical Science*, Wiley Interscience, New York, 1959.
[32] J. M. Kleinberg, *Two algorithms for nearest-neighbor search in high dimensions*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1997, pp. 599–608.
[33] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, *Efficient search for approximate nearest neighbor in high dimensional spaces*, SIAM J. Comput., 30 (2000), pp. 457–474.
[34] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.
[35] J. Matousek, *Lectures on Discrete Geometry*. Springer-Verlag, New York, 2002.
[36] J. Matousek, *On variants of the Johnson-Lindenstrauss lemma*, Random Structures Algorithms, 33 (2008), pp. 142–156.
[37] S. Muthukrishnan and S. C. Sahinalp, *Simple and practical sequence nearest neighbors with block operations*, in Lecture Notes in Comput. Sci. 2373, Springer, Berlin, 2002, pp. 262–278.
[38] I. Pak, *Using stopping times to bound mixing times*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 953–954.
[39] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, *Latent semantic indexing: A probabilistic analysis*, in Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium of Database Systems, 1998, J. Comput. System Sci., 61 (2000), pp. 217–235.
[40] T. Sarlós, *Improved approximation algorithms for large matrices via random projections*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Berkeley, CA, 2006, pp. 143–152.
[41] L. Schulman, *Clustering for edge-cost minimization*, in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2000, pp. 547–555.
[42] P. N. Yianilos, *Data structures and algorithms for nearest neighbor search in general metric spaces*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, 1993, pp. 311–321.
[43] P. N. Yianilos, *Locally lifting the curse of dimensionality for nearest neighbor search (extended abstract)*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, 2000, pp. 361–370.