Sublinear Algorithms With Sublinear Descriptions

Jay Dharmadhikari

May 2025

1 Introduction

In the world of big data, there is an ever-present need to run algorithms on data that does not fit in a traditional computer's memory. Millions of data points generated concurrently by database systems, networks, need to be processed in real time using limited memory and time. This has led to the development of streaming and sketching algorithms, which accomplish a task approximately on some large dataset while saving on space, time, or both. There are many subfields that contribute results in this area, three main ones are:

- *Streaming* algorithms, which receive a data stream and compute information with limited space
- *Sketching* algorithms, which compress large streams into small "sketches" that preserve information
- *Sublinear* algorithms, which estimate properties of a stream by viewing a small fraction of the data

Algorithms of this nature are often randomized approximation algorithms, so there are many cases where a possible solution to a problem is the most obvious. Here, we examine some results where algorithms that are extremely simple to describe can exhibit powerful properties. We also look at some common tricks for streaming algorithms, such as averaging copies of them to improve accuracy.

We can now look at a formal model. A *stream* is a collection of elements $x_1, \ldots, x_n$ that we aim to compute some function $f(x)$ on. The most common (and useful) model is that the elements are integers or real numbers, and the stream is far, far larger than the space available to the algorithm. This resembles many real-world scenarios where we may want to compute a statistic on an event in real-time when the event occurs too fast to process naively. For example, credit card companies should be able to detect within a fraction of a second if a transaction is potentially fraudulent. Another example is healthcare devices that take many measurements and must elicit a proper signal from the noise.

We will settle for outputting an estimate $\tilde{f}(x)$ on the stream. A nice property is that we would like guarantees on how the error behaves, specifically we would like to provide parameters ε and δ which represent the margin of error and the failure probability, respectively. We follow a *relative error* model, which means we design algorithms such that

$$\mathbb{P}\left((1-\varepsilon)f(x) < \tilde{f}(x) < (1+\varepsilon)f(x)\right) > 1-\delta$$

Generally, we will measure the space complexity of computing $\tilde{f}(x)$, which will usually depend on ε, δ, and sometimes n. These notes are adapted from the notes of Jelani Nelson, Thomas Rothvoss, and others.

2 Morris Counting Algorithm

A great problem to start with is counting. Suppose you are running server analytics for Amazon and you want to count how many requests some servers get per day. Of course, there could be many servers that get millions of requests a day, so keeping an exact counter for every server becomes a hassle. Additionally, you don't *need* the exact count for each server – an estimate within 1-2% would suffice. Formally, we want to design a data structure that maintains a single integer n and supports two operations:

- **increment()**: increment n by 1
- **query**: output (an estimate of) n

The naive solution is to store a counter using $\log n$ bits of memory, and it can be shown that any method for outputting n exactly must take at least $\log n$ bits. However, we can use much less space with an estimate, i.e. **query()** will output an estimate $\tilde{n}$ that satisfies

$$\mathbb{P}(|\tilde{n} - n| > \varepsilon n) < \delta$$

for $0 < \varepsilon, \delta < 1$ that are inputs to the algorithm. We can think of ε as the relative error we want to achieve, and δ as the probability of exceeding that error (i.e. failing to estimate correctly.)
An algorithm given by Morris [Mor78] accomplishes this task and works as follows:

1. Initialize $X \leftarrow 0$.
2. When **increment()** is called, increment X with probability $\frac{1}{2^X}$.
3. When **query()** is called, output $2^X - 1$.

Intuitively, X is approximating $\log n$. We can now analyze the error of Morris' algorithm.

Claim. Let X_n denote the value of X after n calls to **increment()**. Then, $\mathbb{E}[2^{X_n}] = n + 1$.
Proof. By induction on n, where the algorithm obviously satisfies the base case of $n = 0$. Then,

$$\begin{aligned}
\mathbb{E}[2^{X_n}] &= \sum_{j=0}^{\infty} \mathbb{E}[2^{X_n} \mid X_{n-1} = j] \cdot \mathbb{P}(X_{n-1} = j) \\
&= \sum_{j=0}^{\infty} \left(2^{j-1} \left(1 - \frac{1}{2^{j-1}} \right) + \frac{1}{2^{j-1}} \cdot 2^j \right) \cdot \mathbb{P}(X_{n-1} = j) \\
&= \sum_{j=0}^{\infty} (2^{j-1} + 1) \cdot \mathbb{P}(X_{n-1} = j) \\
&= \sum_{j=0}^{\infty} 2^{j-1} \cdot \mathbb{P}(X_{n-1} = j) + \sum_{j=0}^{\infty} \mathbb{P}(X_{n-1} = j) \\
&= \mathbb{E}[2^{X_{n-1}}] + \sum_{j=0}^{\infty} \mathbb{P}(X_{n-1} = j) \\
&= ((n-1) + 1) + 1 \\
&= n + 1
\end{aligned}$$

□

This makes it clear why **query()** outputs $2^X - 1$, as we get an unbiased estimator of n.

Now we want to analyze the variance (that is, examine whether the algorithm achieves ε-error). We first need to bound the variance of our estimator $\tilde{n} = 2^X - 1$:

$$\begin{aligned}\mathrm{Var}[2^X - 1] &= \mathbb{E}[(2^{X_n} - 1)^2] - n^2 \\ &= \mathbb{E}[2^{2X_n}] + 1 - 2\mathbb{E}[2^{X_n}] - n^2\end{aligned}$$

Now we claim $\mathbb{E}[2^{2X_n}] = \frac{3}{2}n(n+1) + 1$ using induction:

$$\begin{aligned}\mathbb{E}[2^{2X_n}] &= \sum_{i=0}^{\infty} 2^{2i} \cdot \mathbb{P}(X_n = i) \\ &= \sum_i 2^{2i} \left(\frac{1}{2^{i-1}} \cdot \mathbb{P}(X_{n-1} = i-1) + \left(1 - \frac{1}{2^i}\right) \cdot \mathbb{P}(X_{n-1} = i) \right) \\ &= \sum_i 2^{i+1} \cdot \mathbb{P}(X_{n-1} = i-1) + \sum_i 2^{2i} \cdot \mathbb{P}(X_{n-1} = i) - \sum_i 2^i \cdot \mathbb{P}(X_{n-1} = i) \\ &= 4\mathbb{E}[2^{X_{n-1}}] + \mathbb{E}[2^{2X_{n-1}}] - \mathbb{E}[2^{X_{n-1}}] \\ &= 3\mathbb{E}[2^{X_{n-1}}] + \mathbb{E}[2^{2X_{n-1}}] \\ &= 3n + \mathbb{E}[2^{2X_{n-1}}] \\ &= 3n + \frac{3}{2}(n-1)n + 1 \\ &= \frac{3n(n+1)}{2} + 1\end{aligned}$$

□

This means $\mathrm{Var}[2^X - 1] = \frac{1}{2}n^2 - \frac{1}{2}n - 1 \leq \frac{1}{2}n^2$. We bound the algorithm's error with Chebyshev's inequality:

$$\begin{aligned}\mathbb{P}(|(2^{X_n} - 1) - n| > \varepsilon n) &\leq \frac{\mathrm{Var}[2^{X_n} - 1]}{\varepsilon^2 n^2} \\ &\leq \frac{1}{2}n^2 \cdot \frac{1}{\varepsilon^2 n^2} \\ &= \frac{1}{2\varepsilon^2}\end{aligned}$$

This is not particularly useful, as the failure probability is only $< 1/2$ when $\varepsilon \geq 1$, in which case zero is a valid estimate!

To reduce the variance of the algorithm, we instantiate s copies of the basic Morris algorithm and averaging their outputs (sum them up and multiply by $1/s$). This doesn't change the expectation, but since variations of random variables add, and multiplying a random variable by a constant $c = 1/s$ causes the variance to grow by c^2, the error probability looks like:

$$\mathbb{P}(|(2^{X_n} - 1) - n| > \varepsilon n) \leq \frac{1}{2s\varepsilon^2} < \delta$$

when $s > 1/(2\varepsilon^2\delta) = \Theta(1/\varepsilon^2\delta)$.

Now we reduce the space complexity's dependence on the failure probability δ from $1/\delta$ to $\log(1/\delta)$. Run t copies of the extended Morris algorithm, each with failure probability $\delta = 1/3$ so $s = \Theta(1/\varepsilon^2)$. Then, we output the median of the t copies. Observe that the expected number of copies that will succeed is $2t/3$. For the overall algorithm to fail via a bad median, half the copies or more need to fail, implying that the algorithm deviates from its expectation by $t/6$. Define:

$$X_i = \begin{cases} 1 & \text{if the } i\text{th copy of the extended Morris algorithm succeeds.} \\ 0 & \text{otherwise.} \end{cases}$$

By the Chernoff bound:

$$\mathbb{P}\left(\sum_i X_i \leq \frac{t}{2}\right) \leq \mathbb{P}\left(\left|\sum_i X_i - \mathbb{E}\left[\sum_i X_i\right]\right| \geq t/6\right) \leq 2e^{-t/3} < \delta$$

when t is $\Theta(\log(1/\delta))$.

It remains to analyze the space complexity of the full algorithm, which is running $st = \Theta(\log(1/\delta)/\varepsilon^2)$ copies of a basic Morris counter. When any of the individual counters X reaches the value $\log(stn/\delta)$, the probability that it is incremented during any call to **increment()** is at most $\delta/(nst)$. Then, the probability that it is incremented at all in the next n calls to **increment()** is at most $\delta/(st)$. By the union bound, with probability $1 - \delta$, none of the st counters ever stores a value larger than $\log(stn\delta)$. Thus, with probability $1 - \delta$, the total space complexity is:

$$O(\frac{1}{\varepsilon^2}\log(1/\delta)\log\log(n/(\varepsilon\delta)))$$

If we fix a constant ε and δ, then the total space complexity is $O(\log\log n)$ with constant probability. This is an exponential boost compared to the naive method! □

Two more recent results of note are:

- In [NY22], Nelson and Yu presented a modification to the algorithm which improves the dependence of the space complexity on δ from $\log(1/\delta)$ to $\log\log(1/\delta)$. They further show that this is optimal up to small factors.
- In [Ade+22], it is shown that there is no amortized benefit when storing multiple counters. In other words, k counters cannot be maintained with asymptotically less memory than k times the space used by a single counter.

We believe these two results essentially resolve approximate counting, although it would be exciting to see further development regarding this problem.

3 Distinct Elements

Here we attack the distinct elements problem with a toolkit similar to the previous section. The distinct elements problem, or the F_0 problem, deals with a stream of integers $x_1, \ldots, x_n \in [n]$ and asks us to output the number of distinct elements in the stream. We want to store a small amount of information that will allow us to approximate this value. Similarly to above, if t is the true number of distinct elements, we want to output an estimate $\tilde{t}$ such that $\mathbb{P}(|\tilde{t} - t| > \varepsilon t) < \delta$. The following algorithm is given in [FM85] by Flalojet and Martin:

1. Initialize a hash function $h : [n] \to [0, 1]$ and a value $X \leftarrow \infty$.
2. For each item in the stream x_i, $X = \min(X, h(x_i))$.
3. Output $1/X - 1$.

Note that the algorithm is highly idealized since storing a truly random hash function h takes $\Omega(n)$ bits and all computations of the algorithm are carried out with infinite-precision real numbers. The intuition is that m is the minimum of t i.i.d. $Unif(0, 1)$ random variables.

Claim. $\mathbb{E}[X] = \frac{1}{t+1}$.

Proof. We use the fact that for a random variable with only non-negative values, $\mathbb{E}[X] = \int_0^\infty \mathbb{P}(X > u)du$. Suppose there are items $x_1, \ldots, x_n$ in the stream corresponding to unique integers $y_1, \ldots, y_n$.

$$\begin{aligned}
\mathbb{E}[X] &= \int_0^\infty \mathbb{P}(X > u)du \\
&= \int_0^\infty \mathbb{P}(\forall x_i, h(x_i) > u)du \\
&= \int_0^\infty \prod_{j=0}^{t} \mathbb{P}(h(y_j) > u)du \\
&= \int_0^1 (1-u)^t du \\
&= \frac{1}{t+1}
\end{aligned}$$

□

Claim. $\mathrm{Var}[X] \leq \frac{1}{(t+1)^2}$.

Proof. We analyze $\mathbb{E}[X^2]$.

$$\begin{aligned}
\mathbb{E}[X^2] &= \int_0^1 \mathbb{P}(X^2 > u)du \\
&= \int_0^1 \mathbb{P}(X > \sqrt{u})du \\
&= \int_0^1 (1-\sqrt{u})^t du \\
&= 2\int_0^1 \lambda^t(1-\lambda)d\lambda \quad \text{(substitution } \lambda = 1 - \sqrt{u}\text{)} \\
&= \frac{2}{(t+1)(t+2)}
\end{aligned}$$

Then $\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \frac{t}{(t+1)^2(t+2)} \leq \frac{1}{(t+1)^2}$. □

Just as before, we will average $s = 1/\varepsilon^2\delta$ copies of the basic FM algorithm together. If Z is the random variable describing this variable, we have $\mathbb{E}[Z] = \frac{1}{t+1}$ and $\text{Var}[Z] \leq \frac{1}{s(t+1)^2}$. Then, by Chebyshev's:

$$\mathbb{P}(|Z - \frac{1}{t+1}| > \frac{\varepsilon}{t+1}) \leq \frac{(t+1)^2}{\varepsilon^2}\frac{1}{s(t+1)^2} = \delta$$

To complete the analysis, we show that the estimation also satisfies error bounds. Since

$$\frac{1}{(1 \pm \varepsilon)^{\frac{1}{t+1}}} - 1 = (1 \pm O(\varepsilon))(t+1) - 1 = (1 \pm O(\varepsilon))t \pm O(\varepsilon)$$

we get that $\mathbb{P}\left(\left|\left(\frac{1}{Z} - 1\right) - t\right| > O(\varepsilon)t\right) < \delta$. □

Finally, following the pattern from the Morris algorithm, we will take the median of $q = 36\ln(2/\delta)$ copies of the extended FM algorithm each with failure probability $1/3$. If $\hat{t}$ is this estimator, then:

Claim. $\mathbb{P}(|\hat{t} - t| > \varepsilon t) < \delta$.
Proof. Let

$$Y_i = \begin{cases} 1 & \text{if the } i\text{th copy of the extended FM algorithm succeeds.} \\ 0 & \text{otherwise.} \end{cases}$$

and let $Y = \sum Y_i$. We have $\mathbb{E}[Y] \geq 2q/3$ as each copy had failure probability $1/3$. We are bounding the probability that the median fails, i.e. at least half of the copies have $Y_i = 0$, or in other words that $Y \leq q/2$. Then we get that

$$\mathbb{P}(Y \leq q/2) \leq \mathbb{P}(|Y - \mathbb{E}[Y]| > \frac{1}{4}\mathbb{E}[Y])$$

By Chernoff, this is at most

$$2e^{-\frac{\left(\frac{1}{4}\right)^2 2s/3}{3}} < \delta$$

which completes the analysis. □

The final space required, ignoring the cost of storing a perfect hash function h, is the space required to store $O(\log(1/\delta)/\varepsilon^2)$ real numbers (each copy of the basic FM stores a single real number, the minimum hash seen.)

It should be noted that the FM algorithm presented here is impractical and relies on access to a hash function that may not always be available. Instead, [Bar+02] gives a strategy that stores k small hash values instead of the minimum and gives the same error guarantees with a much weaker restriction on the hash function. Furthermore, HyperLogLog [Fla+07] is the industry standard for this problem, as it can estimate values in the billions with an error of 2% using < 2 kilobytes of space. It extends the FM algorithm to use a Bloom-Filter-esque structure, and is widely used by Google for distinct element estimation.

4 Approximate Median

Another problem is approximating the median element of a data stream. The generalization of this problem, which has to do with approximating any quantile of a stream, has been studied in-depth by the KLL sketch, GK sketch, and more. However, in our quest for extremely simple algorithms, we find the following procedure for finding an relative ε-approximation of the median:

1. Sample t values (with replacement) from the stream $S = x_1, \ldots, x_n$.
2. Return the median of the sample.

Claim. If $t \geq \frac{7}{\varepsilon^2} \log \frac{2}{\delta}$, then the algorithm outputs a y such that $|\text{rank}(y) - n| \leq \varepsilon n$ with probability $\geq 1 - \delta$.
Proof. Partition S into three groups:

- $S_L = \{x \in S : \text{rank(x)} \leq \frac{n}{2} - \varepsilon n\}$
- $S_M = \{x \in S : \frac{n}{2} - \varepsilon n < \text{rank(x)} < \frac{n}{2} + \varepsilon n\}$
- $S_U = \{x \in S : \text{rank(x)} \geq \frac{n}{2} + \varepsilon n\}$

Observe that if less than $t/2$ elements from each of S_L and S_U are in the sample, then the median of the sample is in S_M, meaning it is ε-approximate to the true median. Let $X_i = 1$ if the i-th sample is in S_L and 0 otherwise. Let $X = \sum_i X_i$. By Chernoff, if $t > \frac{7}{\varepsilon^2} \log \frac{2}{\delta}$,

$$\mathbb{P}(X \geq t/2) \leq \mathbb{P}(X \geq (1+\varepsilon)\mathbb{E}[X]) \leq e^{-\varepsilon^2(1/2-\varepsilon)t/3} \leq \delta/2$$

where the first inequality follows from $\mathbb{E}[X] = t(1/2 - \varepsilon)$ and $(1+\varepsilon)\mathbb{E}[X] = t(1+\varepsilon)(1/2-\varepsilon) \leq t/2$.

Similarly, S_U contributes $\geq t/2$ elements with probability $\leq \delta/2$. By union bound, the algorithm returns a ε-approximate median with probability $1 - \delta$. □

Another algorithm by [MMS14] accomplishes an approximate median using *one* unit of memory:

1. Initialize $x = 0$
2. **for each** s_i in S **do**
   - **if** $s_i > x$ **then** $x = x + 1$
   - **else if** $s_i < x$ **then** $x = x - 1$
3. Output x.

This can be generalized to approximate any quantile described as a fraction h/k:

1. Initialize $x = 0$
2. **for each** s_i in S **do**
   - **if** $s_i > x$ **then** $x = x + 1$ with probability $1 - h/k$
   - **else if** $s_i < x$ **then** $x = x - 1$ with probability h/k
3. Output x.

While we will not analyze the algorithms, they squeeze remarkable properties out of the smallest amount of space possible. The provided reference improves their performance while space usage remains $O(1)$.

5 Reservoir Sampling

Many algorithms in streaming (including some discussed above) rely on taking a sample of size k from a stream of size n, without necessarily knowing n. Reservoir sampling [Vit85] and its variants give efficient ways to do this. The simplest form of reservoir sampling on a stream $x_1, \ldots, x_n$ is as follows:

1. Initialize an array R containing the first k items of the stream $x_1, \ldots, x_k$.
2. For each item x_i, generate a random number $j \in \{1, \ldots, i\}$. If j is in $\{1, \ldots, k\}$, set $R[j]$ to x_i.
3. When the stream is finished, return the items in R.

Claim. Reservoir sampling returns a uniform random subset from the stream.
Proof. By induction on the size of the stream n. If $n = k$, then the algorithm returns the only sample possible. Our inductive hypothesis is that in a stream of n elements, any item is in the n'th reservoir R_n with probability k/n. For the $n+1$'th element x_{n+1}, the probability that it is included in the reservoir R_{n+1} is $k/n + 1$. For any other element x_i already in the reservoir,

$$\begin{aligned}\mathbb{P}(x_i \in R_{n+1}) &= \mathbb{P}(x_i \in R_{n+1} | x_i \in R_n) \cdot \mathbb{P}(x_i \in R_n) \\ &= (1 - \frac{1}{n+1}) \cdot \frac{k}{n} \\ &= \frac{k}{n+1}\end{aligned}$$

Thus, after item x_{n+1} is inserted, every item has a $k/n + 1$ chance of being in the reservoir. By induction, the claim holds. □
It may also be useful to sample with replacement. For this, [Par+04] provides a simple adjustment to the scheme above:

1. Initialize reservoir R as empty.
2. For each item x_i, flip k coins where each has probability of heads $1/i$. For each heads, remove a uniform random item from R. After all coins have been flipped, add copies of x_i into R equal to the total number of heads.
3. When the stream is finished, return the items in R.

The proof is similar to above, the reader may examine [Par+04] for more.

Reservoir sampling is an interesting technique from a mathematical, theoretical, and practical perspective. Its ideas have been connected to several other sampling techniques from probability theory, most notably the Fisher-Yates shuffle for drawing subsets of cards from an infinite deck. A natural extension of the problem is *weighted* random sampling, where each item is given a weight corresponding to the probability with which it should be included in the sample. This is a key component of introducing fairness in machine learning algorithms, where training data usually comes in a stream but must be sampled with precision. Because of its importance in streaming algorithms, there have been many improvements to reservoir sampling that make it suitable for practical use.

6 Johnson-Lindenstrauss Lemma

The JL Lemma is one of the most fundamental results in approximations algorithms, with its applications spanning across fields. It shows that n points in high dimensional Euclidean space can be mapped into an $O(\log n/\varepsilon^2)$-dimensional space such that the distance between any two points only changes by at most $(1 \pm \varepsilon)$. Not only does such a map exist, but one can easily be found in randomized polynomial time.

Theorem. For $0 < \varepsilon < 1$ and any set of points $x_1, \ldots, x_n \in \mathbb{R}^d$ there is a linear map $T : \mathbb{R}^d \to \mathbb{R}^k$ with $k = \Theta(\frac{\log n}{\varepsilon^2})$ such that for all $i, j \in [n]$:

$$(1-\varepsilon)||x_i - x_j||_2 \leq ||T(x_i) - T(x_j)||_2 \leq (1+\varepsilon)||x_i - x_j||_2$$

Before we can prove the theorem, we need two facts:

Fact 1. Let $\alpha, \beta \in \mathbb{R}$ and let $g_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $g_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ be two independent Gaussian random variables. Then $\alpha g_1 + \beta g_2$ has distribution $\mathcal{N}(\alpha\mu_1 + \beta\mu_2, \alpha^2\sigma_1^2 + \beta\sigma_2^2)$.

Fact 2. Let $X = (X_1, \ldots, X_d)$ be a d-dimensional vector with all coordinates drawn independently from $\mathcal{N}(0, 1)$ and let $0 < \varepsilon < 1$. Then,

$$\mathbb{P}(|||X||_2^2 - d| \geq \varepsilon d) \leq 2\exp(-\frac{d\varepsilon^2}{8})$$

We skip the proofs of these facts, but the intuition is that the sum of independent Gaussians is Gaussian and that Gaussians are strongly concentrated around their mean. Now we are ready to prove the theorem.

We define our map T as $T(x) = Ax$ where $A \in \mathbb{R}^{k \times d}$ is a matrix where all entries are chosen independently as $A_{ij} \sim \mathcal{N}(0, \frac{1}{k})$. We want to show that T works with high probability for an appropriate choice of k, so we start by examining how Ax behaves for a fixed vector $x \in \mathbb{R}^d$.

Claim. For a fixed $x \in \mathbb{R}^d$, Ax is a random vector where each coordinate is drawn independently from $\mathcal{N}(0, \frac{||x||_2^2}{k})$.

Proof. The i-th coordinate of Ax is

$$(Ax)_i = \sum_{j=1}^{k} A_{ij}x_j \sim \mathcal{N}(0, \frac{||x||_2^2}{k})$$

which is from **Fact 1**. The independence follows from the fact that the rows of A are independent. Next, for any $x \in \mathbb{R}^d$, we have

$$\mathbb{E}[||Ax||_2^2] = \sum_{i=1}^{k} \mathbb{E}[(Ax)_i^2] = k \cdot \frac{||x||_2^2}{k} = ||x||_2^2$$

which means that our map Ax maintains the length of x in expectation. Next, we show that it maintains the length of x with *high probability*.

Claim. For any $x \in \mathbb{R}^d$, $\mathbb{P}(|||Ax||_2 - ||x||_2| > \varepsilon||x||_2) \leq \exp(-\varepsilon^2 k/8)$.

Proof. The claim is invariant under scaling x, so we can assume $||x||_2 = 1$. Then,

$$\begin{aligned}
\mathbb{P}(1-\varepsilon \leq ||Ax||_2 \leq 1+\varepsilon) &= \mathbb{P}((1-\varepsilon)^2 \leq ||Ax||_2^2 \leq (1+\varepsilon)^2) \\
&\geq \mathbb{P}(1-\varepsilon \leq ||Ax||_2^2 \leq 1+\varepsilon) \\
&\geq 1 - 2\exp(-\frac{\varepsilon^2 k}{8})
\end{aligned}$$

The first inequality follows from the fact that $(1+\varepsilon)^2 \geq 1+\varepsilon$ and $(1-\varepsilon)^2 \leq 1-\varepsilon$ for all $0 \leq \varepsilon \leq 1$. The second inequality follows from the fact that Ax is a vector with entries drawn from $\mathcal{N}(0, 1)$, then scaled by $1/\sqrt{k}$, so we can apply **Fact 2**.

Now we are ready to prove the theorem. We apply a union bound on all difference vectors $x_i - x_j$ between points and the claim above:

$$\begin{aligned}\mathbb{P}(\exists i \neq j : |\frac{||Ax_i - Ax_j||_2}{||x_i - x_j||_2} - 1| > \varepsilon) &\leq \sum_{1 \leq i < j \leq n} \mathbb{P}(\frac{||A(x_i - x_j)||_2}{||x_i - x_j||_2} > \varepsilon) \\ &\leq n^2 \cdot 2\exp(\varepsilon^2 k/8) \\ &= \frac{1}{2n}\end{aligned}$$

when we set $k = 8\ln(4n^3)/\varepsilon^2$. □

We can redo the projection $O(n)$ times to arbitrarily reduce the failure probability. There are many such matrices A that give a projection with the same property. A quite surprising result from [Ach03] is that a matrix with boolean entries drawn from $\{-1, 1\}$ works as well!

The applications of JL transforms span:

- Dimensionality reduction for nearest neighbor search
- Compressed sensing
- Machine learning preprocessing
- Graph sparsification

and more.

References

[Ach03] Dimitris Achlioptas. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins". In: *Journal of Computer and System Sciences* 66.4 (2003). Special Issue on PODS 2001, pp. 671–687. DOI: 10.1016/S0022-0000(03)00025-4. URL: https://www.sciencedirect.com/science/article/pii/S0022000003000254.

[Ade+22] Ishaq Aden-Ali et al. "On the Amortized Complexity of Approximate Counting". In: *CoRR* abs/2211.03917 (2022). URL: https://arxiv.org/abs/2211.03917.

[Bar+02] Ziv Bar-Yossef et al. "Counting Distinct Elements in a Data Stream". In: *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*. 2002, pp. 1–10.

[Fla+07] Philippe Flajolet et al. "HyperLogLog: The Analysis of a Near-Optimal Cardinality Estimation Algorithm". In: *Proceedings of the 2007 Conference on Analysis of Algorithms (AofA 07)*. 2007, pp. 127–146. URL: https://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf.

[FM85] Philippe Flajolet and G. Nigel Martin. "Probabilistic Counting Algorithms for Data Base Applications". In: *Journal of Computer and System Sciences* 31.2 (1985), pp. 182–209.

[MMS14] Qiang Ma, S. Muthukrishnan, and Mark Sandler. "Frugal Streaming for Estimating Quantiles: One (or two) memory suffices". In: *arXiv preprint arXiv:1407.1121* (2014). URL: https://arxiv.org/abs/1407.1121.

[Mor78] Robert Morris. "Counting large numbers of events in small registers". In: *Communications of the ACM* 21.10 (1978), pp. 840–842.

[NY22] Jelani Nelson and Huacheng Yu. "Optimal Bounds for Approximate Counting". In: *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. ACM, 2022, pp. 119–127. DOI: 10.1145/3517804.3526225. URL: https://doi.org/10.1145/3517804.3526225.

[Par+04] B. H. Park et al. "Reservoir-Based Random Sampling with Replacement from Data Stream". In: *Proceedings of the 2004 SIAM International Conference on Data Mining (SDM)*. SIAM, 2004, pp. 492–496. DOI: 10.1137/1.9781611972740.53. URL: https://epubs.siam.org/doi/10.1137/1.9781611972740.53.

[Vit85] Jeffrey Scott Vitter. "Random Sampling with a Reservoir". In: *ACM Transactions on Mathematical Software* 11.1 (1985), pp. 37–57. DOI: 10.1145/3147.3165. URL: https://dl.acm.org/doi/10.1145/3147.3165.