Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press



MaxGeomHash bridges MinHash and FracMinHash for scalable similarity estimation using sub-linear *k*-mer sketching

Mahmudur Rahman Hera, David Koslicki and Conrado Martinez

Genome Res. published online November 24, 2026
Access the most recent version at doi:10.1101/gr.282217.126

P<P Published online November 24, 2026 in advance of the print journal.

Accepted Manuscript Peer-reviewed and accepted for publication but not copyedited or typeset; accepted manuscript is likely to differ from the final, published version.

Open Access Freely available online through the *Genome Research* Open Access option.

Creative Commons License This manuscript is Open Access.This article, published in *Genome Research*, is available under a Creative Commons License (Attribution 4.0 International license), as described at http://creativecommons.org/licenses/by/4.0/.

Email Alerting Service Receive free email alerts when new articles cite this article - sign up in the box at the top right corner of the article or **click here.**



To subscribe to *Genome Research* go to:
https://genome.cshlp.org/subscriptions

Published by Cold Spring Harbor Laboratory Press

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

MaxGeomHash bridges MinHash and FracMinHash for scalable similarity estimation using sub-linear k-mer sketching *

Mahmudur Rahman Hera[†] David Koslicki[‡] Conrado Martínez[§]

Abstract

With the surge in sequencing data generated from biological studies, designing scalable computational techniques has become essential. One effective strategy to enable large-scale computation is to extract k-mers from long sequences, and summarize large k-mer sets into random samples (a.k.a. *sketches*). These sketches enable rapid estimation of similarity metrics such as Jaccard or cosine, and thus facilitate scalable computations. Popular sketching tools in bioinformatics range from the MinHash algorithm to generate fixed-size sketches; to FracMinHash algorithm to retain a fraction of all k-mers. Here, we introduce a novel sketching algorithm, MaxGeomHash, which for a specified integer parameter $b \geq 1$, will produce a random sample of size $b \lg(n/b) + \mathcal{O}(b)$ without knowing the total number of k-mers n beforehand. Notably, this is the first permutation-invariant and parallelizable sketching algorithm to date that can produce sub-linear sketches. We also introduce a variant, α-MaxGeomHash, that produces sketches of size $\Theta(n^\alpha)$ for a given $\alpha \in (0, 1)$. We study the algorithm's properties, analyze generated sample sizes, verify theoretical results empirically, provide a fast implementation, and investigate similarity estimate quality. With intermediate-sized samples between constant (MinHash) and linear (FracMinHash), MaxGeomHash balances efficiency (smaller samples need less processing) with accuracy (larger samples yield better estimates). On genomic datasets, we demonstrate that MaxGeomHash enables pairwise similarity estimation at a large scale, and MaxGeomHash sketches enable similarity estimation more accurately than MinHash, and more efficiently than FracMinHash.

Keywords: Random sampling, sketching, k-mers, MinHash, FracMinHash, similarity estimation, dimensionality reduction.

*Authors are listed in alphabetical order.

[†]Center for Advanced Biotechnology & Medicine, Rutgers University, NJ-08854, USA mahmudur.r@rutgers.edu and Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA-16802, USA

Present address: Ray and Stephanie Lane Computational Biology Department Carnegie Mellon University 5000 Forbes Avenue Pittsburgh PA 15213.

[‡]Computer Science and Engineering, Biology, and the Huck Institutes of the Life Sciences, Pennsylvania State University, University Park, PA-16802, USA dmk333@psu.edu

[§]Department of Computer Science, Universitat Politècnica de Catalunya, Barcelona-08034, Spain conrado@cs.upc.edu

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

Introduction

The exponential growth of genomic and metagenomic sequencing data has necessitated the development of scalable computational methods for efficiently processing and comparing biological sequences. Central to many methods is the concept of k-mers (substrings of length k), which serve as fundamental units for sequence comparison and analysis. However, the vast number of distinct k-mers in modern datasets makes exact comparisons computationally prohibitive, driving the need for approximation techniques that trade accuracy for improvements in speed and memory.

Sketching methods have emerged as a powerful solution, creating compact "fingerprints" of sequence data that preserve essential similarity information while reducing computational requirements. MinHash (Broder 1997; Brian D. Ondov et al. 2016) has become ubiquitous in genomic applications, enabling rapid all-versus-all comparisons through fixed-size sketches. However, as we and others have shown (Koslicki and Zabeti 2019; Liu and Koslicki 2022), traditional MinHash exhibits significant limitations when comparing sets of very different sizes, a common scenario in metagenomic analyses where bacterial genomes are compared against complex environmental metagenomic samples.

FracMinHash addresses these limitations by allowing sketch sizes to scale with the data (Irber Jr 2020; Hera, Pierce-Ward, and Koslicki 2023). Rather than maintaining a fixed number of hash values, FracMinHash retains a fraction $s \in [0, 1]$ of all k-mers. This linear scaling of sketch size enables accurate containment estimation between sets of arbitrary sizes.

While FracMinHash provides excellent accuracy, the price it pays is maintaining very large samples (linear in the number of distinct elements). In modern bioinformatics, genomic repositories and metagenomic studies often contain billions to trillions of sequences, resulting in very large sketch sizes. This motivates the need for new sampling algorithms that can achieve a middle ground: maintaining the accuracy benefits of letting samples grow, while keeping sample sizes sub-linear.

In this work, we propose a novel algorithm for random sampling, which we call MaxGeomHash (MGH for short). Given a user-defined parameter b, our algorithm is designed to produce sub-linear random samples from a data stream containing n distinct elements, without the knowledge of n beforehand. We also present a variant, α-MaxGeomHash (α-MGH for short), which also supports sub-linear random samples and allows for controlling the asymptotic growth order of the random samples using a user-specified parameter $\alpha \in (0, 1)$. Both algorithms aim to offer flexible control over sketch size as the number of distinct elements grows. In addition to introducing the algorithms, we investigate their theoretical and practical properties. Specifically, we characterize the expectation and variance of MGH and α-MGH sample sizes and quantify their computational cost. We also investigate their suitability for similarity estimation, including Jaccard, cosine, and other metrics. Finally, we apply these algorithms on simulated and real genomic data for similarity estimation and clustering by considering streams of k-mers extracted from genomic sequences, using an efficient C++ implementation to compute and compare MGH/α-MGH samples directly from FASTA/FASTQ files.

Results

In this section, we present: 1) the complete theory of MaxGeomHash; 2) simulation experiments verifying theoretical expectations and asymptotically unbiased estimators; and 3) real data experiments highlighting the balance provided by our algorithms.

Theory of MaxGeomHash

In this section, we introduce the theory of MaxGeomHash. To allow for the reader to consume the theory, we first outline a brief overview of random sampling algorithms. We then describe the standard MaxGeomHash algorithm (which we refer to as the standard variant henceforth), followed by a variant called α-MaxGeomHash (which we refer to as the α-variant henceforth). We conclude the section by detailing the steps required to estimate similarity/distance metrics with MaxGeomHash sketches.

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

A brief overview of random sampling from data streams

Random sampling has a long history in algorithms and data systems, from early treatments and classic textbook expositions (Knuth 1997; Goodman and Kruskal 1961) to scalable streaming variants such as Reservoir Sampling (Vitter 1985) and its weighted forms (Efraimidis and Spirakis 2006). In large-scale bioinformatics, sketching-based sampling is indispensable for scalable and efficient analysis. Prominent families include MinHash and bottom-k (Broder 1997; Cohen and Kaplan 2007), FracMinHash (Irber et al. 2022; Hera and Koslicki 2025), and Affirmative Sampling (Lumbroso and Martínez 2022); in biological contexts, FracMinHash-style sketches are particularly effective and can be used to estimate a variety of biologically relevant quantities (Hera and Koslicki 2025; Hera and Koslicki 2024).

Model and objective. We observe a long dataset or data stream $\mathcal{Z} = z_1, z_2, \ldots, z_N$ of items drawn from a large *universe* $\mathcal{D}$, with $z_i \in \mathcal{D}$. Let the multiset of distinct elements underlying $\mathcal{Z}$ be $\mathcal{M} = \{x_1^{f_1}, \ldots, x_n^{f_n}\}$, with f_i the frequency of x_i in $\mathcal{Z}$, and let the set of distinct elements be $\mathcal{X} = \{x_1, \ldots, x_n\}$. The (unknown) number $n = |\mathcal{X}|$ is the *cardinality* of $\mathcal{Z}$. Our goal is to obtain a *random sample* $\mathcal{S}$ of distinct elements from $\mathcal{X}$ such that, for $S = |\mathcal{S}|$, every S-subset (a subset containing S elements) of $\mathcal{X}$ is equally likely to be drawn. We focus on methods that operate in one pass over $\mathcal{Z}$ without prior knowledge of n.

A unifying hashed-sampling view. Most random sampling algorithms work by sequentially processing the dataset or data stream $\mathcal{Z}$, one element at a time, and keeping a current sample $\mathcal{S}$ (in all the cases that we study here $\mathcal{S}$ is a random sample from the subset of elements processed so far). If the current item z is present in $\mathcal{S}$, we can update its frequency count and move on to the next. If z is not present, we compute its hash value $h = h(z)$ and use h to decide whether z is added to the sample or discarded. Moreover, and depending on our decision w.r.t. z, we might decide to make additional changes in $\mathcal{S}$, for example, removing some elements from $\mathcal{S}$. Examples of algorithms that fall into the general scheme outlined above include bottom-k and MinHash (Broder 1997) , FracMinHash (Irber et al. 2022; Hera and Koslicki 2024; Hera and Koslicki 2025), ModHash (Broder 1997), and Affirmative Sampling (Lumbroso and Martínez 2022). Here, it is important to note that both MinHash and bottom-k render samples of fixed size, say k. MinHash actually returns k hash values, not the elements themselves: it applies $h_1, \ldots, h_k$ hash functions to each item in the data set, and keeps the minimum values observed for each of the k hash functions. In bottom-k, we apply a hash function h to each item x in the data set, and we keep the k smallest observed hash values. Despite the two algorithms are different (specially in terms of practical efficiency) they render equivalent random samples of size k, and we will loosely use the terms MinHash and bottom-k interchangeably. Related sketching methods such as HyperLogLog (Flajolet et al. 2007) and HyperMinHash (Yu and Weber 2020) also use hash-prefix and leading-zero information to organize hashed elements into buckets, although these methods have different objectives and sketch representations.

In FracMinHash, an item z is sampled (kept in the sample $\mathcal{S}$) iff $h(z) \leq s$ for a given real value $s \in (0, 1)$, where $h(.)$ produces real hash values in the range [0,1]. In ModHash, an item z is sampled iff $h(z) \mod M = 0$, where $h(.)$ produces integer hash values in some range $[0, H-1]$ for some large H. In bottom-k, an item z is sampled iff it is among the k smallest hash values seen so far. If sampling z results in $|\mathcal{S}| = k+1$, then the item with the largest hash value in $\mathcal{S}$ is removed. MinHash applies k independent hash functions, and an item z is sampled if $h_i(z)$ is the minimum hash value among the h_i values observed so far, for some i. Affirmative Sampling —the standard variant— samples an item z if not present in the sample and if $h(z)$ is larger than $\min_{x \in \mathcal{S}} h(x)$; moreover, it evicts the item z^* with the smallest hash value in $\mathcal{S}$ if $h(z)$ is not among the b-largest hash values in $\mathcal{S}$. Initially, the first b items are always included in $\mathcal{S}$; and at any given moment, the sample $\mathcal{S}$ will contain $|\mathcal{S}|$ distinct elements with largest hash values seen so far.

In FracMinHash and ModHash, the expected size of the sample $\mathcal{S}$ is linear in n. For FracMinHash, it will follow a binomial distribution of parameters n and s. Likewise, ModHash produces random samples with their size following a binomial distribution of parameters n and $1/M$. Last but not least, in Affirmative Sampling, the expected size of $\mathcal{S}$ is $\sim b \ln(n/b)$, for a fixed parameter b, in the standard variant; in α-Affirmative Sampling, $0 < \alpha < 1$, the expected size of the sample is $\Theta(n^\alpha)$.

MinHash has the advantage of producing samples of fixed size and the samples/sketches are small and easy to obtain. However, inference based on those sketches is less accurate, especially when they are used to estimate containment and similarity indices between two sets. Conversely, FracMinHash and ModHash give

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

excellent accuracy, but the price they pay is to keep very large samples (linear in the number of distinct elements). Midway, we have Affirmative Sampling, which guarantees very good accuracy and (asymptotically) unbiased estimation, while producing significantly smaller samples (of size $\Theta(\log n)$ or $\Theta(\sqrt{n})$, say).

All these algorithms return random samples of distinct elements, which is easily provable under pragmatic assumptions on the randomness and independence of the hash functions, namely, if h returns values in the range $[0, H-1]$ then, for any z and any $i \in [0, H-1]$, we assume that $\mathbb{P}\{h(z) = i\} = 1/H$. Indeed, if all decisions (to sample/to evict/to discard) are solely based upon the hash values of the items, and these satisfy the randomness and independence assumptions above then the samples will be random.

Dependability. A sampling algorithm is *dependable* (Lumbroso and Martínez 2022) if it allows for exact frequency counts: membership of x_j in $\mathcal{S}$ is fixed at its first occurrence, and any item that is sampled and later removed will never be reinserted. Reservoir sampling (Vitter 1985) is not dependable, whereas all the other algorithms mentioned here are.

Parallel composition. Massive datasets must be processed in parallel by partitioning $\mathcal{Z}$ into p disjoint data subsets or substreams $\mathcal{Z}_1, \ldots, \mathcal{Z}_p$. For order-independent sketches there is a standard merge operation: compute a local sample $\mathcal{S}_i$ from each $\mathcal{Z}_i$ and then aggregate $\mathcal{S} = \phi\Big(\bigcup_{1 \leq i \leq p} \mathcal{S}_i\Big)$, where ϕ is the algorithm-specific pruning rule (e.g., keep the k smallest hash values, or keep all items with $h(x) \leq s$). This equivalence works both ways: running the algorithm on the entire stream $\mathcal{Z}$ or merging local sketches yields the same distribution. MinHash, bottom-k, FracMinHash, and ModHash satisfy this property; Reservoir Sampling (with repetitions) and Affirmative Sampling (both variants) do not, and are not mergeable in this sense. In general, sketches are called *mergeable* iff the sketch $S_{A \cup B}$ for $A \cup B$ is a function of the sketches S_A and S_B of A and B, resp. (Agarwal et al. 2013), and this is a property of fundamental practical relevance since it is crucial for effective parallelization.

Expected cost of random sampling. The *cost* of running a sampling algorithm refers to the number of operations required to scan the entire data stream and produce the random sample. We give the expected cost $M(n, N)$ of the random sampling algorithms in the form $\Theta(N + u(n) \cdot t(S))$, where N is the size of the dataset (or the length of the data stream), n the number of distinct elements ($n \leq N$, in many applications $n \ll N$), $u(n)$ is the expected number of times there is an update of the sample (and of auxiliary data structures, if any), and $t(S)$ is the time to update a sample of size S. This cost is based on the assumption that we can test whether a given item z (or hash value $h = h(z)$) is already present in the sample or not in expected constant time. The expected number of updates $u(n)$ will be at least S, but it is often the case that $u(n) \gg S$, for example, in bottom-k we have $S = k$, but $u(n) = k \ln(n/k) + \mathcal{O}(k)$ (Lumbroso and Martínez 2022); using a min-heap to keep and efficiently update the sample we have $t(S) = \mathcal{O}(\log k)$. This gives a total expected cost for bottom-k of $\mathcal{O}(N + k \log k \log(n/k))$. Likewise, the cost of producing a FracMinHash sketch is $\mathcal{O}(N + ns)$, as $u(n) = S = n \cdot s$ and $t(S) = \Theta(1)$. The expected cost of the new algorithms, MGH and α-MGH, will be investigated in the following sections, where we describe and study the algorithms and their main properties.

At-a-glance comparison. We summarize the main properties that guide the choice of a sampler in practice in Table 1. Dependability is necessary if exact frequency counts are required; mergeability enables distributed processing; the parameter controls the memory footprint; and the last two columns report the expected sample size. Asterisks indicate quantities that are either fixed or bounded by a constant, and n denotes the total number of distinct elements in the original set.

In summary, our algorithm fits neatly between the fixed-length samples of MinHash and the linearly-sized samples of FracMinHash, while retaining mergeability (which is absent from Affirmative Sampling).

The MaxGeomHash algorithm

Here, we describe our algorithm, MaxGeomHash (MGH for short) which is dependable and parallelizable in the sense explained in the previous section. Like Affirmative Sampling, it has the nice property of returning

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

Algorithm	Dep.?	Merg.?	Parameter	Expected $\lvert\mathcal{S}\rvert$ (parameterized)	Expected $\lvert\mathcal{S}\rvert$ (fixed parameter)
MinHash (Broder 1997)	✓	✓	$k \in \mathbb{N}$	k $(*)$	$\mathcal{O}(1)$
ModHash (Broder 1997)	✓	✓	$m \in \mathbb{N}$	n/m	$\mathcal{O}(n)$
Reservoir Sampling (Vitter 1985)	×	×	$k \in \mathbb{N}$	$\leq k$ $(*)$	$\mathcal{O}(1)$
FracMinHash (Irber et al. 2022)	✓	✓	$s \in (0,1)$	$n \cdot s$	$\mathcal{O}(n)$
Affirmative Sampling (Lumbroso and Martínez 2022)	✓	×	$b \in \mathbb{N}$	$b\ln(n/b) + \mathcal{O}(b)$	$\mathcal{O}(\log n)$
α-AS (Lumbroso and Martínez 2022)	✓	×	$\alpha \in (0,1)$	$\Theta(n^\alpha)$	$\mathcal{O}(n^\alpha)$
MGH (this paper)	✓	✓	$b \in \mathbb{N}$	$b\log_2(n/b) + \mathcal{O}(b)$	$\mathcal{O}(\log n)$
α-MGH (this paper)	✓	✓	$\alpha \in (0,1)$	$\frac{2^{1/(1-\alpha)}-1}{2^{\alpha/(1-\alpha)}-1} \cdot n^\alpha + \mathcal{O}(n^\alpha)$	$\mathcal{O}(n^\alpha)$

Table 1: At-a-glance comparison of MaxGeomHash variants with a few random sampling algorithms in the literature. The properties for Reservoir Sampling hold when repeated items are not considered.

samples of variable size, growing with n. In particular, the standard variant of the algorithm, which we examine now, returns samples of expected size $b\lg(n/b) + \mathcal{O}(b)$—here, and in the sequel we use $\lg \equiv \log_2$.

Given a bit string $w \neq 000\cdots0$, let $\mathrm{zpl}(w)$ denote the length of the longest prefix of 0s in w (if $w = 0000\cdots$ then $\mathrm{zpl}(w) = |w|$, by convention). In other words, if $\mathrm{zpl}(w) = j$ then the first j bits of w are 0 (bits 0 to $j-1$), and bit j is 1. We will also use $\mathrm{tail}(w, j)$ to denote the bit string that results from removing the first $j+1$ bits of w (that is, the longest prefix of zeros and the leftmost 1).

For each data stream item z, we compute its hash value $h := h(z)$, and determine the position $i = 1 + \mathrm{zpl}(h)$ of the leftmost 1 in the binary representation of h. Here we assume that the hash function $h(\cdot)$ maps an item z to a ℓ-bit positive integer $h(z)$ in the range $[0, H-1]$ for some large $H = 2^\ell$ (in practice, we used $\ell = 64$). We also set h' as the suffix after the leftmost 1. We use i to index z into a bucket $\mathcal{S}_i$. If $\mathcal{S}_i$ already contains z or h' is not among the b largest hash values in $\mathcal{S}_i$ then we will discard z. If $\mathcal{S}_i$ doesn't yet contain b elements or h' is among the b largest hashes in $\mathcal{S}_i$ then we add z to $\mathcal{S}_i$; in the latter case, the element z^* with the smallest hash value h' is evicted, so that $\mathcal{S}_i$ will contain the $\leq b$ elements in the data stream such that their leftmost 1 is at position i of $h(z)$ and have largest $h' = \mathrm{tail}(h(z), i)$. A frequency counter is kept for all $z \in \mathcal{S}$. The steps of MaxGeomHash are detailed in the supplemental materials (Algorithm 1). A visual overview of MaxGeomHash is given in Figure 1.

As usual in the analysis of sampling/sketching algorithms, randomness comes from the choice of the hash function h. In practice, we can pragmatically assume that for any data item z and any integer $i \in [0, H-1]$, we have $\mathbb{P}\{h(z) = i\} = 1/H$. For a theoretical standpoint, this can be achieved by drawing our hash function uniformly at random from a strongly universal family of hash functions (Carter and Wegman 1979; Mitzenmacher and Vadhan 2013) (for further discussions on universal classes of hash fucntions and applications the reader may consult (Cormen et al. 2022; Motwani and Raghavan 1995; Dasgupta, Papadimitriou, and Vazirani 2006)). Notice also that since we take $H = 2^\ell$ to be very large, we can also safely assume that the probability of *collisions* —that is, when $h(x) = h(y)$ for distinct items x and y in the dataset— is negligible, only $2^{-\ell} \ll 1/n$. As the expected number of collisions $n/2^\ell$ is almost 0, we will not need to bother about them in practice.

Let $x_1, \ldots, x_n$ denote the distinct elements in $\mathcal{Z}$. Let $\rho_i = 1 + \mathrm{zpl}(\mathrm{hash}(x_i))$. Because of our probabilistic model for h, $h(x_i)$ is a random bitstring of lenght ℓ, and the ρ_i are i.i.d. geometric random variables with parameter $1/2$. Therefore, the probability that an item z has to be considered for bucket $\mathcal{S}_i$ is $1/2^i$. Using this, we can compute the expected sample size and variance.

Theorem 1. *MaxGeomHash with parameter $b \geq 1$ will produce a random sample $\mathcal{S}$ of size $S = |\mathcal{S}| = \sum_{1 \leq i \leq R} |\mathcal{S}_i|$ such that*

$$\mathbb{E}[S] = \mathbb{E}\left[\sum_{i=1}^{R} |\mathcal{S}_i|\right] = b\lg(n/b) + \mathcal{O}(b)$$

where $R = \max\{\rho_j \mid 1 \leq j \leq n\}$. Moreover, $\mathbb{V}[S] = \Theta(1)$. Here, the expectation and variance are taken over the choice of the hash function, selected uniformly at random from a strongly universal family of hash

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

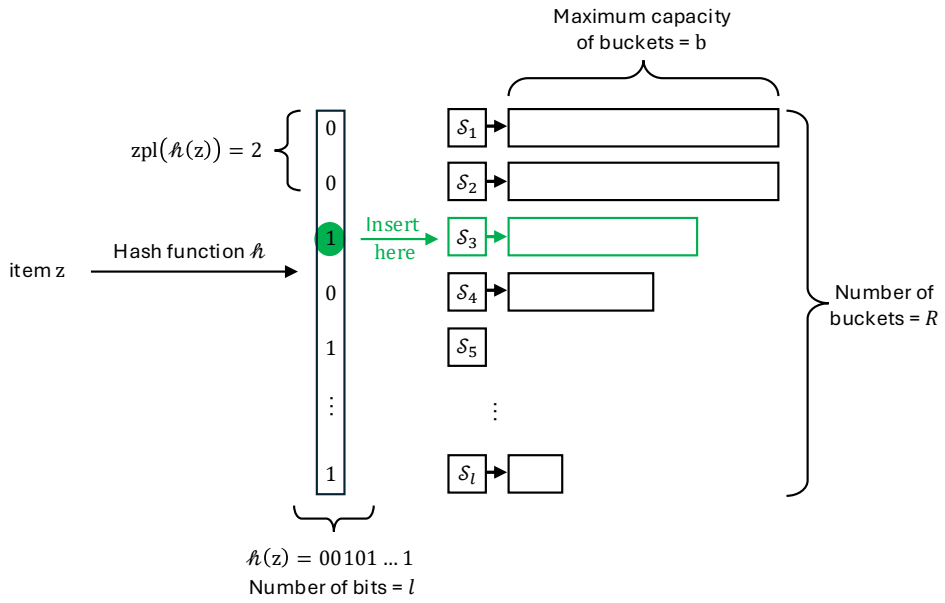

Figure 1: Overview of MaxGeomHash. A hash function $h(\cdot)$ maps an item z to a bit string of length ℓ. The zero prefix length (zpl) of the hash value $h(z)$ determines in which bucket i we should sample (or not) z. Each bucket $\mathcal{S}_i \equiv \text{bucket}[i]$ has a maximum capacity of b, where $b \geq 1$. Of all items assigned to a bucket, only the items with the b largest hash values are retained (similar to bottom-k sketches). The number of buckets can be at most ℓ, the number of bits in the hash value, although this is unlikely to happen in practice if $\lg n \ll \ell$. The largest index R of a non-empty bucket is given by the maximum of n i.i.d. geometric random variables (hence the "MaxGeom" in the algorithm's name).

functions. In other words, the result relies upon the assumption that for any data item z and any value h, $0 \leq h < 2^\ell$, we have $\mathbb{P}\{h(z) = h\} = 2^{-\ell}$.

The expected cost $M_b(n, N)$ of MGH when processing a data stream of N items such that n of them are distinct is dominated by the time needed to scan the data stream $\mathcal{O}(N)$, plus the time to update the buckets i that contain b elements. We expect $N/2^i$ items to be directed to bucket i, but most will be discarded with cost $\Theta(1)$, as they are repetitions or because their hash h' is not among the b larger in $\mathcal{S}_i$. Like in a bottom-k approach, we expect $\sim b \ln(n/2^i b)$ updates in bucket $\mathcal{S}_i$ (Lumbroso and Martínez 2022), and the cost of each of these updates is $\mathcal{O}(\log b)$. Summing over all possible i, we get that the expected cost is

$$M_b(n, N) \in \mathcal{O}\big(N + b \cdot \log b \cdot (\log n)^2\big).$$

The α-MaxGeomHash algorithm

We now discuss a variant of MaxGeomHash, called α-MaxGeomHash (α-MGH for short), that renders samples of expected size $\Theta(n^\alpha)$ for a fixed parameter $\alpha \in (0, 1)$. The algorithm is exactly as Algorithm 1, but instead of keeping at most b distinct elements on each $\mathcal{S}_i$, we will keep up to $\lfloor 2^{\beta i} \rfloor$ in bucket i, with $\beta := \frac{\alpha}{1-\alpha}$.

Theorem 2. *The variant α-MGH of MaxGeomHash with parameter $\alpha \in (0, 1)$ will produce a random sample $\mathcal{S}$ of size S such that*

$$\mathbb{E}[S] = \left(\frac{2^{1/(1-\alpha)} - 1}{2^{\alpha/(1-\alpha)} - 1}\right) \cdot n^\alpha + \mathcal{O}(n^\alpha) \approx \left(\frac{1-\alpha}{\alpha \ln 2} + \frac{3}{2} + \frac{\alpha \ln 2}{12(1-\alpha)}\right) \cdot n^\alpha.$$

Moreover $\mathbb{V}[S] = \Theta(n^\alpha)$. Here, the expectation and variance are taken over the choice of the hash function, selected uniformly at random from a strongly universal family of hash functions.

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

We can also express expected cost $M_\alpha(n, N)$ in the same form as we have discussed for the other algorithms, namely, $M_\alpha(n, N) = \mathcal{O}(N + u(n)t(S))$. Since the expected total number of updates $u(n)$ is $\mathcal{O}(n^\alpha)$ and we can bound the cost of updating any bucket by $\mathcal{O}(\log(n^\alpha)) = \mathcal{O}(\log n)$, we conclude that the expected cost is

$$M_\alpha(n, N) \in \mathcal{O}(N + n^\alpha \log n).$$

Similarity estimation with MGH and α-MGH

Given two sets A and B and random samples $\mathcal{S} = \{\mathcal{S}_i\}_{1 \le i \le R_A}$ and $\mathcal{T} = \{\mathcal{T}_i\}_{1 \le i \le R_B}$, we will use them to estimate $\text{Jaccard}(A, B) = |A \cap B|/|A \cup B|$. We use A_i and B_i to denote the subsets of A and B of elements x with $\text{zpl}(x) + 1 = i$. If we use the same hash function to sample from A and from B, then if an item $x \in A$ is considered for bucket $\mathcal{S}_i$ then it will be also considered for bucket $\mathcal{T}_i$ if x also appears in B, and vice versa. By definition, we have $A_i \cap A_j = \emptyset$ whenever $i \neq j$, and $\bigcup_{i \ge 1} A_i = A$; likewise, $\{B_i\}_{i \ge 1}$ gives a partition of B. One important result (Martínez, Viola, and Wang 2023) establishes that if $A \cap B = \emptyset$ then $\mathcal{S} + \mathcal{T}$ is a random sample of $A + B$ (this also holds for $\mathcal{S}_i + \mathcal{T}_i$). But $\mathcal{S}_i \cup \mathcal{T}_i$ is not a random sample of $A_i \cup B_i$, nor $\mathcal{S} \cup \mathcal{T}$ a random sample of $A \cup B$. Therefore, we need to apply a "filtering" or "refining" step. In particular, we will keep only the b elements with larger hash values (à la bottom-k). So for each i, we will compute $\mathcal{S}_i \cup \mathcal{T}_i$ but keep the b elements with larger hash values h' (or only $|\mathcal{S}_i \cup \mathcal{T}_i|$ elements if that cardinality is smaller than b). Let us call this new sample $\mathcal{U}_i$. This is indeed a random sample of $A_i \cup B_i$. Moreover, we will compute $\mathcal{V}_i = \mathcal{S}_i \cap \mathcal{T}_i$. Notice that $|\mathcal{V}_i| \le b$, since both $\mathcal{S}_i$ and $\mathcal{T}_i$ contain at most b elements each.

All $\mathcal{U}_i$ are mutually disjoint ($\mathcal{U}_i \cap \mathcal{U}_j = \emptyset$ if $i \neq j$) which entails that $\mathcal{U} = \bigcup_{1 \le i \le R_{A \cup B}} \mathcal{U}_i$ is a random sample of $A \cup B$. Likewise $\mathcal{V} = \bigcup_i \mathcal{V}_i = A \cap B \cap \mathcal{U}$, hence (see (Martínez, Viola, and Wang 2023) for a detailed proof) $|\mathcal{V}|/|\mathcal{U}|$ is an unbiased estimator of $\text{Jaccard}(A, B)$. In fact, we can use $\mathcal{U}$ and $\mathcal{V}$ to estimate many other set similarity and set containment measures (Martínez, Viola, and Wang 2023).

The arguments also work *verbatim* if instead of buckets with maximum capacity equal to b, their capacity is given by f_i, e.g., $f_i = \lfloor 2^{\beta i} \rfloor$, as in α-MGH. When computing $\mathcal{U}_i$ we would just need to do the unions $\mathcal{S}_i \cup \mathcal{T}_i$, then retain the elements with the f_i larger h' values (or less if there are not f_i elements in $\mathcal{S}_i \cup \mathcal{T}_i$).

In order to merge two samples $\mathcal{S}$ and $\mathcal{T}$ we should proceed as described above, taking advantage that the two samples are already classified into buckets, and computing the unions/intersections of the buckets $\mathcal{S}_i$ and $\mathcal{T}_i$ while only keeping the b (or $f_i = \lfloor 2^{\beta i} \rfloor$) elements with larger h' hash values, doing it in one single pass. But from the point of view of the result, the process is equivalent to applying MGH (or α-MGH) to $\mathcal{S} \cup \mathcal{T}$; indeed, we have the following important property:

$$\text{MGH}(A \cup B) = \text{MGH}(\text{MGH}(A) \cup \text{MGH}(B))$$

This holds for any variant of the algorithm, whether the capacity of the buckets is b, $\lfloor 2^{\beta i} \rfloor$ or any function f_i. In other words, we can obtain a random sample of $A \cup B$ from random samples $\text{MGH}(A)$ and $\text{MGH}(B)$, by simply performing their union and applying MGH to the result (that is, keeping at most f_i elements with largest hash values on each bucket i).

In order to find which items in $\text{MGH}(A \cup B)$ belong to $A \cap B$, we exploit this simple identity:

$$(A \cap B) \cap \text{MGH}(A \cup B) = \text{MGH}(A) \cap \text{MGH}(B) \cap \text{MGH}(\text{MGH}(A) \cup \text{MGH}(B)).$$

More generally, given datasets A and B, and random samples $\text{MGH}(A)$ and $\text{MGH}(B)$, respectively, we want a random sample M for $A \otimes B$ and the subset M_P of elements in M that sastify a certain property P. Using P also to refer to the subset of elements in $A \otimes B$ that satisfy P, we need to obtain $M_P = M \cap P$. What we have discussed above is that for $\otimes \equiv \cup$, $\text{MGH}(A \otimes B) = \text{MGH}(\text{MGH}(A) \otimes \text{MGH}(B))$ and $M_{A \cap B} = \text{MGH}(A \otimes B) \cap (A \cap B) = \text{MGH}(\text{MGH}(A) \otimes \text{MGH}(B)) \cap (\text{MGH}(A) \cap \text{MGH}(B))$, hence there is a practical efficient way to obtain $M_P = M \cap P = M \cap \text{MGH}(A) \cap \text{MGH}(B)$, and from there an efficient algorithm to estimate similarities using the similarioty of the samples. Indeed, the results of (Hera and Koslicki 2025; Martínez, Viola, and Wang 2023; Sanz-González and Martínez 2025) apply, showing that we can use MGH or α-MGH samples to get (asymptotically) unbiased estimates of the Jaccard similarity of A and B, of their cosine similarity, of the containment index, and many other similarity measures, like Kulczynski 1, Kulczynski 2, or Sørensen-Dice. A list and definition of these metrics can be found in (Martínez, Viola, and Wang 2023, Table 3).

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

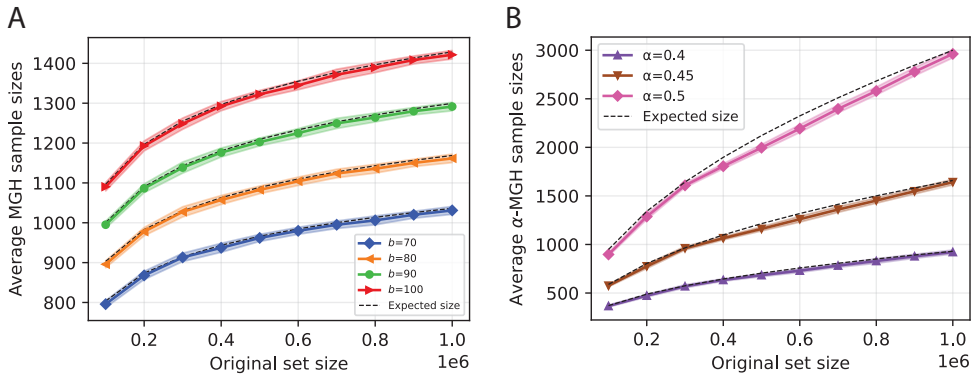

Figure 2: Average sample sizes, recorded from 50 independent trials. Samples are computed using MGH (averages shown in (A)) and α-MGH (averages shown in (B)). The solid lines show the averages, and the shaded regions around the solid lines (very narrow) highlight one standard deviation. Theoretical expectations (excluding lower-order terms) are shown in dashed black lines, closely tracked by the average recorded from simulations.

For some similarity measures like Jaccard, containment index, or Kulczynski 2, the estimates are unbiased; for others, like cosine or Kulczynski 1, the bias is $\mathcal{O}(\mathbb{E}[1/S])$, with S the size of the sample—which is a random variable for MGH and α-MGH (and also for FracMinHash). Also, the variance for all the different similarity measures mentioned above is $\mathcal{O}(\mathbb{E}[1/S])$.

Theorem 3. *Let σ_u be any of the following similarity measures: Jaccard, containment, Kulczynski 2, Sørensen-Dice, or correlation coefficient. Then*

$$\mathbb{E}[\sigma'_u(MGH(A), MGH(B))] = \sigma_u(A, B),$$

where σ'_u is defined as σ with a final "filtering" step. For example, for Jaccard we have $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ and $J'(M_A, M_B) = \frac{|MGH(M_A \cup M_B) \cap M_A \cap M_B|}{|MGH(M_A \cup M_B)|}$. The filtered versions of the other similarity measures are defined analogously.

Let σ_b denote the cosine similarity, or Kulczynski 1. Then $\sigma'_b(MGH(A), MGH(B))$ is asymptotically unbiased:

$$bias_{MGH}(A,B) = \mathbb{E}[\sigma'_b(MGH(A), MGH(B))] - \sigma_b(A,B) = \mathcal{O}\left(\mathbb{E}\left[\frac{1}{S}\right]\right),$$

with $S = |MGH(MGH(A) \cup MGH(B))|$. Finally, for any similarity measure σ among those mentioned above

$$\mathbb{V}[\sigma'(MGH(A), MGH(B))] = \Theta\left(\mathbb{E}\left[\frac{1}{S}\right]\right),$$

where S is as above.

The theorem easily follows from the results in (Martínez, Viola, and Wang 2023) and the straightforward generalization of Theorem 3 in (Sanz-González and Martínez 2025). The bias, if any, the variance, and the mean square error (recall that MSE = bias^2 + variance) of all our estimates based on MGH or α-MGH samples will all go to 0 because $\mathbb{E}[1/S] = 1/\mathbb{E}[S] + o(1/\mathbb{E}[S])$ and $\mathbb{E}[S] \to \infty$ if $n \to \infty$. Details of the proof that $\mathbb{E}[1/S] \sim 1/\mathbb{E}[S]$ can be found in the supplemental materials section at the end. In general, it is not true, but because $\mathbb{E}[S] \to \infty$, plus $\mathbb{V}[S] = \mathcal{O}(\mathbb{E}[S])$, it is not difficult to obtain this result.

MaxGeomHash samples grow sub-linearly with the original set

First, we validate the theoretical expectations of MGH and α-MGH sample sizes using simulated experiments. In these experiments, we generated random sets of 10K to 1M elements consisting of randomly generated

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

alphanumeric strings of length 10. We generated MGH samples with parameters $b \in \{70, 80, 90, 100\}$, and α-MGH samples with parameters $\alpha \in \{0.4, 0.45, 0.5\}$. For each choice of b and α, we ran 50 independent trials using different hash seeds and recorded the resulting sample size. The average sample sizes from these 50 trials are shown in Figure 2 (A shows the averages for MGH, and B shows the averages for α-MGH). The theoretical expectations outlined in Theorems 1 and 2 are also shown in Figure 2 using dashed black lines. When plotting these expected sizes, we excluded the smaller order terms (specifically, $\epsilon_{n,b}$ for the case of MGH, and $\mathcal{O}(\log n)$ for the case of α-MGH) from calculation.

The plots clearly show that the growth of the MGH and α-MGH samples aligns closely with the theoretically derived expectation. As the size of the original set grows, the sample size scales sub-linearly, as we expect them to. Figure 2 also shows one standard deviation as shaded regions around the corresponding solid lines. Notably, the variance we observe for MGH and α-MGH samples is quite small – which highlights the stability of the size of the samples produced by MaxGeomHash.

MaxGeomHash provides more stability than Affirmative Sampling

As outlined in the Background section, the only other sub-linear (super-constant) sketching algorithm is Affirmative Sampling. In this section, we show that MGH sketches are more stable than sketches computed using Affirmative Sampling. We use AS to denote Affirmative Sampling henceforth. There are two versions of Affirmative Sampling – the plain version is referred to as AS, and the α-version (being able to produce sketches of size $\mathcal{O}(n^\alpha)$, in expectation) is referred to as α-AS.

For the comparisons we present in this section, we generated a random set of 100,000 elements (each a random alphanumeric string) and computed sketches of it using AS and MGH, and the α-variant of both algorithms. Sketch sizes are shown in Figure 3A–D. We used $k = 100$ to run AS, $b = 70$ to run MGH, and used $\alpha = 0.5$ to run α-AS, where $\alpha = 0.4$ was used for α-MGH. We selected these parameter values to produce sketches of comparable sizes for AS and MGH, and likewise for α-AS and α-MGH.

First, when we ran the sketching algorithms, we processed the elements in 100 random orders. Figure 3A and Figure 3B show that simply by processing the elements in a different order, AS and α-AS produce different sketches despite using the same hash function (controlled by the random seed). On the other hand, MGH and α-MGH produce the same sketch each time. This "order invariant" property exhibited by MGH is important for reproducing the same results when rerunning experiments and plays a crucial role in multithreaded programming, where the order in which elements are processed often cannot be determined beforehand.

We next investigate the sketches generated by AS and MGH when the elements are always processed in the same order, but using 100 different hash functions (each with a different seed). We show the sizes of AS and MGH sketches in Figure 3C, and the sizes of α-AS and α-MGH sketches in Figure 3D. For these instances, we note that the sizes of the sketches generated by MGH and α-MGH observe some variability (as expected when using different hash functions). However, the variability is significantly less than that observed by AS and α-AS sketches. In fact, the sketches generated by α-AS are quite unstable, sometimes resulting in nearly empty sketches, and sometimes resulting in very large sketches compared to the expected size. Our counterpart α-MGH, on the other hand, generates sketches whose sizes are very well concentrated around its mean value. This behavior is also explained by the theoretical results: the variance of the size of α-MGH sketches, as characterized by Theorem 2, is $\Theta(n^\alpha)$, whereas the variance of the size of α-AS sketches is $\Theta(n^{2\alpha})$ (Lumbroso and Martínez 2022, Table 1). As the variance is much higher, the sizes of α-AS sketches span a much wider range than α-MGH.

The stability of our algorithm is further illustrated in Figure 3E–H, where we show the estimated Jaccard using AS and MGH. For these experiments, we generated two random sets, each comprising 100,000 random alphanumeric strings, with a Jaccard similarity of 0.5. We used AS and MGH sketches to estimate the Jaccard similarity. In Figure 3E and Figure 3F, we generated the sketches using the same hash function, but used 100 different random orders to process the elements (e and f are the same setting as a and b, respectively). Here, we note that since MGH sketches are always the same, they always estimate the same Jaccard. On the other hand, when we run AS, the resulting sketches differ each time due to their sensitivity to the order in which elements are processed. As a result, AS sketches estimate a range of Jaccard values. As α-AS sketches are more unstable, we observe that the resulting Jaccard estimate also varies, further highlighting the challenges in adopting α-AS for similarity estimation.

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

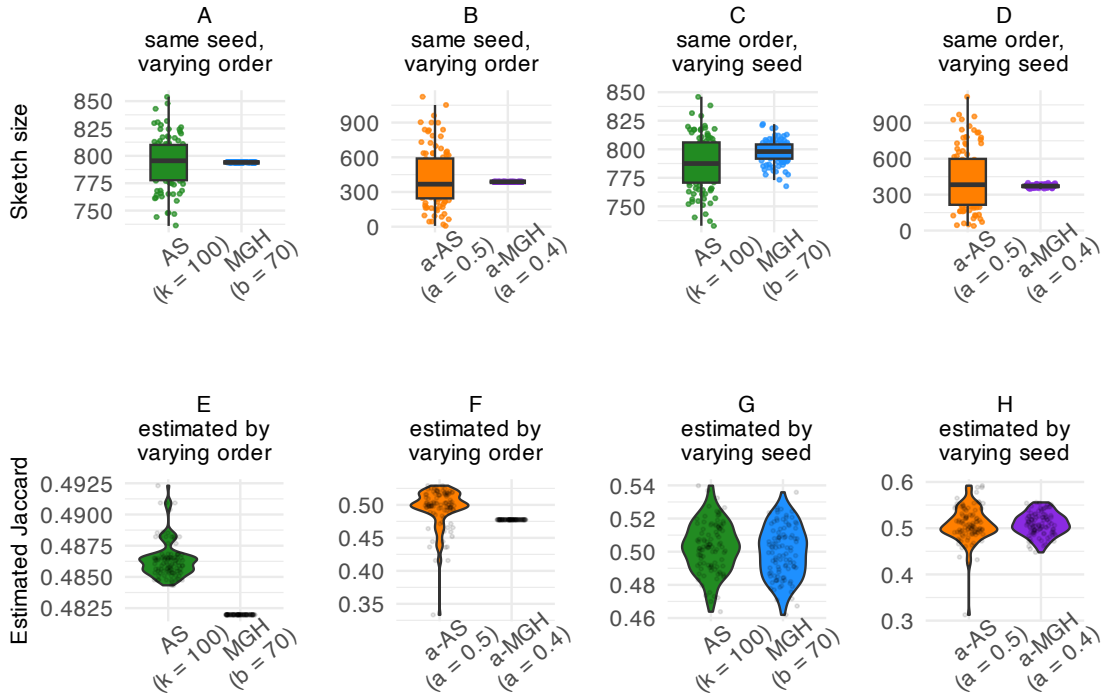

Figure 3: Comparison of AS and MGH sketches on random sets. (A–B) Sketch sizes across 100 random processing orders (fixed hash): AS and α-AS are order-sensitive, while MGH and α-MGH are order-invariant. (C–D) Sketch sizes across 100 hash functions (fixed order): MGH and α-MGH show lower size variability than AS and α-AS at the selected parameters. (E–F) Jaccard estimates (true value 0.5) across random orders: MGH and α-MGH give identical estimates across orders, whereas AS and α-AS vary. (G–H) Jaccard estimates (true value 0.5) across hash functions: AS and MGH show similar variability, while α-AS shows a wider range of estimates than α-MGH. Parameters were selected so that the resulting random samples are similar in sizes.

And finally, when we keep the order of processing the same but use 100 different hash functions to estimate the Jaccard of the same two sets, we find that AS and MGH have a similar degree of variability in estimated Jaccard, but that is not the case with the α-variants, namely α-AS estimates of Jaccard scores span a much wide range that α-MGH, despite the expected sketch sizes being quite similar for both algorithms. This is a direct result of the larger variability of α-AS sketch sizes, as explained earlier.

Together, these results highlight two reasons to prefer MGH over AS: similarity/distance scores estimated using MGH are more stable than those estimated using AS, and results prepared using MGH can be safely recreated owing to the "order invariant" property missing in AS.

MaxGeomHash samples can estimate similarity across the entire range

We next present experiments investigating the quality of estimated similarity scores using MGH and α-MGH samples and the steps outlined earlier. We show only Jaccard estimation results; experiments with other similarity scores were performed, but omitted for brevity.

We randomly generated 5,000 pairs of sets such that the Jaccard similarity between each pair varied from 0.0 to 1.0. Each individual set contains 100,000 alphanumeric strings of length 10. For each pair, we estimated the Jaccard similarity using MGH samples with parameters $b \in \{70, 80, 90, 100\}$, and α-MGH samples with parameters $\alpha \in \{0.4, 0.45, 0.5\}$. These parameters were selected arbitrarily to illustrate MGH

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

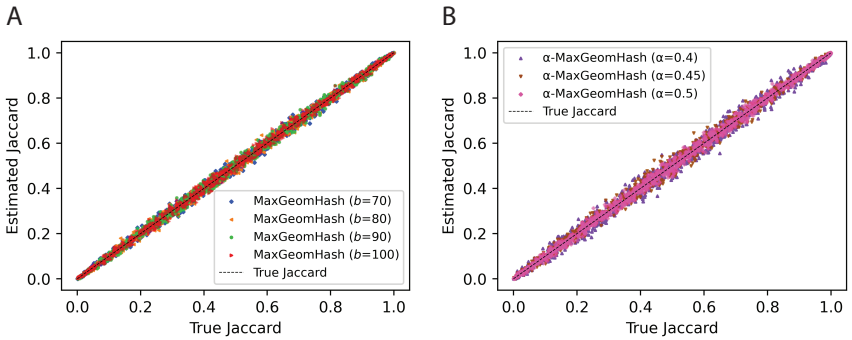

Figure 4: Estimated Jaccard scores against the true Jaccard scores. Estimated Jaccard scores are computed using MGH (shown in Panel A) and α-MGH (shown in Panel B) samples.

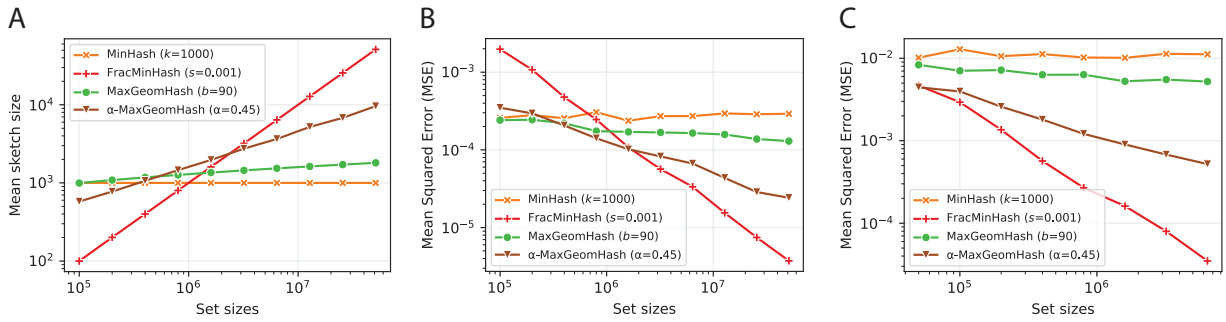

Figure 5: Sketch size and estimation accuracy of sketching algorithms across varying set sizes. (A) Average sketch sizes for four sketching algorithms against sizes of the original sets, computed from 50 independent trials. (B) Mean squared error (MSE) in estimating the Jaccard similarity of two sets, where the sizes of the two sets are equal. The number of elements in the original sets is shown on the horizontal axis. MSE was averaged from 500 independent hash seeds. (C) Mean squared error (MSE) in estimating the containment index of two sets. Size of the smaller set is shown in the horizontal axis. The larger set is 40x larger than the smaller set. Containment of the smaller set in the larger set was estimated using MaxGeomHash sketches. MSE was averaged from 500 independent hash seeds.

and α-MGH's ability to provide for a balance between MinHash and FracMinHash. In practice, the users select the parameter to fit their need for accuracy and available computational resources. In Figure 4, we show these estimated Jaccard values against the true Jaccard scores. Estimated values using MGH samples is shown in Figure 4A and α-MGH shown in Figure 4B. For visual clarity, only 1000 random points are plotted.

Estimated scores highly correlate with the true values (minimum R^2 of 0.9932 over all 7 parameter settings). We note slight variability in the estimated Jaccard values, with the variance decreasing as the sketches become larger as b and α become larger. This behavior is expected in sampling/sketching techniques that trade efficiency for tolerable error. Nonetheless, estimated values remain tightly aligned with true scores without systematic bias, indicating that MGH and α-MGH samples provide accurate, unbiased similarity estimation across the entire range.

MaxGeomHash allows asymptotically unbiased similarity estimation while providing a balance between MinHash and FracMinHash

We next compare MaxGeomHash with two other widely used sketching methods, MinHash and FracMinHash. We generated pairs of random sets (A, B) where the number of elements in A and B is the same and the Jaccard similarity between A and B is fixed at 0.5. We varied the number of elements in A and B from 100K to 50M (million). We focus on slightly larger sets (in contrast to the results presented in Figure 2) to

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

capture the range where FracMinHash sketches start to become larger than MinHash sketches.

For each of these (A, B) pairs, we computed sketches using four sketching algorithms: MinHash, FracMinHash, MGH, and α-MGH. For MinHash and FracMinHash, we used the default parameters ($k = 1000, s = 0.001$, respectively) set in the software tools Mash (Brian D. Ondov et al. 2016) and sourmash (Hera and Koslicki 2025), respectively. These default parameters are widely used in many bioinformatics analyses. To compute MGH and α-MGH sketches, we used $b = 90$, and $\alpha = 0.45$, respectively – which provide a good balance between MinHash and FracMinHash. We note that this specific choice of $b = 90$ and $\alpha = 0.45$ was determined by MinHash and FracMinHash parameters. This parameter choice does not restrict the generality of the results: for any choice of parameters for MinHash and FracMinHash, we can identify b and α values for the two versions of MaxGeomHash that can offer an accuracy-efficiency balance between MinHash and FracMinHash.

For each algorithm, we ran 500 independent trials. In each trial, we computed sketches of the sets A and B using a distinct seed and estimated the Jaccard similarity using these sketches. The average size of the sketches is shown against the number of elements in A and B in Figure 5A. From the estimated values, we also computed the mean squared error, MSE (note that the true Jaccard is fixed at 0.5), and show how the MSE changes as the sets grow large in Figure 5B.

Panels in Figure 5 show complementary results: as sketch size grows, MSE decreases. MinHash sketches are fixed in size, so MSE remains relatively constant. For the other three algorithms, sketch size grows with $|A|$ and $|B|$, and MSE diminishes towards zero – indicating that FracMinHash, MGH, and α-MGH are asymptotically unbiased. With the fastest growing sketch size, FracMinHash exhibits the quickest decrease in MSE. MGH and α-MGH sketches grow slower than FracMinHash sketches, so their MSE decays less quickly, while consuming fewer computational resources than FracMinHash.

We also investigated the use of MGH for estimating the containment index. Because containment is primarily used when two sets differ substantially in size, we generated pairs (A, B) where B is 40 times larger than A. We fixed the containment of A (the smaller set) in B (the larger set) at 0.5 and increased the sizes of both sets in twofold increments. As above, we used the four sketching algorithms to estimate the containment in 500 independent trials. The resulting MSE is shown in Figure 5C; all other experimental settings are the same as those used for Figure 5B. The sketch sizes follow the same trends as those shown in Figure 5A and are omitted for brevity. We observe a similar pattern to the Jaccard results: the MSE decreases most rapidly for FracMinHash and remains approximately constant for MinHash, while MGH and α-MGH provide an intermediate tradeoff, with MSE decreasing as the set sizes grow while using smaller sketches than FracMinHash. We repeated analogous experiments for cosine and Kulczynski 2 similarities and for fixed similarity values ranging from 0.1 to 0.9 in increments of 0.1, and observed similar trends; these results are omitted for brevity.

MaxGeomHash can estimate genomic similarity scores at scale

In this section, we use our algorithm to estimate similarity scores from real genome sequences. For this set of experiments, we obtained 143,614 representative genomes from the GTDB database (Parks et al. 2022). Out of 10.3 billion possible pairs, we restricted our analysis to 20.6 million candidate pairs that have a containment score larger than 0.001. We used KMC (Kokot, Długosz, and Deorowicz 2017) to enumerate all canonical 31-mers, and used these counts to compute the exact Jaccard and containment scores. We used MinHash, MaxGeomHash (both variants), and FracMinHash to estimate the similarity metrics. For fair comparison, we wrote a C++ program to read sequences from genome files, split them into k-mers, compute sketches, and store them as plain text files of hash values. Details of how we acquired the data, of our sketch-generator program, and how we parameters of the sketching algorithms are elaborated in the supplemental materials.

We show the relative absolute error in the Jaccard estimates for the 20.6 million pairs we consider in Figure 6A. The heatmaps depict large errors in red, and accurate estimates in blue. We notice that the constant sketches produced by MinHash results in large relative errors (mostly red), whereas FracMinHash sketches produce highly accurate estimates (mostly blue) by allowing the sketches to grow linearly. With sub-linear growth, the two variants of MaxGeomHash can obtain estimates that are in between MinHash and FracMinHash (less red, more yellow). Figure 6B shows average relative error for all four methods, and numerically quantify the visual presentation given by Figure 6A. In Figure 6C and Figure 6D, we

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

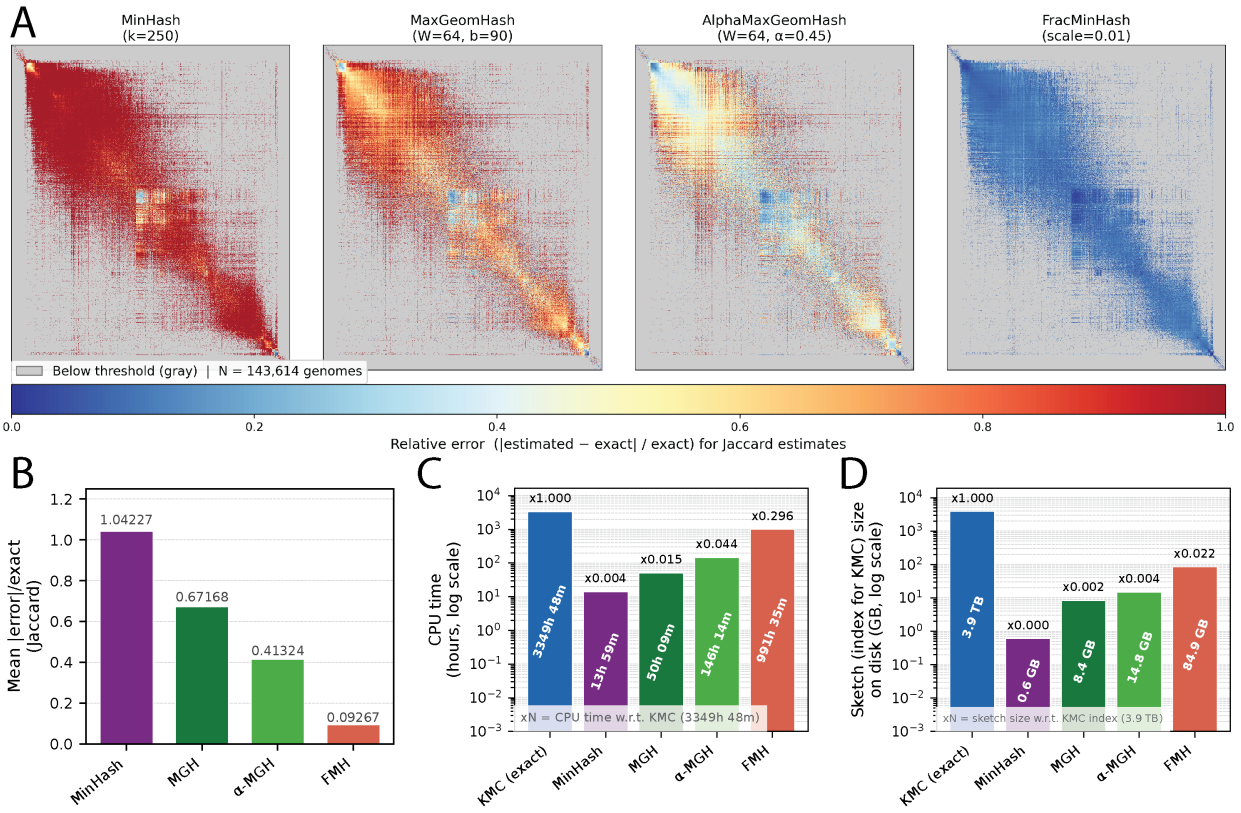

Figure 6: Accuracy–resource trade-offs across sketching methods. (A) Error heatmaps show MinHash is least accurate, FracMinHash most accurate, and MaxGeomHash intermediate in estimating Jaccard. (B) Average error reiterates this trend. (C-D) Runtime and storage show MinHash is most efficient, FracMinHash most costly, with MaxGeomHash providing a balance.

show the CPU time consumed to compute the pairwise similarity scores, and the disk space required to store the sketches (or for KMC, to store the binary k-mer-index), respectively. These plots show that the sketching methods consume a small fraction of what would be needed if we consider all k-mers. MinHash consumes very small amount of resources, whereas FracMinHash consumes the most. In both time and memory, MaxGeomHash variants are between MinHash and FracMinHash, and offer bioinformaticians a range of choices to better manage accuracy-resources trade-offs. Other similar results using estimates of the containment index are included in the supplemental materials for brevity.

MaxGeomHash can compute similarity tree of mammal genomes

We conclude with a real biological data application: estimating a similarity tree (a proxy for phylogenetic tree) among ten mammals. The phylogenetic relationships between mammals are well-established (Upham, Esselstyn, and Jetz 2019). We assess whether k-mer-based sketching techniques can approximate the known phylogeny.

We downloaded ten mammal genome assemblies from Ensembl (Zerbino et al. 2018), extracted canonical k-mers ($k = 31$), and computed k-mer-sketches using four algorithms: MinHash, FracMinHash, MaxGeomHash, and α-MaxGeomHash. We used the same parameters as in Figure 5, with motivation detailed in the corresponding text. We computed pairwise Jaccard similarity by loading all sketches in memory, then converted Jaccard scores to genome-wide average mutation rates (i.e., one minus average nucleotide identity, ANI) assuming a simple mutation model using methods from (Hera, Pierce-Ward, and Koslicki 2023). We then computed a similarity tree from these pairwise mutation rates using UPGMA (Unweighted Pair Group Method with Arithmetic Mean), a standard distance-based method for constructing rooted phylogenetic trees under

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

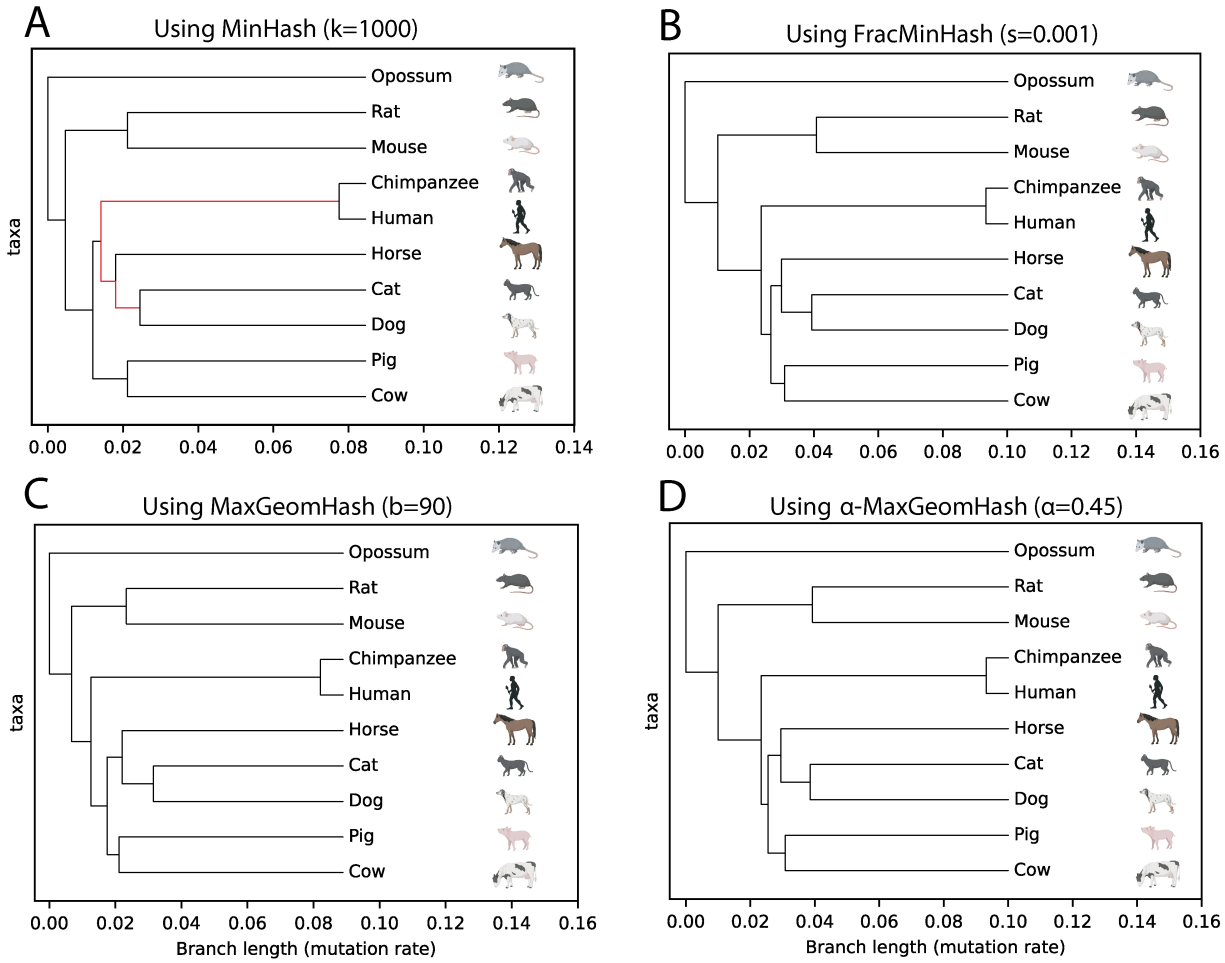

Figure 7: Similarity tree of ten mammal genomes estimated using four sketching methods using k=31-mer-sketches. The red branch in (A) shows that by using MinHash, Carnivora mammals Cat and Dog are placed close to Primates, whereas in reality, they resemble Pig, Cow, and Horse more, and belong in the Laurasiatheria clade. The other three methods consume more resources and correct this mistake, where MGH and α-MGH offer better resource usage compared to FracMinHash (time & memory listed in Table 2). Icons of the mammals are collected from BioRender.com.

a molecular clock assumption (Felsenstein 2004; Nei and Kumar 2000). We also experimented with the Neighbor Joining algorithm and observed similar trends, but opted to present the UPGMA-generated trees for clarity. Estimated trees for each sketching method are shown in Figure 7.

All methods correctly identify Opossum as the outgroup (most distantly related to the others) and correctly group similar organisms: Rat and Mouse (Rodentia), Human and Chimpanzee (Primates), Cat and Dog (Carnivores), and Pig and Cow (Cetartiodactyla). However, all methods incorrectly place rodents and Primates far apart, though these clades are sister groups.

The major discrepancy between the MinHash tree and the others (Figure 7A) is that with only 1000 hashes, Carnivora (Cat and Dog) are placed close to Primates (Human and Chimpanzee), whereas they actually belong in Laurasiatheria with Pig, Cow, and Horse. Using larger sketches, FracMinHash, MGH, and α-MGH correct this error (illustrated in Figure 7B, Figure 7C, and Figure 7D, respectively) and are thus more accurate than MinHash. However, FracMinHash requires more computation due to linear sketch growth, whereas MGH and α-MGH reduce computational costs by producing smaller sketches that grow sub-linearly.

Computational resources are reported in Table 2. We used the same program mentioned in the previous

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

	Computing sketches		Storing sketches	Computing all-pair Jaccard	
Algorithm name	CPU Time (s)	Memory (GB)	Disk space	CPU Time (s)	Memory (MB)
bottom-k MinHash (k = 1000)	4673.3	1.397	160 KB	0.01	4.203
FracMinHash (s=0.001)	3970.84	1.421	360 MB	20.64	977.621
MaxGeomHash (b=90)	4365.62	1.397	878 KB	0.04	5.836
α-MaxGeomHash (α=0.45)	4465.38	1.399	18 MB	0.94	43.953

Table 2: Computational resources used by four sketching algorithms in computing the similarity tree of ten mammals. The trees are shown in Figure 7.

section to compute the sketches. FracMinHash is the fastest to compute despite a larger size, since its logic is simple – whether to keep a hash value doesn't depend on other hash values. The other three methods require maintaining a min (max) heap-like data structure, resulting in slightly higher CPU time. Memory usage for computing sketches is similar across methods, with FracMinHash consuming slightly more due to larger sketches. Disk space for storing sketches starkly highlights size differences: MinHash requires the least space, FracMinHash the most. Finally, computational resources for loading sketches and computing pairwise Jaccard values (and genome-wide average mutation rates) follow sketch sizes, with MinHash consuming the least resources, FracMinHash the most, and MGH and α-MGH providing balance.

We conclude by highlighting that MGH and α-MGH, using parameters $b = 90$ and $\alpha = 0.45$, respectively, can generate clusters that are as accurate as FracMinHash, yet they consume many times less resources when performing pairwise similarity estimation (516x and 22x faster, respectively; 167x and 22x lightweight in memory, respectively; 419x and 22x less storage, respectively). Overall, MaxGeomHash sketches can estimate a similarity tree more efficiently than FracMinHash, and more accurately than MinHash when the parameters are selected carefully. We note that while we used similarity (phylogenetic) tree estimation as an example to highlight this advantage of MaxGeomHash, such benefits can be had in any analysis where sketches are used to estimate similarity/distance.

Discussion

We introduced MaxGeomHash and α-MaxGeomHash, both sketching algorithms that are one-pass, dependable, order-independent, and mergeable. Without knowing the number of distinct elements n, MGH yields an expected distinct element sample size of $b \lg(n/b) + \mathcal{O}(b)$ while α-MGH yields $\Theta(n^\alpha)$, $\alpha \in (0, 1)$. The parameter b in MaxGeomHash allows the user to control the sketch size while keeping the growth of sketches logarithmic, whereas the parameter α in α-MaxGeomHash allows for changing the asymptotic growth order of the sketch itself.

Importantly, MaxGeomHash and α-MaxGeomHash are the first dependable, order-independent, parallelizable, sub-linear sampling methods to date. Unlike existing sub-linear methods such as Affirmative Sampling (Lumbroso and Martínez 2022), which is sensitive to the order of data stream processing and cannot be reliably parallelized, our approach produces identical samples regardless of data partitioning or thread execution order.

This situates MGH and α-MGH directly between MinHash (constant sketch size) and FracMinHash (linear sketch size), obtaining the more accurate scaling feature of FracMinHash with the advantage of smaller sketch sizes. We analyzed sample-size expectation/variance and computational cost, and obtained (asymptotically) unbiased estimators for the Jaccard index, and leveraged recent theory results that automatically extend these estimators to a variety of other metrics (containment, cosine, Kulczynski 1, etc.) as well as provide explicit error bounds and concentration inequalities. Empirical evaluations on simulated and real genomic data confirm our theoretical predictions and demonstrate practical utility, including efficient pairwise similarity computation.

These properties make MGH and α-MGH well-suited for the diverse sketching applications that form a core substrate for sequence search, clustering, quality assessment, phylogenetics, and metagenomic surveillance. Workflows that currently use FracMinHash or MinHash (such as Mash screen (Brian D Ondov et al. 2019), sourmash gather (Irber et al. 2022), Skani (Shaw and William Yu 2023), YACHT (Koslicki, White,

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

et al. 2024), fmh-funcprofiler (Hera, Liu, et al. 2024), sylph (Shaw and Yu 2025), etc.) could be retooled to use MGH to shrink memory/IO budgets without sacrificing accuracy guarantees.

Similarly, recent work relating sketching based metrics to biologically informative metrics like ANI (Hera, Pierce-Ward, and Koslicki 2023), AAI (Pierce-Ward et al. 2022), and d_N/d_S(Rodriguez, Hera, and Koslicki 2025) translate directly to MGH: one merely replaces a FracMinHash estimate of these metrics with the MGH counterpart. Recent large-scale computational biology projects, like the Logan effort (Chikhi et al. 2024) prioritize storage efficiency. Utilizing MGH in such infrastructures and projects can substantially reduce persistent index size with only modest (and exactly quantifiable) accuracy changes, while retaining compatibility with downstream operations on such sketches.

The theoretical analysis of MaxGeomHash can be extended in several ways, for example by better characterizing the error terms we used in Theorem 1 and Theorem 2. A more comprehensive characterization has already been recently attempted (Font et al. 2027), which can be complemented by studying the estimates of the similarity/distance metrics in a similar fashion.

Lastly, we provide an efficient C++ implementation for computing and comparing MGH/α-MGH samples directly from FASTA/FASTQ files, making our methods accessible to the bioinformatics community. Future improvements on the implementation front include refining our C++ implementation, which is not yet fully optimized. There remains headroom from cache-aware bucket layouts, succinct per-bucket priority structures, SIMD-friendly hash partitioning, and task-parallel refinement. These engineering improvements should further reduce runtime and memory while preserving the theoretical guarantees established for MGH and α-MGH.

Methods

In this section, we give the implementation details that were not included in the main text for the sake of brevity.

Implementation of the sketching algorithms

We implemented all sketching algorithms both using Python and using C++. The Python implementation was used for all the simulation experiments (experiments presented in the Results section. The C++ implementation was used to compute the k-mer-sketches of the mammal genomes (experiments presented in Figure 7. In the Python implementation, we used MurMurHash3 function implemented in the Python package "mmh3" (Senuma 2025). We used 64-bit unsigned hash values for our purposes, which is a common practice in bioinformatics applications (Hera and Koslicki 2025; Cox and Hinkley 1974). To maintain the items with the largest/smallest hash values in the non-empty buckets, we used the package "heapq" in Python. For our C++ implementation, we used a public C++ implementation of the same hash function, and also used unsigned 64-bit hash values. We used priority queues, implemented in the Standard Template Library of C++, to maintain the non-empty buckets in our C++ implementation.

Storing k-mer-sketches

We store all sketches in a simple, human-readable plain-text format consisting of a metadata header followed by algorithm-specific data records. Each file begins with several comment lines (prefixed by '#') that record the sketch version, algorithm name, k-mer size, hash seed, hash function, and the parameters relevant to the particular sketching algorithm. After the header, each file lists one record per line. For the MaxGeomHash and α-MaxGeomHash sketch, each line contains four comma-separated fields: the bucket index, the hash value h of an item, the meaningful portion of the hash value h', and a frequency count, although it would suffice to store the bucket index followed by a list of b or f_i trimmed h' hash values. Consequently, neither the frequency counts nor the full hash values are necessary in all applications. This representation encodes all records in the sample produced by MaxGeomHash.

The bottom-k and FracMinHash sketches use simpler formats, each storing a single hash value per line. For bottom-k, the file contains the k smallest hash values observed in the sequence, as specified by the 'params.k'. For FracMinHash, the file lists all hashes that fall below a threshold derived from the sampling scale parameter ('params.scale'), effectively implementing downsampling by a fixed fraction. This

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

minimalistic and uniform design across algorithms ensures ease of debugging and straightforward integration into similarity estimation.

Comparing pairs of k-mer-sketches

When comparing two sketches, we first check if their metadata indicate compatibility – if hash function or seeds differ, or if k-mer sizes differ, or if the parameters of the sketching algorithm employed to compute the sketches differ, then the two sketches are not compatible with each other, and the similarity estimation does not proceed further. If the two sketches are found to be compatible, the algorithm-specific rules are used to compute a similarity metric, such as the cosine similarity or the Jaccard index.

Recording running times, memory usages, and storage usages

We used the standard "/usr/bin/time -v" command to record the CPU time usage, as well as the peak memory usage (maximum resident set size). For the storage usage, the bash utility "find" was used to get the total space required to store the sketches in plain text files.

Data acquisition from GTDB and curation

For the results presented in Figure 7, we used GTDB r226 version (Parks et al. 2022). GTDB provides a phylogenetically consistent bacterial and archaeal taxonomy derived from genomes deposited in GenBank. We selected the high-quality, "representative genome" (i.e. GTDB rep.) for each species cluster. We downloaded the 143,614 genomes as compressed FASTA files. For pairwise comparison, we first identified candidate genome pairs using an all-vs-all FracMinHash sketch screen (scale = 0.01, $k = 31$) with a maximum containment threshold of 0.001, yielding approximately 20.6 million candidate pairs out of the 10.3 billion possible pairs. All downstream comparisons were restricted to these candidate pairs.

Sketching algorithms used to estimate similarity scores on the GTDB representative genomes

We used KMC3 to enumerate all canonical 31-mers for the genomes in our dataset. Using KMC3's k-mer-counts, we computed the Jaccard similarity and containment index as the ground truth. We then used MinHash, FracMinHash, and MaxGeomHash to estimate the Jaccard similarity and the containment index, and compared the estimates with the ground truth. From KMC3's k-mer-counts, we found that the number of distinct 31-mers per genome ranged from 142,000 to 15.9 million (median 2.86M, mean 3.21 M, and standard deviation 1.85M). For these numbers, we found that FracMinHash sketches with 1/100 sampling rate generates very accurate estimates, while MinHash sketches with 250 k-mers yield reasonable accuracy. We ran MGH and α-MGH with parameters $b = 90$ and $\alpha = 0.45$, as these choices provide reasonable accuracy-resource trade-off.

Pairwise containment scores computed on GTDB representative genomes

Similar to the results using Jaccard estimates presented in Figure 6, we also computed the heatmap and accuracy plots for the containment estimates. These results are given in Figures 8 and 9.

Obtaining the mammal genomes

We obtained the ten mammal genomes from Ensembl (Zerbino et al. 2018) Release-112. A list of these mammals and their Ensembl IDs is given in Table 3.

Code availability

A C++ tool to generate MaxGeomHash sketches from genome sequences in standard FASTA format is available as Supplemental Code 1, and also available online at github.com/KoslickiLab/kmer-sketch. Code to

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

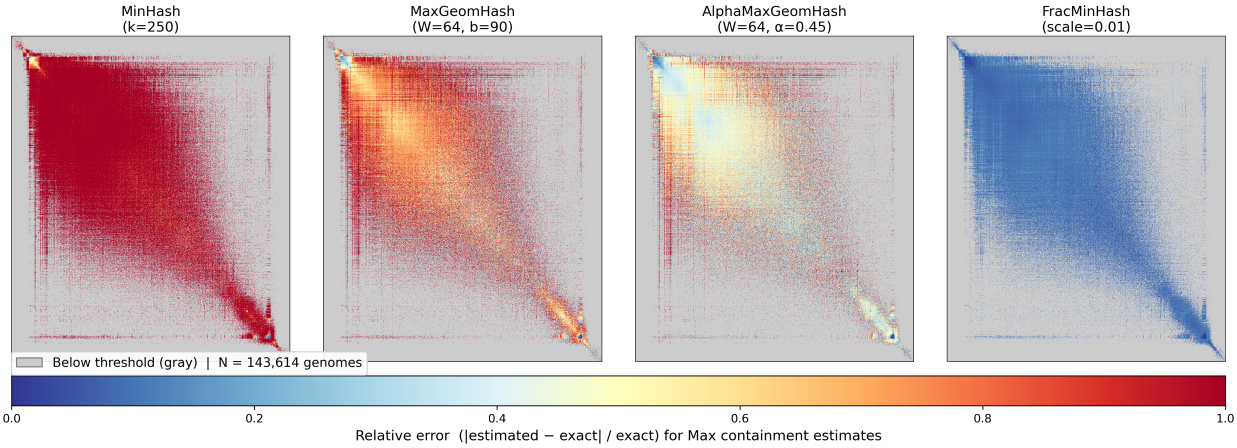

Figure 8: Error heatmaps show MinHash is least accurate, FracMinHash most accurate, and MaxGeomHash intermediate in estimating the containment index.

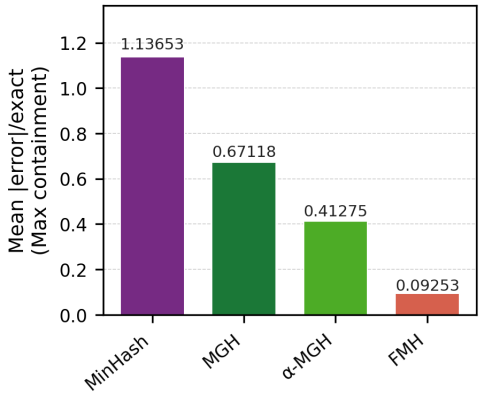

Figure 9: Average relative error in estimating the containment index for four sketching methods.

reproduce the analyses, experiments, and the plots presented in this manuscript is available as Supplemental Code 2, and also available online at github.com/KoslickiLab/MaxGeomHash.

Competing interest statement

The authors have no competing interests to declare that are relevant to the content of this article.

Acknowledgements

The research of David Koslicki has been supported by NIH NIGMS project R01GM146462. The research of Conrado Martínez has been supported by funds from the MOTION Project (Project PID2020-112581GB-C21) of the Spanish *Ministerio de Ciencia e Innovación* (MCIN/AEI/10.13039/501100011033), and the PLASMA Project (Project ANR-25-CE48-3555) of the French *Agence Nationale de la Recherche*. The work was initiated when Mahmudur Rahman Hera was a PhD student at Penn State. During this time, Mahmudur Rahman Hera was supported by NIH NIGMS project R01GM146462. The authors gratefully acknowledge Anat Kreimer for supporting the completion of this work during Mahmudur Rahman Hera's postdoctorate appointment at Rutgers.

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

Standard name	Scientific name	Ensembl assembly ID
Human	*Homo sapiens*	GRCh38
Chimpanzee	*Pan troglodytes*	Pan_tro_3.0
House mouse	*Mus musculus*	GRCm39
Norway rat	*Rattus norvegicus*	mRatBN7.2
Dog	*Canis lupus familiaris*	ROS_Cfam_1.0
Cat	*Felis catus*	Felis_catus_9.0
Cow	*Bos taurus*	ARS-UCD1.3
Pig	*Sus scrofa*	Sscrofa11.1
Horse	*Equus caballus*	EquCab3.0
Gray short-tailed opossum	*Monodelphis domestica*	ASM229v1

Table 3: Mammalian reference genomes downloaded from Ensembl (release 112) using `wget`.

Author contributions: All authors agree that each author contributed equally to this work. As such, their names appear in alphabetical order by last name. A fine-grained contributions summary is as follows.

Mahmudur Rahman Hera: Software (lead); Validation (lead); Investigation (lead); Data curation (lead); Visualization (lead); Formal analysis (supporting); Conceptualization (supporting); Methodology (supporting); Writing – original draft (co-lead); Writing – review & editing (supporting).

David Koslicki: Supervision (lead); Resources (lead); Funding acquisition (lead); Investigation (supporting); Visualization(supporting); Formal analysis (supporting); Conceptualization (supporting); Methodology (supporting); Writing – original draft (supporting); Writing – review & editing (lead).

Conrado Martínez: Conceptualization (lead); Formal analysis (lead); Methodology (lead); Writing – original draft (co-lead); Writing – review & editing (supporting); supervision (supporting).

References

Agarwal, Pankaj K. et al. (Dec. 2013). "Mergeable summaries". In: *ACM Trans. Database Syst.* 38.4. ISSN: 0362-5915. DOI: 10.1145/2500128. URL: https://doi.org/10.1145/2500128.

Broder, Andrei Z. (1997). "On the resemblance and containment of documents". In: *Proc. Compression and Complexity of SEQUENCES*. Ed. by Bruno Carpentieri et al. IEEE, pp. 21–29. DOI: 10.1109/SEQUEN.1997.666900.

Carter, J. Lawrence and Mark N. Wegman (1979). "Universal Classes of Hash Functions". In: *Journal of Computer and System Sciences* 18.2, pp. 143–154. DOI: 10.1016/0022-0000(79)90044-8.

Chikhi, Rayan et al. (2024). "Logan: planetary-scale genome assembly surveys life's diversity". In: *bioRxiv*, pp. 2024–07.

Cohen, Edith and Haim Kaplan (2007). "Bottom-k sketches: better and more efficient estimation of aggregates". In: *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '07. San Diego, California, USA: ACM, pp. 353–354. ISBN: 9781595936394. DOI: 10.1145/1254882.1254926.

Cormen, Thomas H. et al. (2022). *Introduction to Algorithms*. 4th ed. MIT Press.

Cox, David R. and D. V. Hinkley (1974). *Theoretical Statistics*. London: Chapman & Hall.

Dasgupta, Sanjoy, Christos Papadimitriou, and Umesh Vazirani (2006). *Algorithms*. McGraw-Hill Education.

Efraimidis, Pavlos S. and Paul G. Spirakis (2006). "Weighted Random Sampling with a Reservoir". In: *Information Processing Letters* 97.5, pp. 181–185. DOI: 10.1016/j.ipl.2005.11.003. URL: https://doi.org/10.1016/j.ipl.2005.11.003.

Felsenstein, Joseph (2004). *Inferring phylogenies*. Sunderland, Massachusetts, USA: Sinauer Associates, Inc.

Flajolet, Ph. et al. (2007). "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm". In: *Proceedings of 2007 Int. Conference on Analysis of Algorithms (AofA 07)*. Ed. by Ph. Jacquet. Vol. AH. Discrete Mathematics & Theoretical Computer Science (Proceedings), pp. 127–146.

Font, Àlex et al. (2027). "A Detailed Analysis of the Size of MaxGeomHash Sketches". In: *Similarity Search and Applications*. Ed. by Klaus Schoeffmann et al. Cham: Springer Nature Switzerland, pp. 52–66. ISBN: 978-3-032-42224-8.

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

Goodman, Leo A. and William H. Kruskal (1961). "Measures of Association for Cross Classifications IV: Simplification of Asymptotic Variances". In: *Journal of the American Statistical Association* 56.296, pp. 779–802. DOI: 10.1080/01621459.1961.10482149.

Hera, Mahmudur Rahman and David Koslicki (2024). "Cosine Similarity Estimation Using FracMinHash: Theoretical Analysis, Safety Conditions, and Implementation". In: *24th Int. Workshop on Algorithms in Bioinformatics (WABI 2024)*. Ed. by Solon P. Pissis and Wing-Kin Sung. Vol. 312. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:16. DOI: 10.4230/LIPICS.WABI.2024.6.

Hera, Mahmudur Rahman and David Koslicki (2025). "Estimating similarity and distance using FracMinHash". In: *Algorithms Mol. Biol.* 20.1, p. 8. DOI: 10.1186/S13015-025-00276-8.

Hera, Mahmudur Rahman, Shaopeng Liu, et al. (2024). "Metagenomic functional profiling: to sketch or not to sketch?" In: *Bioinformatics* 40.Supplement_2, pp. ii165–ii173.

Hera, Mahmudur Rahman, N Tessa Pierce-Ward, and David Koslicki (2023). "Deriving confidence intervals for mutation rates across a wide range of evolutionary distances using FracMinHash". In: *Genome Research* 33.7, pp. 1061–1068.

Irber, Luiz et al. (2022). "Lightweight compositional analysis of metagenomes with FracMinHash and minimum metagenome covers". In: *bioRxiv*. DOI: 10.1101/2022.01.11.475838. URL: https://www.biorxiv.org/content/early/2022/01/17/2022.01.11.475838.

Irber Jr, Luiz Carlos (2020). *Decentralizing indices for genomic data*. University of California, Davis.

Knuth, Donald E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3rd ed. Reading, MA, USA: Addison-Wesley. ISBN: 978-0-201-89684-8.

Kokot, Marek, Maciej Długosz, and Sebastian Deorowicz (2017). "KMC 3: counting and manipulating k-mer statistics". In: *Bioinformatics* 33.17, pp. 2759–2761.

Koslicki, David, Stephen White, et al. (2024). "YACHT: an ANI-based statistical test to detect microbial presence/absence in a metagenomic sample". In: *Bioinformatics* 40.2, btae047.

Koslicki, David and Hooman Zabeti (2019). "Improving minhash via the containment index with applications to metagenomic analysis". In: *Applied Mathematics and Computation* 354, pp. 206–215.

Liu, Shaopeng and David Koslicki (2022). "CMash: fast, multi-resolution estimation of k-mer-based Jaccard and containment indices". In: *Bioinformatics* 38.Supplement_1, pp. i28–i35.

Lumbroso, Jérémie O. and Conrado Martínez (2022). "Affirmative Sampling: Theory and Applications". In: *33rd Int. Conf. on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms (AofA 2022)*. Ed. by Mark D. Ward. Vol. 225. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 12:1–12:17. DOI: 10.4230/LIPICS.AOFA.2022.12.

Martínez, Conrado, Alfredo Viola, and Jun Wang (2023). "Unbiased Similarity Estimators Using Samples". In: *Similarity Search and Applications - 16th International Conference, SISAP 2023, A Coruña, Spain, October 9-11, 2023, Proceedings*. Ed. by Oscar Pedreira and Vladimir Estivill-Castro. Vol. 14289. Lecture Notes in Computer Science. Springer, pp. 56–63. DOI: 10.1007/978-3-031-46994-7_5.

Mitzenmacher, Michael and Salil Vadhan (2013). "Why simple hash functions work: Exploiting the entropy in a data stream". In: *Theory of Computing* 9.30, pp. 897–945.

Motwani, Rajeev and Prabhakar Raghavan (1995). *Randomized Algorithms*. Cambridge University Press. DOI: 10.1017/CBO9780511814075.

Nei, Masatoshi and Sudhir Kumar (2000). *Molecular evolution and phylogenetics*. Oxford University Press.

Ondov, Brian D et al. (2019). "Mash Screen: high-throughput sequence containment estimation for genome discovery". In: *Genome Biology* 20.1, p. 232.

Ondov, Brian D. et al. (2016). "Mash: fast genome and metagenome distance estimation using MinHash". In: *Genome Biology* 17.1, p. 132.

Parks, Donovan H et al. (2022). "GTDB: an ongoing census of bacterial and archaeal diversity through a phylogenetically consistent, rank normalized and complete genome-based taxonomy". In: *Nucleic Acids Research* 50.D1, pp. D785–D794.

Pierce-Ward, N Tessa et al. (2022). *Amino Acid k-mers enable assembly-and alignment-free sequence analysis*. Tech. rep. Manubot. URL: https://bluegenes.github.io/2022-paper-protein-kmers.

Rodriguez, Judith, Mahmudur Rahman Hera, and David Koslicki (2025). "Leveraging FracMinHash Containment for Genomic dN/dS". In: *bioRxiv preprint*. URL: https://doi.org/10.1101/2025.11.12.688019.

Sanz-González, Hugo and Conrado Martínez (2025). "Fast and Accurate Estimates for External Clustering Validation Measures". In: *Similarity Search and Applications Conference - 18th International Conference,*

Downloaded from genome.cshlp.org on November 29, 2026 . Published by Cold Spring Harbor Laboratory Press

SISAP 2025, Reykjavík, Iceland, October 1-3, 2025, Proceedings. Ed. by Giuseppe Amato et al. Vol. 16134. Lecture Notes in Computer Science. Springer, pp. 248–262. DOI: 10.1007/978-3-032-06069-3_20.

Senuma, Hajime (Jan. 2025). "mmh3: A Python extension for MurmurHash3". In: *Journal of Open Source Software* 10.105, p. 6124. ISSN: 2475-9066. DOI: 10.21105/joss.06124.

Shaw, Jim and Yun William Yu (2023). "Fast and robust metagenomic sequence comparison through sparse chaining with skani". In: *Nature Methods* 20.11, pp. 1661–1665.

Shaw, Jim and Yun William Yu (2025). "Rapid species-level metagenome profiling and containment estimation with sylph". In: *Nature Biotechnology* 43.8, pp. 1348–1359.

Upham, Nathan S, Jacob A Esselstyn, and Walter Jetz (2019). "Inferring the mammal tree: species-level sets of phylogenies for questions in ecology, evolution, and conservation". In: *PLoS biology* 17.12, e3000494.

Vitter, Jeffrey S. (1985). "Random Sampling with a Reservoir". In: *ACM Trans. Math. Softw.* 11.1, pp. 37–57. DOI: 10.1145/3147.3165.

Yu, Yun William and Griffin M Weber (2020). "Hyperminhash: Minhash in loglog space". In: *IEEE Transactions on Knowledge and Data Engineering* 34.1, pp. 328–339.

Zerbino, Daniel R et al. (2018). "Ensembl 2018". In: *Nucleic Acids Research* 46.D1, pp. D754–D761.