Locality-Sensitive Hashing (LSH) for scalable clustering in single-cell RNA sequencing

Paolo Carnevali

Chan-Zuckerberg Initiative

November 2017

Single-cell RNA sequencing

- In single-cell RNA sequencing, one measures gene expression levels for individual cells.

Single-cell RNA sequencing

- In single-cell RNA sequencing, one measures gene expression levels for individual cells.
- The gene expression levels for a single cell form a *cell expression vector* with one element per gene, which can be:
    1. A raw read count (number of sequencing reads that mapped to the gene).
    2. A read count normalized in some way.
    3. A read count transformed in some way -for example, via logarithmic tranformations.

Single-cell RNA sequencing

- In single-cell RNA sequencing, one measures gene expression levels for individual cells.
- The gene expression levels for a single cell form a *cell expression vector* with one element per gene, which can be:
  1. A raw read count (number of sequencing reads that mapped to the gene).
  2. A read count normalized in some way.
  3. A read count transformed in some way -for example, via logarithmic tranformations.
- The cell expression vectors for all the cells in an experiment form an *expression matrix*.

Clustering for single-cell RNA sequencing

- The goal of clustering is to find groups (clusters) of cells that have ”similar” expression vectors according to some definition of similarity.

Clustering for single-cell RNA sequencing

- The goal of clustering is to find groups (clusters) of cells that have ”similar” expression vectors according to some definition of similarity.
- In a grossly oversimplified picture of cell populations, each cluster corresponds to a ”cell type”.

Clustering for single-cell RNA sequencing

- The goal of clustering is to find groups (clusters) of cells that have ”similar” expression vectors according to some definition of similarity.
- In a grossly oversimplified picture of cell populations, each cluster corresponds to a ”cell type”.
- In reality, things are much more complex, and clustering is only the first step in the process of extracting information from cell expression data.

Clustering for single-cell RNA sequencing

- The goal of clustering is to find groups (clusters) of cells that have ”similar” expression vectors according to some definition of similarity.
- In a grossly oversimplified picture of cell populations, each cluster corresponds to a ”cell type”.
- In reality, things are much more complex, and clustering is only the first step in the process of extracting information from cell expression data.
- This presentation will not cover clustering. It will only be concerned with the computational step necessary to rapidly find pairs of cells that have sufficiently ”similar” expression vectors.

Scaling up

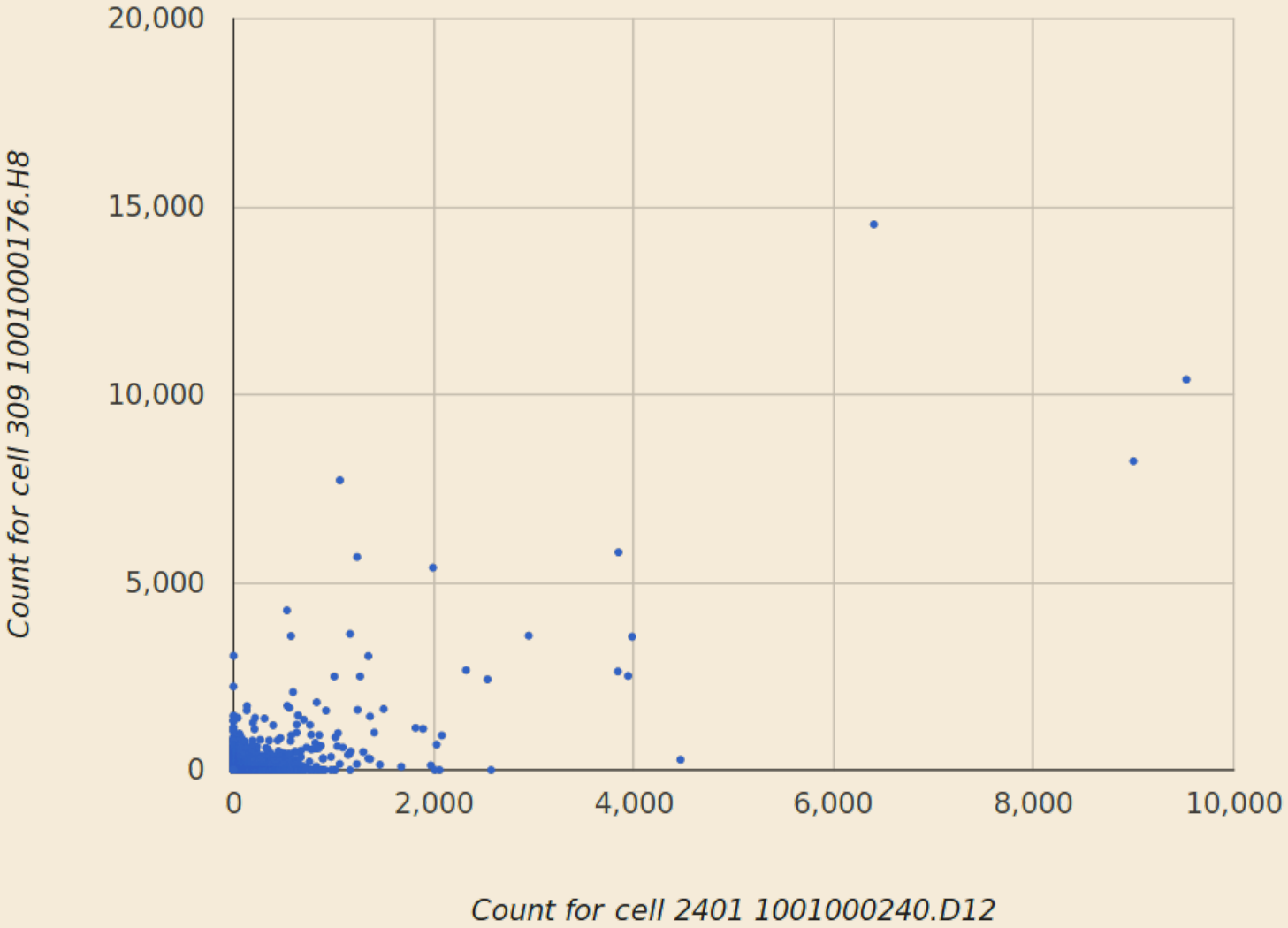
- We will call n the number of genes in an experiment.
- n is mostly determined by the species, ($n \approx 2 \times 10^4$ for human cells).

Scaling up

- We will call n the number of genes in an experiment.
- n is mostly determined by the species, ($n \approx 2 \times 10^4$ for human cells).
- We will call N the number of cells in an experiment.
- Today, $10^2 \lesssim N \lesssim 10^5$ typically, but experiment size is increasing.
- For the Human Cell Atlas, N is expected to exceed $\approx 10^8$.
- We need to be able to scale up the number of cells but not the number of genes.

Comparing cell expression vectors

Given two cells with expression vectors x and y, we need a definition of the similarity between the two cells.



Cell similarity

- The most commonly used measure of cell similarity is the Pearson Correlation Coefficient.

Cell similarity

- The most commonly used measure of cell similarity is the Pearson Correlation Coefficient.
- For two cells with expression vectors x and y, it is defined as the ratio:

$$r(x, y) = \frac{covariance(x, y)}{\sigma(x)\sigma(y)},$$

where $\sigma(x)$ and $\sigma(y)$ are the standard deviations of the components of x and y (including the zero components!).

Cell similarity

- The most commonly used measure of cell similarity is the Pearson Correlation Coefficient.
- For two cells with expression vectors x and y, it is defined as the ratio:

$$r(x, y) = \frac{covariance(x, y)}{\sigma(x)\sigma(y)},$$

  where $\sigma(x)$ and $\sigma(y)$ are the standard deviations of the components of x and y (including the zero components!).
- For the cells shown in the previous slide, $r \approx 0.71$.

Interpretation of the Pearson Correlation Coefficient as a cosine

Interpretation of the Pearson Correlation Coefficient as a cosine

- Define the averages of the components of x and y, $\bar{x}$ and $\bar{y}$.

Interpretation of the Pearson Correlation Coefficient as a cosine

- Define the averages of the components of x and y, $\bar{x}$ and $\bar{y}$.
- Define shifted expression vectors with zero mean and sum $X = x - \bar{x}$, $Y = y - \bar{y}$.

Interpretation of the Pearson Correlation Coefficient as a cosine

- Define the averages of the components of x and y, $\bar{x}$ and $\bar{y}$.
- Define shifted expression vectors with zero mean and sum $X = x - \bar{x}$, $Y = y - \bar{y}$.
- It can be shown that the similarity between the two cells, $r(x, y)$, equals $\cos\theta$, where θ is the angle between the X and Y vectors.

Finding pairs of similar cells

- In most types of analysis, including clustering, we need to find pairs of cells that have high similarity.

Finding pairs of similar cells

- In most types of analysis, including clustering, we need to find pairs of cells that have high similarity.
- The simplest and most common way to do this is to directly compute the similarity for all possible pairs of cells, and store the pairs that have similarity above the desired threshold.

Finding pairs of similar cells

- In most types of analysis, including clustering, we need to find pairs of cells that have high similarity.
- The simplest and most common way to do this is to directly compute the similarity for all possible pairs of cells, and store the pairs that have similarity above the desired threshold.
- The computation for each pair can be performed in time proportional to the total number of non-zero expression vector entries for the two cells.

Finding pairs of similar cells

- In most types of analysis, including clustering, we need to find pairs of cells that have high similarity.
- The simplest and most common way to do this is to directly compute the similarity for all possible pairs of cells, and store the pairs that have similarity above the desired threshold.
- The computation for each pair can be performed in time proportional to the total number of non-zero expression vector entries for the two cells.
- With optimized C++ code and using sparse representations for the cell expression vectors, this typically takes $\approx 20\mu s$ per pair.

Finding pairs of similar cells

- In most types of analysis, including clustering, we need to find pairs of cells that have high similarity.
- The simplest and most common way to do this is to directly compute the similarity for all possible pairs of cells, and store the pairs that have similarity above the desired threshold.
- The computation for each pair can be performed in time proportional to the total number of non-zero expression vector entries for the two cells.
- With optimized C++ code and using sparse representations for the cell expression vectors, this typically takes $\approx 20\mu s$ per pair.
- The number of pairs to be considered increases with the square of the number of cells, so the total time required is $\approx 10s$ for 1000 cells (good), but ≈ 4 months for 10^6 cells (bad).

Locality-Sensitive Hashing (LSH)

- LSH is a probabilistic approach to finding pairs of similar objects in a large set.

Locality-Sensitive Hashing (LSH)

- LSH is a probabilistic approach to finding pairs of similar objects in a large set.
- It can be much faster than direct methods, but it does not guarantee that all pairs of similar objects will be found.
- However, the probability that a pair will be found is a non-decreasing function of the similarity between the items in the pair.

Locality-Sensitive Hashing (LSH)

- LSH is a probabilistic approach to finding pairs of similar objects in a large set.
- It can be much faster than direct methods, but it does not guarantee that all pairs of similar objects will be found.
- However, the probability that a pair will be found is a non-decreasing function of the similarity between the items in the pair.
- I will explain later the reason for the name Locality-Sensitive Hashing.

Locality-Sensitive Hashing history and references

Locality-Sensitive Hashing history and references

- Originally developed for the Alta Vista search engine, for the special case of Jaccard similarity (MinHash algorithm), by Broder (1997), *Compression and Complexity of Sequences: Proceedings*.

Locality-Sensitive Hashing history and references

- Originally developed for the Alta Vista search engine, for the special case of Jaccard similarity (MinHash algorithm), by Broder (1997), *Compression and Complexity of Sequences: Proceedings*.
- Mathematical foundation by Indyk and Motwani (1998), *Proceedings of 30th Symposium on Theory of Computing*. This work also introduced the name *Locality-Sensitive Hashing*.

Locality-Sensitive Hashing history and references

- Originally developed for the Alta Vista search engine, for the special case of Jaccard similarity (MinHash algorithm), by Broder (1997), *Compression and Complexity of Sequences: Proceedings*.
- Mathematical foundation by Indyk and Motwani (1998), *Proceedings of 30th Symposium on Theory of Computing*. This work also introduced the name *Locality-Sensitive Hashing*.
- Since then used in a variety of applications.

Locality-Sensitive Hashing history and references

- Originally developed for the Alta Vista search engine, for the special case of Jaccard similarity (MinHash algorithm), by Broder (1997), *Compression and Complexity of Sequences: Proceedings*.
- Mathematical foundation by Indyk and Motwani (1998), *Proceedings of 30th Symposium on Theory of Computing*. This work also introduced the name *Locality-Sensitive Hashing*.
- Since then used in a variety of applications.
- An excellent and intuitive presentation is in Chapter 3 of Leskovec, Rajaraman, and Ullman (2010), ”Mining of Massive DataSets”, *Cambridge University Press*, also freely available on the Internet by arrangement with the publisher (see especially sections 3.5.4 and 3.7.2).

Locality-Sensitive Hashing for the cosine similarity

- Although a general mathematical theory for LSH exists, the specifics are different depending on which distance/similarity measure is considered.

Locality-Sensitive Hashing for the cosine similarity

- Although a general mathematical theory for LSH exists, the specifics are different depending on which distance/similarity measure is considered.
- The original and most used formulation is the one for Jaccard similarity (MinHash algorithm).

Locality-Sensitive Hashing for the cosine similarity

- Although a general mathematical theory for LSH exists, the specifics are different depending on which distance/similarity measure is considered.
- The original and most used formulation is the one for Jaccard similarity (MinHash algorithm).
- In this presentation, I will focus on the case where our items are vectors in a Cartesian n-dimensional space and the relevant similarity measure is the cosine of the angle between two of our vectors X and Y.

Locality-Sensitive Hashing for the cosine similarity

- Although a general mathematical theory for LSH exists, the specifics are different depending on which distance/similarity measure is considered.
- The original and most used formulation is the one for Jaccard similarity (MinHash algorithm).
- In this presentation, I will focus on the case where our items are vectors in a Cartesian n-dimensional space and the relevant similarity measure is the cosine of the angle between two of our vectors X and Y.
- This is the case of interest to us, where X and Y are the shifted expression vectors of two cells.

A key observation

- Consider two vectors X and Y in our n-dimensional space at an angle θ. For simplicity we can visualize our vectors as starting at the origin.
- Randomly pick a hyperplane through the origin in our n-dimensional space.
- What is the probability that our vectors X and Y lie on the same side of the hyperplane?

A key observation

- Consider two vectors X and Y in our n-dimensional space at an angle θ. For simplicity we can visualize our vectors as starting at the origin.
- Randomly pick a hyperplane through the origin in our n-dimensional space.
- What is the probability that our vectors X and Y lie on the same side of the hyperplane?
- A bit of reflection reveals that the probability is $p(\theta) = 1 - \frac{\theta}{\pi}$, assuming θ is measured in radians.

A key observation

- Consider two vectors X and Y in our n-dimensional space at an angle θ. For simplicity we can visualize our vectors as starting at the origin.
- Randomly pick a hyperplane through the origin in our n-dimensional space.
- What is the probability that our vectors X and Y lie on the same side of the hyperplane?
- A bit of reflection reveals that the probability is $p(\theta) = 1 - \frac{\theta}{\pi}$, assuming θ is measured in radians.
- For example, if $X = Y$, $\theta = 0$, the two vectors certainly lie on the same side of the hyperplane, and in fact the above formula gives $p = 1$.

A key observation

- Consider two vectors X and Y in our n-dimensional space at an angle θ. For simplicity we can visualize our vectors as starting at the origin.
- Randomly pick a hyperplane through the origin in our n-dimensional space.
- What is the probability that our vectors X and Y lie on the same side of the hyperplane?
- A bit of reflection reveals that the probability is $p(\theta) = 1 - \frac{\theta}{\pi}$, assuming θ is measured in radians.
- For example, if $X = Y$, $\theta = 0$, the two vectors certainly lie on the same side of the hyperplane, and in fact the above formula gives $p = 1$.
- Conversely, if $X = -Y$, $\theta = \pi$, the two vectors never lie on the same side of the hyperplane, and in fact the above formula gives $p = 0$.

Not convinced?

The following figure from the book referenced above might help:

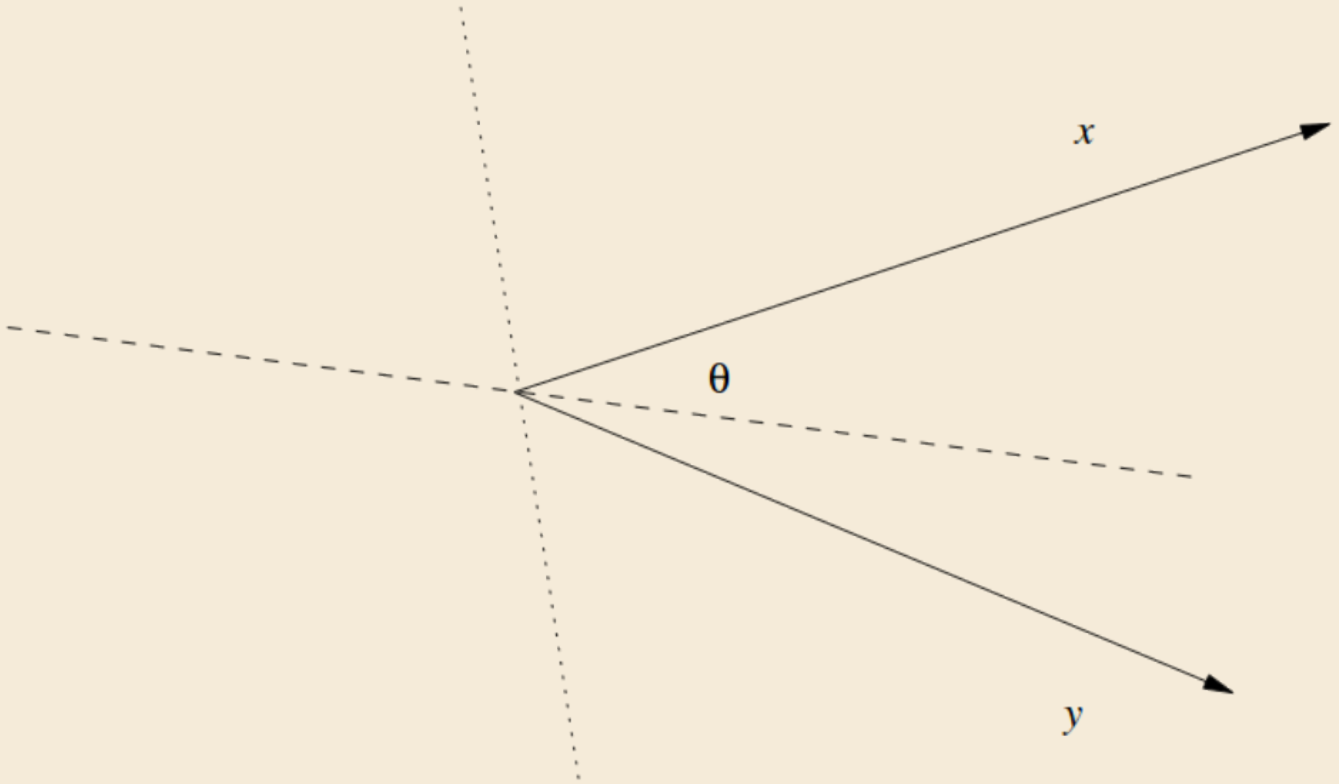


Figure 3.12: Two vectors make an angle θ

Many random hyperplanes

- What happens if we repeat this using m random hyperplanes?

Many random hyperplanes

- What happens if we repeat this using m random hyperplanes?
- For each of the hyperplanes, the two vectors are on the same side with probability $p(\theta) = 1 - \frac{\theta}{\pi}$.

Many random hyperplanes

- What happens if we repeat this using m random hyperplanes?
- For each of the hyperplanes, the two vectors are on the same side with probability $p(\theta) = 1 - \frac{\theta}{\pi}$.
- Call k the number of hyperplanes that have the two vectors on the same side.

Many random hyperplanes

- What happens if we repeat this using m random hyperplanes?
- For each of the hyperplanes, the two vectors are on the same side with probability $p(\theta) = 1 - \frac{\theta}{\pi}$.
- Call k the number of hyperplanes that have the two vectors on the same side.
- Each of the random hyperplanes is picked independently, so k has a binomial distribution with m tries and probability $p = p(\theta)$:

$$P(k) = \binom{m}{k} p^k (1-p)^{m-k}$$

An unbiased estimator of θ

- The mean of the binomial distribution gives the expected value of k:

$$\bar{k} = pm = \left(1 - \frac{\theta}{\pi}\right) m$$

An unbiased estimator of θ

- The mean of the binomial distribution gives the expected value of k:

$$\bar{k} = pm = \left(1 - \frac{\theta}{\pi}\right) m$$

- This gives an unbiased estimator for $\bar{\theta}$:

$$\bar{\theta} = \pi \left(1 - \frac{k}{m}\right)$$

How accurate is the estimator?

- The standard deviation of the binomial distribution is

$$\sigma(k) = \sqrt{mp(1-p)}$$

- Therefore the standard deviation of the θ estimate is

$$\sigma(\bar{\theta}) = \frac{\pi}{m}\sqrt{mp(1-p)} = \pi\sqrt{\frac{p(1-p)}{m}}$$

How accurate is the estimator?

- The standard deviation of the binomial distribution is

$$\sigma(k) = \sqrt{mp(1-p)}$$

- Therefore the standard deviation of the θ estimate is

$$\sigma(\bar{\theta}) = \frac{\pi}{m}\sqrt{mp(1-p)} = \pi\sqrt{\frac{p(1-p)}{m}}$$

- The estimator becomes better as m increases.
- We can make the estimator as accurate as we like, at the cost of increasing m.
- However the error goes down only as $O(1/\sqrt{m})$.

The estimator for $r(x, y) = \cos\theta$

- We have an estimator for θ, but we need an estimator for the similarity between two cells $r(x, y) = \cos\theta$.

The estimator for $r(x, y) = \cos\theta$

- We have an estimator for θ, but we need an estimator for the similarity between two cells $r(x, y) = \cos\theta$.
- We can use $\bar{r} = cos\bar{\theta}$, but a non-linear transformation is involved, and therefore:
  - The $\bar{r}$ estimator is not unbiased.
  - The computation of its standard deviation is not straighforward.

The estimator for $r(x, y) = \cos\theta$

- We have an estimator for θ, but we need an estimator for the similarity between two cells $r(x, y) = \cos\theta$.
- We can use $\bar{r} = cos\bar{\theta}$, but a non-linear transformation is involved, and therefore:
  - The $\bar{r}$ estimator is not unbiased.
  - The computation of its standard deviation is not straighforward.
- However, when $m \to \infty$:
  - The estimator $\bar{r}$ becomes unbiased.
  - Its standard deviation is given by

$$\sigma(\bar{r}) = \left|\frac{dr}{d\theta}\right| \sigma(\bar{\theta}) = \pi \sin\theta \sqrt{\frac{p(1-p)}{m}}$$

Properties of the $\bar{r}$ estimator

Properties of the $\bar{r}$ estimator

- It is unbiased for large m.

Properties of the $\bar{r}$ estimator

- It is unbiased for large m.
- Its standard deviation is independent of the number of genes n and decreases when the number of hyperplanes m increases.

Properties of the $\bar{r}$ estimator

- It is unbiased for large m.
- Its standard deviation is independent of the number of genes n and decreases when the number of hyperplanes m increases.
- Its distribution becomes Gaussian at large m.

Properties of the $\bar{r}$ estimator

- It is unbiased for large m.
- Its standard deviation is independent of the number of genes n and decreases when the number of hyperplanes m increases.
- Its distribution becomes Gaussian at large m.
- Its standard deviation goes to zero for high similarity, when $p \to 1$, $\theta \to 0$.

Properties of the $\bar{r}$ estimator

- It is unbiased for large m.
- Its standard deviation is independent of the number of genes n and decreases when the number of hyperplanes m increases.
- Its distribution becomes Gaussian at large m.
- Its standard deviation goes to zero for high similarity, when $p \to 1$, $\theta \to 0$.
- For a given m, its standard deviation is highest for $p = 1/2$, when $\theta = \pi/2$, the similarity is zero, and the standard deviation becomes $\pi/(2\sqrt{m})$.

Properties of the $\bar{r}$ estimator

- It is unbiased for large m.
- Its standard deviation is independent of the number of genes n and decreases when the number of hyperplanes m increases.
- Its distribution becomes Gaussian at large m.
- Its standard deviation goes to zero for high similarity, when $p \to 1$, $\theta \to 0$.
- For a given m, its standard deviation is highest for $p = 1/2$, when $\theta = \pi/2$, the similarity is zero, and the standard deviation becomes $\pi/(2\sqrt{m})$.
- For example, for $m = 1024$, the maximum standard deviation is $\pi/64 \approx 0.049$

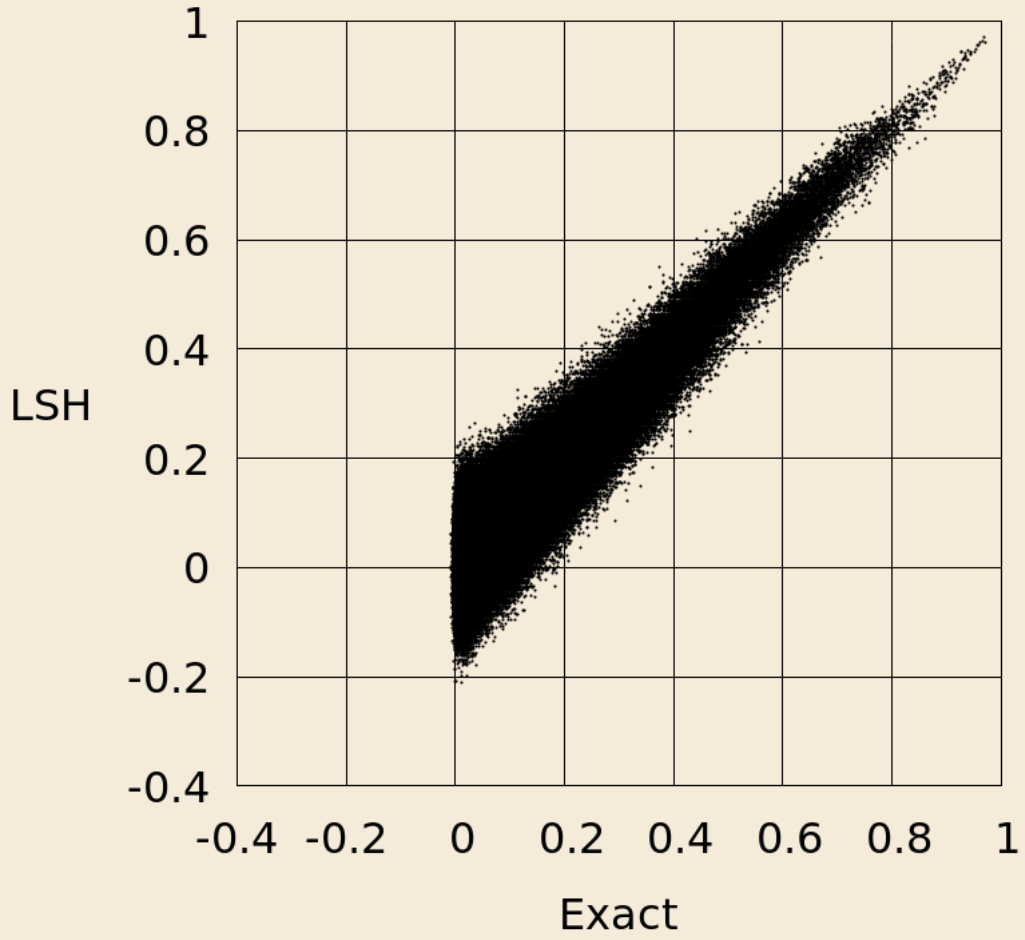
How well does it work in practice?

- The next slide shows a comparison of $r(x, y)$ computed exactly with the result of the estimator, using LSH with $m = 1024$ random hyperplanes.

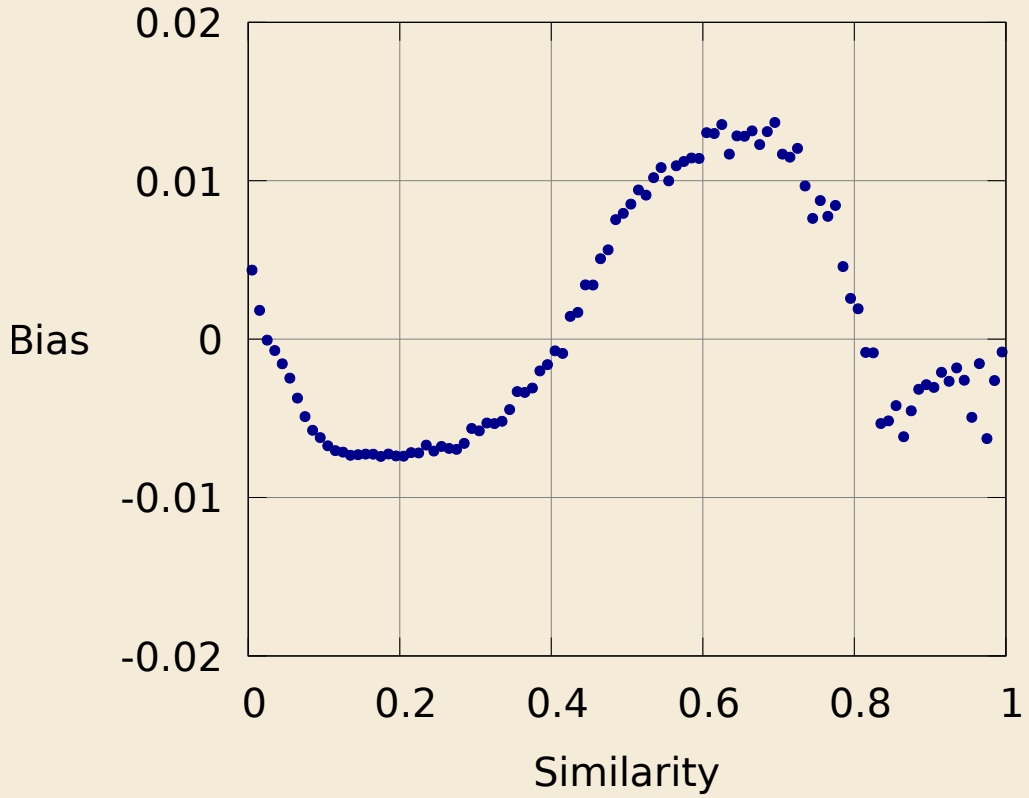
How well does it work in practice?

- The next slide shows a comparison of $r(x, y)$ computed exactly with the result of the estimator, using LSH with $m = 1024$ random hyperplanes.
- Scatter plot of $\approx 6 \times 10^5$ cell pairs, randomly downsampled from a run with $\approx 4 \times 10^3$ cells (data courtesy of S. Darmanis, publication pending).
- Horizontal axis: exact cell similarity $r(x, y)$.
- Vertical axis: value computed using the $\bar{r}$ estimator.

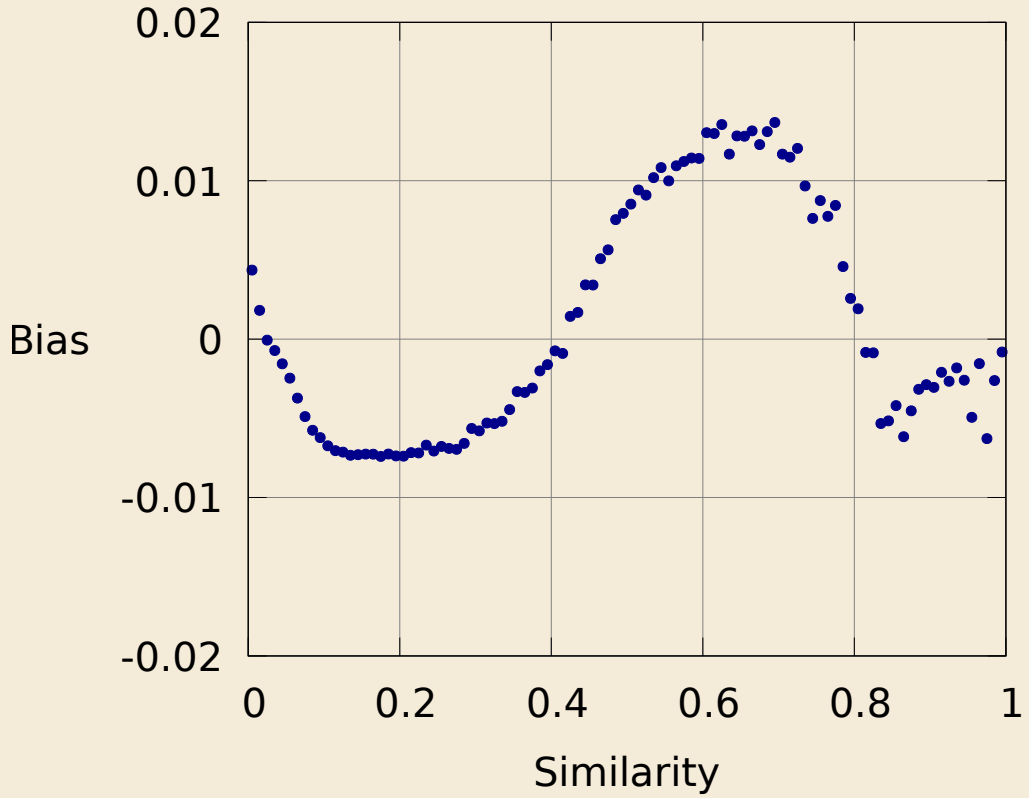
Similarity computation: estimated versus exact, $m = 1024$



Estimator bias, $m = 1024$

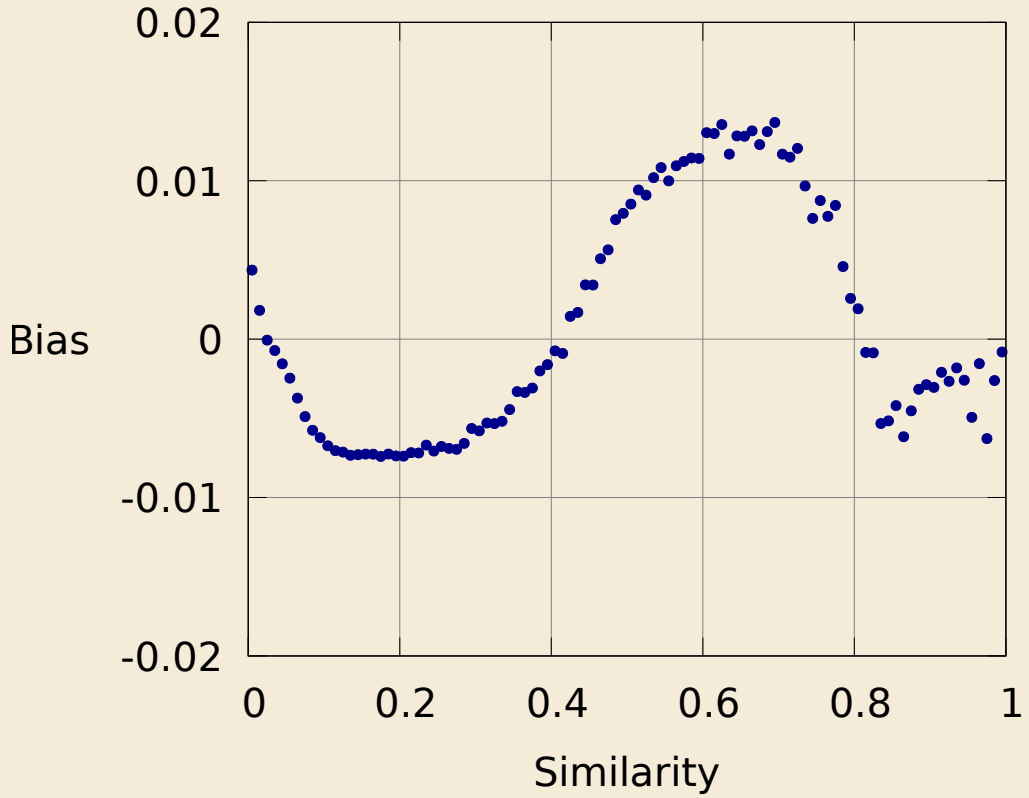


Estimator bias, $m = 1024$



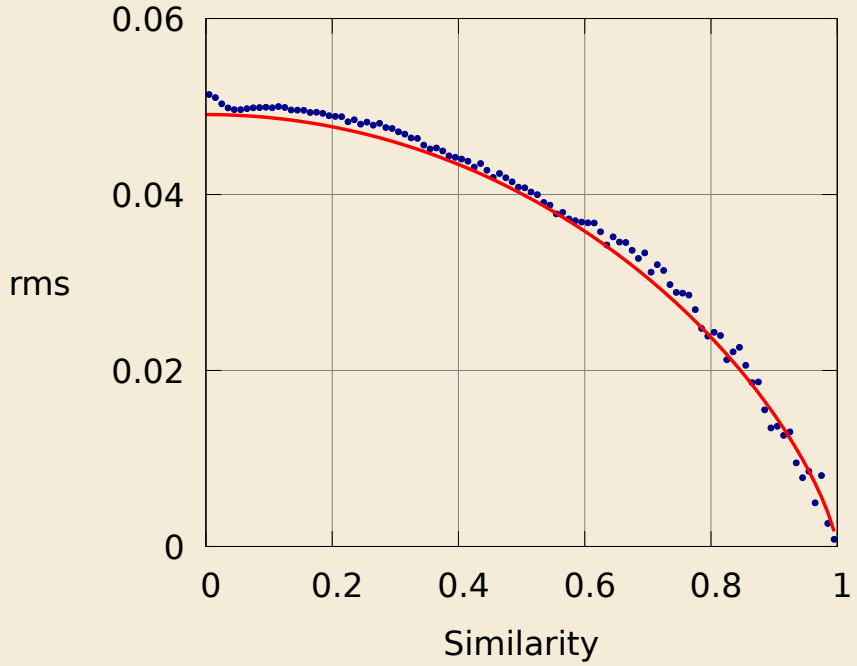
- The theory only guarantees a zero-bias estimator when $m \to \infty$.

Estimator bias, $m = 1024$



- The theory only guarantees a zero-bias estimator when $m \to \infty$.
- For $m = 1024$ the bias is negligible for all practical purposes.

Estimator standard deviation, $m = 1024$



- The actual standard deviation of the estimator (blue points) is in good agreement with the theoretical prediction (red line).
- The error incurred in using the estimator for $m = 1024$ is acceptable for clustering and other analyses.

”All pairs” algorithm for computing cell similarities using LSH

”All pairs” algorithm for computing cell similarities using LSH

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.

”All pairs” algorithm for computing cell similarities using LSH

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. For each cell, compute the sign of the scalar product of the shifted expression vector with each of the m random vectors and store it as a bit vector (the cell *LSH signature*).

”All pairs” algorithm for computing cell similarities using LSH

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. For each cell, compute the sign of the scalar product of the shifted expression vector with each of the m random vectors and store it as a bit vector (the cell *LSH signature*).
3. For each pair of cells, count the number k of signature bits that are identical for the two cells. This gives the number of hyperplanes k, out of the m, for which the two cells are on the same side of the hyperplane. Given k, we can estimate the similarity for the pair as $\bar{r} = \cos\bar{\theta}$, $\bar{\theta} = \pi(1 - k/m)$.

”All pairs” algorithm for computing cell similarities using LSH

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. For each cell, compute the sign of the scalar product of the shifted expression vector with each of the m random vectors and store it as a bit vector (the cell *LSH signature*).
3. For each pair of cells, count the number k of signature bits that are identical for the two cells. This gives the number of hyperplanes k, out of the m, for which the two cells are on the same side of the hyperplane. Given k, we can estimate the similarity for the pair as $\bar{r} = \cos\bar{\theta}$, $\bar{\theta} = \pi(1 - k/m)$.

   This algorithm still has complexity $O(N^2)$, because step 3 is looping over all pairs. We will see later how we can do better, again using LSH.

Efficient implementation of step 3

Using hardware POPCOUNT instruction (count number of set bits), each loop iteration processes 64 bits:

```
for(uint64_t i=0; i<wordCount; i++)
    mismatchCount += __builtin_popcountll(x[i] ^ y[i]);
```

Efficient implementation of step 3

Using hardware POPCOUNT instruction (count number of set bits), each loop iteration processes 64 bits:

```
for(uint64_t i=0; i<wordCount; i++)
    mismatchCount += __builtin_popcountll(x[i] ^ y[i]);
```

Using ”g++ -O3 -msse4.2”, the loop compiles as:

```
.L3:    movq    (%rdi,%r8,8), %rcx
        xorq    (%rsi,%r8,8), %rcx
        addq    $1, %r8
        popcntq %rcx, %rcx
        addq    %rcx, %rax
        cmpq    %r8, %rdx
        jne     .L3
```

Efficient implementation of step 3

Using hardware POPCOUNT instruction (count number of set bits), each loop iteration processes 64 bits:

```
for(uint64_t i=0; i<wordCount; i++)
    mismatchCount += __builtin_popcountll(x[i] ^ y[i]);
```

Using "g++ -O3 -msse4.2", the loop compiles as:

```
.L3:    movq    (%rdi,%r8,8), %rcx
        xorq    (%rsi,%r8,8), %rcx
        addq    $1, %r8
        popcntq %rcx, %rcx
        addq    %rcx, %rax
        cmpq    %r8, %rdx
        jne     .L3
```

- With data in cache, the entire loop runs in $\approx 13ns \approx 30$ cycles for $m = 1024$, $wordCount = 16$.
- ≈ 2 cycles per iteration.
- This is ≈ 1500 times faster than direct computation.

Benchmarks, $m = 1024$

Number of cells	3.6×10^3	5.3×10^4	3.3×10^5	1.3×10^6
Exact computation	127			
LSH, total	6.4	111	910	13482
LSH signatures	6.3	95	289	1660
Loop over pairs	0.08	16	621	11822

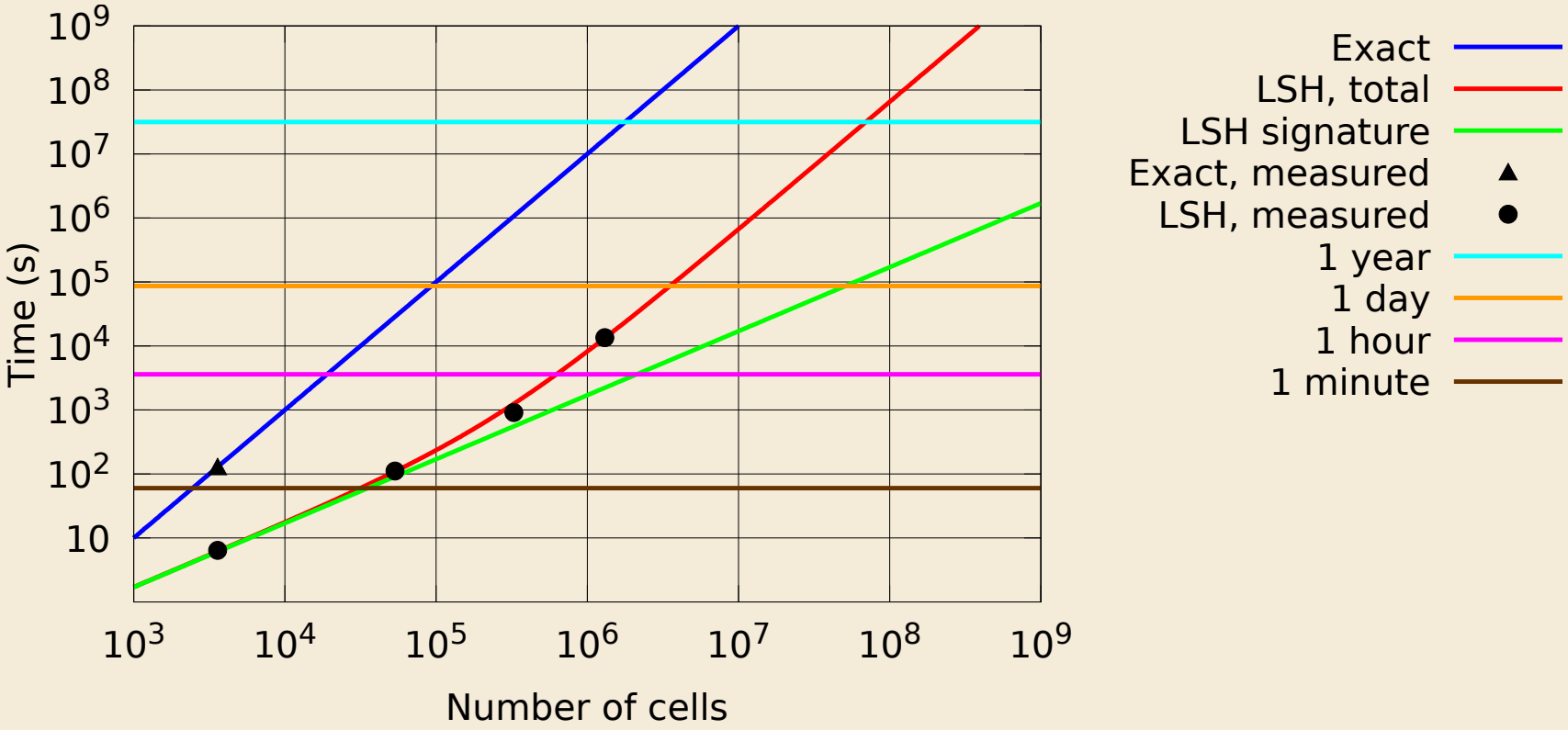
- Elapsed times in seconds on a Lenovo ThinkPad laptop with 2.70GHz Intel® Core™ i7-6820HQ processor, single-threaded.
- Benchmark done excluding time to store similarities, which varies depending on criteria used for storing and typically adds $\approx 5 - 10\%$.

Performance model, $m = 1024$

Benchmark results are consistent with this simple performance model:

Time for exact similarity computation (per pair, for 2500 average expressed genes)	$20\mu s$	$O(N^2)$
Time to compute LSH signature (per cell, normalized to 2500 expressed genes)	1.7 ms	$O(N)$
Time for LSH similarity computation (per pair, excluding storing)	13 ns	$O(N^2)$

Performance summary, $m = 1024$



Performance model suggestions

Performance model suggestions

- At $N \lesssim 10^5$, performance of the all pairs LSH algorithm is dominated by the linear term (step 2).

Performance model suggestions

- At $N \lesssim 10^5$, performance of the all pairs LSH algorithm is dominated by the linear term (step 2).
- At larger N, it approaches its asymptotic scaling $O(N^2)$ but ≈ 1500 times faster than direct computation.

Performance model suggestions

- At $N \lesssim 10^5$, performance of the all pairs LSH algorithm is dominated by the linear term (step 2).
- At larger N, it approaches its asymptotic scaling $O(N^2)$ but ≈ 1500 times faster than direct computation.
- Without using parallelism, the all pairs LSH algorithm can do $\approx 10^4$ cells interactively, $\approx 3 \times 10^6$ overnight.

Performance model suggestions

- At $N \lesssim 10^5$, performance of the all pairs LSH algorithm is dominated by the linear term (step 2).
- At larger N, it approaches its asymptotic scaling $O(N^2)$ but ≈ 1500 times faster than direct computation.
- Without using parallelism, the all pairs LSH algorithm can do $\approx 10^4$ cells interactively, $\approx 3 \times 10^6$ overnight.
- Performance of the full LSH algorithm (see final portion of this presentation) will be somewhere between the green line ($O(N)$) and the red line ($O(N^2)$ for large N).

Performance model suggestions

- At $N \lesssim 10^5$, performance of the all pairs LSH algorithm is dominated by the linear term (step 2).
- At larger N, it approaches its asymptotic scaling $O(N^2)$ but ≈ 1500 times faster than direct computation.
- Without using parallelism, the all pairs LSH algorithm can do $\approx 10^4$ cells interactively, $\approx 3 \times 10^6$ overnight.
- Performance of the full LSH algorithm (see final portion of this presentation) will be somewhere between the green line ($O(N)$) and the red line ($O(N^2)$ for large N).
- The full LSH algorithm will enable processing the large number of cells expected for the Human Cell Atlas, when taking advantage of parallelism.

Interpretation as dimensionality reduction

- We are using the similarity between cell signatures to approximate the exact similarity.

Interpretation as dimensionality reduction

- We are using the similarity between cell signatures to approximate the exact similarity.
- This is a form of dimensionality reduction: instead of representing each cell using its expression vector (n dimensions, continuous variables) we represent it using its signature (m dimensions, boolean variables).

Interpretation as dimensionality reduction

- We are using the similarity between cell signatures to approximate the exact similarity.
- This is a form of dimensionality reduction: instead of representing each cell using its expression vector (n dimensions, continuous variables) we represent it using its signature (m dimensions, boolean variables).
- Each signature value corresponds to a spherical polygon on the unit hypersphere in n-dimensional space.

Interpretation as dimensionality reduction

- We are using the similarity between cell signatures to approximate the exact similarity.
- This is a form of dimensionality reduction: instead of representing each cell using its expression vector (n dimensions, continuous variables) we represent it using its signature (m dimensions, boolean variables).
- Each signature value corresponds to a spherical polygon on the unit hypersphere in n-dimensional space.
- We could do clustering in signature space, and there may be some advantages to that. However I have not yet explored this possibility, and for now I am only using LSH to speed up the calculation of cell similarities.

Can we do better than $O(N^2)$?

Can we do better than $O(N^2)$?

- Yes, we can, by exploiting the discrete nature of the signature vectors.
- This is where LSH realizes its full potential.

A bad hash function

- Pick a subset of q bits of the cell signatures.

A bad hash function

- Pick a subset of q bits of the cell signatures.
- These bits form an integer $h(x)$ which is a function of the cell expression vector x, and we can therefore consider it a hash function.

A bad hash function

- Pick a subset of q bits of the cell signatures.
- These bits form an integer $h(x)$ which is a function of the cell expression vector x, and we can therefore consider it a hash function.
- This hash function has properties that are extremely undesirable for ordinary hash functions, because it is prone to collisions for cells with high similarity.

A bad hash function

- Pick a subset of q bits of the cell signatures.
- These bits form an integer $h(x)$ which is a function of the cell expression vector x, and we can therefore consider it a hash function.
- This hash function has properties that are extremely undesirable for ordinary hash functions, because it is prone to collisions for cells with high similarity.
- The reason is that, if two cells are very similar ($r(x, y)$ is close to 1), there is high probability that $h(x) = h(y)$. We will compute this collision probability, which is an increasing function of $r(x, y)$, in one of the next slides.

A locality-sensitive hash function

A locality-sensitive hash function

- This property is actually very useful in our context: if we create a hash table using such a hash function, cells that are very similar are likely to experience a collision and therefore end up in the same bucket.

A locality-sensitive hash function

- This property is actually very useful in our context: if we create a hash table using such a hash function, cells that are very similar are likely to experience a collision and therefore end up in the same bucket.
- If we want to find highly similar pairs, we can search over pairs that are in the same bucket, rather than over all possible pairs.

A locality-sensitive hash function

- This property is actually very useful in our context: if we create a hash table using such a hash function, cells that are very similar are likely to experience a collision and therefore end up in the same bucket.
- If we want to find highly similar pairs, we can search over pairs that are in the same bucket, rather than over all possible pairs.
- This is the basic idea of LSH, from which the name (because the hash function is likely not to change under a small local perturbation of the cell expression vector).

A locality-sensitive hash function

- This property is actually very useful in our context: if we create a hash table using such a hash function, cells that are very similar are likely to experience a collision and therefore end up in the same bucket.
- If we want to find highly similar pairs, we can search over pairs that are in the same bucket, rather than over all possible pairs.
- This is the basic idea of LSH, from which the name (because the hash function is likely not to change under a small local perturbation of the cell expression vector).
- We need to make this more quantitative.

Collision probability

- Given two cells with similarity $r(x, y) = \cos\theta$, where θ is the angle between the shifted expression vectors, the collision probability is the probability that the q bits of the two cells are identical.

Collision probability

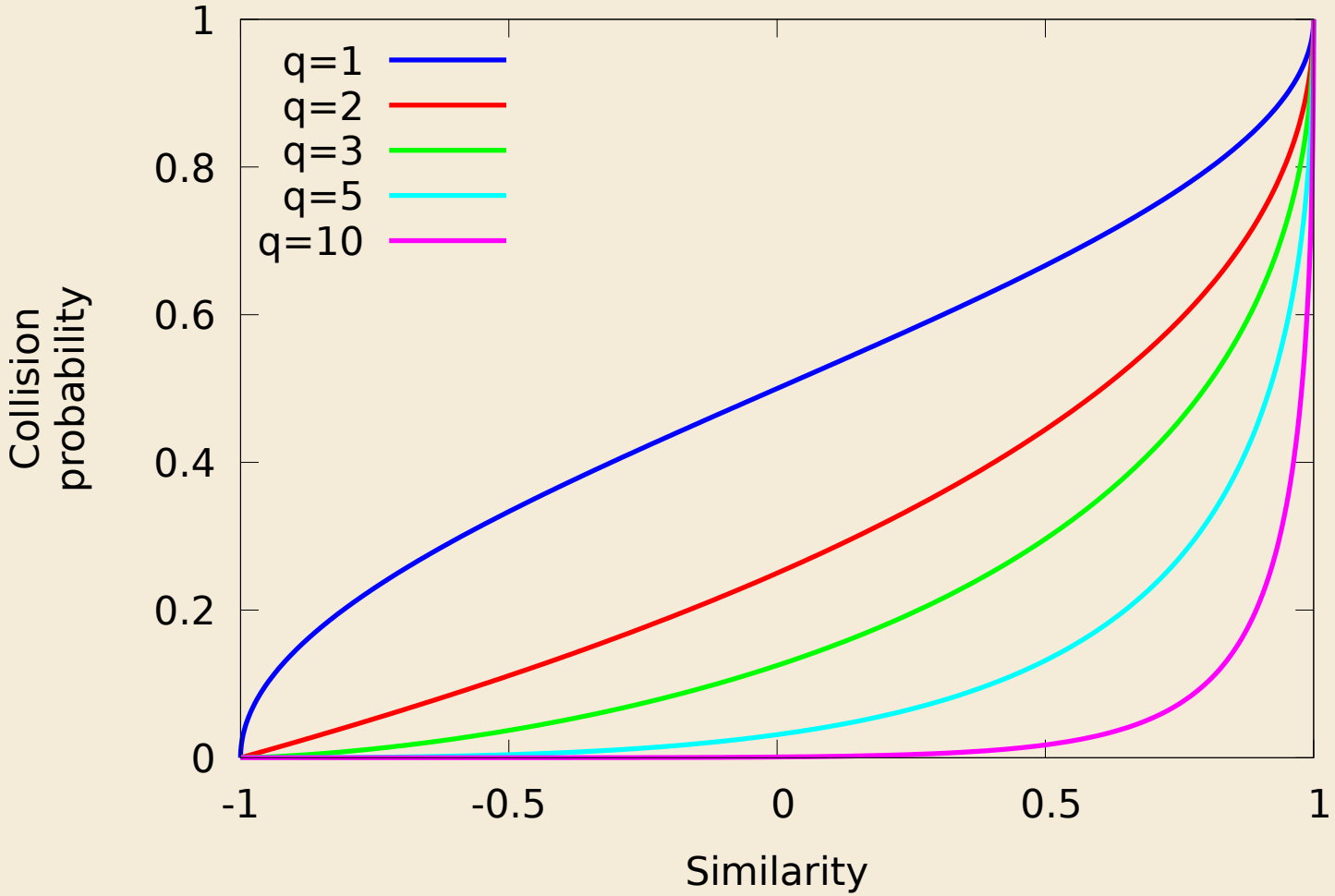
- Given two cells with similarity $r(x, y) = \cos\theta$, where θ is the angle between the shifted expression vectors, the collision probability is the probability that the q bits of the two cells are identical.
- The probability that a single bit is identical equals the probability that the two cells are on the same side of the corresponding hyperplane, $1 - \frac{\theta}{\pi}$.

Collision probability

- Given two cells with similarity $r(x, y) = \cos\theta$, where θ is the angle between the shifted expression vectors, the collision probability is the probability that the q bits of the two cells are identical.
- The probability that a single bit is identical equals the probability that the two cells are on the same side of the corresponding hyperplane, $1 - \frac{\theta}{\pi}$.
- The hyperplanes corresponding to the q bits are all random and uncorrelated, and therefore the probability that all the q bits are identical is

$$P_{collision}(\theta) = \left(1 - \frac{\theta}{\pi}\right)^q$$

Collision probability increases with cell similarity



Iterate over sets of q bits

- We can iterate the process Q times, using Q mutually disjoint subsets of q bits out of the total m .

Iterate over sets of q bits

- We can iterate the process Q times, using Q mutually disjoint subsets of q bits out of the total m .
- What is the probability that a pair of cells appears in the same bucket at least one of the Q times?

Iterate over sets of q bits

- We can iterate the process Q times, using Q mutually disjoint subsets of q bits out of the total m .
- What is the probability that a pair of cells appears in the same bucket at least one of the Q times?
  - Probability the two cells are not in the same bucket in a given iteration:

$$P_a(\theta) = 1 - P_{collision}(\theta) = 1 - \left(1 - \frac{\theta}{\pi}\right)^q$$

Iterate over sets of q bits

- We can iterate the process Q times, using Q mutually disjoint subsets of q bits out of the total m .
- What is the probability that a pair of cells appears in the same bucket at least one of the Q times?
  - Probability the two cells are not in the same bucket in a given iteration:

$$P_a(\theta) = 1 - P_{collision}(\theta) = 1 - \left(1 - \frac{\theta}{\pi}\right)^q$$

  - Probability the two cells are not in the same bucket in all Q iterations:

$$P_b(\theta) = P_a^Q(\theta) = \left[1 - \left(1 - \frac{\theta}{\pi}\right)^q\right]^Q$$

Iterate over sets of q bits

- We can iterate the process Q times, using Q mutually disjoint subsets of q bits out of the total m .
- What is the probability that a pair of cells appears in the same bucket at least one of the Q times?
  - Probability the two cells are not in the same bucket in a given iteration:

$$P_a(\theta) = 1 - P_{collision}(\theta) = 1 - \left(1 - \frac{\theta}{\pi}\right)^q$$

  - Probability the two cells are not in the same bucket in all Q iterations:

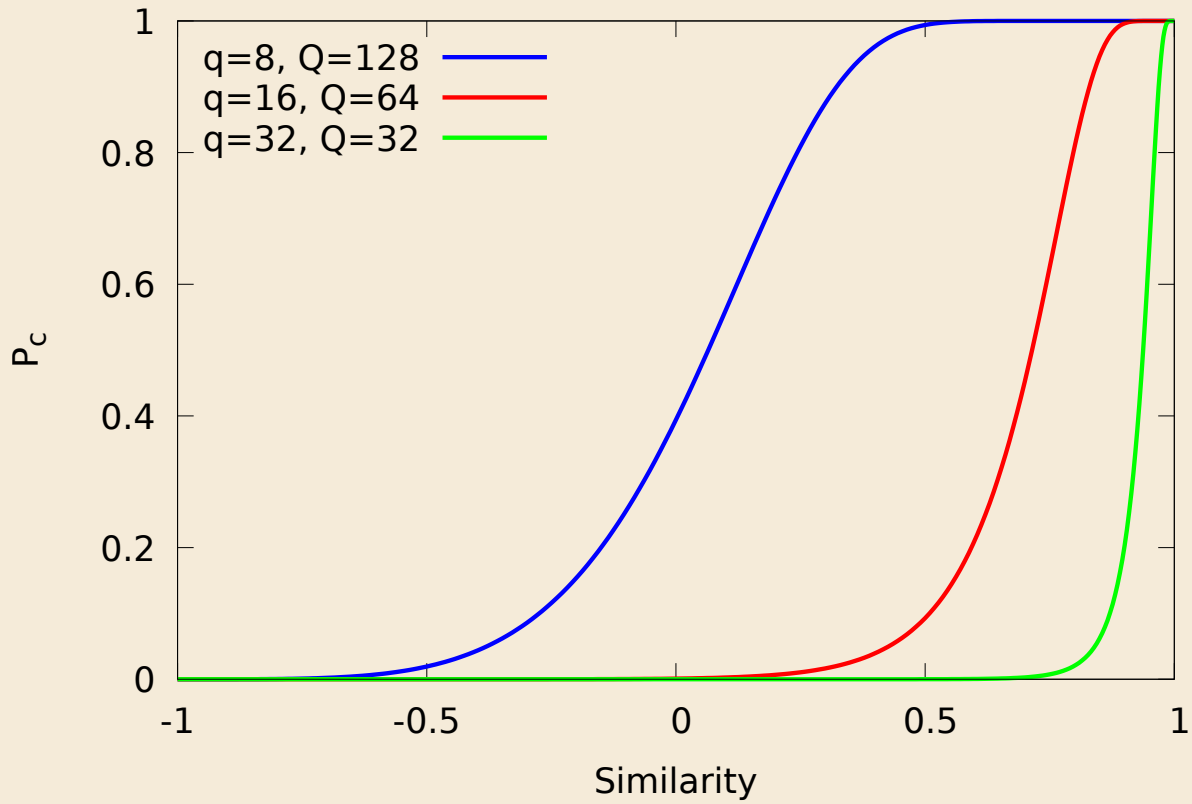
$$P_b(\theta) = P_a^Q(\theta) = \left[1 - \left(1 - \frac{\theta}{\pi}\right)^q\right]^Q$$

  - Probability the two cells are in the same bucket at least one of the Q times:

$$P_c(\theta) = 1 - P_b(\theta) = 1 - \left[1 - \left(1 - \frac{\theta}{\pi}\right)^q\right]^Q$$

P_c versus cell similarity

- For selected values of q, Q such that $qQ = 1024$.
- Cell pairs with high similarity are virtually certain to be the same bucket at least once.
- We can tune q and Q depending on what our threshold for ”interesting” cell similarity is.



Full LSH algorithm to find cell pairs with high similarities

Full LSH algorithm to find cell pairs with high similarities

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.

Full LSH algorithm to find cell pairs with high similarities

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. Compute all the cell signatures.

Full LSH algorithm to find cell pairs with high similarities

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. Compute all the cell signatures.
3. Iterate Q times, each time using a different subset of q signature bits as the hash function:

Full LSH algorithm to find cell pairs with high similarities

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. Compute all the cell signatures.
3. Iterate Q times, each time using a different subset of q signature bits as the hash function:
    - Create a hash table using the current hash function.

Full LSH algorithm to find cell pairs with high similarities

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. Compute all the cell signatures.
3. Iterate Q times, each time using a different subset of q signature bits as the hash function:
   - Create a hash table using the current hash function.
   - Iterate over all cell pairs in each bucket. For each pair use the $\bar{r}$ estimator to compute an approximate similarity. If the similarity is greater than the desired threshold, store that pair.

Full LSH algorithm to find cell pairs with high similarities

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. Compute all the cell signatures.
3. Iterate Q times, each time using a different subset of q signature bits as the hash function:
   - Create a hash table using the current hash function.
   - Iterate over all cell pairs in each bucket. For each pair use the $\bar{r}$ estimator to compute an approximate similarity. If the similarity is greater than the desired threshold, store that pair.

- Steps 1 and 2 are the same as for the preliminary LSH algorithm that iterates over all pairs.

Full LSH algorithm to find cell pairs with high similarities

1. Randomly pick m random vectors in n-dimensional space, each defining a hyperplane orthogonal to it.
2. Compute all the cell signatures.
3. Iterate Q times, each time using a different subset of q signature bits as the hash function:
   - Create a hash table using the current hash function.
   - Iterate over all cell pairs in each bucket. For each pair use the $\bar{r}$ estimator to compute an approximate similarity. If the similarity is greater than the desired threshold, store that pair.

- Steps 1 and 2 are the same as for the preliminary LSH algorithm that iterates over all pairs.
- But in step 3 we are no longer iterating over all pairs.

Complexity of the full LSH algorithm

- Complexity analysis of the full LSH algorithm is not possible as it is heavily dependent on the amount of clustering present.

Complexity of the full LSH algorithm

- Complexity analysis of the full LSH algorithm is not possible as it is heavily dependent on the amount of clustering present.
- In the limit of extreme clustering, all cells end up in the same bucket and we saved nothing.

Complexity of the full LSH algorithm

- Complexity analysis of the full LSH algorithm is not possible as it is heavily dependent on the amount of clustering present.
- In the limit of extreme clustering, all cells end up in the same bucket and we saved nothing.
- But in general, the number of cell pairs that will have to be considered will be a small fraction of the total, assuming judicious choices of q and Q are made, and that special treatment is given to buckets with unreasonably large numbers of cells.

Complexity of the full LSH algorithm

- Complexity analysis of the full LSH algorithm is not possible as it is heavily dependent on the amount of clustering present.
- In the limit of extreme clustering, all cells end up in the same bucket and we saved nothing.
- But in general, the number of cell pairs that will have to be considered will be a small fraction of the total, assuming judicious choices of q and Q are made, and that special treatment is given to buckets with unreasonably large numbers of cells.
- Work is in progress. Actual performance in practice will be somewhere between $O(N)$ and $O(N^2)$. It is reasonable to expect $O(N \log N)$.

Incremental version of the LSH algorithm

- If new cells are added to the system, can we update the list of similar cell pairs without redoing the entire computation?

Incremental version of the LSH algorithm

- If new cells are added to the system, can we update the list of similar cell pairs without redoing the entire computation?
- We can, if we keep the hash tables created at each of the Q iterations. This is a small amount of information compared to the cell expression vectors.

Incremental version of the LSH algorithm

- If new cells are added to the system, can we update the list of similar cell pairs without redoing the entire computation?
- We can, if we keep the hash tables created at each of the Q iterations. This is a small amount of information compared to the cell expression vectors.
- When we add a new cell, we compute its signature, and update each of the Q hash tables. At the same time, we consider all cell pairs consisting of the newly added cell and one of the cells that are in the same bucket in at least one of the Q iterations.

Incremental version of the LSH algorithm

- If new cells are added to the system, can we update the list of similar cell pairs without redoing the entire computation?
- We can, if we keep the hash tables created at each of the Q iterations. This is a small amount of information compared to the cell expression vectors.
- When we add a new cell, we compute its signature, and update each of the Q hash tables. At the same time, we consider all cell pairs consisting of the newly added cell and one of the cells that are in the same bucket in at least one of the Q iterations.
- This can support incremental clustering calculations, assuming that the clustering algorithm used also supports incremental update.

It takes a village...

Thank you for your insight, help, support, teachings, which made this work possible.

- **CZI:**
  - Andrey Kislyuk
  - Anthony Sabathia
  - Athena Chavarria
  - Brian Pinkerton
  - Bruce Martin
  - Cat Chang
  - Cori Bargmann
  - Ed Ruiz
  - James Wang
  - Kathleen Kiang
  - Landon Patton
  - Marcus Kinsella
  - Michael Chung
  - Priscilla Chan
  - Yun-Fang Juan
- **BioHub:**
  - James Webber
  - Jim Karkanias
  - Joshua Batson
  - Olga Botvinnik
  - Spyros Darmanis
- **U.C. Santa Cruz:**
  - Jim Kent
  - Maximilian Haeussler
- **Other:**
  - Mark Zuckerberg
  - Trilby Steiger