

SetSketch: Filling the Gap between MinHash and HyperLogLog

Otmar Ertl
Dynatrace Research
Linz, Austria
otmar.ertl@dynatrace.com

ABSTRACT

MinHash and HyperLogLog are sketching algorithms that have become indispensable for set summaries in big data applications. While HyperLogLog allows counting different elements with very little space, MinHash is suitable for the fast comparison of sets as it allows estimating the Jaccard similarity and other joint quantities. This work presents a new data structure called SetSketch that is able to continuously fill the gap between both use cases. Its commutative and idempotent insert operation and its mergeable state make it suitable for distributed environments. Fast, robust, and easy-to-implement estimators for cardinality and joint quantities, as well as the ability to use SetSketch for similarity search, enable versatile applications. The presented joint estimator can also be applied to other data structures such as MinHash, HyperLogLog, or HyperMinHash, where it even performs better than the corresponding state-of-the-art estimators in many cases.

PVLDB Reference Format:

Otmar Ertl. SetSketch: Filling the Gap between MinHash and HyperLogLog. PVLDB, 14(11): 2244 - 2257, 2021.
doi:10.14778/3476249.3476276

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/dynatrace-research/set-sketch-paper>.

1 INTRODUCTION

Data sketches [16] that are able to represent sets of arbitrary size using only a small, fixed amount of memory have become important and widely used tools in big data applications. Although individual elements can no longer be accessed after insertion, they are still able to give approximate answers when querying the cardinality or joint quantities which may include the intersection size, union size, size of set differences, inclusion coefficients, or similarity measures like the Jaccard and the cosine similarity.

Numerous such algorithms with different characteristics have been developed and published over the last two decades. They can be classified based on following properties [53] and use cases:

Idempotency: Insertions of values that have already been added should not further change the state of the data structure. Even though this seems to be an obvious requirement, it is not satisfied by many sketches for sets [11, 47, 58, 67, 69].

Commutativity: The order of insert operations should not be relevant for the final state. If the processing order cannot be guaranteed, commutativity is needed to get reproducible results. Many data structures are not commutative [13, 32, 65].

Mergeability: In large-scale applications with distributed data sources it is essential that the data structure supports the union set operation. It allows combining data sketches resulting from partial data streams to get an overall result. Ideally, the union operation is idempotent, associative, and commutative to get reproducible results. Some data structures trade mergeability for better space efficiency [11, 13, 65].

Space efficiency: A small memory footprint is the key purpose of data sketches. They generally allow to trade space for estimation accuracy, since variance is typically inversely proportional to the sketch size. Better space efficiencies can sometimes also be achieved at the expense of recording speed, e.g. by compression [21, 36, 62].

Recording speed: Fast update times are crucial for many applications. They vary over many orders of magnitude for different data structures. For example, they may be constant like for the HyperLogLog (HLL) sketch [30] or proportional to the sketch size like for minwise hashing (MinHash) [6].

Cardinality estimation: An important use case is the estimation of the number of elements in a set. Sketches have different efficiencies in encoding cardinality information. Apart from estimation error, robustness, speed, and simplicity of the estimation algorithm are important for practical use. Many algorithms rely on empirical observations [34] or need to combine different estimators for different cardinality ranges, as is the case for the original estimators of HLL [30] and HyperMinHash [72].

Joint estimation: Knowing the relationship among sets is important for many applications. Estimating the overlap of both sets in addition to their cardinalities eventually allows the computation of joint quantities such as Jaccard similarity, cosine similarity, inclusion coefficients, intersection size, or difference sizes. Data structures store different amounts of joint information. In that regard, for example, MinHash contains more information than HLL. Extracting joint quantities is more complex compared to cardinalities, because it involves the state of two sketches and requires the estimation of three unknowns in the general case, which makes finding an efficient, robust, fast, and easy-to-implement estimation algorithm challenging. If a sketch supports cardinality estimation and is mergeable, the joint estimation problem can be solved using the inclusion-exclusion principle $|U \cap V| = |U| + |V| - |U \cup V|$. However, this naive approach is inefficient as it does not use all information available.

Locality sensitivity: A further use case is similarity search. If the same components of two different sketches are equal with a probability that is a monotonic function of some similarity measure, it can be used for locality-sensitive hashing (LSH) [2, 35, 43, 73], an

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476276

indexing technique that allows querying similar sets in sublinear time. For example, the components of MinHash have a collision probability that is equal to the Jaccard similarity.

The optimal choice of a sketch for sets depends on the application and is basically a trade-off between estimation accuracy, speed, and memory efficiency. Among the most widely used sketches are MinHash [6] and HLL [30]. Both are idempotent, commutative, and mergeable, which is probably one of the reasons for their popularity. The fast recording speed, simplicity, and memory-efficiency have made HLL the state-of-the-art algorithm for cardinality estimation. In contrast, MinHash is significantly slower and is less memory-efficient with respect to cardinality estimation, but is locality-sensitive and allows easy and more accurate estimation of joint quantities.

1.1 Motivation and Contributions

The motivation to find a data structure that supports more accurate joint estimation and locality sensitivity like MinHash, while having a better space-efficiency and a faster recording speed that are both closer to those of HLL, has led us to a new data structure called SetSketch. It fills the gap between HLL and MinHash and can be seen as a generalization of both as it is configurable with a continuous parameter between those two special cases. The possibility to fine-tune between space-efficiency and joint estimation accuracy allows better adaptation to given requirements.

We present fast, robust, and simple methods for cardinality and joint estimation that do not rely on any empirically determined constants and that also give consistent errors over the full cardinality range. The cardinality estimator matches that of both MinHash or HLL in the corresponding limit cases. Together with the derived corrections for small and large cardinalities, which do not require any empirical calibration, the estimator can also be applied to HLL and generalized HyperLogLog (GHLL) sketches over the entire cardinality range. Since its derivation is much simpler than that in the original paper based on complex analysis [30], this work also contributes to a better understanding of the HLL algorithm.

The presented joint estimation method can be also specialized and used for other data structures. In particular, we derived a new closed-form estimator for MinHash that dominates the state-of-the-art estimator based on the number of identical components. Furthermore, our approach is able to improve joint estimation from HLL, GHLL, and HyperMinHash sketches except for very small sets compared to the corresponding state-of-the-art estimators. Given the popularity of all those sketches, specifically of MinHash and HLL, these side results could therefore have a major impact in practice, as more accurate estimates can be made from existing and persisted data structures. We also found that a performance optimization actually developed for SetSketch can be applied to HLL to speed up recording of large sets.

The presented methods have been empirically verified by extensive simulations. For the sake of reproducibility the corresponding source code including a reference implementation of SetSketch is available at <https://github.com/dynatrace-research/set-sketch-paper>. An extended version of this paper with additional appendices containing mathematical proofs and more experimental results is also available [28].

1.2 MinHash

MinHash [6] maps a set S to an m -dimensional vector (K_1, \dots, K_m) using

$$K_i := \min_{d \in S} h_i(d).$$

Here h_i are independent hash functions. The probability, that components K_{U_i} and K_{V_i} of two different MinHash sketches for sets U and V match, equals the Jaccard similarity J

$$P(K_{U_i} = K_{V_i}) = \frac{|U \cap V|}{|U \cup V|} = J.$$

This locality sensitivity makes MinHash very suitable for set comparisons and similarity search, because the Jaccard similarity can be directly and quickly estimated from the fraction of equal components with a root-mean-square error (RMSE) of $\sqrt{J(1-J)/m}$. MinHash also allows the estimation of cardinalities [12, 13] and other joint quantities [15, 19].

Meanwhile, many improvements and variants have been published that either improve the recording speed or the memory efficiency. One permutation hashing [40] reduces the costs for adding a new element from $O(m)$ to $O(1)$. However, there is a high probability of uninitialized components for small sets leading to large estimation errors. This can be remedied by applying a finalization step called densification [44, 63, 64] which may be expensive for small sets [27] and also prevents further aggregations. Alternatively, fast similarity sketching [17], SuperMinHash [25], or weighted minwise hashing algorithms like BagMinHash [26] and ProbMinHash [27] specialized to unweighted sets can be used instead. Compared to one permutation hashing with densification, they are mergeable, allow further element insertions, and even give more accurate Jaccard similarity estimates for small set sizes.

To shrink the memory footprint, b-bit minwise hashing [39] can be used to reduce all MinHash components from typically 32 or 64 bits to only a few bits in a finalization step. Although this loss of information must be compensated by increasing the number of components m , the memory efficiency can be significantly improved if precise estimates are needed only for high similarities. However, the need of more components increases the computation time and the sketch cannot be further aggregated or merged after finalization.

Besides the original application of finding similar websites [33], MinHash is nowadays widely used for nearest neighbor search [35], association-rule mining [14], machine learning [41], metagenomics [3, 22, 46, 51], molecular fingerprinting [57], graph embeddings [7], or malware detection [50, 60].

1.3 HyperLogLog

The HyperLogLog (HLL) data structure consists of m integer-valued registers K_1, K_2, \dots, K_m similar to MinHash. While MinHash typically uses at least 32 bits per component, HLL only needs 5-bit or 6-bit registers to count up to billions of distinct elements [30, 34]. The state for a given set S is defined by

$$K_i := \max_{d \in S} \lfloor 1 - \log_b h_i(d) \rfloor \quad \text{with } h_i(d) \sim \text{Uniform}(0, 1) \quad (1)$$

where h_i are m independent hash functions. $K_i = 0$ is used for initialization and the representation of empty sets. The original HLL uses the base $b = 2$, which makes the logarithm evaluation very cheap. We refer to the generalized HyperLogLog (GHLL) when using any $b > 1$ [12, 53]. Definition (1) leads to a recording time of

$O(m)$. Therefore, to have a constant time complexity, HLL implementations usually use stochastic averaging [30] which distributes the incoming data stream over all m registers

$$K_i := \max_{d \in S: h_1(d)=i} \lfloor 1 - \log_b h_2(d) \rfloor,$$

where h_1 and h_2 are independent uniform hash functions mapping to $\{1, 2, \dots, m\}$ and the interval $(0, 1)$, respectively. Instead of using two hash functions, implementations typically calculate a single hash value that is split into two parts. The original cardinality estimator [30] was inaccurate for small and large cardinalities. Therefore, a series of improvements have been proposed [34, 59, 70], which finally ended in an estimator that does not require empirical calibration [23, 24] and that is already successfully used by the Redis in-memory data store.

HLL is not very suitable for joint estimation or similarity search. The reason is that there is no simple estimator for the Jaccard similarity and no closed-form expression for the collision probability like for MinHash. The inclusion-exclusion principle gives significantly worse estimates for joint quantities than MinHash using the same memory footprint [18]. A maximum likelihood (ML) based method [23, 24] for joint estimation was proposed, that performs significantly better, but requires solving a three-dimensional optimization problem, which does not easily translate into a fast and robust algorithm. A computationally less expensive method was proposed in [49] for estimating inclusion coefficients. However, it relies on precomputed lookup tables and does not properly account for stochastic averaging, which can lead to large errors for small sets.

These joint estimation approaches have proven that HLL also encodes some joint information. This might be one reason why HLL's asymptotic space efficiency, when considering cardinality estimation only, is not optimal [53, 54]. Lossless compression [21, 36, 62] is able to improve the memory efficiency at the expense of recording speed and is therefore rarely used in practice. Approaches using lossy compression [69] are not recommended as their insert operations are not idempotent and commutative.

Nevertheless, due to its simplicity, HLL has become the state-of-the-art cardinality estimation algorithm, especially if the data is distributed. This is evidenced by the many databases like Redis, Oracle, Snowflake, Microsoft SQL Server, Google BigQuery, Vertica, Elasticsearch, Aerospike, or Amazon Redshift that use HLL under the hood to realize approximate distinct count queries. HLL is also used for many applications like metagenomics [1, 22, 46], graph analysis [4, 55, 56], query optimization [31], or fraud detection [10].

1.4 HyperMinHash

HyperMinHash [72] is the first approach to combine some properties of HLL and MinHash in one data structure. It can be seen as a generalization of HLL and one permutation hashing. HyperMinHash supports cardinality and joint estimation by using larger registers and hence more space than HLL. Unfortunately, the theoretically derived Jaccard similarity estimator is relatively complex and too expensive for practical use. Therefore, an approximation based on empirical observations was proposed. However, the original estimation approach is not optimal, as our results presented later have shown that even the naive inclusion-exclusion principle works better in some scenarios.

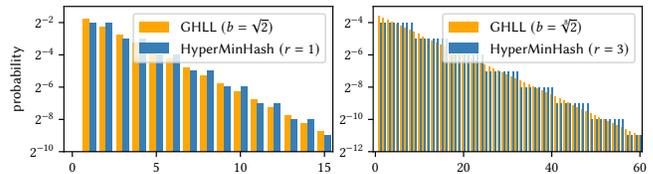


Figure 1: Probabilities of register update values for GHLL and HyperMinHash with equivalent configurations.

HyperMinHash with parameter r is very similar to GHLL with base $b = 2^{2^{-r}}$. The reason is that HyperMinHash approximates the probability distribution of GHLL with probabilities that are just powers of $\frac{1}{2}$ as shown in Figure 1. This has the advantage that hash values can be mapped to corresponding update values using only a few CPU instructions. Like those of HLL and GHLL, the register values of HyperMinHash are not locality-sensitive.

1.5 Further Related Work

A simple, though obviously not very memory-efficient, way to combine the properties of MinHash and HLL is to use them both in parallel [9, 52]. Probably the best alternative to MinHash and HLL which also works for distributed data and which even supports binary set operations is the ThetaSketch [18]. The downsides are a significantly worse memory efficiency compared to HLL in terms of cardinality estimation and that it is not locality-sensitive like MinHash.

2 METHODOLOGY

The basics of SetSketch follow directly from the properties introduced in Section 1. Similar to MinHash and HLL we consider a mapping from a set S to m integer-valued registers K_1, K_2, \dots, K_m . Furthermore, we assume that register values are obtained through some hashing procedure and that they are identically distributed. Since we want to estimate the cardinality from the register values, the distribution of K_i should only depend on the cardinality $n = |S|$

$$P(K_i \leq k) = F(k; n).$$

Mergeability requires that the register value $K_{U \cup V, i}$ of the union of two sets U and V can be computed directly from the corresponding individual register values $K_{U, i}$ and $K_{V, i}$ using some binary operation $K_{U \cup V, i} = \varphi(K_{U, i}, K_{V, i})$. To enable cardinality estimation, register values need to have some monotonic dependence on the cardinality. Without loss of generality, we consider non-decreasing monotonicity. Therefore, and because $|U \cup V| \geq |U|$ and $|V| \geq |W| \Rightarrow |U \cup V| \geq |U \cup W|$, we want φ to satisfy $\varphi(x, y) \geq x$ and $y \geq z \Rightarrow \varphi(x, y) \geq \varphi(x, z)$, respectively. The only binary operation with these properties and that is also idempotent and commutative, is the maximum function [28]. Therefore, we require $K_{U \cup V, i} = \max(K_{U, i}, K_{V, i})$.

For two disjoint sets U and V with cardinalities $|U| = n_U$ and $|V| = n_V$ we have $|U \cup V| = n_U + n_V$. In this case $K_{U, i}$ and $K_{V, i}$ are independent and therefore $P(K_{U \cup V, i} \leq k) = P(\max(K_{U, i}, K_{V, i}) \leq k) = P(K_{U, i} \leq k) \cdot P(K_{V, i} \leq k)$ which results in a functional equation

$$F(k; n_U + n_V) = F(k; n_U) \cdot F(k; n_V). \quad (2)$$

Algorithm 1: SetSketch

```
Data:  $S$   
Result:  $K_1, K_2, \dots, K_m$   
 $(K_1, K_2, \dots, K_m) \leftarrow (0, 0, \dots, 0)$   
 $K_{\text{low}} \leftarrow 0$   
 $w \leftarrow 0$   
forall  $d \in S$  do  
  initialize pseudorandom number generator with seed  $d$   
  for  $j \leftarrow 1$  to  $m$  do  
     $x_j \leftarrow$  generate  $j$ -th smallest out of  $m$  exponentially distributed random  
    values with rate  $a$  using (7) for SetSketch1 or (8) for SetSketch2  
    if  $x_j > b^{-K_{\text{low}}}$  then break  
     $k \leftarrow \max(0, \min(q + 1, \lfloor 1 - \log_b x_j \rfloor))$   
    if  $k \leq K_{\text{low}}$  then break  
     $i \leftarrow$  sample from  $\{1, 2, \dots, m\}$  without replacement  
    if  $k > K_i$  then  
       $K_i \leftarrow k$   
       $w \leftarrow w + 1$   
      if  $w \geq m$  then  
         $K_{\text{low}} \leftarrow \min(K_1, K_2, \dots, K_m)$   
         $w \leftarrow 0$ 
```

To allow estimation with constant relative error over a wide range of cardinalities, the location of the distribution should increase logarithmically with the cardinality n , while its shape should remain essentially the same. For a discrete distribution this can be enforced by the functional equation

$$F(k; n) = F(k + 1; nb). \quad (3)$$

Multiplying the cardinality with the constant $b > 1$, which corresponds to an increment on the logarithmic scale, shifts the distribution to the right by 1.

The system of functional equations composed of (2) and (3) has the solution

$$P(K_i \leq k) = F(k; n) = e^{-nab^{-k}} \quad (4)$$

with some constant $a > 0$ [28]. For a set with a single element, in particular, this is

$$P(K_i \leq k \mid n = 1) = F(k; 1) = e^{-ab^{-k}}. \quad (5)$$

Therefore, K_i needs to be distributed as $K_i \sim \lfloor 1 - \log_b X \rfloor$ where $X \sim \text{Exp}(a)$ is exponentially distributed with rate a . This directly leads to the definition of our new SetSketch data structure, which sets the state for a given set S as

$$K_i := \max_{d \in S} \lfloor 1 - \log_b h_i(d) \rfloor \quad \text{with } h_i(d) \sim \text{Exp}(a), \quad (6)$$

where $h_i(d)$ are hash functions with exponentially distributed output. This definition is very similar to that of HLL without stochastic averaging (1) where the hash values are distributed uniformly instead of exponentially.

2.1 Ordered Register Value Updates

Updating SetSketch registers based on (6) is not very efficient, because processing a single element requires m hash function evaluations and m is typically in the hundreds or thousands. Therefore, Algorithm 1 uses an alternative method based on ideas from our previous work [25–27] to reduce the average time complexity for adding an element from $\mathcal{O}(m)$ to $\mathcal{O}(1)$ for sets significantly larger than m .

Since register values are increasing with cardinality, only the smallest hash values $h_i(d)$ will be relevant for the final state according to (6). In particular, if $K_{\text{low}} \leq \min(K_1, \dots, K_m)$ denotes some lower bound of the current state, only hash values $h_i(d) \leq b^{-K_{\text{low}}}$ will be able to alter any register. As more elements are added to the SetSketch, the register values increase and so does their common minimum, allowing K_{low} to be set to higher and higher values. For large sets, $b^{-K_{\text{low}}}$ will eventually become so small that almost all hash values of further elements will be greater. Therefore, computing the hash values in ascending order would allow to stop the processing of an element after a hash value is greater than $b^{-K_{\text{low}}}$, and thus to achieve an asymptotic time complexity of $\mathcal{O}(1)$.

For that, we consider all hash values $h_i(d)$ as random values generated by a pseudorandom number generator that was seeded with element d . This perspective allows us to use any other random process that assigns exponentially distributed random values to each $h_i(d)$. In particular, we can use an appropriate ascending random sequence $0 < x_1 < x_2 < \dots < x_m$ whose values are randomly shuffled and assigned to $h_1(d), h_2(d), \dots, h_m(d)$. Shuffling corresponds to sampling without replacement and can be efficiently realized as described in [26, 27] based on the Fisher-Yates algorithm [29]. The random values x_j must be chosen such that $h_i(d)$ are eventually exponentially distributed as required by (6).

One possibility to achieve that is to use exponentially distributed spacings [20]

$$x_j \sim x_{j-1} + \frac{1}{m+1-j} \text{Exp}(a) \quad \text{with } x_0 = 0. \quad (7)$$

In this way the final hash values $h_i(d)$ will be statistically independent and the state of SetSketch will look like as if it was generated using m independent hash functions.

Alternatively, the domain of the exponential distribution with rate a can be divided into m intervals $[\gamma_{j-1}, \gamma_j)$ with $\gamma_0 = 0$ and $\gamma_m = \infty$. Setting $\gamma_j := \frac{1}{a} \log(1 + j/(m-j))$ ensures that an exponentially distributed random value $X \sim \text{Exp}(a)$ has equal probability to fall into any of these m intervals $P(X \in [\gamma_{j-1}, \gamma_j)) = \frac{1}{m}$ [28]. Hence, if the points are sampled in ascending order according to

$$x_j \sim \text{Exp}(a; \gamma_{j-1}, \gamma_j), \quad (8)$$

where $\text{Exp}(a; \gamma_{j-1}, \gamma_j)$ denotes the truncated exponential distribution with rate a and domain $[\gamma_{j-1}, \gamma_j)$, the shuffled assignment will lead to hash values $h_i(d)$ that are exponentially distributed with rate a . However, in contrast to the first approach, the hash values $h_i(d)$ will be statistically dependent, because there is always exactly one point sampled from each interval. This correlation will be less significant for large sets $n \gg m$, where it is unlikely that one element is responsible for the values of different registers at the same time.

Dependent on which method is used in conjunction with (6) to calculate the register values, we refer to SetSketch1 for the uncorrelated approach using exponential spacings and to SetSketch2 for the correlated approach using sampling from disjoint intervals.

2.2 Lower Bound Tracking

Different strategies can be used to keep track of a lower bound K_{low} as needed for an asymptotic constant-time insert operation. Since maintenance of the minimum of all register values with a small worst case complexity would require additional space by using

either a histogram [23] or binary trees [26, 27], we decided to simply update the lower bound regularly by scanning the whole register array as suggested in [61] which takes $O(m)$ time. However, since this approach is inefficient for small bases b , Algorithm 1 counts the number of register modifications w instead of the number of register values greater than the minimum. After every m register updates when $w \geq m$, K_{low} is updated and set to the current minimum of all register values, and w is reset. By definition, this contributes at most amortized constant time to every register increment and hence does not change the expected time complexity.

2.3 Parameter Configuration

SetSketch has 4 parameters, m , b , a , and q , that need to be set appropriately. If we have m registers, which are able to represent all values from $\{0, 1, 2, \dots, q, q+1\}$ with some nonnegative integer q , the memory footprint will be $m \lceil \log_2(q+2) \rceil$ bits without special encoding. m determines the accuracy of cardinality estimates, because, as shown later, the RMSE is in the range $[1/\sqrt{m}, 1.04/\sqrt{m}]$ for $b \leq 2$. While the choice of the base b has only little influence on cardinality estimation, joint estimation and locality sensitivity are significantly improved as $b \rightarrow 1$. The cardinality range, for which accurate estimation is possible, is controlled by the parameters a and q . Since the support of distribution (4) is unbounded, a and q need to be chosen such that register values smaller than 0 or greater than $q+1$ are very unlikely and can be ignored for the expected cardinality range. The lower bound of this range is typically 1 as we want SetSketches to be able to represent any set with at least one element.

If we choose $a \geq \log(m/\varepsilon)/b$ for some $\varepsilon \ll 1$, it is guaranteed that negative register values occur only with a maximum probability of ε [28]. Therefore, the error introduced by using zero as initial value as done in Algorithm 1 is negligible for small enough values of ε . In practice, setting $a = 20$ is a good choice in most cases. Even in the extreme case with $b \rightarrow 1$ and $m = 2^{20}$, the probability is still less than 0.22% to encounter at least one negative value. Similarly, setting $q \geq \lceil \log_b \frac{mn_{\max} a}{\varepsilon} \rceil$ guarantees that register values greater than $q+1$ occur only with a maximum probability of ε for all cardinalities up to n_{\max} [28]. A small b implies a larger value of q and therefore also a larger memory footprint, if the cardinality range is fixed.

To give a concrete example, consider a SetSketch with parameters $m = 4096$, $b = 1.001$, $a = 20$, and $q = 2^{16} - 2 = 65534$. The last parameter ensures that two bytes are sufficient to represent a single register and the whole data structure takes 8 kB. The probability that there is at least one register with negative value is 8.28×10^{-6} for a set with just a single element. Furthermore, the probability that any register value is greater than $q+1$ is 2.93×10^{-6} for $n = 10^{18}$. Therefore, a SetSketch using this configuration is suitable to represent any set with up to 10^{18} distinct elements. The expected error of cardinality estimates is approximately $1/\sqrt{m} \approx 1.56\%$.

3 ESTIMATION

In this section we present methods for cardinality and joint estimation. We will assume that register values are statistically independent which is only true for SetSketch1 and only a good approximation for SetSketch2 in the case of large sets with $n \gg m$. However,

our experiments presented later have shown that the estimators derived under this assumption also perform well for SetSketch2 over the entire cardinality range. The correlation even has a positive effect and reduces the estimation errors for small sets.

We introduce some special functions and corresponding approximations that are used for the derivation of the estimators. The functions $\xi_b^1(x)$ and $\xi_b^2(x)$ defined by

$$\xi_b^s(x) := \frac{\log b}{\Gamma(s)} \sum_{k=-\infty}^{\infty} b^{s(x-k)} e^{-b^{x-k}} \approx 1, \quad (9)$$

where Γ denotes the gamma function, are both very close to 1 for $b \leq 2$ [28]. For $b = 2$ we have $\max_x |\xi_b^1(x) - 1| \leq 10^{-5}$ and $\max_x |\xi_b^2(x) - 1| \leq 10^{-4}$, which shows that these approximations introduce only a small error. Smaller values of b lead to even smaller errors as both functions converge quickly towards 1 as $b \rightarrow 1$. Another approximation that we will use is

$$\zeta_b(x_1, x_2) := \sum_{k=-\infty}^{\infty} e^{-b^{x_1-k}} - e^{-b^{x_2-k}} \approx x_2 - x_1. \quad (10)$$

The relative error $|\frac{\zeta_b(x_1, x_2) - (x_2 - x_1)}{x_2 - x_1}|$ is smaller than 10^{-5} for $b = 2$ and approaches zero quickly as $b \rightarrow 1$ [28].

3.1 Cardinality Estimation

Instead of using the straightforward way of estimating the cardinality using the ML method based on (4), we derive a closed-form estimator, based on our previous work [23, 24], that is simpler to implement, cheaper to evaluate, and gives almost identical results. The expectation of $(ab^{-K_i})^s$ with $s \in \{1, 2\}$ and K_i distributed according to (4) is given by

$$\begin{aligned} \mathbb{E}((ab^{-K_i})^s) &= \sum_{k=-\infty}^{\infty} (ab^{-k})^s (e^{-nab^{-k}} - e^{-nab^{-k+1}}) \\ &= (1-b^{-s}) \sum_{k=-\infty}^{\infty} (ab^{-k})^s e^{-nab^{-k}} \\ &= \frac{1-b^{-s}}{n^s} \sum_{k=-\infty}^{\infty} b^{s(\log_b(na) - k)} e^{-b(\log_b(na) - k)} \\ &= \frac{(1-b^{-s}) \Gamma(s)}{n^s \log b} \xi_b^s(\log_b(na)) \approx \frac{(1-b^{-s}) \Gamma(s)}{n^s \log b}. \end{aligned} \quad (11)$$

Here the asymptotic identity (9) was used as approximation in the final step. We now consider the statistic $X_m = \frac{\log b}{1-1/b} \frac{1}{m} \sum_{i=1}^m ab^{-K_i}$. Using (11) for the cases $s = 1$ and $s = 2$ we get for its expectation and its variance [28]

$$\mathbb{E}(X_m) \approx \frac{1}{n} \quad \text{and} \quad \text{Var}(X_m) \approx \frac{1}{mn^2} \left(\frac{b+1}{b-1} \log(b) - 1 \right).$$

The delta method [8] gives for $m \rightarrow \infty$

$$\mathbb{E}(X_m^{-1}) \approx n \quad \text{and} \quad \text{Var}(X_m^{-1}) \approx \frac{n^2}{m} \left(\frac{b+1}{b-1} \log(b) - 1 \right)$$

which suggests to use X_m^{-1} as estimator for the cardinality n

$$\hat{n} = \frac{m(1-1/b)}{a \log(b) \sum_{i=1}^m b^{-K_i}}. \quad (12)$$

This estimator can be quickly evaluated, if the powers of b are precalculated and stored in a lookup table for all possible exponents which are known to be from $\{0, 1, \dots, q+1\}$.

The corresponding relative standard deviation (RSD) $\sqrt{\text{Var}(\hat{n})}/n$ is given by $\sqrt{\frac{1}{m} \left(\frac{b+1}{b-1} \log(b) - 1 \right)}$. It is minimized for $b \rightarrow 1$, where it equals $1/\sqrt{m}$, and increases slowly with b . For $b = 2$ the expected error is still as low as $1.04/\sqrt{m}$. However, as already discussed, smaller values of b require larger values of q and hence more space.

Optimal memory efficiency, measured as the product of the variance and the memory footprint, is typically obtained for values of b ranging from 2 to 4. Theoretically, if register values are compressed, a better memory efficiency can be achieved for $b \rightarrow \infty$ [53].

Values significantly greater than 2 invalidate the approximation (9) which was used for the derivation of estimator (12). To reduce the estimation error in this regime, random offsets, which correspond to register-specific values of parameter a , would be necessary [42, 53]. This work, however, focuses on $b \leq 2$. Although small values of b are inefficient for cardinality estimation, they contain, as we will show, more information about how sets relate to each other, which ultimately justifies slightly larger memory footprints.

3.2 Joint Estimation

The relationship of two sets U and V can be characterized by three quantities. Without loss of generality we choose the cardinalities $n_U = |U|$, $n_V = |V|$, and the Jaccard similarity $J = \frac{|U \cap V|}{|U \cup V|}$ with the natural constraint $J \in [0, \min(\frac{n_U}{n_V}, \frac{n_V}{n_U})]$ for parameterization. Other quantities such as

$$|U \cup V| = \frac{n_U + n_V}{1+J}, \quad (\text{union size})$$

$$|U \cap V| = \frac{(n_U + n_V)J}{1+J}, \quad (\text{intersection size})$$

$$|U \setminus V| = \frac{n_U - n_V J}{1+J}, \quad |V \setminus U| = \frac{n_V - n_U J}{1+J}, \quad (\text{difference sizes})$$

$$\frac{|U \cap V|}{\sqrt{|U||V|}} = \frac{(n_U + n_V)J}{\sqrt{n_U n_V (1+J)}}, \quad (\text{cosine similarity})$$

$$\frac{|U \cap V|}{|U|} = \frac{(n_U + n_V)J}{n_U (1+J)}, \quad \frac{|U \cap V|}{|V|} = \frac{(n_U + n_V)J}{n_V (1+J)}, \quad (\text{inclusion coefficients})$$

can be expressed in terms of only these three variables. If we have two SetSketches representing two different sets, we would like to find estimates for n_U , n_V , and J , which would allow us to estimate any other quantity of interest.

While we can use the cardinality estimator derived in the previous section to get estimates for n_U and n_V , respectively, the estimation of J is more challenging. One possibility is to leverage the mergeability of data sketches and use the inclusion-exclusion principle. Assume \hat{n}_U , \hat{n}_V , and $\hat{n}_{U \cup V}$ are cardinality estimates of $|U|$, $|V|$, and $|U \cup V|$, respectively. Then the inclusion-exclusion principle allows estimating the intersection size as $\hat{n}_U + \hat{n}_V - \hat{n}_{U \cup V}$. This can be used to estimate J as

$$\hat{J}_{\text{in-ex}} = \frac{\hat{n}_U + \hat{n}_V - \hat{n}_{U \cup V}}{\hat{n}_{U \cup V}}. \quad (13)$$

To ensure the natural constraint $J \in [0, \min(\frac{n_U}{n_V}, \frac{n_V}{n_U})]$ and the non-negativity of estimated intersection and difference sizes, it might be necessary to trim $\hat{J}_{\text{in-ex}}$ to the range $[0, \min(\hat{n}_U/\hat{n}_V, \hat{n}_V/\hat{n}_U)]$.

In the following a new joint estimation approach is derived that dominates the inclusion-exclusion principle as our experiments have shown. Using (4), the joint cumulative distribution function of registers K_{U_i} and K_{V_i} of two SetSketches representing sets U and V , respectively, is given by

$$\begin{aligned} & P(K_{U_i} \leq k_U \wedge K_{V_i} \leq k_V) \\ &= P(K_{U \setminus V, i} \leq k_U) P(K_{V \setminus U, i} \leq k_V) P(K_{U \cap V, i} \leq \min(k_U, k_V)) \\ &= e^{-\frac{n_U - n_V J}{1+J} a b^{-k_U}} e^{-\frac{n_V - n_U J}{1+J} a b^{-k_V}} e^{-\frac{(n_U + n_V)J}{1+J} a b^{-\min(k_U, k_V)}} \end{aligned}$$

$$= \begin{cases} e^{-a(b^{-k_U} \frac{n_U - n_V J}{1+J} + b^{-k_V} n_V)} & k_U \geq k_V, \\ e^{-a(b^{-k_V} \frac{n_V - n_U J}{1+J} + b^{-k_U} n_U)} & k_U \leq k_V. \end{cases}$$

It can be used to calculate the probability that $K_{U_i} > K_{V_i}$

$$\begin{aligned} & P(K_{U_i} > K_{V_i}) = \sum_{k=-\infty}^{\infty} P(K_{U_i} = k+1 \wedge K_{V_i} \leq k) \\ &= \sum_{k=-\infty}^{\infty} P(K_{U_i} \leq k+1 \wedge K_{V_i} \leq k) - P(K_{U_i} \leq k \wedge K_{V_i} \leq k) \\ &= \sum_{k=-\infty}^{\infty} e^{-ab^{-k}(\frac{n_U - n_V J}{b(1+J)} + n_V)} - e^{-ab^{-k} \frac{n_U + n_V}{1+J}} \\ &= \zeta_b \left(\log_b \left(a \left(\frac{n_U - n_V J}{b(1+J)} + n_V \right) \right), \log_b \left(a \frac{n_U + n_V}{1+J} \right) \right) \\ &\approx \log_b \left(a \frac{n_U + n_V}{1+J} \right) - \log_b \left(a \left(\frac{n_U - n_V J}{b(1+J)} + n_V \right) \right) = p_b \left(\frac{n_U - n_V J}{n_U + n_V} \right). \end{aligned}$$

Here we used the approximation (10) which is asymptotically equal as $b \rightarrow 1$ and introduces a negligible error if $b \leq 2$. p_b is defined as $p_b(x) := -\log_b(1 - x \frac{b-1}{b})$. Complemented by $P(K_{U_i} < K_{V_i})$, which is obtained analogously, and $P(K_{U_i} = K_{V_i}) = 1 - P(K_{U_i} > K_{V_i}) - P(K_{U_i} < K_{V_i})$ we have

$$\begin{aligned} & P(K_{U_i} > K_{V_i}) \approx p_b(u - vJ), \quad P(K_{U_i} < K_{V_i}) \approx p_b(v - uJ), \\ & P(K_{U_i} = K_{V_i}) \approx 1 - p_b(u - vJ) - p_b(v - uJ), \end{aligned} \quad (14)$$

where we introduced the relative cardinalities $u = \frac{n_U}{n_U + n_V}$ and $v = \frac{n_V}{n_U + n_V}$ with $u + v = 1$. The approximation of $P(K_{U_i} = K_{V_i})$ is also always nonnegative [28].

If the cardinalities n_U and n_V are known, the ML method can be used to estimate J . The corresponding log-likelihood function as function of J using the approximated probabilities is

$$\begin{aligned} \log \mathcal{L}(J) &= D_+ \log(p_b(u - vJ)) + D_- \log(p_b(v - uJ)) \\ &\quad + D_0 \log(1 - p_b(u - vJ) - p_b(v - uJ)), \end{aligned}$$

where $D_+ := |\{i : K_{U_i} > K_{V_i}\}|$, $D_- := |\{i : K_{U_i} < K_{V_i}\}|$, and $D_0 := |\{i : K_{U_i} = K_{V_i}\}|$ are the number of registers in the sketch of U that are greater than, less than, or equal to those in the sketch of V , respectively. Since this log-likelihood function is cheap to evaluate requiring only 5 logarithm evaluations and is strictly concave on the domain $J \in [0, \min(\frac{n_U}{n_V}, \frac{n_V}{n_U})] = [0, \min(\frac{u}{v}, \frac{v}{u})]$ at least for all $b \leq e \approx 2.718$ [28], the ML estimate for J can be quickly and robustly found using standard univariate optimization algorithms like Brent's method [5].

Since the ML estimator is asymptotically efficient, the RMSE of the ML estimate is expected to be equal to $I^{-1/2}(J)$ as $m \rightarrow \infty$. $I(J)$ denotes the Fisher information with respect to J for known cardinalities n_U and n_V , which can be derived as [28]

$$I(J) = \frac{m(b-1)^2}{b^2 \log^2(b)} \left(\frac{(ub^p p_b(u-vJ))^2}{p_b(u-vJ)} + \frac{(vb^p p_b(v-uJ))^2}{p_b(v-uJ)} + \frac{(ub^p p_b(u-vJ) + vb^p p_b(v-uJ))^2}{1 - p_b(u-vJ) - p_b(v-uJ)} \right).$$

The asymptotic RMSE can be compared with the Jaccard estimator of MinHash for which the RMSE is $\sqrt{J(1-J)/m}$. The corresponding ratio is shown in Figure 2 for the cases $n_U = n_V$ and $n_U = 0.5n_V$, and various values of b . For $n_U = n_V$ the curves approach 1 as $b \rightarrow 1$. If $b = 1.001$ the difference from 1 is already very small and hence the estimation error for J will be almost the same as for MinHash with the same number of registers m . Given that 2-byte registers are sufficient for $b = 1.001$ as discussed in Section 2.3, this significantly improves the space efficiency compared to MinHash which uses 4- or even 8-byte components. As shown in Figure 2 for

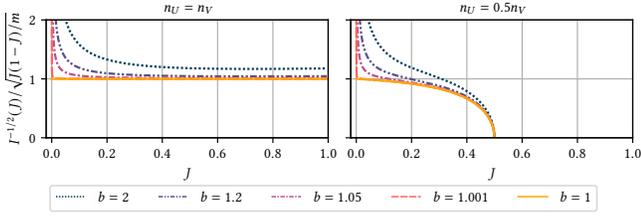


Figure 2: Asymptotic RMSE of the new estimation approach with known cardinalities n_U and n_V relative to the RMSE of MinHash $\sqrt{J(1-J)/m}$ with equal number of registers m .

$n_U = 0.5n_V$, the error can be even smaller than that of the MinHash Jaccard estimator. The reason is that our estimation approach incorporates n_U and n_V and also if register values are greater or smaller while the classic MinHash estimator only considers the number of equal registers.

If the true cardinalities n_U and n_V are not known, we simply replace them by corresponding estimates \hat{n}_U and \hat{n}_V using (12). As a consequence, the RMSE will apparently increase and $I^{-1/2}(J)$ will only represent an asymptotic lower bound as $m \rightarrow \infty$. Nevertheless, as supported by our experiments presented later, the estimation error is indistinguishable from the case with known cardinalities if the sets have equal size, thus if $n_U = n_V$. The reason is that D_+ and D_- are identically distributed in this case, which makes the log-likelihood function symmetric with respect to u and v . Since this also implies $u = v = \frac{1}{2}$ and estimates $\hat{u} = \hat{n}_U / (\hat{n}_U + \hat{n}_V)$ and $\hat{v} = \hat{n}_V / (\hat{n}_U + \hat{n}_V)$ for u and v satisfy $\hat{u} + \hat{v} = 1$ by definition, they can be written as $\hat{u} = \frac{1}{2} + \varepsilon$ and $\hat{v} = \frac{1}{2} - \varepsilon$, respectively, with some estimation error ε . Therefore and due to the symmetry of the log-likelihood function, the ML estimator is an even function of ε . The estimated Jaccard similarity will therefore differ only by an $O(\varepsilon^2)$ term from the estimate based on the true cardinalities n_U and n_V . Since $\varepsilon \ll 1$ for sufficiently large m , this second order term can be ignored in practice.

3.3 Locality Sensitivity

A collision probability, that is a monotonic function of some similarity measure, is the foundation of LSH [2, 35, 43, 73]. Rewriting of (14) and using $u + v = 1$ gives for the probability of a register to be equal in two different SetSketches

$$P(K_{Ui} = K_{Vi}) \approx \log_b(1 + J(b-1) + \frac{(b-1)^2}{b}(u-vJ)(v-uJ)).$$

Using $0 \leq (u-vJ)(v-uJ) \leq \frac{1}{4}(1-J)^2$ [28] leads to

$$\log_b(1 + J(b-1)) \leq P(K_{Ui} = K_{Vi}) \leq \log_b(1 + J(b-1) + (1-J)^2 \frac{(b-1)^2}{4b}).$$

These bounds are illustrated in Figure 3 for $b \in \{2, 1.2, 1.001\}$. They are very tight for large J . Both bounds approach $P(K_{Ui} = K_{Vi}) = J$ as $b \rightarrow 1$. Estimating $P(K_{Ui} = K_{Vi})$ by D_0/m and resolving for J results in corresponding lower and upper bound estimators

$$\hat{J}_{\text{low}} := \max\left(0, 2^b \frac{D_0/m+1}{b-1} - 1\right), \quad \hat{J}_{\text{up}} := \frac{b^{D_0/m} - 1}{b-1}. \quad (15)$$

The tight boundaries, especially for large J , make SetSketch an interesting alternative to MinHash for LSH. Like b-bit minwise hashing [39] and odd sketches [47], SetSketches are more memory

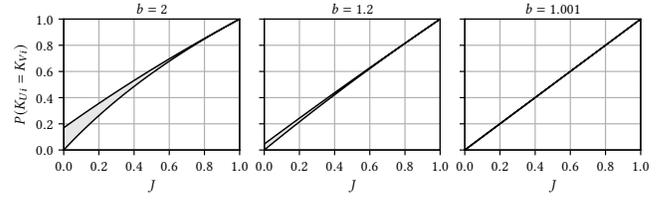


Figure 3: The range of possible collision probabilities of SetSketch registers as function of J .

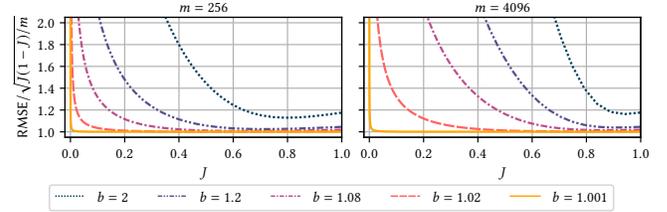


Figure 4: The RMSE of \hat{J}_{up} for the case $n_U = n_V$ relative to the RMSE of MinHash $\sqrt{J(1-J)/m}$.

efficient than MinHash, but unlike them, SetSketches can be further aggregated. The RMSE of MinHash is $\sqrt{J(1-J)/m}$. As long as the RMSE of SetSketch is not significantly greater than that, the additional error from using SetSketches instead of MinHash is negligible. For comparison, we consider the worst case scenario with $|U| = |V| \Leftrightarrow u = v = \frac{1}{2}$, which maximizes the collision probability, while using \hat{J}_{up} , which is based on the minimum possible collision probability, for estimation. Figure 4 shows the theoretical results for different values of b and m . Even for large values like $b = 2$ the RMSE is only increased by less than 20%, for all similarities greater than 0.7 or 0.9 for the cases $m = 256$ and $m = 4096$, respectively. The difference becomes smaller with decreasing b . The RMSE almost matches that of MinHash for $b = 1.001$, for which 2-byte registers suffice as mentioned earlier, again showing great potential for space savings compared to MinHash.

When searching for nearest neighbors with LSH, a set of candidates is determined first, which is then further filtered. For filtering, the presented more precise joint estimation approach can be used instead of (15) to reduce the false positive rate.

4 COMPARISON

This section compares MinHash and HLL with SetSketch, and in particular shows that they relate to SetSketches with $b = 1$ and $b = 2$, respectively.

4.1 MinHash

MinHash can be regarded as an extreme case of SetSketch1 with $b \rightarrow 1$ for which the statistic $e^{-ab^{-K}}$ approaches a continuous uniform distribution according to (5). Therefore, since the registers of SetSketch1 are also statistically independent, the transformation $K'_i = 1 - e^{-ab^{-K_i}}$ makes SetSketch1 equivalent to MinHash with values K'_i as $b \rightarrow 1$. Applying this transformation to (12) for $b \rightarrow 1$ and using $\lim_{b \rightarrow 1} (1 - 1/b)/\log(b) = 1$ indeed leads to the

cardinality estimator of MinHash [12, 13] with a RSD of $1/\sqrt{m}$

$$\hat{n} = \frac{m}{\sum_{i=1}^m -\log(1-K'_i)}. \quad (16)$$

The presented joint estimation method can also be applied to MinHash. However, as MinHash uses the minimum instead of the maximum for state updates we have to redefine D_+ and D_- as $D_+ := |\{i : K'_{U_i} < K'_{V_i}\}|$, $D_- := |\{i : K'_{U_i} > K'_{V_i}\}|$. For $b \rightarrow 1$, the ML estimate can be explicitly expressed as [28]

$$\hat{j} = \frac{u^2(D_0+D_-)+v^2(D_0+D_+)-\sqrt{(u^2(D_0+D_-)-v^2(D_0+D_+))^2+4D_-D_+u^2v^2}}{2muv} \quad (17)$$

and has an asymptotic RMSE of [28]

$$\text{RMSE}(\hat{j}) \stackrel{m \rightarrow \infty}{\approx} I^{-1/2}(J) = \sqrt{\frac{J(1-J)}{m}} \sqrt{1 - \frac{(u-v)^2 J}{uv(1-J)^2}} \leq \sqrt{\frac{J(1-J)}{m}}.$$

This shows that this estimator outperforms the state-of-the-art Jac-card estimator, which has a RMSE of $\sqrt{J(1-J)/m}$. Our experiments showed that this estimator also works better, if the cardinalities are unknown and need to be estimated using (17). Therefore, this approach is a much less expensive alternative to the ML approach described in [15], which considers the full likelihood as a function of all 3 parameters and requires solving a three-dimensional optimization problem.

The state of SetSketch2 is logically equivalent to that of SuperMinHash as $b \rightarrow 1$. SuperMinHash also uses correlation between components and is able to reduce the variance of the standard estimator for J by up to a factor of 2 for small sets [25].

4.2 HyperLogLog

The generalized HyperLogLog (GHLL) with stochastic averaging, which includes HLL as a special case with $b = 2$, is similar to SetSketch with $a = 1/m$. Under the Poisson model [23, 30], which assumes that the cardinality n is not fixed but Poisson distributed with mean λ , the register values will be distributed as $P(K_i \leq k) = e^{-\lambda m^{-1} b^{-k}}$ for $k \geq 0$ [28]. Comparison with (4) shows that the register values of GHLL are distributed as those of SetSketch for $a = 1/m$ and $n = \lambda$ provided that all register values are nonzero. Therefore, the cardinality estimator (12) can be used to estimate λ in this case. An unbiased estimator for the Poisson parameter λ is also an unbiased estimator for the true cardinality n [28]. Since (12) is asymptotically unbiased as $m \rightarrow \infty$, it can be used to estimate n . And indeed, (12) corresponds to the cardinality estimator with a RSD of $\sqrt{3 \log(2) - 1}/\sqrt{m} \approx 1.04/\sqrt{m}$ presented in [30] for HLL with base $b = 2$.

For small cardinalities, when many registers are zero, the value distribution will differ significantly from (4) and the estimator will fail. However, it can be fixed by applying the same trick as presented for the case $b = 2$ in [23]. If registers values are limited to the range $\{0, 1, \dots, q+1\}$, and $C_k := |\{i : K_i = k\}|$ is the histogram of register values, the corrected estimator can be written as [28]

$$\hat{n}_{\text{corr}} = \frac{m(1-1/b)}{a \log(b) (m\sigma_b(C_0/m) + (\sum_{k=1}^q C_k b^{-k}) + mb^{-q} \tau_b(1-C_{q+1}/m))}. \quad (18)$$

The functions σ_b and τ_b are defined as converging series

$$\begin{aligned} \sigma_b(x) &:= x + (b-1) \sum_{k=1}^{\infty} b^{k-1} x b^k, \\ \tau_b(x) &:= 1 - x + (b-1) \sum_{k=0}^{\infty} b^{-k-1} (x b^{-k} - 1). \end{aligned}$$

\hat{n}_{corr} also includes a correction for very large cardinalities, for which register update values greater than $q+1$, exceeding the value range of registers, are more likely to occur. The corrected estimator could also be applied to misconfigured SetSketches that do not allow to ignore the probability of registers with values less than 0 or greater than $q+1$ as discussed in Section 2.3. To save computation time, the values of $\sigma_b(C_0/m)$ and $\tau_b(C_{q+1}/m)$ can be tabulated for all possible values of $C_0, C_{q+1} \in \{0, 1, \dots, m\}$ for fixed m and b .

Due to the similarity of the register value distribution, the proposed joint estimation method also works well for GHLL, provided that all register values are from $\{1, 2, \dots, q\}$. Since the estimator relies only on the relative order, it can be even applied as long as there are no registers having value 0 or $q+1$ simultaneously in both GHLL sketches. Registers equal to $q+1$ can be easily avoided by choosing q sufficiently large. Registers, that have never been updated and thus are zero in both sketches, are expected for union cardinalities $|U \cup V|$ smaller than mH_m , where $H_m := \sum_{i=1}^m 1/i$ is the m -th harmonic number. This follows directly from the coupon collector's problem [48]. If the registers do not satisfy the prerequisites for applying the new estimation approach, the inclusion-exclusion principle (13) could still be used as fallback.

4.3 HyperMinHash

The similarity with GHLL as discussed in Section 1.4 allows using the proposed joint estimation approach for HyperMinHash as well. If the condition of not too small cardinalities is satisfied, the experiments presented below have shown that the new approach also outperforms the original estimator of HyperMinHash, especially when the cardinalities of the two sets are very different.

5 EXPERIMENTS

For the sake of reproducibility, we used synthetic data for our experiments to verify SetSketch and the proposed estimators. The wide and successful application of MinHash, HLL, and many other probabilistic data structures has proven that the output of high-quality hash functions [66] is indistinguishable from uniform random values in practice. This justifies to simply use sets consisting of random 64-bit integers for our experiments. In this way arbitrary many random sets of predefined cardinality can be easily generated, which is fundamental for the rigorous verification of the presented estimators. Real-world data sets usually do not include thousands of different sets with the same predefined cardinality.

5.1 Implementation

Both SetSketch variants as well as GHLL have been implemented in C++. The corresponding source code, including scripts to reproduce all the presented results, has been made available at <https://github.com/dynatrace-research/set-sketch-paper>. We used the Wyrand pseudorandom number generator [71] which is extremely fast, has a state of 64 bits, and passes a series of statistical quality tests [38]. Seeded with the element of a set, it is able to return a sequence of 64-bit pseudorandom integers. The random bits are used very economically. Only if all 64 bits are consumed, the next bunch of 64 bits will be generated. Sampling with replacement was realized as constant-time operation as described in [27] based on Fisher-Yates shuffling [29]. The algorithm proposed in [37] was used

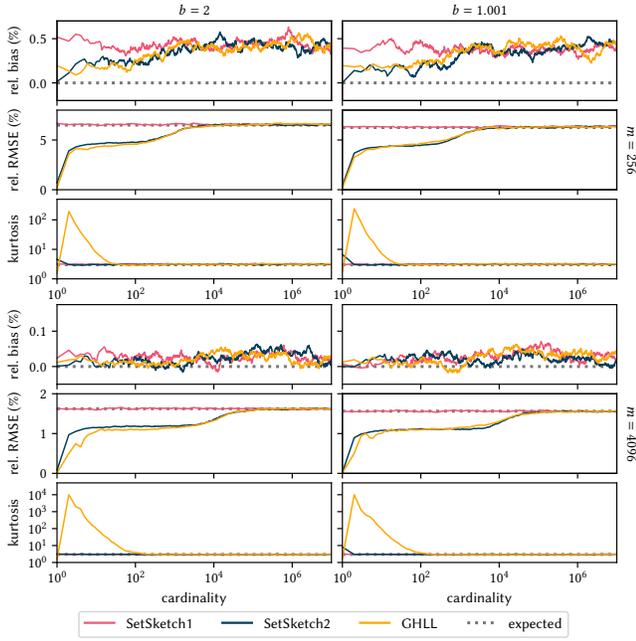


Figure 5: The relative bias, the relative RMSE, and the kurtosis of \hat{n} (12) for SetSketch1 and SetSketch2 and of \hat{n}_{corr} (18) for GHLL.

to sample random integers from intervals. The ziggurat method [45] as implemented in the Boost.Random C++ library [68] was used to obtain exponentially distributed random values. The algorithm described in [27] was applied to efficiently sample from a truncated exponential distribution as needed for SetSketch2. Our implementation precalculates and stores all relevant powers of b^{-1} in a sorted array, which is then used to find register update values as defined by (6) using binary search instead of expensive logarithm evaluations. This also allows to limit the search space to values greater than K_{low} , which further saves time with increasing cardinality.

5.2 Cardinality Estimation

Our cardinality estimation approach was tested by running 10 000 simulation cycles for different SetSketch and GHLL configurations. In each cycle we generated 10 million 64-bit random integer values and inserted them into the data structure. The use of 64 bits makes collisions very unlikely and allows regarding the number of inserted random values as the cardinality of the recorded set. During each simulation cycle the cardinality was estimated for a predefined set of true cardinality values which have been chosen to be roughly equally spaced on the log-scale.

We investigated SetSketch1, SetSketch2, and GHLL with stochastic averaging using $m = 256$ and $m = 4096$ registers. We considered different bases $b = 1.001$ and $b = 2$, for which we used 6-bit ($q = 62$) and 2-byte registers ($q = 2^{16} - 2 = 65534$), respectively. For both SetSketch variants, we used $a = 20$ and (12) for cardinality estimation. The corrected estimator (18) was used for GHLL.

Figure 5 shows the relative bias, the relative RMSE, and the kurtosis of the cardinality estimates over the true cardinality. The

bias is significantly smaller than the RMSE in all cases. Furthermore, when comparing $m = 256$ and $m = 4096$, the bias decreases more quickly than the RMSE as $m \rightarrow \infty$. Hence, the bias can be ignored in practice for reasonable values of m .

The independence of register values in SetSketch1 leads to a constant error over the entire cardinality range, that matches well with the theoretically derived RSD from Section 3.1. In accordance with the discussion in Section 3.1, the error is only marginally improved when decreasing b from 2 to 1.001. The error curves of SetSketch2 and GHLL are very similar and show a significant improvement for cardinalities smaller than m . This is attributed to the correlation of register values of SetSketch2 and the stochastic averaging of GHLL, respectively, both of which have a similar positive effect. However, when comparing the kurtosis, which is expected to be equal to 3 for normal-like distributions, we observed huge values for GHLL. This means that GHLL is very susceptible to outliers for small cardinalities. This is not surprising, as stochastic averaging only updates a single register per element. For a set with cardinality $n = 2$, for example, it happens with $1/m$ probability that both elements update the same register, which makes the state indistinguishable from that of a set with just a single element and obviously results in a large 50% error. We also tested the ML method for cardinality estimation and obtained visually almost identical results [28] which are therefore omitted in Figure 5 and which prove the efficiency of estimators (12) and (18).

5.3 Joint Estimation

To verify the presented joint estimation approach, we generated many pairs of random sets with predefined relationship. Each pair of sets (U, V) was constructed by generating three sets S_1, S_2 , and S_3 of 64-bit random numbers with fixed cardinalities n_1, n_2 , and n_3 , respectively, which are merged according to $U = S_1 \cup S_3$ and $V = S_2 \cup S_3$. The use of 64-bit random numbers allows ignoring collisions and all three sets can be considered to be distinct. By construction, we have $J = \frac{n_3}{n_1+n_2+n_3}$, $n_U = n_1+n_3$, and $n_V = n_2+n_3$, which guarantees both cardinalities, the Jaccard similarity, and hence also other joint quantities to be the same for all generated pairs (U, V) . After recording both sets using the data structure under consideration and applying the proposed joint estimation approach, we finally compared the estimates with the prescribed true values for various joint quantities.

Figure 6 shows the relative RMSE when estimating the Jaccard similarity, cosine similarity, inclusion coefficients, intersection size, and difference sizes using different approaches from SetSketch1 with $a = 20$ and $b \in \{1.001, 2\}$. The union cardinality was fixed $|U \cup V| = 10^6$ and the Jaccard similarity was selected from $J \in \{0.01, 0.1, 0.5\}$. Each data point shows the result after evaluating 1000 pairs of randomly generated sets. The charts were obtained by varying the ratio $|U \setminus V|/|V \setminus U|$ over $[10^{-3}, 10^3]$ while keeping J and $|U \cup V|$ fixed. The symmetry allowed us to perform the experiments only for ratios from $[1, 10^3]$.

Figure 6 clearly shows that the proposed joint estimator dominates the inclusion-exclusion principle for all investigated joint quantities. The difference is more significant for small set overlaps like for $J = 0.01$. Comparing the results for $b = 1.001$ and $b = 2$ shows that joint estimation can be significantly improved

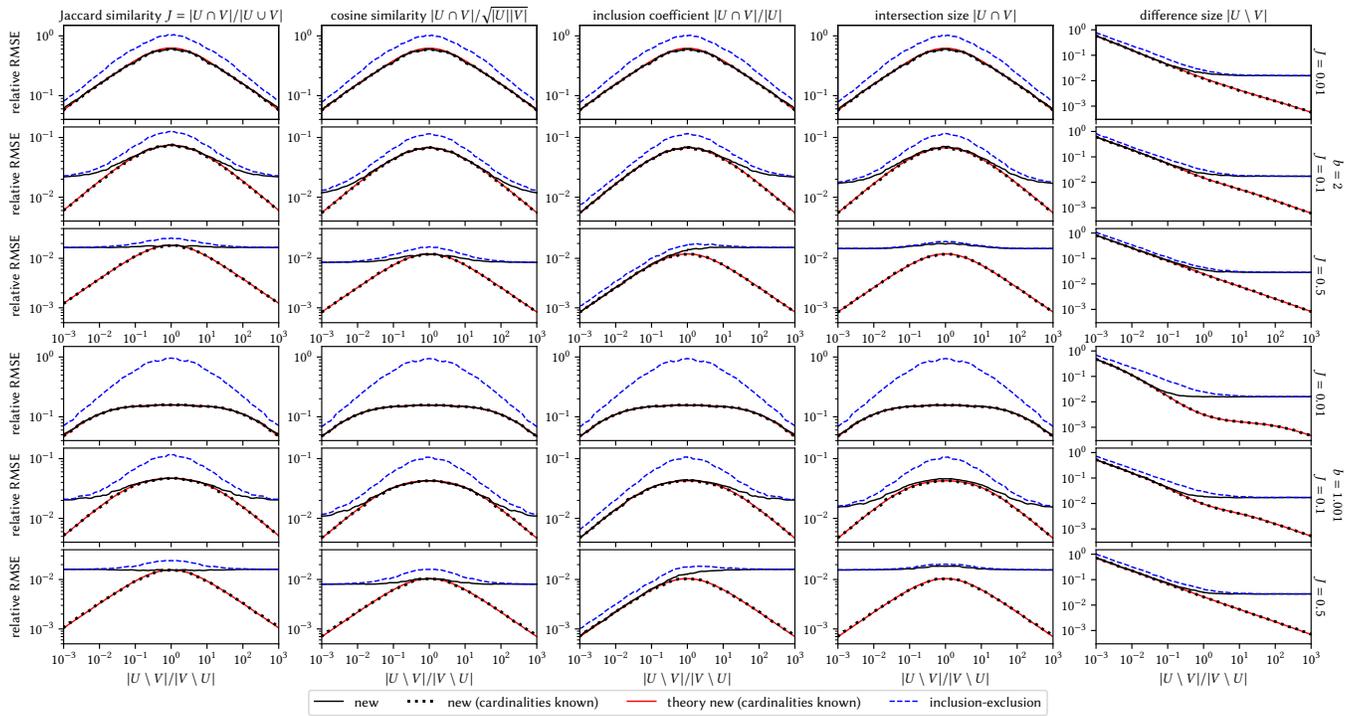


Figure 6: The relative RMSE of various estimated joint quantities when using SetSketch1 with $b \in \{1.001, 2\}$ and $m = 4096$ for sets with a fixed union cardinality of $|U \cup V| = 10^6$.

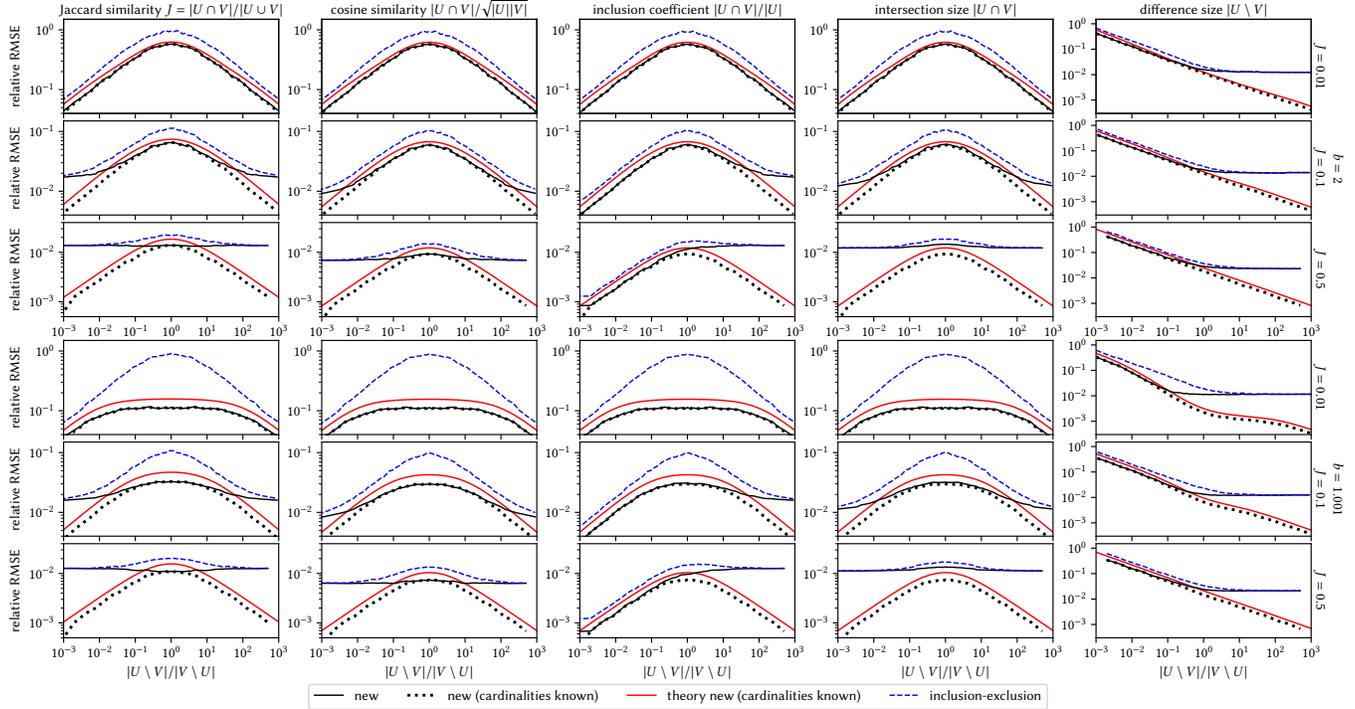


Figure 7: The relative RMSE of various estimated joint quantities when using SetSketch2 with $b \in \{1.001, 2\}$ and $m = 4096$ for sets with a fixed union cardinality of $|U \cup V| = 10^3$.

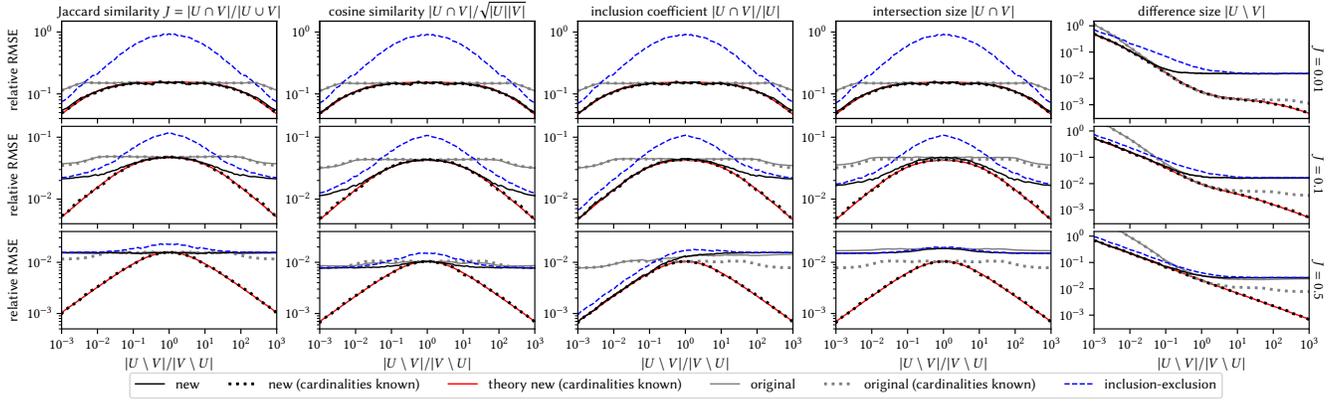


Figure 8: The relative RMSE of various estimated joint quantities when using MinHash with $m = 4096$ for sets with a fixed union cardinality of $|U \cup V| = 10^6$.

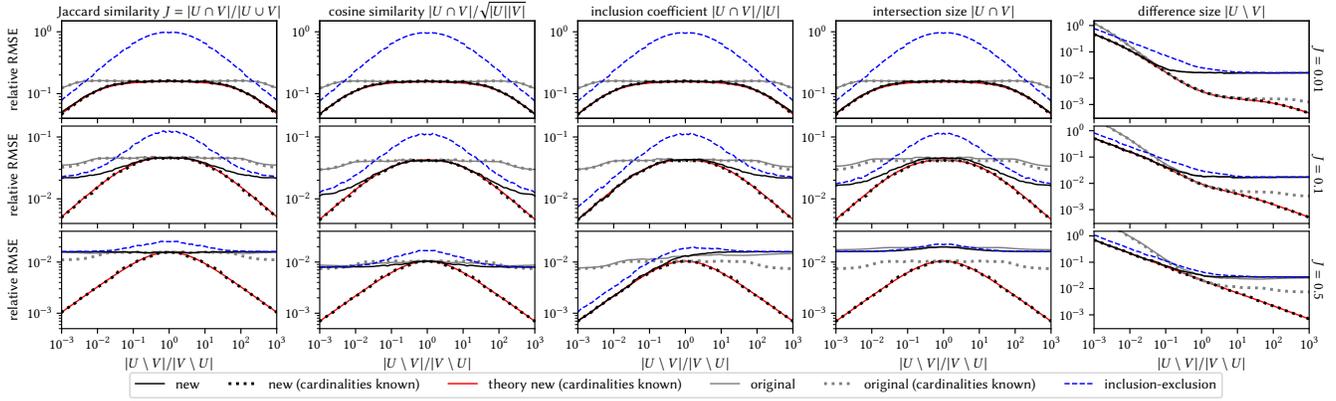


Figure 9: The relative RMSE of various estimated joint quantities when using HyperMinHash with $m = 4096$ and $r = 10$, which corresponds to $b = 2^{-2^{10}} \approx 1.000677$, for sets with a fixed union cardinality of $|U \cup V| = 10^6$.

when using smaller bases b . Knowing the true values of n_U and n_V further reduces the estimation error for which we observed perfect agreement with the theoretically derived RMSE. If n_U and n_V are known, any other joint quantity g can be expressed as a function of J . The corresponding ML estimate is therefore given by $\hat{g} = g(\hat{J})$. Variable transformation of the Fisher information given in Section 3.2, allows to calculate the corresponding asymptotic RMSE of g which is $I^{-1/2}(J)|g'(J)|$ as $m \rightarrow \infty$. As also predicted for the case $n_U = n_V$, the estimation error of J is the same regardless of whether the true cardinalities are known or not.

When running the same simulations for SetSketch2 and GHLL also with bases $b \in \{1.001, 2\}$, we got almost identical charts [28], which therefore have been omitted here. The reason why our estimation approach also worked for GHLL is that the union cardinality was fixed at 10^6 which is large enough to not have any registers that are zero in both sketches as discussed in Section 4.2. In particular, this shows that our approach can significantly improve joint estimation from HLL sketches with $b = 2$ for which the inclusion-exclusion principle is the state-of-the-art approach for joint estimation.

We repeated all simulations with a fixed union cardinality of $|U \cup V| = 10^3$. For SetSketch1, we obtained the same results as for $|U \cup V| = 10^6$ [28]. However, the errors for SetSketch2 shown in Figure 7 are significantly smaller and also lower than theoretically predicted. The reason for this improvement is, as before for cardinality estimation, the statistical dependence between register values in case of small sets. For $b = 1.001$, the error is reduced by a factor of up to $\sqrt{2}$ when estimating the Jaccard similarity using our new approach. This is expected as SetSketch2 corresponds to SuperMinHash [25] as $b \rightarrow 1$, for which the variance is known to be approximately 50% smaller than for MinHash, if $|U \cup V|$ is smaller than m . Our joint estimator failed for GHLL in this case [28], because $|U \cup V| = 10^3$ is significantly smaller than $H_m m$ with $m = 4096$, and hence, the condition for its applicability is not satisfied as discussed in Section 4.2.

We also applied the new approach to MinHash and used the explicit estimation formula (17). Figure 8 only shows the results for $|U \cup V| = 10^6$, because the results are very similar for $|U \cup V| = 10^3$, as expected [28]. The results are also almost indistinguishable from those obtained for SetSketch with $b = 1.001$ shown in Figure 6.

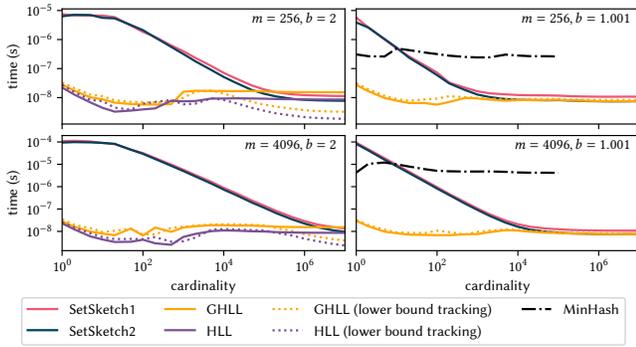


Figure 10: The average recording time per element as function of set cardinality.

Thus, SetSketch is able to give almost the same estimation accuracy using significantly less space as 2-byte registers are sufficient for $b = 1.001$. We also analyzed the state-of-the-art MinHash estimator based on the fraction of equal components. For the case that the cardinalities are not known, they have been estimated using (16). Our new estimator has a significantly better overall performance in both cases with known and unknown cardinalities, respectively. Only for inclusion coefficients and difference sizes the original method led to a slightly smaller error for $J = 0.5$ and $|U \setminus V|/|V \setminus U| > 1$. However, the new estimator clearly dominated, if sets have a small overlap. In contrast to the original estimator, it also dominates the inclusion-exclusion principle in all cases.

Finally, due to its similarity to GHLL for which our approach was able to improve joint estimation, we also considered HyperMinHash. Figure 9 shows the results for a HyperMinHash with parameter $r = 10$ which corresponds to a base of $b = 2^{-2^{10}} \approx 1.000677$ as discussed in Section 1.4. The original estimator of HyperMinHash led to similar results as the original estimator of MinHash (compare Figure 8). In contrast to the original estimator, which is based on empirically determined constants, our approach is solely based on theory and never performed worse than the inclusion-exclusion principle. Furthermore, since the estimation error was significantly reduced in many cases, our estimator seems to be superior to the original estimation approach. The results for the case that the cardinalities are known show perfect agreement with the theoretical RMSE originally derived for SetSketch1. Therefore, at least for large sets, HyperMinHash seems to encode joint information equally well as SetSketch with corresponding base. However, the big advantage of SetSketch is that the same estimator can be applied for any cardinalities, while estimation from GHLL or HyperMinHash sketches requires special treatment of small sets. The original HyperMinHash estimator delegates to a second estimator in this case.

5.4 Performance

The runtime behavior of a sketch is crucial for its practicality. Therefore, we measured the recording time for sets with cardinalities up to 10^7 . Instead of generating a set first, we simply generated 64-bit random numbers on the fly using the very fast Wyrand generator [71]. As hashing of more complex items is usually more expensive than generating random values, this experimental setup amplifies

the runtime differences compared to reality, where elements also need to be loaded from main memory. Furthermore, we excluded initialization costs, which are comparable for all considered data structures and which are a negligible factor for large cardinalities. For each cardinality considered, we performed 1000 simulation runs. In each run, we measured the time needed to generate the corresponding number of random elements and to insert them into the data structure. Afterwards, we calculated the average recording time per element. We measured the performance for SetSketch1, SetSketch2, GHLL with $b \in \{1.001, 2\}$, MinHash, and HLL and sketch sizes $m \in \{256, 4096\}$ on a Dell Precision 5530 notebook with an Intel Core i9-8950HK processor.

Figure 10 summarizes the results. The recording speed was roughly independent of the set size for HLL and GHLL due to stochastic averaging. We also implemented variants of both algorithms that use lower bound tracking as described in Section 2.2. If update values are not greater than the current lower bound, they will not be able to change any register. This can avoid many relatively costly random accesses to individual registers. This optimization, which is simpler than other approaches [23, 61], led to a significant performance improvement for HLL and GHLL with $b = 2$ at large cardinalities. However, this optimization did not speed up recording for GHLL with $b = 1.001$, because the calculation of register update values is the bottleneck here as it is more expensive for small b .

For MinHash the recording time was also independent of the cardinality, but many orders of magnitude slower due to its $O(m)$ insert operation, which was also the reason why we only simulated sets up to a size of 10^5 . The insert operations of both SetSketch variants have almost identical performance characteristics. As expected, insertions are quite slow for small sets. However, with increasing cardinality, the tracked lower bound K_{low} will also increase and finally leads to better and better recording speeds. For large sets, SetSketch is several orders of magnitude faster than MinHash and SetSketch2 even achieves the performance of the non-optimized (without lower bound tracking) versions of HLL and GHLL. We observed a quicker decay for $b = 1.001$ than for $b = 2$. The reason is that register values are updated more frequently for smaller b , which allows K_{low} to be raised earlier.

6 CONCLUSION

We have presented a new data structure for sets that combines the properties of MinHash and HLL and allows fine-tuning between estimation accuracy and memory efficiency. The presented estimators for cardinality and joint quantities do not require empirical calibration, give consistent errors over the full cardinality range without having to consider special cases such as small sets, and can be evaluated in a numerically robust fashion. The simple estimation from SetSketches, plus locality sensitivity as a bonus, compensate for the slower recording speed compared to HLL and GHLL for small sets. The developed joint estimator can also be straightforwardly applied to existing MinHash, HLL, GHLL, and HyperMinHash data structures to obtain more accurate results for joint quantities than with the corresponding state-of-the-art estimators in many cases. We expect that our estimation approach will also work for other set similarity measures or joint quantities that have not been covered by our experiments.

REFERENCES

- [1] D. N. Baker and B. Langmead. 2019. Dashing: fast and accurate genomic distances with HyperLogLog. *Genome Biology* 20, 265 (2019).
- [2] M. Bawa, T. Condie, and P. Ganesan. 2005. LSH Forest: self-tuning indexes for similarity search. In *Proceedings of the 14th International Conference on World Wide Web (WWW)*. 651–660.
- [3] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy. 2015. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotechnology* 33 (2015), 623–630.
- [4] P. Boldi, M. Rosa, and S. Vigna. 2011. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*. 625–634.
- [5] R. P. Brent. 1973. *Algorithms for minimization without derivatives*. Prentice-Hall.
- [6] A. Z. Broder. 1997. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences*. 21–29.
- [7] F. Bérés, D. M. Kelen, R. Pálovics, and A. A. Benczúr. 2019. Node embeddings in dynamic graphs. *Applied Network Science* 4, 64 (2019).
- [8] G. Casella and R. L. Berger. 2002. *Statistical Inference* (2nd ed.). Duxbury, Pacific Grove, CA.
- [9] R. Castro Fernandez, J. Min, D. Nava, and S. Madden. 2019. Lazo: A Cardinality-Based Method for Coupled Estimation of Jaccard Similarity and Containment. In *Proceedings of the 35th International Conference on Data Engineering (ICDE)*. 1190–1201.
- [10] Y. Chabchoub, R. Chiky, and B. Dogan. 2014. How can sliding HyperLogLog and EWMA detect port scan attacks in IP traffic? *EURASIP Journal on Information Security* 2014, 5 (2014).
- [11] A. Chen, J. Cao, L. Shepp, and T. Nguyen. 2011. Distinct Counting With a Self-Learning Bitmap. *J. Amer. Statist. Assoc.* 106, 495 (2011), 879–890.
- [12] P. Clifford and I. A. Cosma. 2012. A Statistical Analysis of Probabilistic Counting Algorithms. *Scandinavian Journal of Statistics* 39, 1 (2012), 1–14.
- [13] E. Cohen. 2015. All-Distances Sketches, Revisited: HIP Estimators for Massive Graphs Analysis. *IEEE Transactions on Knowledge and Data Engineering* 27, 9 (2015), 2320–2334.
- [14] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. 2001. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering* 13, 1 (2001), 64–78.
- [15] R. Cohen, L. Katzir, and A. Yehezkel. 2017. A Minimal Variance Estimator for the Cardinality of Big Data Set Intersection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 95–103.
- [16] Graham Cormode. 2017. Data sketching. *Communications of the ACM* 60, 9 (2017), 48–55.
- [17] S. Dahlgaard, M. B. T. Knudsen, and M. Thorup. 2017. Fast Similarity Sketching. In *Proceedings of the IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. 663–671.
- [18] A. Dasgupta, K. J. Lang, L. Rhodes, and J. Thaler. 2016. A Framework for Estimating Stream Expression Cardinalities. In *Proceedings of the 19th International Conference on Database Theory (ICDT)*. 6:1–6:17.
- [19] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapyuk. 2002. Mining database structure; or, how to build a data quality browser. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 240–251.
- [20] L. Devroye. 1986. Uniform and Exponential Spacings. In *Non-Uniform Random Variate Generation*. Springer, New York, NY, 206–245.
- [21] M. Durand. 2004. *Combinatoire analytique et algorithmique des ensembles de données*. Ph.D. Dissertation, École Polytechnique, Palaiseau, France.
- [22] R. A. L. Elworth, Q. Wang, P. K. Kota, C. J. Barberan, B. Coleman, A. Balaji, G. Gupta, R. G. Baraniuk, A. Shrivastava, and T. J. Treangen. 2020. To Petabytes and beyond: recent advances in probabilistic and signal processing algorithms and their application to metagenomics. *Nucleic Acids Research* 48, 10 (2020), 5217–5234.
- [23] O. Ertl. 2017. New cardinality estimation algorithms for HyperLogLog sketches. (2017). arXiv:1702.01284 [cs.DS]
- [24] O. Ertl. 2017. New Cardinality Estimation Methods for HyperLogLog Sketches. (2017). arXiv:1706.07290 [cs.DS]
- [25] O. Ertl. 2017. SuperMinHash - A New Minwise Hashing Algorithm for Jaccard Similarity Estimation. (2017). arXiv:1706.05698 [cs.DS]
- [26] O. Ertl. 2018. BagMinHash - Minwise Hashing Algorithm for Weighted Sets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1368–1377.
- [27] O. Ertl. 2020. ProbMinHash - A Class of Locality-Sensitive Hash Algorithms for the (Probability) Jaccard Similarity. *IEEE Transactions on Knowledge and Data Engineering* (2020). <https://doi.org/10.1109/TKDE.2020.3021176>
- [28] O. Ertl. 2021. SetSketch: Filling the Gap between MinHash and HyperLogLog (extended version). (2021). arXiv:2101.00314 [cs.DS]
- [29] R. A. Fisher and F. Yates. 1938. *Statistical Tables for Biological, Agricultural and Medical Research*. Oliver and Boyd Ltd., Edinburgh.
- [30] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. 2007. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the International Conference on the Analysis of Algorithms (AoA)*. 127–146.
- [31] M. J. Freitag and T. Neumann. 2019. Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates. In *Proceedings of the 9th Conference on Innovative Data Systems Research (CIDR)*.
- [32] A. Helmi, J. Lumbroso, C. Martínez, and A. Viola. 2012. Data Streams as Random Permutations: the Distinct Element Problem. In *Proceedings of the 23rd International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods for the Analysis of Algorithms (AofA)*.
- [33] M. Henzinger. 2006. Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 284–291.
- [34] S. Heule, M. Nunkesser, and A. Hall. 2013. HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT)*. 683–692.
- [35] P. Indyk and R. Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*. 604–613.
- [36] K. J. Lang. 2017. Back to the Future: an Even More Nearly Optimal Cardinality Estimation Algorithm. (2017). arXiv:1708.06839 [cs.DS]
- [37] D. Lemire. 2019. Fast Random Integer Generation in an Interval. *ACM Transactions on Modeling and Computer Simulation* 29, 1 (2019), 3:1–3:12.
- [38] D. Lemire. 2020. *TestingRNG: Testing Popular Random-Number Generators*. Retrieved July 18, 2021 from <https://github.com/lemire/testingRNG>
- [39] P. Li and C. König. 2010. B-Bit Minwise Hashing. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*. 671–680.
- [40] P. Li, A. Owen, and C.-H. Zhang. 2012. One Permutation Hashing. In *Proceedings of the 25th Conference on Neural Information Processing Systems (NIPS)*. 3113–3121.
- [41] P. Li, A. Shrivastava, J. Moore, and A. C. König. 2011. Hashing Algorithms for Large-Scale Learning. In *Proceedings of the 24th Conference on Neural Information Processing Systems (NIPS)*. 2672–2680.
- [42] A. Lukasiewicz and P. Uznański. 2020. Cardinality estimation using Gumbel distribution. (2020). arXiv:2008.07590 [cs.DS]
- [43] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. 2007. Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*. 950–961.
- [44] T. Mai, A. Rao, M. Kapilevich, R. A. Rossi, Y. Abbasi-Yadkori, and R. Sinha. 2019. On Denitification for Minwise Hashing. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence (UAI)*. 831–840.
- [45] G. Marsaglia and W. W. Tsang. 2000. The Ziggurat Method for Generating Random Variables. *Journal of Statistical Software* 5, 8 (2000).
- [46] G. Marçais, B. Solomon, R. Patro, and C. Kingsford. 2019. Sketching and Sublinear Data Structures in Genomics. *Annual Review of Biomedical Data Science* 2, 1 (2019), 93–118.
- [47] M. Mitzenmacher, R. Pagh, and N. Pham. 2014. Efficient Estimation for High Similarities Using Odd Sketches. In *Proceedings of the 23rd International Conference on World Wide Web (WWW)*. 109–118.
- [48] M. Mitzenmacher and E. Upfal. 2005. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press.
- [49] A. Nazi, B. Ding, V. Narasayya, and S. Chaudhuri. 2018. Efficient Estimation of Inclusion Coefficient Using Hyperloglog Sketches. In *Proceedings of the 44th International Conference on Very Large Data Bases (VLDB)*. 1097–1109.
- [50] N. Nissim, O. Lahav, A. Cohen, Y. Elovici, and L. Rokach. 2019. Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud. *Computers & Security* 87, 101590 (2019).
- [51] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy. 2016. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology* 17, 132 (2016).
- [52] A. Pascoe. 2013. *Hyperloglog and MinHash-A union for intersections*. Technical Report. AdRoll. Retrieved July 18, 2021 from <https://tech.nextroll.com/media/hllminhash.pdf>
- [53] S. Pettie and D. Wang. 2020. Information Theoretic Limits of Cardinality Estimation: Fisher Meets Shannon. (2020). arXiv:2007.08051 [cs.DS]
- [54] S. Pettie, D. Wang, and L. Yin. 2020. Simple and Efficient Cardinality Estimation in Data Streams. (2020). arXiv:2008.08739 [cs.DS]
- [55] B. W. Priest. 2020. DegreeSketch: Distributed Cardinality Sketches on Massive Graphs with Applications. *arXiv preprint arXiv:2004.04289* (2020).
- [56] B. W. Priest, R. Pearce, and G. Sanders. 2018. Estimating Edge-Local Triangle Count Heavy Hitters in Edge-Linear Time and Almost-Vertex-Linear Space. In *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC)*.
- [57] D. Probst and J.-L. Reymond. 2018. A probabilistic molecular fingerprint for big data settings. *Journal of Cheminformatics* 10, 66 (2018).
- [58] Y. Qi, P. Wang, Y. Zhang, Q. Zhai, C. Wang, G. Tian, J. C. S. Lui, and X. Guan. 2020. Streaming Algorithms for Estimating High Set Similarities in LogLog Space. *IEEE Transactions on Knowledge and Data Engineering* (2020). <https://doi.org/10.1109/TKDE.2020.2969423>

- [59] J. Qin, D. Kim, and Y. Tung. 2016. LogLog-Beta and More: A New Algorithm for Cardinality Estimation Based on LogLog Counting. (2016). arXiv:1612.02284 [cs.DS]
- [60] E. Raff and C. Nicholas. 2017. An Alternative to NCD for Large Sequences, Lempel-Ziv Jaccard Distance. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1007–1015.
- [61] P. Reviriego, V. Bruschi, S. Pontarelli, D. Ting, and G. Bianchi. 2020. Fast Updates for Line-Rate HyperLogLog-Based Cardinality Estimation. *IEEE Communications Letters* 24, 12 (2020), 2737–2741.
- [62] B. Scheuermann and M. Mauve. 2007. Near-optimal compression of probabilistic counting sketches for networking applications. In *Proceedings of the 4th ACM SIGACT-SIGOPS International Workshop on Foundation of Mobile Computing*.
- [63] A. Shrivastava. 2017. Optimal Densification for Fast and Accurate Minwise Hashing. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 3154–3163.
- [64] A. Shrivastava and P. Li. 2014. Improved Densification of One Permutation Hashing. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI)*. 732–741.
- [65] D. Ting. 2014. Streamed Approximate Counting of Distinct Elements: Beating Optimal Batch Methods. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 442–451.
- [66] R. Urban. 2020. *SMhasher: Hash function quality and speed tests*. Retrieved July 18, 2021 from <https://github.com/rurban/smhasher>
- [67] P. Wang, Y. Qi, Y. Zhang, Q. Zhai, C. Wang, J. C. S. Lui, and X. Guan. 2019. A Memory-Efficient Sketch Method for Estimating High Similarities in Streaming Sets. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 25–33.
- [68] S. Watanabe and J. Maurer. 2020. *Boost.Random C++ Library*. Retrieved July 18, 2021 from https://www.boost.org/doc/libs/1_75_0/doc/html/boost_random.html
- [69] Q. Xiao, S. Chen, Y. Zhou, and J. Luo. 2020. Estimating Cardinality for Arbitrarily Large Data Stream With Improved Memory Efficiency. *IEEE/ACM Transactions on Networking* 28, 2 (2020), 433–446.
- [70] Y. Zhao, S. Guo, and Y. Yang. 2016. Hermes: An Optimization of HyperLogLog Counting in real-time data processing. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. 1890–1895.
- [71] Wang Yi. 2021. *Wyhash*. Retrieved July 18, 2021 from <https://github.com/wangyifudan/wyhash/tree/399078afad15f65ec86836e1c3b0a103d178f15b>
- [72] Y. W. Yu and G. M. Weber. 2020. HyperMinHash: MinHash in LogLog space. *IEEE Transactions on Knowledge and Data Engineering* (2020). <https://doi.org/10.1109/TKDE.2020.2981311>
- [73] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. In *Proceedings of the 42nd International Conference on Very Large Data Bases (VLDB)*. 1185–1196.