DOI:10.1145/1646353.1646379

Faster Dimension Reduction

By Nir Ailon and Bernard Chazelle

Abstract

Data represented geometrically in high-dimensional vector spaces can be found in many applications. Images and videos, are often represented by assigning a dimension for every pixel (and time). Text documents may be represented in a vector space where each word in the dictionary incurs a dimension. The need to manipulate such data in huge corpora such as the web and to support various query types gives rise to the question of how to represent the data in a lower-dimensional space to allow more space and time efficient computation. Linear mappings are an attractive approach to this problem because the mapped input can be readily fed into popular algorithms that operate on linear spaces (such as principal-component analysis, PCA) while avoiding the curse of dimensionality.

The fact that such mappings even exist became known in computer science following seminal work by Johnson and Lindenstrauss in the early 1980s. The underlying technique is often called "random projection." The complexity of the mapping itself, essentially the product of a vector with a dense matrix, did not attract much attention until recently. In 2006, we discovered a way to "sparsify" the matrix via a computational version of Heisenberg's Uncertainty Principle. This led to a significant speedup, which also retained the practical simplicity of the standard Johnson–Lindenstrauss projection. We describe the improvement in this article, together with some of its applications.

1. INTRODUCTION

Dimension reduction, as the name suggests, is an algorithmic technique for reducing the dimensionality of data. From a programmer's point of view, a d-dimensional array of real numbers, after applying this technique, is represented by a much smaller array. This is useful because many data-centric applications suffer from exponential blowup as the underlying dimension grows. The infamous curse of dimensionality (exponential dependence of an algorithm on the dimension of the input) can be avoided if the input data is mapped into a space of logarithmic dimension (or less); for example, an algorithm running in time proportional to 2^d in dimension d will run in linear time if the dimension can be brought down to log d. Common beneficiaries of this approach are clustering and nearest neighbor searching algorithms. One typical case involving both is, for example, organizing a massive corpus of documents in a database that allows one to respond quickly to similar-document searches. The clustering is used in the back-end to eliminate (near) duplicates, while nearest-neighbor queries are processed at the front-end. Reducing the dimensionality of the data helps the system respond faster to both queries and data updates. The idea, of course, is to retain the basic metric properties of the data set (e.g., pairwise distances) while reducing its size. Because this is technically impossible to do, one will typically relax this demand and tolerate errors as long as they can be made arbitrarily small.

The common approaches to dimensionality reduction fall into two main classes. The first one includes *data-aware* techniques that take advantage of prior information about the input, principal-component analysis (PCA) and compressed sensing being the two archetypical examples: the former works best when most of the information in the data is concentrated along a few fixed, unknown directions in the vector space. The latter shines when there exists a basis of the linear space over which the input can be represented sparsely, i.e., as points with few nonzero coordinates.

The second approach to dimension reduction includes *data-oblivious* techniques that assume no prior information on the data. Examples include sketches for data streams, locality sensitive hashing, and random linear mappings in Euclidean space. The latter is the focus of this article. Programmatically, it is equivalent to multiplying the input array by a random matrix. We begin with a rough sketch of the main idea.

Drawing on basic intuition from both linear algebra and probability, it may be easy to see that mapping high-dimensional data into a random lower-dimensional space via a linear function will produce an approximate representation of the original data. Think of the directions contained in the random space as samples from a population, each offering a slightly different view of a set of vectors, given by their projection therein. The collection of these narrow observations can be used to learn about the approximate geometry of these vectors. By "approximate" we mean that properties that the task at hand may care about (such as distances and angles between vector) will be slightly distorted. Here is a small example for concreteness. Let $a_1, \ldots, a_d$ be independent random variables with mean 0 and unit variance (e.g., a Gaussian $N(0, 1)$). Given a vector $x = (x_1, \ldots, x_d)$, consider the inner product $Z = \Sigma_i a_i x_i$: the expectation of Z is 0 but its variance is precisely the square of the Euclidean length of x. The number Z can be interpreted as a "random projection" in one dimension: the variance allows us to "read off" the length of x. By sampling in this way several times, we can increase our confidence, using the law of

A previous version of this paper appeared in *Proceedings of the 38th ACM Symposium on Theory in Computing* (May 2006, Seattle, WA).

large numbers. Each sample corresponds to a dimension. The beauty of the scheme is that we can now use it to handle many distances at once.

It is easy to see that randomness is necessary if we hope to make meaningful use of the reduced data; otherwise we could be given as input a set of vectors belonging to the kernel of any fixed matrix, thus losing all information. The size of the distortion as well as the failure probability are user-specified parameters that determine the target (low) dimension. How many dimensions are sufficient? Careful quantitative calculation reveals that, if all we care about is distances between pairs of vectors and angles between them—in other words, the Euclidean *geometry* of the data—then a random linear mapping to a space of dimension *logarithmic* in the size of the data is sufficient. This statement, which we formalize in Section 1.1, follows from Johnson and Lindenstrauss's seminal work.[25] The consequence is quite powerful: If our database contains n vectors in d dimensions, then we can replace it with one in which data contains only $\log n$ dimensions! Although the original paper was not stated in a computational language, deriving a naïve pseudocode for an algorithm implementing the idea in that paper is almost immediate. This algorithm, which we refer to as JL for brevity, has been studied in theoretical computer science in many different contexts. The main theme in this study is improving efficiency of algorithms for high-dimensional geometric problems such as clustering,[37] nearest neighbor searching,[24, 27] and large scale linear algebraic computation.[18, 28, 35, 36, 38]

For many readers it may be obvious that these algorithms are directly related to widely used technologies such as web search. For others this may come as a surprise: Where does a Euclidean space hide in a web full of textual documents? It turns out that it is very useful to represent text as vectors in high-dimensional Euclidean space.[34] The dimension in the latter example can be as high as the number of words in the text language!

This last example illustrates what a *metric embedding* is: a mapping of objects as points in metric spaces. Computer scientists care about such embeddings because often it is easier to design algorithms for metric spaces. The simpler the metric space is, the friendlier it is for algorithm design. *Dimensionality* is just one out of many measures of simplicity. We digress from JL by mentioning a few important results in computer science illustrating why embedding input in simple metric spaces is useful. We refer the reader to Linial et al.[29] for one of the pioneering works in the field.

- The Traveling Salesman Problem (TSP), in which one wishes to plan a full tour of a set of cities, with given costs of traveling between any two cities is an archetype of a computational hardness which becomes easier if the cities are embedded in a metric space,[14] and especially in a low-dimensional Euclidean one.[7, 30]
- Problems such as combinatorial optimization on graphs become easier if the nodes of the graph can be embedded in ℓ_1 space. (The space ℓ_p^d is defined to be the set $\mathbf{R}^d$ endowed with a norm, where the norm of a vector x (written as $\|x\|_p$) is given by $(\Sigma_{i=1}^{d} |x_i|^p)^{1/p}$. The metric is given by the distance between pairs of vectors x and y taken to be $\|x-y\|_p$.)
- Embedding into tree metrics (where the distance between two nodes is defined by the length of the path joining them) is useful for solving network design optimization problems.

The JL algorithm linearly embeds an input which is already in a high-dimensional Euclidean space ℓ_2^d into a lower-dimensional ℓ_p^k space for any $p \geq 1$, and admits a naïve implementation with $O(dk)$ running time per data vector; in other words, the complexity is proportional to the number of random matrix elements.

Our modification of JL is denoted FJLT, for *Fast-Johnson–Lindenstrauss-Transform*. Although JL often works well, it is the computational bottleneck of many applications, such as approximate nearest neighbor searching.[24, 27] In such cases, substituting FJLT yields an immediate improvement. Another benefit is that implementing FJLT remains extremely simple. Later in Section 3 we show how FJLT helps in some of the applications mentioned above. Until then, we concentrate on the story of FJLT itself, which is interesting in its own right.

1.1. A brief history of a quest for a faster JL

Before describing our result, we present the original JL result in detail, as well as survey results related to its computational aspects. We begin with the central lemma behind JL.[25] The following are the main variables we will be manipulating:

X—a set of vectors in Euclidean space (our input dataset). In what follows, we use the term *points* and *vectors* interchangeably.

n—the size of the set X.

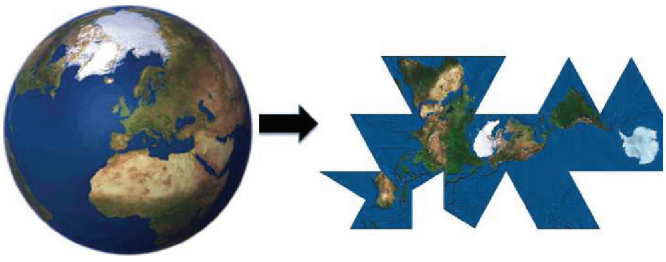
d—the dimension of the Euclidean space (typically very big).

k—the dimension of the space we will reduce the points in X to (ideally, much smaller than d).

ε—a small tolerance parameter, measuring to what is the maximum allowed distortion rate of the metric space induced by the set X in Euclidean m-space (the exact definition of distortion will be given below).

In JL, we take k to be $c\varepsilon^{-2} \log n$, for some large enough

Figure 1. Embedding a spherical metric onto a planar one is no easy task. The latter is more favorable as input to printers.



absolute constant c. We then choose a random subspace of dimension k in $\mathbf{R}^d$ (we omit the mathematical details of what a random subspace is), and define Φ to be the operation of projecting a point in $\mathbf{R}^d$ onto the subspace. We remind the reader that such an operation is linear, and is hence equivalently representable by a matrix. In other words, we've just defined a random matrix. Denote it by Φ.

The JL Lemma states that with high probability, for all pairs of points $x, y \in X$ simultaneously,

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

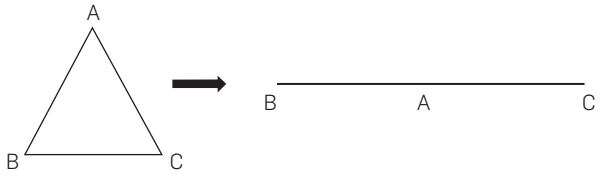
This fact is useful provided that $k < d$, which will be implied by the assumption

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

Informally, JL says that projecting the n points on a random low-dimensional subspace should, *up to a distortion of* $1 \pm \varepsilon$, preserve pairwise distances. The mapping matrix of size $\Phi = k \times d$ can be implemented in a computer program as follows: The first row is a random unit vector chosen uniformly in $\mathbf{R}^d$; 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, etc. The high-level proof idea is to show that for each pair $x, y \in X$ the probability of (1) being violated is order of $1/n^2$. A standard union bound over the number of pairs of points in X then concludes the proof.

It is interesting to pause and ask whether the JL theorem should be intuitive. The answer is both yes and no. Low-dimensional geometric intuition is of little help. Take an equilateral triangle ABC in the plane (Figure 2), no matter how you project it into a line, you get three points in a row, two of which form a distance at least twice the smallest one. The distortion is at least 2, which is quite bad. The problem is that, although the expected length of each side's projection is identical, the variance is high. In other words, the projected distance is rarely close to the average. If, instead of $d = 2$, we choose a high dimension d and project down to $k = c\varepsilon^{-2} \log n$ dimensions, the three projected lengths of ABC still have the same expected value, but crucially their (identical) variances are now very small. Why? Each such length (squared) is a sum of k independent random variables, so its distribution is almost normal with variance proportional to k (this is a simple case of the central limit theorem). This fact alone explains each factor in the expression for k: ε^{-2} ensures the desired distortion; $\log n$ reduces the error probability to $n^{-c'}$, for constant c' growing with c, which allows us to apply a union bound over all $\binom{n}{2}$ pairs of distances in X.

Figure 2. A triangle cannot be embedded onto a line while simultaneously preserving distances between all pairs of vertices.



Following Johnson and Lindenstrauss,[25] various researchers suggested simplifications of the original JL design and of their proofs (Frankl and Maehara,[20] DasGupta and Gupta,[17] Indyk and Motwany[24]). These simplifications slightly change the distribution from which Φ is drawn and result in a better constant c and simpler proofs. These results, however, do not depart from the original JL from a computational point of view, because the necessary time to apply Φ to a vector is still order of nk.

A bold and ingenious attempt to reduce this cost was taken by Achlioptas.[1] He noticed that the only property of Φ needed for the transformation to work is that $(\Phi_i \cdot x)^2$ be tightly concentrated around the mean $1/d$ for all unit vectors $x \in \mathbf{R}^d$, where Φ_i is the ith row of Φ. The distribution he proposed is very simple: Choose each element of Φ uniformly from the following distribution:

$$\begin{cases} \sqrt{3}/d & \text{with probability } 1/6; \\ 0 & 2/3; \\ -\sqrt{3}/d & 1/6. \end{cases}$$

The nice property of this distribution is that it is relatively sparse: on average, a fraction 2/3 of the entries of Φ are 0. Assuming we want to apply Φ on many points in $\mathbf{R}^d$ in a real-time setting, we can keep a linked list of all the nonzeros of Φ during preprocessing and reap the rewards in the form of a threefold speedup in running time.

Is Achlioptas's result optimal, or is it possible to get a super constant speedup? This question is the point of departure for this work. One idea to obtain a speedup, aside from sparsifying Φ, would be to reduce the target dimension k, and multiply by a smaller matrix Φ. Does this have a chance of working? A lower bound of Alon[5] provides a negative answer to this question, and dashes any hope of reducing the number of rows of Φ by more than a factor of $O(\log(1/\varepsilon))$. The remaining question is hence whether the matrix can be made sparser than Achlioptas's construction. This idea has been explored by Bingham and Mannila.[11] They considered sparse projection heuristics, namely, fixing most of the entries of Φ as zeroes. They noticed that in practice such matrices Φ seem to give a considerable speedup with little compromise in distortion for data found in certain applications. Unfortunately, it can be shown that sparsifying Φ by more than a constant factor (as implicitly suggested in Bingham and Mannila's work) will not work for all inputs. Indeed, a sparse matrix will typically distort a sparse vector. The intuition for this is given by an extreme case: If both Φ and the vector x are very sparse, the product Φx may be null, not necessarily because of cancellations, but more simply because each multiplication $\Phi_{ij} x_j$ is itself zero.

1.2. The random densification technique

In order to prevent the problem of simultaneous sparsity of Φ and x, 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 be both concentrated. The look of

frustration on the face of any musician who has to wrestle with the delay from a digital synthesizer can be attributed to the Uncertainty Principle.

Before we show how to use this principle, we must stop and ask: what are the tools we have at our disposal? We may write the matrix Φ as a product of matrices, or, algorithmically, apply a chain of linear mappings on an input vector. With that in mind, an interesting family of matrices we can apply to an input vector is the *orthogonal* family of d-by-d matrices. Such matrices are *isometries*: The Euclidean geometry suffers no distortion from their application.

With this in mind, we precondition the random k-by-d mapping with a Fourier transform (via an efficient FFT algorithm) in order to isometrically densify any sparse vector. To prevent the inverse effect, i.e., the sparsification of dense vectors, we add a little randomization to the Fourier transform (see Section 2 for details). The reason this works is because sparse vectors are rare within the space of all vectors. Think of them as forming a tiny ball within a huge one: if you are inside the tiny ball, a random transformation is likely to take you outside; on the other hand, if you are outside to begin with, the transformation is highly unlikely to take you inside the tiny ball.

The resulting FJLT shares the low-distortion characteristics of JL but with a lower running time complexity.

2. THE DETAILS OF FJLT

In this section we show how to construct a matrix Φ drawn from FJLT and then prove that it works, namely:

1. It provides a low distortion guarantee. (In addition to showing that it embeds vectors in low-dimensional ℓ_2^k, we will show it also embeds in ℓ_1^k.)
2. Applying it to a vector is efficiently computable.

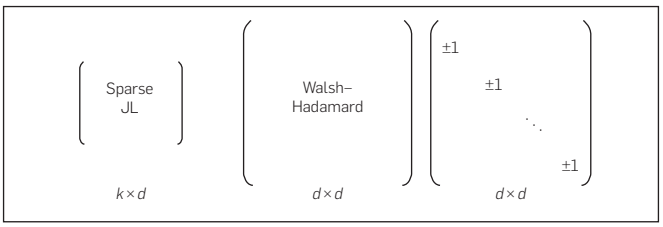
The first property is shared by the standard JL and its variants, while the second one is the main novelty of this work.

2.1. Constructing Φ

We first make some simplifying assumptions. We may assume with no loss of generality that d is a power of two, $d = 2^h > k$, and that $n\,\Omega\, d = \Omega(\varepsilon^{-1/2})$; otherwise the dimension of the reduced space is linear in the original dimension. Our random embedding $\Phi \sim \text{FJLT}(n, d, \varepsilon, p)$ is a product of three real-valued matrices (Figure 3):

$$\Phi = PHD.$$

Figure 3. FJLT.



The matrices P and D are chosen randomly whereas H is deterministic:

- P is a k-by-d matrix. Each element is an independent mixture of 0 with an unbiased normal distribution of variance $1/q$, where

$$q = \min\left\{\Theta\left(\frac{\varepsilon^{p-2}\log^p n}{d}\right), 1\right\}.$$

  In other words, $P_{ij} \sim N(0, 1/q)$ with probability q, and $P_{ij} = 0$ with probability $1 - q$.
- 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.
- 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 corresponds to the discrete Fourier transform over the additive group GF $(2)^d$: its FFT is very simple to compute and requires only $O(d \log d)$ steps. It follows that the mapping Φx of *any* point $x \in \mathbb{R}^d$ can be computed in time $O(d \log d + |P|)$, where $|P|$ is the number of nonzero entries in P. The latter is $O(\varepsilon^{-2} \log n)$ not only on average but also with high probability. Thus we can assume that the running time of $O(d \log d + qd\varepsilon^{-2} \log n)$ is worst-case, and not just expected.

The FJLT Lemma. *Given a fixed set X of n points in $\mathbf{R}^d$, $\varepsilon < 1$, and $p \in \{1, 2\}$, draw a matrix Φ from* FJLT. *With probability at least* 2/3, *the following two events occur*:

1. *For any $x \in X$,*
$$(1-\varepsilon)\alpha_p \|x\|_2 \le \|\Phi x\|_p \le (1+\varepsilon)\alpha_p \|x\|_2,$$
where $\alpha_1 = k\sqrt{2\pi^{-1}}$ and $\alpha_2 = k$.
2. *The mapping Φ: $\mathbf{R}^d \to \mathbf{R}^k$ requires*
$$O(d \log d + \min\{d\varepsilon^{-2} \log n, \varepsilon^{p-4} \log^{p+1} n\})$$
operations.

Remark: By repeating the construction $O(\log(1/\delta))$ times we can ensure that the probability of failure drops to δ for any desired $\delta > 0$. By *failure* we mean that either the first or the second part of the lemma does not hold.

2.2. Showing that Φ works

We sketch a proof of the FJLT Lemma. Without loss of generality, we can assume that $\varepsilon < \varepsilon_0$ for some suitably small ε_0. Fix $x \in X$. The inequalities of the lemma are invariant under scaling, so we can assume that $\|x\|_2 = 1$. Consider the random variable $u = HDx$, denoted by $(u_1, \ldots, u_d)^T$. The first coordinate u_1 is of the form $\Sigma_{i=1}^d a_i x_i$, where each $a_i = \pm d^{-1/2}$ is chosen independently and uniformly. We can use a standard tail estimate technique to prove that, with probability at least, say, 0.95,

$$\max_{x \in X} \|HDx\|_\infty = O\left(d^{-1/2}\sqrt{\log n}\right). \quad (3)$$

It is important to intuitively understand what (3) means. Bounding $\|HDx\|_\infty$ is tantamount to bounding the magnitude of all coordinates of HDx. This can be directly translated to a *densification* property. To see why, consider an extreme case: If we knew that, say, $\|HDx\|_\infty < 1$, then we would automatically steer clear of the sparsest case possible, in which x is null in all but one coordinate (which would have to be 1 by the assumption $\|x\|_2 = \|HDx\|_2 = 1$).

To prove (3), we first make the following technical observation:

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

Setting $t = sd$ above, we now use the technical observation together with Markov's inequality to conclude that, for any $s > 0$,

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

for $s = \Theta(d^{-1/2}\sqrt{\log n})$. A union bound over all $nd \le n^2$ coordinates of the vectors $\{HDx | x \in X\}$ leads to (3). We assume from now on that (3) holds with s as the upper bound; in other words, $\|u\|_\infty \le s$, where $u = HDx$. Assume now that u is fixed. It is convenient (and immaterial) to choose s so that $m \stackrel{\text{def}}{=} s^{-2}$ is an integer.

It can be shown that $\|u\|_2 = \|x\|_2$ by virtue of both H and D (and their composition) being isometries (i.e., preserve ℓ_2 norms). Now define,

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

The vector y is the final mapping of x using Φ. It is useful to consider each coordinate of y separately. All coordinates share the same distribution (though not as independent random variables). Consider y_1. By definition of FJLT, it 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; then draw random i.i.d. variables $r_1, \ldots, r_d$ from $N(0, 1/q)$. Set $y_1 = \sum_{j=1}^d r_j b_j u_j$ and let $Z = \sum_{j=1}^d b_j u_j^2$. It can be shown that the conditional variable $(y_1 | Z = z)$ is distributed $N(0, z/q)$ (this follows a well known fact known as the 2-stability of the normal distribution). 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 $Z_1 (= Z), Z_2, \ldots, Z_k$. It now follows that the expectation of Z satisfies:

$$\mathbf{E}[Z] = \sum_{j=1}^d u_j^2 \mathbf{E}[b_j] = q. \tag{4}$$

Let u^2 formally denote $(u_1^2, \ldots, u_d^2) \in (\mathbf{R}^+)^d$. By our assumption that (3) holds, u^2 lies in the d-dimensional polytope:

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

Let $u^* \in \mathbf{R}^d$ denote a vector such that u^{*2} is a vertex of $\mathcal{P}$. By symmetry of these vertices, 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 vector u^* will be convenient for identifying extremal cases in the analysis of Z. By extremal we mean the most *problematic* case, namely, the sparsest possible under assumption (3) (recall that the whole objective of *HD* was to alleviate sparseness).

We shall use Z^* to denote the random variable Z corresponding to the case $u = u^*$. We observe that $Z^* \sim m^{-1}B(m, q)$; in words, the binomial distribution with parameters m, q divided by the constant m. Consequently,

$$\mathbf{var}(Z^*) = q(1-q)/m. \tag{5}$$

In what follows, we divide our discussion between the ℓ_1 and the ℓ_2 cases.

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 the moments of Z over the random b_j's.

LEMMA 1. *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: The case $q = 1$ is trivial because Z is constant and equal to 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 $\mathcal{P}$. So it suffices to prove the moment upper bounds for Z^*, which conveniently behaves like a (scaled) binomial. By standard bounds on the binomial moments,

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

proving the first part of the lemma.

By Jensen's inequality and (4),

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

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 $u = u^*$. 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 $x \ge 0$,

$$\begin{aligned}\mathbf{E}\left[\sqrt{Z^*}\right] &= \sqrt{q}\,\mathbf{E}\left[\sqrt{Z^*/q}\right] \\ &\ge \sqrt{q}\left(1 + \frac{1}{2}\mathbf{E}[Z^*/q - 1] - \mathbf{E}[(Z^*/q - 1)^2]\right).\end{aligned} \tag{6}$$

By (4), $\mathbf{E}[Z^*/q - 1] = 0$ and, using (5),

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

Plugging this into (6) shows that $\mathbf{E}\left[\sqrt{Z^*}\right] \ge \sqrt{q}(1-\varepsilon)$, as desired. ✹

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

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

On the other hand, by Lemma 1, we note that

$$(1-\varepsilon)\sqrt{2/\pi} \le \mathbf{E}[|y_1|] \le \sqrt{2/\pi}. \quad (7)$$

Next, we prove that $\|y\|_1$ is sharply concentrated around its mean $\mathbf{E}[\|y\|_1] = k\mathbf{E}[|y_1|]$. To do this, we begin by bounding the moments of $|y_1| = |\Sigma_j b_j r_j u_j|$. Using conditional expectations, we can show that, for any integer $t \ge 0$,

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

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

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

It follows that the moment generating function satisfies

$$\mathbf{E}[e^{\lambda|y_1|}] = 1 + \lambda\,\mathbf{E}[|y_1|] + \sum_{t>1}\mathbf{E}[|y_1|^t]\lambda^t/t! \le 1 + \lambda\,\mathbf{E}[|y_1|] + \sum_{t>1} O(t)^t\lambda^t/t!.$$

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

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

Using independence, we find that

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

Meanwhile, Markov's inequality and (7) imply that

$$\mathbf{Pr}[\|y\|_1 \ge (1+\varepsilon)\mathbf{E}[\|y\|_1] \le \mathbf{E}[e^{\lambda\|y\|_1}]/e^{\lambda(1+\varepsilon)\mathbf{E}[\|y\|_1]} \le e^{-\lambda\varepsilon\mathbf{E}[\|y\|_1]+O(\lambda^2 k)} \le e^{-\Omega(\varepsilon^2 k)},$$

for some $\lambda = \Theta(\varepsilon)$. The constraint $\lambda < \lambda_0$ corresponds to ε being smaller than some absolute constant. The same 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 0.95. By (7), this implies that $k\mathbf{E}[|y_1|]$ is itself concentrated around $\alpha_1 = k\sqrt{2/\pi}$ with a relative error at most ε; rescaling ε by a constant factor and ensuring (3) proves the ℓ_1 claim of the first part of the FJLT lemma.

The ℓ_2 case: We set

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

for a large enough constant c_1.

LEMMA 2. *With probability at least* $1 - \frac{1}{20n}$,

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

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, hence achieves its maximum at the vertices of the polytope $\mathcal{P}$ (same as in the proof of Lemma 1). As argued before, therefore, $\mathbf{E}[e^{\lambda Z}] \le \mathbf{E}[e^{\lambda Z^*}]$. We conclude the proof of the first part with a union bound on standard tail estimates on the scaled binomial Z^* that we derive from bounds on its moment generating function $\mathbf{E}[e^{\lambda Z^*}]$ (e.g., Alon and Spencer[6]). For the second part, let $S = \Sigma_{i=1}^k Z_i$. Again, the moment generating function of S is bounded above by that of $S^* \sim m^{-1}B(mk, q)$—all Z_i's are distributed as Z^*—and the desired concentration bound follows. ✹

We assume from now on that the premise of Lemma 2 holds for all choices of $x \in X$. A union bound shows that this happens with probability of at least 0.95. For each $i = 1, \ldots, k$ the random variable $y_i^2 q / Z_i$ is distributed as χ^2 with one degree of freedom. It follows that, conditioned on Z_i, the expected value of y_i^2 is Z_i/q and the moment generating function of y_i^2 is

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

Given any $0 < \lambda < \lambda_0$, for fixed λ_0, for large enough ξ, the moment generating function converges and is equal to

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

We use here the fact that $Z_i/q = O(1)$, which we derive from the first part of Lemma 2. By independence, therefore,

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

and hence

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{k} y_i^2 > (1+\varepsilon)\sum_{i=1}^{k} Z_i/q\right] &= \mathbf{Pr}\left[e^{\lambda\sum_{i=1}^{k} y_i^2} > e^{(1+\varepsilon)\lambda\sum_{i=1}^{k} Z_i/q}\right] \\ &\le \mathbf{E}\left[e^{\lambda\sum_{i=1}^{k} y_i^2}\right] / e^{(1+\varepsilon)\lambda\sum_{i=1}^{k} Z_i/q} \\ &\le e^{-\varepsilon\lambda\sum_{i=1}^{k} Z_i/q + \xi\lambda^2\sum_{i=1}^{k}(Z_i/q)^2}.\end{aligned} \quad (8)$$

If we plug

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

into (8) and assume that ε is smaller than some global ε_0, we avoid convergence issues (Lemma 2). By that same lemma, 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. We set $k = c\varepsilon^{-2}\log n$ for some large enough c and use a union bound, possibly rescaling ε, to conclude the ℓ_2 case of the first part of the FJLT lemma.

Running Time: The vector Dx requires $O(d)$ steps, since D is diagonal. Computing $H(Dx)$ takes $O(d \log d)$ time using the FFT for Walsh–Hadamard. Finally, computing $P(H\,Dx)$ requires $O(|P|)$ time, where $|P|$ is the number of nonzeros in P. This number is distributed in $B(nk, q)$. It is now immediate to verify that

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

A Markov bound establishes the desired complexity of the FJLT. This concludes our sketch of the proof of the FJLT lemma. ■

3. APPLICATIONS

3.1. Approximate nearest neighbor searching

Given a metric space (U, d_U) and a finite subset (database) $P \subseteq U$, the problem of *ε-approximate nearest neighbor* (ε-ANN) searching is to preprocess P so that, given a query $x \in U$, a point $p \in P$ satisfying

$$d_U(x,p) \le (1+\varepsilon)d_U(x,q), \quad \text{for all} \quad q \in P,$$

can be found efficiently. In other words, we are interested in a point p further from x by a factor at most $(1 + \varepsilon)$ of the distance to its nearest neighbor.

This problem has received considerable attention. There are two good reasons for this: (i) ANN boasts more applications than virtually any other geometric problem[23]; (ii) allowing a small error ε makes it possible to break the curse of dimensionality.[24, 27]

There is abundant literature on (approximate) nearest neighbor searching.[8–10, 12, 13, 15, 16, 19, 21–24, 26, 27, 33, 39, 40] 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 Indyk[23] for a recent survey).

The first algorithms with query times of poly(d, log n) and polynomial storage (for fixed ε) were those of Indyk and Motwani[24] in the Euclidean space case, and Kushilevitz et al.[27] in the Hamming cube case. Using JL, Indyk et al. provide a query time of $O(\varepsilon^{-2}d \log n)$ with $n^{O(\varepsilon^{-2})}$ storage and preprocessing. A discrete variant of JL was used by Kushilevitz et al. in the Hamming cube case. We mention here that the dimension reduction overwhelms the running time of the two algorithms. In order to improve the running time in both cases, we used two main ideas in Ailon and Chazelle.[2] The first idea applied to the discrete case. It used an observation related to the algebraic structure of the discrete version of JL used in Kushilevitz et al.[27] to obtain a speedup in running time. This observation was only applicable in the discrete case, but suggested the intuitive idea that a faster JL should be possible in Euclidean space as well, thereby motivating the search for FJLT. Indeed, by a straightforward application in Indyk et al.'s algorithm (with $p = 1$), the running time would later be improved using FJLT to $O(d \log d + \varepsilon^{-3}\log^2 n)$. Notice the *additive* form of this last expression in some function $f = f(d)$ and $g = g(n, \varepsilon)$, instead of a *multiplicative* one.

3.2. Fast approximation of large matrices

Large matrices appear in virtually every corner of science. Exact algorithms for decomposing or solving for large matrices are often inhibitively expensive to perform. This may change given improvements in matrix multiplication technology, but it appears that we will have to rely on matrix approximation strategies for a while, at least in the general case. It turns out that FJLT and ideas inspired by it play an important role in recent developments.

We elaborate on an example from a recent solution of Sarlós[36] to the problem of ℓ_2 regression (least square fit of an overdetermined linear system). Prior to that work (and ours), Drineas et al.[18] showed that, by downsampling (choosing only a small subset and discarding the rest) from the set of equations of the linear regression, an approximate solution to the problem could be obtained by solving the downsampled problem, the size of which depends only on the dimension d of the original solution space. The difficulty with this method is that the downsampling distribution depends on norms of rows of the left-singular vector matrix of the original system. Computing this matrix is as hard as the original regression problem and requires $O(m^2d)$ operations, with m the number of equations. To make this solution more practical, Sarlós observed that multiplying the equation matrix on the left by the $m \times m$ orthogonal matrix HD (as defined above in the definition of FJLT) implicitly multiplies the left-singular vectors by HD as well. By an analysis similar to the one above, the resulting left-singular matrix can be shown to have almost uniform row norm. This allows use of Drineas et al.'s ideas with uniform sampling of the equations. Put together, these results imply the first $o(m^2d)$ running time solution for worst-case approximate ℓ_2 regression.

In a recent stream of papers, authors Liberty, Martinsson, Rokhlin, Tygert and Woolfe[28, 35, 38] design and analyze fast algorithms for low-dimensional approximation algorithms of matrices, and demonstrate their application to the evaluation of the SVD of numerically low-rank matrices. Their schemes are based on randomized transformations akin to FJLT.

4. BEYOND FJLT

The FJLT result gives rise to the following question: What is a lower bound, as a function of n, d and ε, on the complexity of computing a JL-like random linear mapping? By this we mean a mapping that distorts pairwise Euclidean distances among any set of n points in d dimension by at most $1 \pm \varepsilon$. The underlying model of computation can be chosen as a linear circuit,[32] manipulating complex-valued intermediates by either adding two or multiplying one by (random) constants, and designating n as input and $k = O(\varepsilon^{-2} \log n)$ as output (say, for $p = 2$). It is worth observing that any lower bound in $\Omega(\varepsilon^{-2} \log n \min\{d, \log^2 n\})$ would imply a similar lower bound on the complexity of computing a Fourier transform. Such bounds are known only in a very restricted model[31] where constants are of bounded magnitude.

As a particular case of interest, we note that, whenever $k = O(d^{1/3})$, the running time of FJLT is $O(d \log d)$. In a more recent paper, Ailon and Liberty[3] improved this bound and showed that it is possible to obtain a JL-like random mapping in time $O(d \log d)$ for $k = O(d^{1/2-\delta})$ and any $\delta > 0$. Their transformation borrows the idea of preconditioning a Fourier transform with a random diagonal matrix from FJLT, but uses it differently and takes advantage of stronger measure concentration bounds and tools from error correcting codes over fields of characteristic 2. The same authors together with Singer consider the following inverse problem[4]: Design randomized linear time computable transformations that require the mildest assumptions possible on data to ensure successful dimensionality reduction. C

References

1. Achlioptas, D. Database-friendly random projections: Johnson–Lindenstrauss with binary coins. *J. Comput. Syst. Sci. 66*, 4 (2003), 671–687.
2. Ailon, N., Chazelle, B. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. *SIAM J. Comput. 39*, 1 (2009), 302–322.
3. Ailon, N., Liberty, E. Fast dimension reduction using rademacher series on dual bch codes. *Discrete and Computational Geometry* (2008).
4. Ailon, N., Liberty, E., Singer, A. Dense fast random projections and lean walsh transforms. *APPROX-RANDOM*, 2008, 512–522.
5. Alon, N. Problems and results in extremal combinatorics–I. *Discrete Math. 273*, 1–3 (2003), 31–53.
6. Alon, N., Spencer, *J. The Probabilistic Method*. John Wiley, 2nd edition, 2000.
7. Arora, S. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM 45*, 5 (1998), 753–782.
8. Arya, S., Mount, D.M. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (Austin, TX, United States, 1993), 271–280.
9. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM 45*, 6 (1998), 891–923.
10. Bern, M.W. Approximate closest-point queries in high dimensions. *Inf. Process. Lett. 45*, 2 (1993), 95–99.
11. Bingham, E., Mannila, H. Random projection in dimensionality reduction: Applications to image and text data. In *Knowledge Discovery and Data Mining*, 2001, 245–250.
12. Borodin, A., Ostrovsky, R., Rabani, Y. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the 31st Annual Symposium on the Theory of Computing* (*STOC*) (1999), 312–321.
13. Chan, T.M. Approximate nearest neighbor queries revisited. *Discrete Comput. Geometry 20*, 3 (1998), 359–373.
14. Christofides, N. Worst-case analysis of a new heuristic for the travelling salesman problem. *Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh*, 1976, 388.
15. Clarkson, K.L. An algorithm for approximate closest-point queries. In *Proceedings of the 10th Annual ACM Symposium on Computational Geometry* (*SoCG*) (1994), 160–164.
16. Clarkson, K.L. Nearest neighbor queries in metric spaces. *Discrete Comput. Geometry 22*, 1 (1999), 63–93.
17. DasGupta, S., Gupta, A. An elementary proof of the Johnson–Lindenstrauss lemma. *Technical Report, UC Berkeley*, 1999, 99–106.
18. Drineas, P., Mahoney, M.W., Muthukrishnan, S. Sampling algorithms for ℓ_2 regression and applications. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (Miami, FL, United States, 2006).
19. Farach-Colton, M., Indyk, P. Approximate nearest neighbor algorithms for Hausdorff metrics via embeddings. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science* (*FOCS*) (1999), 171–180.
20. Frankl, P., Maehara, H. The Johnson–Lindenstrauss lemma and the sphericity of some graphs. *J. Comb. Theory Ser. A 44* (1987), 355–362.
21. Indyk, P. On approximate nearest neighbors in non-Euclidean spaces. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science* (*FOCS*) (1998), 148–155.
22. Indyk, P. Dimensionality reduction techniques for proximity problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (2000), 371–378.
23. Indyk, P. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*. CRC, 2004.
24. Indyk P., Motwani, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing* (*STOC*) (1998), 604–613.
25. Johnson, W.B., Lindenstrauss, J. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math. 26* (1984), 189–206.
26. Kleinberg, J.M. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (*STOC*) (1997), 599–608.
27. Kushilevitz, E., Ostrovsky, R., Rabani, Y. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput. 30*, 2 (2000), 457–474.
28. Liberty, E., Woolfe, F., Martinsson, P.-G., Rokhlin, V., Tygert, M. Randomized algorithms for the low-rank approximation of matrices. *Proc. Natl. Acad. Sci.* (*PNAS*) *104*, 51 (2007), 20167–20172.
29. Linial, N., London, E., Rabinovich, Y. The geometry of graphs and some of its algorithmic applications. *Combinatorica 15*, 2 (1995), 215–245.
30. Mitchell, J.S.B. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem. *SIAM J. Comput. 28*, 4 (1999), 1298–1309.
31. Morgenstern, J. Note on a lower bound on the linear complexity of the fast fourier transform. *J. ACM 20*, 2 (1973), 305–306.
32. Morgenstern, J. The linear complexity of computation. *J. ACM 22*, 2 (1975), 184–194.
33. Muthukrishnan, S., Sahinalp, S.C. Simple and practical sequence nearest neighbors with block operations. In *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching* (*CPM*) (2002), 262–278.
34. Papadimitriou, C., Raghavan, P., Tamaki, H., Vempala, S. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the 17th Annual Symposium of Database Systems* (1998), 159–168.
35. Rokhlin, V., Tygert, M. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Science* (*PNAS*) *105*, 36 (2008), 13212–13217.
36. Sarlós, T. 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).
37. Schulman, L. Clustering for edge-cost minimization. In *Proceedings of the 32nd Annual Symposium on Theory of Computing* (*STOC*) (2000), 547–555.
38. Woolfe, F., Liberty, E., Rokhlin, V., Tygert, M. A fast randomized algorithm for the approximation of matrices. *Appl. Comput. Harmon. Anal. 25* (2008), 335–366.
39. Yianilos, P.N. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (1993), 311–321.
40. Yianilos, P.N. Locally lifting the curse of dimensionality for nearest neighbor search (extended abstract). In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (2000), 361–370.

Nir Ailon (nailon@gmail.com), Google Research.

Bernard Chazelle (chazelle@cs.princeton.edu), Department of Computer Science, Princeton University.