HyperMinHash: Jaccard index sketching in LogLog space

Extended Abstract

Y. William Yu
Harvard Medical School DBMI
Boston, Massachusetts
william_yu@hms.harvard.edu

Griffin Weber
Harvard Medical School DBMI
Boston, Massachusetts
griffin_weber@hms.harvard.edu

ABSTRACT
In this extended abstract, we describe and analyse a streaming probabilistic sketch, HYPERMINHASH, to estimate the Jaccard index (or Jaccard similarity coefficient) over two sets A and B. HyperMinHash can be thought of as a compression of standard MinHash by building off of a HyperLogLog count-distinct sketch. Given Jaccard index δ, using k buckets of size O(log(l) + log log(|A ∪ B|)) (in practice, typically 2 bytes) per set, HyperMinHash streams over A and B and generates an estimate of the Jaccard index δ with error O(1/l + √k/δ). This improves on the best previously known sketch, MinHash, which requires the same number of storage units (buckets), but using O(log(|A ∪ B|)) bit per bucket. For instance, our new algorithm allows estimating Jaccard indices of 0.01 for set cardinalities on the order of 10^{19} with relative error of around 5% using 64KB of memory; the previous state-of-the-art MinHash can only estimate Jaccard indices for cardinalities of 10^{10} with the same memory consumption. Alternately, one can think of HyperMinHash as an augmentation of b-bit MinHash that enables streaming updates, unions, and cardinality estimation (and thus intersection cardinality by way of Jaccard), while using log log extra bits.

CCS CONCEPTS
• Theory of computation → Sketching and sampling:

KEYWORDS
MinHash, sketching, streaming, loglog, Jaccard

ACM Reference Format:
Y. William Yu and Griffin Weber. 2017. HyperMinHash: Jaccard index sketching in LogLog space: Extended Abstract. In Proceedings of arXiv preprint (arXiv 2017). ACM, New York, NY, USA, 11 pages. https://doi.org/00.000/000_0

1 INTRODUCTION
Many questions in data science can be rephrased in terms of the number of items in a database that satisfy some Boolean formula. For example, "how many participants in a political survey are independent and have a favorable view of the federal government?", or "how many of the source IP addresses used in a DDoS attack today were also used last month?" In this paper, we consider the design of approximate streaming sketches to answer questions phrased in conjunctive normal form (an AND of ORs); this is of course equivalent to estimating the cardinality of intersections of unions of a collection of sets. The literature already has near-optimal probabilistic data structures for approximating the 'count-distinct' problem [1, 4-6], which is equivalent to finding the cardinality of unions of sets (i.e. ORs in CNF). Thus, we in particular focus on the problem of estimating Jaccard index[7], a proxy for set similarity that when coupled with union cardinality, allows estimation of intersection sizes.

1.1 Jaccard index
Given two sets A and B, where |A| = n and |B| = m, and n > m without loss of generality, the Jaccard index is defined as

δ(X, Y) = \frac{|A \cap B|}{|A \cup B|}. \tag{1}

Clearly, if paired with a good count-distinct union estimator for |A ∪ B|, this allows us to estimate intersection sizes as well. Though Jaccard originally defined this index to measure ecological diversity in 1902 [7], in more modern times, it has been used as a proxy for the document similarity problems. In 1997, Broder introduced min-wise hashing (colloquially known as 'MinHash') [2], a technique for quickly estimating the resemblance of documents by looking at the Jaccard index of ‘shingles’ (collections of phrases) contained in the documents.

1.2 MinHash
MinHash relies on a simple fact: if you apply a random permutation to the universe of elements, the chance that the smallest items under this permutation in sets A and B are the same is precisely the Jaccard index. To see this, consider a random permutation of A ∪ B. The minimum element will come from either A \setminus B, B \setminus A, or A \cap B, all disjoint sets. If the minimum element lies in A \setminus B, then min(A) \neq B, so min(A) \neq min(B); the same is of course true by symmetry for B \setminus A. Conversely, if min(A ∪ B) ∈ A \cap B, then clearly min(A) = min(B). Because the permutation is random, every element has an equal probability of being the minimum, and thus

P (min(A) = min(B)) = \frac{|A \cap B|}{|A \cup B|}. \tag{2}

While using a single random permutation produces an unbiased estimator of δ(X, Y), it is a Bernoulli 0/1 random variable with high variance. So, instead of using a single permutation, one averages k trails. The expected fraction of matches is also an unbiased estimator of the Jaccard index, but with variance decreased by a factor of 1/k. Though the theoretical justification is predicated on having a true random permutation, in practice we approximate that by using good
random hash functions instead. A good hash function will specify a nearly total ordering on the universe of items, and provided we use $\theta(nk)$ bits for the hash function output, the probability of accidental collision of min-hashes is exponentially small.

Though theoretically easy to analyze, this scheme has a number of drawbacks, chief amongst them the requirement of having $k$ random hash functions, which means that the computational complexity is $\theta(nk)$ to generate the sketch. To address this, several variants of MinHash have been proposed [3].

1. **k-hash functions.** The scheme described above, which has the shortcoming of using $\theta(nk)$ computation to generate the sketch.

2. **k-minimum values.** A single hash function is used, but instead of storing the single minimum value, we store the smallest $k$ values for each set (also known as the KMV sketch [1]). Sketch generation time is reduced to $O(n \log k)$, but we also incur an $O(k \log k)$ sorting penalty when computing Jaccard index.

3. **k-partition.** Another 1-permutation MinHash variant, $k$-partition stochastically averages by first deterministically partitioning a set into $k$ parts using the first couple bits of the hash value, and then stores the minimum value in each partition [9]. $k$-partition has the advantage of $O(n)$ sketch generation time and $O(k)$ Jaccard index computation time, at the cost of some difficulty in the analysis.

It is important here to remark that for all of the above variants, MinHash sketches of $A$ and $B$ can be losslessly combined to form the MinHash sketch of $A \cup B$. Using order statistics, it is additionally possible to estimate the union cardinalities [1], so these sketches can all be used to directly estimate intersection size in addition to just Jaccard index. This also of course directly implies that streaming updates are permitted, so preprocessing incurs no additional space requirement.

### 1.3 log log space complexity

All of the variants of MinHash given in the last section require logarithmic bits per bucket in order to prevent accidental collisions (i.e. we want to ensure that when two hashes match, they came from identical elements), though in the case of $k$-partition, some of those bits can be stored implicitly in the bucket identity. However, in the similar problem of cardinality estimation of unique items (the ‘count-distinct’ problem), literature over the last several decades produced several streaming sketches that require sub-logarithmic bits per bucket; indeed, the LogLog, SuperLogLog, and HyperLogLog family of sketches requires, as given in the name, only $O(\log \log (n))$ bits per bucket by storing only the position of the first 1 bit of a uniform hash function [4–6].

We wanted to do the same thing for the Jaccard index problem. First note that HyperLogLog union cardinalities can be used to compute intersection cardinalities using the inclusion-exclusion principle, but that the relative error is then in the size of the union (as opposed to the size of the Jaccard index for MinHash) and compounds when taking the intersections of multiple sets; for small intersections, the error is often too great to be practically feasible. Alternately, when unions and streaming updates are not necessary, the more recent advance of $b$-bit MinHash [8] solves exactly this problem while using only a ‘constant’ number of bits per bucket.

$b$-bit MinHash operates in the same way as standard MinHash, but after computing the minimum hash value, stores only the lowest order $b$ bits. Indeed, for very large Jaccard similarity $\delta(A, B) > 0.5$, Li, et al. determined that even using 1 bit per bucket was asymptotically optimal. In general, for small Jaccard similarity, $b$-bit MinHash needs $O(\log(1/\delta))$ bits, without any dependence on the sizes of the sets. For estimating the Jaccard similarity between exactly two sets, $b$-bit MinHash is nearly asymptotically optimal [10].

However, $b$-bit MinHash, while great for Jaccard index, loses many of the benefits of standard MinHash. Because $b$-bit MinHash only takes the lowest order $b$ bits of the minimum hash value after finding the minimum, it also requires $\log(n)$ bits per bucket during the sketch generation phase, the same as standard MinHash. This also implies that sketches cannot be merged together, so union cardinalities cannot be estimated.

Some of these shortcomings can be overcome by pairing a $b$-bit MinHash sketch with a HyperLogLog count-distinct sketch. HyperLogLog uses $\log \log (n)$ bits per bucket to estimate union cardinalities, and combined, these two allow for accurate estimation of intersection sizes, not just Jaccard index. However, this still does not permit the usage of unions or streaming updates, so more complex predicates (e.g. $|(A \cup B) \cap C|$) still cannot be evaluated and the data structure still requires $O(\log(n) + \log \log(n))$ bits per bucket during the sketch generation phase.

We resolved this issue, by building a new sketch, HyperMinHash, as a hybrid of sorts between HyperLogLog and $k$-partition MinHash. Using the same amount of space as $b$-bit MinHash + HyperLogLog, we achieve better streaming performance (using only $O(\log \log(n) \log(1/\delta))$ bits per bucket at all stages of the process), the ability to take unions of sketches, and count-distinct cardinality estimation. In Table 1, we summarize some of the properties of the various methods we’ve described above.

The paper is organized as follows: first, we present the detailed algorithm. Then, we prove error and scaling bounds. Finally, we give some practical and empirical optimizations and experiments, along with pseudocode for implementation.

### 2 HYPERMINHASH IN THEORY

#### 2.1 Intuition

MinHash works under the premise that two sets will have identical minimum value with probability equal to the Jaccard distance, because they can only share a minimum value if that minimum value corresponds to a member of the intersection of those two sets. If we have a total ordering on the union of both sets, the fraction of equal buckets is an unbiased estimator for Jaccard distance. However, with limited precision hash functions, there is some chance of accidental collision, when the value does not correspond to a member of the intersection. In order to get close to a true total ordering, the space of potential hashes must be on the order of the size of the union, and thus we must store $O(\log n)$ bits.

Note, however, that the minimum of of a collection of uniform $[0, 1]$ random variables $X_1, \ldots, X_n$ is much more likely to be a small number than a large one (the insight behind most count-distinct sketches [1]). HyperMinHash operates identically to MinHash, but
Table 1: Comparison of key features against other methods.

| Method                        | Bits per bucket† | Unions‡ | Jaccard index§ | Intersection size | Streaming updates |
|-------------------------------|------------------|---------|----------------|-------------------|-------------------|
| MinHash                      | log(\(n\))      | ✓       | ✓              | ✓                 | ✓                 |
| b-bit MinHash                | \(\log(1/\delta)\) | ✓       | ✓              | ✓                 | ✓                 |
| HyperLogLog                  | \(\log \log(\(n\))\) | ✓       | ✓              | ✓                 | ✓                 |
| HyperLogLog + MinHash        | \(\log(\(n\)) + \log(\log(\(n\)))\) | ✓       | ✓              | ✓                 | ✓                 |
| HyperLogLog + b-bit MinHash  | \(\log(\(n\)) + \log(1/\delta)\) | ✓       | ✓              | ✓                 | ✓                 |
| HyperMinHash                 | \(\log(\(n\)) + \log(1/\delta)\) | ✓       | ✓              | ✓                 | ✓                 |

†: \(n\) is the cardinality of the sets and \(\delta\) is the Jaccard indexes, where applicable.

‡: Where applicable, \(\Theta(1/\varepsilon^2)\) buckets are required for union cardinality estimation with relative error \(\varepsilon\).

§: All of the MinHash based methods also require \(\Theta(1/\delta)\) buckets to give accurate Jaccard indexes.

Figure 1: HyperMinHash generates sketches in the same fashion as one-permutation MinHash. It begins by hashing each object in the set to a uniformly random number between 0 and 1, encoded in binary. Then, the hashed values are partitioned by the first \(p\), and the minimum value within each partition is taken. Each value is specified by a tuple; the first part is the minimum of \(P_q\), where \(q\) bits (and store \(2^q\) + 1 if there is no 1 bit in the first \(2^q\) bits), and \(r\) bits following that (Figure 1). We will give up on having a total ordering so long as the number of accidental collisions in the minimum values is low.

To analyze the performance of HyperMinHash compared to random-permutation MinHash (or equivalently 0-collision standard MinHash) it suffices to consider the expected number of accidental collisions. In this intuitive analysis here, we will only analyze the simple case of collisions while using only a single bucket, but the instead of storing the minimum values with fixed precision, it effectively uses an adaptive precision that increases resolution when the values are smaller by using initial loglog counters (from LogLog cardinality estimation) and then storing a fixed number of bits beyond that (similar in spirit to b-bit minhash). More precisely, after dividing up the items into \(k\) partitions, we store the position of the leading 1 bit in the first \(2^q\) bits (and store \(2^q + 1\) if there is no 1 bit in the first \(2^q\) bits), and \(r\) bits following that (Figure 1). We will give up on having a total ordering so long as the number of accidental collisions in the minimum values is low.

To analyze the performance of HyperMinHash compared to random-permutation MinHash (or equivalently 0-collision standard MinHash) it suffices to consider the expected number of accidental collisions. In this intuitive analysis here, we will only analyze the simple case of collisions while using only a single bucket, but the
same flavor of argument holds for multiple partitions. The HyperLogLog part of the sketch results in collisions whenever two items match in order of magnitude (Figure 2).

Figure 2: HyperLogLog sections, used alone, result in collisions whenever the minhashes match in order of magnitude.

By pairing it with an additional \( r \)-bit bucket, our collision space is narrowed to precisely the diagonal row of \( b \)-bit minHash, but only assuming that we store the leading 1 indicator as large as it might be. An explicit exact formula for the expected number of collisions is

\[
\mathbb{E}[C] = \sum_{i=1}^{\infty} \sum_{j=0}^{2^r-1} \left( \frac{2^r + j}{2^{i+r}} \right)^n \left( \frac{2^r + j + 1}{2^{i+r}} \right)^m - \left( \frac{2^r + j}{2^{i+r}} \right)^n \left( \frac{2^r + j + 1}{2^{i+r}} \right)^m, \tag{3}
\]

though finding a closed formula is rather more difficult.

Intuitively, suppose that our hash value is \((12, 01011101)\) for \(X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\). By pairing it with an additional 6-bit bucket, our collision space is narrowed to precisely the diagonal row of \(b\)-bit minHash, but only assuming that we store the leading 1 indicator as large as it might be. An explicit exact formula for the expected number of collisions is

\[
\mathbb{E}[C] = \sum_{i=1}^{\infty} \sum_{j=0}^{2^r-1} \left( \frac{2^r + j}{2^{i+r}} \right)^n \left( \frac{2^r + j + 1}{2^{i+r}} \right)^m - \left( \frac{2^r + j}{2^{i+r}} \right)^n \left( \frac{2^r + j + 1}{2^{i+r}} \right)^m, \tag{3}
\]

Though finding a closed formula is rather more difficult.

Intuitively, suppose that our hash value is \((12, 01011101)\) for partition 01. This implies that the original bitstring of the minimum hash was 0.0100000000000101011101 · · · . Then a uniform random hash in \([0, 1]\) collides with this number with probability \(2^{-(27+12+8)} = 2^{-21}\). So we expect to need cardinalities on the order of \(2^{21}\) before having many collisions. But of course, as the cardinalities of \(A\) and \(B\) increase, so does the expected value of the leading 1 in the bitstring, as analyzed in the construction of HyperLogLog [6]. Thus, the collision probabilities remain roughly constant as cardinalities increase, at least until we reach the precision limit of the LogLog counters.

But of course, we store only a finite number of bits for the leading 1 indicator (often 6 bits). Because it’s a LogLog counter, storing 6 bits is sufficient for set cardinalities up to \(O(2^6 = 2^{64})\). This increases our collision surface though, as we might have collisions in the lower left region near the origin (Figure 4). We can directly compute the collision probability (and similarly the variance) by summing together the probability mass in these boxes, replacing the infinite sum with a finite sum (Lemma 2.6). For more sensitive estimations, we can actually subtract out the expected number of collisions to un-bias the estimation. However, as before, finding a closed form of this equation is difficult. In the next section we will prove bounds on the expectation and variance in the number of collisions.

2.2 Proofs

The main result of this paper bounds the expectation and variance of accidental collision, given two HyperMinHash sketches of disjoint sets. First, we rigorously define the full HyperMinHash sketch.

**Definition 2.1.** We will define \(f_{p,q,r}(A) : S \rightarrow \{(1, \ldots, 2^q) \times (0, 1)^r\}\) to be the HyperMinHash sketch constructed from Figure 1, where \(A\) is a set of hashable objects and \(p, q, r \in \mathbb{N}\), and let \(f_{p,q,r}(A)_i : S \rightarrow \{(1, \ldots, 2^q) \times (0, 1)^r\}\) be the value of the \(i\)th bucket in the sketch.

More precisely, let \(h(x) : S \rightarrow [0, 1]\) be a uniformly random hash function.

Let \(\rho_q(x) = \min([-\log_q(x)] + 1, 2^q)\).

Let \(\sigma_r(x) = \lfloor x 2^r \rfloor\).

Let \(\tilde{\hat{h}}_{q,r}(x) = (\rho_q(x), \sigma_r(x 2^q \rho_q(x) - 1))\).

Then, we will define

\[
f_{p,q,r}(A)_i = \tilde{\hat{h}}_{q,r} \min_{a \in A} \min_{i \leq \rho_q(a) < (i+1)2^p} (h(a)2^p - i)
\]
Definition 2.2. Let $A, B$ be hashable sets with $|A| = n, |B| = m$, $n > m$, and $A \cap B = \emptyset$. Then we can define an indicator variable for collisions in bucket $i$ of their respective HyperMinHash sketches

$$Z_{p, q, r}(A, B, i) = \mathbb{1}_{(p_{q}, r(A)) = (p_{q}, r(B))}.$$  

(4)

Our main theorems follow:

Theorem 2.3. $C = \sum_{i=0}^{2^{q}-1} Z_{p, q, r}(A, B, i)$ is the number of collisions between the HyperMinHash sketches of two disjoint sets $A$ and $B$. Then the expectation

$$\mathbb{E}[C] \leq 2^{p} \left( \frac{5}{2^{r}} + \frac{n}{2^{p+2q+r}} \right).$$  

(5)

Theorem 2.4. Given the same setup as in Theorem 2.3,

$$\text{Var}(C) \leq \mathbb{E}[C]^2 + \mathbb{E}[C].$$  

(6)

Theorem 2.3 allows us to correct for the number of random collisions before computing Jaccard distance, and Theorem 2.4 tells us that the standard deviation in the number of collisions is approximately the expectation.

Before proving these theorems, we will start by proving a simpler proposition.

Proposition 2.5. Consider a HyperMinHash sketch with only 1 bucket on two disjoint sets $A$ and $B$. i.e. $f_{0, q, r}(A)$ and $f_{0, q, r}(B)$. Let $\gamma(n, m) \sim Z_{0, q, r}(A, B, 0)$. Naturally, as a good hash function results in uniform random variables, $\gamma$ is only dependent on the cardinalities $n$ and $m$. We claim that

$$\mathbb{E}[\gamma(n, m)] \leq \frac{6}{2^{q}} + \frac{n}{2^{2q+r}}$$  

(7)

Proving this will require a few technical lemmas, which we’ll then use to prove the main theorems.

Lemma 2.6.

$$\mathbb{E}[\gamma(n, m)] = \mathbb{P}(f_{0, q, r}(A_0) = f_{0, q, r}(B_0))$$

$$= \sum_{i=1}^{2^{q}-1} \sum_{j=0}^{2^{p}-1} \left[ \left( 1 - \frac{2^{r} + j}{2^{q+r}} \right)^{n} - \left( 1 - \frac{2^{r} + j + 1}{2^{q+r}} \right)^{n} \right]$$

$$+ \sum_{i=2^{q}}^{2^{q}-1} \sum_{j=0}^{2^{p}-1} \left[ \left( 1 - \frac{j}{2^{q+r}} \right)^{m} - \left( 1 - \frac{j + 1}{2^{q+r}} \right)^{m} \right].$$

Proof. Let $a_1, \ldots, a_n$ be random variables corresponding to the hashed values of items in $A$. Then $a_i \in [0, 1]$ are uniform r.v. Similarly, $b_1, \ldots, b_m$, drawn from hashed values of $B$ are uniform $[0, 1]$ r.v. Let $x = \min(a_1, \ldots, a_n)$ and $y = \min(b_1, \ldots, b_m)$. Then we have probability density functions

$$pdf(x) = n(1-x)^{n}, \text{ for } x \in [0, 1]$$  

(8)

$$pdf(y) = m(1-y)^{m}, \text{ for } y \in [0, 1]$$  

(9)

and cumulative density functions

$$cdf(x) = 1 - (1-x)^{n}, \text{ for } x \in [0, 1]$$  

(10)

$$cdf(y) = 1 - (1-y)^{m}, \text{ for } y \in [0, 1].$$  

(11)

We are particularly interested in the joint probability density function

$$pdf(x, y) = n(1-x)^{n}m(1-y)^{m}, \text{ for } (x, y) \in [0, 1]^2.$$  

(12)

The probability mass enclosed in a square along the diagonal $S = \{ s_1, s_2 \}^2 \subset [0, 1]^2$ is then precisely

$$\mu(S) = \int_{s_1}^{s_2} \int_{s_1}^{s_2} n(1-x)^{n}m(1-y)^{m}dydx$$

$$= \left[ (1-s_2)^{m} - (1-s_1)^{m} \right] \left[ (1-s_2)^{m} - (1-s_1)^{m} \right].$$

(14)

Recall $f_{0, q, r}(A_0) \in \{1, \ldots, 2^{q} \} \times [0, 1] \equiv \{1, \ldots, 2^{q} \} \times [0, \ldots, 2^{r} - 1]$, so given $f_{0, q, r}(A_0) = (i, j)$, $x = 0.0000\ldots$, in the binary expansion, unless $i = 2^{q}$, in which case the binary expansion is $x = 0.0000\ldots$. That in turn gives $s_1 < x < s_2$, where $s_1 = \frac{2^{r}j}{2^{q+r}}, s_2 = \frac{2^{r}j+1}{2^{q+r}}$ when $i < 2^{q}$, and $s_1 = \frac{2^{r}j}{2^{q+r}}, s_2 = \frac{j+1}{2^{q+r}}$. Collisions happen precisely when $s_1 < x, y < s_2$.

Finally, using the $s_1, s_2$ formulas above, it suffices to sum the probability of collision over the image of $f$, so

$$\mathbb{E}[\gamma(n, m)] = \sum_{i=1}^{2^{q}-1} \sum_{j=0}^{2^{p}-1} \mu((s_1, s_2)).$$

(15)

Substituting in for $s_1, s_2$, and $\mu$ completes the proof. Note also that this is precisely the sum of the probability mass in the red and purple squares along the diagonal in Figure 4. \hfill $\square$

Figure 4: In practice, HyperMinHash has a limited number of bits for the loglog counters, so there’s a final lower left bucket at the precision limit.
pdf of min(Y₁, ..., Yₘ)

pdf of min(X₁, ..., Xₙ)

Figure 5: We’ll upper bound the collision probability of hyperminhash by dividing it into these four regions of integration: (a) the Top Right orange box, (b), the magenta ray covering intermediate boxes, (c) the black strip covering all but the final purple box, and (d) the final purple subbucket by the origin.

While Lemma 2.6 allows us to explicitly compute $E\gamma(m, n)$, the lack of a closed form solution makes reasoning about it difficult. Here, we’ll upper bound the expectation by integrating over four regions of the unit square that cover all the collision boxes (Figure 5). For ease of notation, let $\bar{\epsilon} = 2^n$ and $\bar{q} = 2^m$.

- The Top Right box $TR = [\frac{1}{\bar{\epsilon} + 1}, 1]^2$ (in orange in Figure 5).
- The magenta triangle from the origin bounded by the lines $y = \frac{1}{\bar{q}}x$ and $y = \frac{\bar{q} + 1}{\bar{q}}x$ with $0 < x < \frac{1}{\bar{\epsilon}}$, which we will denote $RAY$.
- The black strip near the origin covering all the purple boxes except the one on the origin, bounded by the lines $y = x - \frac{1}{\bar{q}}$, $y = x + \frac{1}{\bar{q}}$, and $\frac{1}{\bar{\epsilon}} < x < \frac{1}{\bar{q}}$, which we will denote $STRIP$.
- The Bottom Left purple box $BL = [0, \frac{1}{\bar{\epsilon}}]^2$.

**Lemma 2.7.**

\[ \mu(TR) \leq \frac{1}{\bar{\epsilon}} \]

**Proof.** By Equation 14,

\[ \mu(TR) = \int_{\frac{1}{\bar{\epsilon}} + 1}^{1} \int_{\frac{1}{\bar{\epsilon}} + 1} n(1-x)^{n-1} m(1-y)^{m-1} dy dx \]

\[ = \left[ -(1-x)^n \right]_{\frac{1}{\bar{\epsilon}} + 1}^{1} \left[ -(1-y)^m \right]_{\frac{1}{\bar{\epsilon}} + 1}^{1} \]

\[ = \frac{1}{(\bar{\epsilon}+1)^{n+m}} \leq \frac{1}{\bar{\epsilon}} \]

\[ \square \]

**Lemma 2.8.**

\[ \mu(BL) \leq \frac{n}{\bar{\epsilon} \bar{q}} \]

**Proof.**

\[ \mu(BL) = \int_{0}^{\frac{\bar{\epsilon}}{\bar{\epsilon}+1}} \int_{0}^{\frac{\bar{q}}{\bar{q}+1}} n(1-x)^{n-1} m(1-y)^{m-1} dy dx \]

\[ = \left[ -(1-x)^n \right]_{0}^{\frac{\bar{\epsilon}}{\bar{\epsilon}+1}} \left[ -(1-y)^m \right]_{0}^{\frac{\bar{q}}{\bar{q}+1}} \]

\[ = 1 - \left( 1 - \frac{1}{\bar{\epsilon}} \right)^n \left( 1 - \frac{1}{\bar{\bar{q}}} \right)^m \]

For $n, m < \bar{\epsilon} \bar{q}$, we note that the linear binomial approximation is actually a strict upper bound (as can be trivially verified through the Taylor expansion), so

\[ \mu(BL) \leq \frac{nm}{\bar{\epsilon} \bar{q}} \leq \frac{n}{\bar{\epsilon}} \]

\[ \square \]

**Lemma 2.9.**

\[ \mu(RAY) \leq \frac{3}{\bar{\epsilon}} \]

**Proof.** Unfortunately, the ray is not aligned to the axes, so we cannot integrate $x$ and $y$ separately.

\[ \mu(RAY) = \int_{0}^{\frac{\bar{\epsilon}}{\bar{\epsilon}+1}} \int_{\frac{1}{\bar{\epsilon}+1}}^{\frac{\bar{\epsilon}}{\bar{\epsilon}+1}} n(1-x)^{n-1} m(1-y)^{m-1} dy dx \]

\[ = \int_{0}^{\frac{\bar{\epsilon}}{\bar{\epsilon}+1}} n(1-x)^{n-1} \left( \frac{1}{\bar{\epsilon}+1} x \right)^m - \left( \frac{\bar{\epsilon}}{\bar{\epsilon}+1} x \right)^m \right| \]

\[ (16) \]

Using the elementary difference of powers formula, note that for $0 < \alpha \leq \beta \leq 1$,

\[ \alpha^m - \beta^m = (\alpha - \beta) \left( \sum_{i=1}^{m} \alpha^{m-i} \beta^{i-1} \right) \leq (\alpha - \beta) m \beta^{m-1} \]

Applying this to Eq. 16, we can conclude with a bit of symbolic manipulation that

\[ \mu(RAY) \leq \int_{0}^{\frac{\bar{\epsilon}}{\bar{\epsilon}+1}} n(1-x)^{n-1} \left( \frac{2 \bar{\epsilon} + 1}{\bar{\epsilon}(\bar{\epsilon} + 1)} \right)^m \left( \frac{1 - \bar{\epsilon}}{\bar{\epsilon}+1} x \right)^{m-1} \right| \]

\[ \leq \frac{2 \bar{\epsilon} + 1}{\bar{\epsilon}(\bar{\epsilon} + 1)} \int_{0}^{\frac{\bar{\epsilon}}{\bar{\epsilon}+1}} nm \left( \frac{1 - \bar{\epsilon}}{\bar{\epsilon}+1} x \right)^{n+m-2} x dx \]

With a straight-forward integration by parts, we get

\[ R \leq - \frac{2 \bar{\epsilon} + 1}{\bar{\epsilon}^2} \cdot \frac{nm}{n + m - 1} \cdot \frac{\bar{\epsilon}}{\bar{\epsilon} + 1} \cdot \left( \frac{1 - \bar{\epsilon}^2}{(\bar{\epsilon} + 1)^2} \right)^{n+m-1} \]

\[ = - \frac{2 \bar{\epsilon} + 1}{\bar{\epsilon}^2} \cdot \frac{nm}{n + m - 1} \cdot \bar{\epsilon} + 1 \cdot \frac{1}{\bar{\epsilon} + 1} \cdot \left( \frac{1 - \bar{\epsilon}^2}{(\bar{\epsilon} + 1)^2} \right)^{n+m-1} \]

but note that the first two terms above are negative, and so can be upper bounded by 0, resulting in

\[ P_{RAY} \leq \left( \frac{2 \bar{\epsilon} + 1}(\bar{\epsilon} + 1) \right)^m \cdot \frac{nm}{(n + m)(n + m - 1)} \leq \frac{3}{\bar{\epsilon}} \]

\[ \square \]
Recall that the dependence into the new indicator variables given in Lemmas 2.10 and 2.8 by substituting \( \tilde{\alpha} \) because those boxes are also covered by the Magenta Ray from 2.5. Notice first that

\[
\mathbb{E}_y(n, m) \leq \frac{6}{\varepsilon} + \frac{n}{\varepsilon q} = \frac{6}{2^\ell} + \frac{n}{2^{2\ell + \tau}}
\]

Proof of Proposition 2.5. We need only sum up the bounds from Lemmas 2.7, 2.8, 2.9, and 2.10 to conclude

\[
\mathbb{E}_y(n, m) \leq \frac{6}{\varepsilon} + \frac{n}{\varepsilon q} = \frac{6}{2^\ell} + \frac{n}{2^{2\ell + \tau}}
\]

Proof of Theorem 2.3. Let \( A_i, B_i \) be the \( i \)th partitons of \( A \) and \( B \) respectively. Recall that \( |A| = n \geq m = |B| \). For ease of notation, let’s define \( \tilde{\phi} = 2^\phi \).

Recall that

\[
C = \sum_{j=0}^{\tilde{p}-1} \mathbb{E}_{Z_{p, q}, r}(A, B, j).
\]

We will first bound \( \mathbb{E}_{Z_{p, q}, r}(A, B, j) \) using the same techniques used in Proposition 2.5. Notice first that \( Z_{p, q, r}(A, B, j) \) effectively rescales the minimum hash values from \( Z_{0, q, r}(A, B, j) = y(n, m) \) down by a factor of \( 2^p \); i.e. we scale down both the axes in Figure 5.

Now, we no longer need the Top Right bound from Lemma 2.7 because those boxes are also covered by the Magenta Ray from Lemma 2.9. Furthermore, we scale down the bounds of integration in Lemmas 2.10 and 2.8 by substituting \( \tilde{q} \leftarrow q \tilde{\phi} \).

Summing these together, we readily conclude

\[
\mathbb{E}_{Z_{p, q}, r}(A, B, j) \leq \frac{5}{\varepsilon} + \frac{n}{\varepsilon q \tilde{\phi}} = \frac{5}{2^\ell} + \frac{n}{2^{2\ell + 2\tau + \tau}}
\]

Then by linearity of expectation,

\[
\mathbb{E}C \leq 2^\phi \left( \frac{5}{2^\ell} + \frac{n}{2^{2\ell + 2\tau + \tau}} \right)
\]

Proof of Theorem 2.4. We can decompose \( C \) into

\[
C = \sum_{\alpha_1 + \cdots + \alpha_p = n} \sum_{\beta_1 + \cdots + \beta_p = m} \mathbb{E}_{Z_{p, q}, r}(A, B, j)_{|A_i| = \alpha_i \land |B_i| = \beta_i}
\]

by conditioning on the full multinomial distribution and removing the dependence into the new indicator variables.

For ease of notation in the following, we will use \( \tilde{\alpha}, \tilde{\beta} \) to denote the event \( \forall i, |A_i| = \alpha_i \) and \( \forall i, |B_i| = \beta_i \) respectively. Additionally, let \( \tilde{Z}(j) = Z_{p, q, r}(A, B, j) \). So,

\[
C = \sum_{\tilde{\alpha}, \tilde{\beta}} \sum_{j=0}^{\tilde{p}-1} \mathbb{I}_{\tilde{\alpha}, \tilde{\beta}} \tilde{Z}(j)
\]

Then

\[
\mathbb{E}\text{Var}(C) = \sum_{\tilde{\alpha}, \tilde{\beta}} \sum_{j=0}^{\tilde{p}-1} \mathbb{I}_{\tilde{\alpha}, \tilde{\beta}} \tilde{Z}(j)
\]

and vice versa, because they are disjoint indicator variables. As such, for \( \tilde{\alpha}, \tilde{\beta} \neq (\tilde{\alpha}_2, \tilde{\beta}_2) \),

\[
\mathbb{E}\text{Var}(C) = \sum_{\tilde{\alpha}, \tilde{\beta}} \sum_{j=0}^{\tilde{p}-1} \mathbb{I}_{\tilde{\alpha}, \tilde{\beta}} \tilde{Z}(j)
\]

implying that

\[
\mathbb{E}\text{Var}(C) = \sum_{\tilde{\alpha}, \tilde{\beta}} \sum_{j=0}^{\tilde{p}-1} \mathbb{I}_{\tilde{\alpha}, \tilde{\beta}} \tilde{Z}(j)
\]

Note that the first term is simply

\[
\sum_{j=0}^{\tilde{p}-1} \mathbb{I}_{\tilde{\alpha}, \tilde{\beta}} \tilde{Z}(j)
\]

and because \( \tilde{Z} \) is a \{0, 1\} Bernoulli r.v.,

\[
\sum_{j=0}^{\tilde{p}-1} \mathbb{E}\tilde{Z}(j) = \sum_{j=0}^{\tilde{p}-1} \mathbb{E}\tilde{Z}(j) = \mathbb{E}C
\]

Moving on, from the covariance formula, for independent random variables \( X_1, X_2, Y \),

\[
\text{Cov}(X_1, X_2 Y) = \mathbb{E}(X_1 X_2 Y) - \mathbb{E}(X_1 Y) \mathbb{E}(X_2 Y)
\]

\[
= \mathbb{E}(X_1) \mathbb{E}(X_2) \mathbb{E}(Y^2) - \mathbb{E}(Y) \mathbb{E}(X_1) \mathbb{E}(X_2) \mathbb{E}(Y)
\]
Thus the second term of the summation can be bounded as follows:

\[
\sum_{\hat{a}, \hat{b}} \sum_{0 \leq j_1 < j_2 \leq F - 1} \text{Cov} \left( \mathbb{1}_{\hat{a}, \hat{b}} \hat{Z} \left( j_1 | \hat{a}, \hat{b} \right), \mathbb{1}_{\hat{a}, \hat{b}} \hat{Z} \left( j_2 | \hat{a}, \hat{b} \right) \right)
\]

\[
= \sum_{\hat{a}, \hat{b}} \sum_{0 \leq j_1 < j_2 \leq F - 1} \mathbb{E} \left[ \hat{Z} \left( j_1 | \hat{a}, \hat{b} \right) \right] \mathbb{E} \left[ \hat{Z} \left( j_2 | \hat{a}, \hat{b} \right) \right] \text{Var} \left( \mathbb{1}_{\hat{a}, \hat{b}} \right)
\]

\[
\leq \sum_{\hat{a}, \hat{b}} \sum_{j = 0}^{F - 1} \mathbb{E} \left[ \hat{Z} \left( j \hat{a}, \hat{b} \right) \right] \mathbb{E} \left[ \hat{Z} \left( j \hat{a}, \hat{b} \right) \right] \text{Var} \left( \mathbb{1}_{\hat{a}, \hat{b}} \right)
\]

\[
= \sum_{\hat{a}, \hat{b}} \mathbb{E} \left[ C \hat{a}, \hat{b} \right] ^2 \text{Var} \left( \mathbb{1}_{\hat{a}, \hat{b}} \right)
\]

\[
= \sum_{\hat{a}, \hat{b}} \mathbb{E} \left[ C \hat{a}, \hat{b} \right] ^2 \mathbb{P} \left( \hat{a}, \hat{b} \right) \left( 1 - \mathbb{P} \left( \hat{a}, \hat{b} \right) \right)
\]

\[
\leq \sum_{\hat{a}, \hat{b}} \mathbb{E} \left[ C \hat{a}, \hat{b} \right] ^2 \mathbb{P} \left( \hat{a}, \hat{b} \right)
\]

\[
= \mathbb{E} [C] ^2
\]

We conclude that

\[
\text{Var}(C) \leq \mathbb{E} [C] ^2 + \mathbb{E} [C]
\]

\[\blacksquare\]

3 HYPERMINHASH IN PRACTICE

Here, we present full algorithms to match a naive implementation of HyperMinHash as described in the previous Theory section. A Python implementation is available at https://github.com/yunwilliamyu/hyperminhash.

3.1 Implementation optimizations

We recommend several optimizations for practical implementations of HyperMinHash. First, it is mathematically equivalent to:

1. Pack the hashed tuple into a single word; this enables Jaccard index computation while using only one comparison per bucket instead of two.

2. Use the max instead of min of the subbuckets. This allows us to take the union of two sketches while using only one comparison per bucket.

These recommendations should be self-explanatory, and are simply minor engineering optimizations, which we do not use in our prototyping, as they do not affect accuracy. However, while we can exactly compute the number of expected collisions through Lemma 2.6, this computation is slow and often results in floating point errors unless BigInts are used. In practice, two ready solutions present themselves:

1. We can ignore the bias and simply add it to the error. As the bias and standard deviation of the error are the same order of magnitude, this only doubles the absolute error in the estimation of Jaccard index. For large Jaccard indexes, this does not matter.

2. We also present a fast, numerically stable, algorithm to approximate the expected number of collisions (Algorithm 6).
We can also use other k-minimum value count-different cardinality estimators, which we empirically found useful for large cardinalities.  

Algorithm 3 Estimate Cardinality. Note that the left parts of all the buckets can be passed directly into a HyperLogLog estimator. We can also use other k-minimum value count-different cardinality estimators, which we empirically found useful for large cardinalities.

```
function EstimateCardinality(S, p, q, r)
    Initialize |S| integer registers \( b_1 = b_2 = \ldots = b_{|S|} = 0 \).
    for \( i \in \{1, \ldots, |S|\} \) do
        \( b_i \leftarrow S_i[0] \)
    end for
    \( R \leftarrow \text{HyperLogLogCardinalityEstimator}\{b_1\}, q\)
    if \( R < 1024|S| \) then
        return \( R \)
    else
        Initialize |S| real registers \( r_1, \ldots, r_{|S|} \).
        for \( i \in \{1, \ldots, |S|\} \) do
            \( r_i \leftarrow 2^{-S_i[0]} \cdot (1 + S_i[1]) \)
        end for
        if \( \sum r_i = 0 \) then
            return \( \infty \)
        else
            return \( \frac{|S|^2}{\sum r_i} \)
        end if
    end if
end function
```

Algorithm 4 Compute Jaccard Index. Note that the correction factor \( EC \) is generally not needed, except for really small Jaccard index. Additionally, for most practical purposes, it is safe to substitute ApproximateExpectedCollisions for ExpectedCollisions.

```
function JaccardIndex(S, T, p, q, r)
    assert \( |S| = |T| \)
    \( C \leftarrow 0 \)
    \( N \leftarrow 0 \)
    for \( i \in \{1, \ldots, |S|\} \) do
        \( S_i \leftarrow T_i \)
        \( C \leftarrow C + 1 \)
    end if
    if \( S_i = (0, 0) \) and \( T_i = (0, 0) \) then
        \( N \leftarrow N + 1 \)
    end if
    \( n \leftarrow \text{EstimateCardinality}(S, q) \)
    \( m \leftarrow \text{EstimateCardinality}(T, q) \)
    \( EC \leftarrow \text{ApproximateExpectedCollisions}(n, m, p, q, r) \)
    return \( (C - \text{EC}) / N \)
end function
```

Algorithm 5 Expected collisions. Note that because of floating point error, BigInts must be used for large \( n \) and \( m \).

```
function ExpectedCollisions(n, m, p, q, r)
    \( x \leftarrow 0 \)
    for \( i \in \{1, \ldots, 2^q\} \) do
        for \( j \in \{1, \ldots, 2^q\} \) do
            if \( i \neq 2^q \) then
                \( b_1 \leftarrow \frac{2^x + j}{2^x + j + 1} \)
                \( b_2 \leftarrow \frac{2^y + j}{2^y + j + 1} \)
            else
                \( b_1 \leftarrow \frac{j}{2^y + j + 1} \)
                \( b_2 \leftarrow \frac{j + 1}{2^y + j + 1} \)
            end if
            \( Pr_x \leftarrow (1 - b_1)^n - (1 - b_1)^n \)
            \( Pr_y \leftarrow (1 - b_2)^m - (1 - b_2)^m \)
            \( x \leftarrow x + Pr_x Pr_y \)
        end for
    end for
    return \( x \cdot 2^p \)
end function
```

Algorithm 6 Fast numerically stable approximation to Algorithm 5. Generally underestimates collisions.

```
function ApproximateExpectedCollisions(n, m, p, q, r)
    if \( n < m \) then
        \( \text{SWAP}(x, y) \)
    end if
    if \( n > 2^{q+r} \) then
        return \( \text{ERROR: cardinality too large for approximation.} \)
    else if \( n > 2^{q+5} \) then
        \( \phi \leftarrow \frac{4n/m}{(1+n/m)^2} \)
        \( \text{return } 0.16991948715973909375315012348 \cdot 2^{p-r} \phi \)
    else
        \( \text{return } \text{ExpectedCollisions}(n, m, p, q, 0) \cdot 2^{-r} \)
    end if
end function
```

We can however approximate the number of expected collisions using the following procedure, which is empirically asymptotically correct (Algorithm 6):

1. For \( n < 2^{q+5} \), we approximate by taking the number of expected HyperLogLog collisions and dividing it by \( 2^q \). In each HyperLogLog box, we are interested in collisions along \( 2^q \) boxes along the diagonal \( 4 \). For this approximation, we simply assume that the joint probability density function is almost uniform within the box; this is not completely accurate, but pretty close in practice.

2. For \( 2^{q+5} < n < 2^{q+5}p \), we noted empirically that the expected number of collisions approached \( 0.1699 \cdot 2^{p-r} \) for \( n = m \) as \( n \to \infty \). Furthermore, the number of collisions is dependent on \( n \) and \( m \) by a factor of \( \frac{4n/m}{(1+n/m)^2} \) from 2.9, which for \( n, m \gg 1 \) can be approximated by \( \frac{4n/m}{(1+n/m)^2} \). This approximation is primarily needed because of floating point errors when \( n \to \infty \).

3. Unfortunately, around \( n > 2^{q+5}p \), the number of collisions starts increasing and these approximations fail. However, note that for reasonable values of \( q = 6, p = 15 \), this problem only appears when \( n > 2^{30} \approx 10^{26} \).
3.2 Experimental validation

For the sake of completeness, we include some experimental validation showing the behavior of Jaccard index estimation as the union cardinality grows. In Figure 6, we allocate 256 bytes for both two standard MinHash sketch and a HyperMinHash sketch. For a fixed sketch size and cardinality range, HyperMinHash is more accurate; or, for fixed sketch size and bucket number, HyperMinHash can access exponentially larger set cardinalities. As an aside, with expected collision correction, both methods can be improved, but only until the expected number of collisions reaches the number of bins (i.e. when the uncorrected Jaccard estimate reaches 1), so for these experiments, we have omitted expected error collision for all methods.

4 DISCUSSION AND CONCLUSION

We have introduced HyperMinHash, a sketch for estimating Jaccard distance using log log space, and made available a prototype Python implementation at https://github.com/yunwilliamyu/hyperminhash. It can be thought of as a compression scheme for MinHash that reduces the number of bits per bucket to \( \log \log(n) \) from \( \log(n) \) by using insights from HyperLogLog and \( b \)-bit MinHash. As with the original MinHash, it retains variance on the order of \( k/\delta \), where \( k \) is the number of buckets and \( \delta \) is the Jaccard index between two sets. However, it also introduces \( 1/l^2 \) variance, where \( l = 2^r \), because of the increased number of collisions, which matches the requirements of \( b \)-bit MinHash.

For practical parameters of \( p = 15, q = 6, r = 10 \), the HyperMinHash sketch will use up 64KiB memory per set, and allow for estimating Jaccard indices of 0.01 for set cardinalities on the order of \( 10^{19} \) with accuracy around 5%. HyperMinHash is to our knowledge the first streaming summary sketch capable of directly estimating union cardinality, Jaccard index, and intersection cardinality in log log space, able to be applied to arbitrary Boolean formulas in conjunctive normal form with error rates bounded by the final result size.

ACKNOWLEDGMENTS

This study was supported by National Institutes of Health (NIH) Big Data to Knowledge (BD2K) awards U54HG007963 from the National Human Genome Research Institute (NHGRI) and U01CA198934 from the National Cancer Institute (NCI). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. We additionally thank Adam Sealfon and Daphne Ippolito for useful comments and advice.
REFERENCES

[1] Ziv Bar-Yossef, Ts Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. 2002. Counting distinct elements in a data stream. In International Workshop on Randomization and Approximation Techniques in Computer Science. Springer, 1–10.

[2] Andrei Z Broder. 1997. On the resemblance and containment of documents. In Compression and Complexity of Sequences 1997. Proceedings. IEEE, 21–29.

[3] Edith Cohen. 2016. Min-Hash Sketches. (2016).

[4] Marianne Durand and Philippe Flajolet. 2003. Loglog counting of large cardinalities. In European Symposium on Algorithms. Springer, 605–617.

[5] Philippe Flajolet. 2004. Counting by coin tossings. In Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making: Springer, 1–12.

[6] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In AofA: Analysis of Algorithms. Discrete Mathematics and Theoretical Computer Science, 137–156.

[7] Paul Jaccard. 1902. Lois de distribution florale dans la zone alpine. Bull Soc Vaudoise Sci Nat 38 (1902), 69–130.

[8] Ping Li and Christian König. 2010. b-Bit minwise hashing. In Proceedings of the 19th international conference on World wide web. ACM, 671–680.

[9] Ping Li, Art Owen, and Cun-Hui Zhang. 2012. One permutation hashing. In Advances in Neural Information Processing Systems. 3113–3121.

[10] Rasmus Pagh, Morten Stöckel, and David P Woodruff. 2014. Is min-wise hashing optimal for summarizing set intersection?. In Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 109–120.