KHyperLogLog: Estimating Reidentifiability and Joinability of Large Data at Scale

Pern Hui Chia*, Damien Desfontaines*†, Irippuge Milinda Perera*, Daniel Simmons-Marengo*,
Chao Li*, Wei-Yen Day*, Qiushi Wang*, Miguel Guevara*
*Google, †ETH Zürich
*{pernhc, milinda, dasm, chaoli, wday, qiushi, mgt}@google.com, †damien@desfontain.es

***Abstract*—Understanding the privacy relevant characteristics of data sets, such as reidentifiability and joinability, is crucial for data governance, yet can be difficult for large data sets. While computing the data characteristics by brute force is straightforward, the scale of systems and data collected by large organizations demands an efficient approach. We present KHyperLogLog (KHLL), an algorithm based on approximate counting techniques that can estimate the reidentifiability and joinability risks of very large databases using linear runtime and minimal memory. KHLL enables one to measure reidentifiability of data quantitatively, rather than based on expert judgement or manual reviews. Meanwhile, joinability analysis using KHLL helps ensure the separation of pseudonymous and identified data sets. We describe how organizations can use KHLL to improve protection of user privacy. The efficiency of KHLL allows one to schedule periodic analyses that detect any deviations from the expected risks over time as a regression test for privacy. We validate the performance and accuracy of KHLL through experiments using proprietary and publicly available data sets.**

I. Introduction

Understanding and monitoring the privacy state of production systems is a crucial element of privacy engineering. Data-flow analysis enables one to know which data processing workflows are reading which data and generating which outputs. Static and dynamic code analyses help to understand what binaries do at a more granular level e.g., whether they use pre-approved APIs, whether they read or write sensitive data types, or whether they use safe defaults. Potential violations of privacy policies can be surfaced at code review time before an engineer is allowed to run workflows in production systems.

While data flow analysis and code analysis are powerful tools, characterizing the data to assess their sensitivity (such as we propose in this paper) can often be useful or necessary. While human reviewers often have good intuitions and context about the sensitivity of data, there are obvious limitations: humans may make mistakes, and are limited in how much they can review. An automated analysis system can be accurate and scalable. Where humans would have to settle for evaluating a system or data set once, automated systems can be re-run. This provides regression testing for privacy characteristics and enables data custodians to be confident about the properties of their systems.

Automatic evaluation becomes challenging as data set size increases and data becomes increasingly heterogeneous. While brute force approaches will work for smaller data sets, their runtime and memory requirements become unworkable when run on petabytes of data.

We identify two characteristics of data sets that are often useful during privacy impact assessments: reidentifiability and joinability, and develop a new, scalable, automated approach to measuring them. While these terms may have different connotations, we define and use them consistently throughout the paper.

Reidentifiability is the potential that some supposedly anonymous or pseudonymous data sets could be deanonymized to recover the identities of users. A good practice is to keep reidentifiable data sets guarded with strict access control and access audit trails. As companies collect more information to build useful services it can be difficult to manually determine when a data set becomes reidentifying and requires more careful handling. The ability to estimate the reidentifiability of data sets automatically and efficiently reduces the work required of data custodians to manually label the different data sets.

Joinability measures whether data sets are linkable by unexpected join keys. Sometimes it is necessary to retain multiple data sets with different ID spaces. In those cases data custodians should avoid linking the two data sets to respect the choices of users who maintain separate identities. As an example, consider a website that can be used either signed-in or signed-out. A user may choose

to use the website signed-out to separate activities from their signed-in identity. If the website operator maintains data sets about activities of both signed-in and signed-out users, it might accidentally include granular information (e.g. web browser user agent) in both data sets that could allow the signed-in and signed-out identities to be linked. In that case, we would say that the identities in the two data sets are joinable.

There are various existing metrics for computing reidentifiability and joinability. For example, k-anonymity [1] and l-diversity [2] are two popular measures for reidentifiability. Meanwhile, data similarity are commonly measured using Jaccard index [3] or containment [4]. Two data sets are joinable if there exist a pair of data fields, similar in content, that can serve as a join key. Yet, managing reidentifiability and joinability risks at scale is more challenging than it appears. The naive approach requires memory proportional to the size of the data set, which becomes extremely difficult as data set sizes climb into the petabytes. Our experiments reveal how costly this naive approach is even for data sets in the order of gigabytes (see Section VIII). Linear runtime and sublinear memory are necessary for large-scale data analysis.

Contributions. In this paper we present the KHyperLogLog (KHLL) algorithm and demonstrate how it can be used to efficiently characterize both the reidentifiability and joinability of very large data sets. Adapted from the field of approximate counting, KHLL produces quantitative measures for reidentifiability and joinability using only a single pass over the data set and minimal memory. Both reidentifiability and joinability of data sets can be estimated using the compact data structures (colloquially known as "sketches") of KHLL rather than raw data. In addition, the approach is format-agnostic, allowing it to analyze any data set without modification. We have validated that KHLL is fast, parallelizable and accurate on both proprietary and public data sets.

This paper starts by describing the design goals and challenges in Section II, and how KHLL can be used improve protection of user privacy in organizations with large data sets in Section III. We provide some background on approximate counting in Section IV before presenting our KHLL algorithm in Section V. Next, we describe the use of KHLL for reidentifiability analysis in Section VI and joinability analysis in Section VII. We evaluate the performance and accuracy of KHLL empirically in Section VIII and Section IX. We present the related work in Section X before concluding in Section XI.

II. Quantifying Reidentifiability and Joinability at Scale

Our main goal is to design an efficient approach for quantifying the reidentifiability and joinability risks of large data sets. Specifically, it should help mitigate the risk of mistakes by engineers (e.g., adding additional fields to data sets without realizing they are overly unique or pose joinability risks) particularly as complex production systems evolve. We assume that everyone using KHLL wants to measure the privacy risks of their data sets, and so we don't defend against users attempting to get KHLL to under-report the risks of their data.

We introduce the metrics for reidentifiability and joinability that will be used in the rest of this paper. These metrics are defined on individual data fields and can be directly extended to any combinations of fields.

A. Reidentifiability by Uniqueness Distribution

Let $F = \{f_i\}$ be the set of all values of a field, and $ID = \{id_j\}$ be the set of all user IDs. Let $\{(f_i, id_j)\} \in F \times ID$ denote the pairs of F and ID values as found in a given data set D. Specifically, let $ID[f_i] = \{id_j : (f_i, id_j) \in D\}$ be the set of user IDs associated with a given field value f_i in D.

Definition 1. The uniqueness of a field value f_i with respect to ID is given by the number of unique IDs associated with f_i i.e., $|ID[f_i]|$.

Definition 2. The uniqueness distribution of F with respect to ID is estimated by the histogram of the uniqueness of individual values in F in data set D.

Different from k-anonymity [1] which computes the minimum number k of unique IDs associated with any values in F, we propose to keep the entire distribution of k so that it will be easy to compute the fraction of values in F with high reidentifiability risks, and thus the potential impact to the data when one would like to protect the data with k-anonymity or its variants.

In practice, the uniqueness distribution can be skewed. A few values in F might appear with high frequency while other values may pose high reidentifiability risks as they associate with only a small number of user IDs.

As an example, imagine a log that contains the User Agent (UA) of users who visit a site. UA is an HTTP header field which describes the application type, operating system, software vendor and software version of an HTTP request. To gauge the reidentifiability of UAs, one can estimate the uniqueness distribution by counting

the number of unique IDs that individual UA strings associate with. We expect a high percentage of raw UA strings to be associated with only one or a few user IDs and thus reidentifying [5].

B. Joinability by Containment

Let F_1 and F_2 represent the sets of values of two fields in data sets D_1 and D_2 respectively. Let $|F_1|$ denote the number of unique values in F_1, i.e., the cardinality of F_1, and $F_1 \cap F_2$ denote the set of values in both F_1 and F_2 (the intersection). We measure the joinability of D_1 and D_2 through F_1 and F_2 using the containment metric.

Definition 3. The containment of F_1 in F_2 is the ratio between the number of unique values of the intersection of F_1 and F_2, and the number of unique values in F_1 i.e., $|F_1 \cap F_2|/|F_1|$.

Note that containment is similar to the Jaccard Index but it is asymmetric. Unlike the Jaccard Index which computes the ratio between the number of unique values in the intersection of F_1 and F_2, and the union of F_1 and F_2, containment uses the number of unique values in either F_1 or F_2 as the denominator. This difference is important when F_1 and F_2 differ in size. Imagine one data set that contains a small subset of the users from a larger data set. The Jaccard Index will always be small and would not report joinability risk even when all values of F_1 are contained in F_2.

C. Scalability Requirements

While both uniqueness distribution and containment are easy to compute on small data sets, the computation will need to scale to handle very large data sets. In addition to hosting user content for digital services, organizations collect data for providing physical services (e.g., healthcare and location-based services), improving user experience and service availability, and anti-spam and fraud detection purposes. The scale of data can be huge, both in terms of the number of data fields and rows, and the number of databases.

It would be a Herculean task for human reviewers to manually oversee all product iterations and changes to data strategies. In a similar fashion, individual well-intentioned groups of engineers also find it hard to keep up with the increasingly large number of policy and regulatory requirements.

An ideal system for measuring reidentifiability and joinability that is scalable will need to use efficient and parallelizable algorithms. Also, as increasingly heterogenous data is collected and used it will need an approach agnostic to data types and formats to handle data sets generated by different engineering teams.

III. Applying KHLL to Protect User Privacy

Data custodians can use KHLL in a number of different ways to improve user privacy.

Quantitative Measurement: KHLL can be used to quantitatively measure the reidentifiability risk of a data set. This can inform data custodians about the sensitivity of data and its risks so they can plan a suitable and appropriate data strategy (e.g., anonymization, access controls, audit trails) at points in the life cycle of the data, including data collection, usage, sharing and retention (or deletion).

Exploring Data Strategies: The efficiency of KHLL provides data custodians a powerful analysis tool for exploring different data sanitization strategies. Many data analysis tasks (e.g., experimentation to improve service availability, anti-spam and fraud detection) can use a projection (view) of a high-dimensional data set that is protected with k-anonymity [1] (or related techniques such as l-diversity [2] or anatomy [6]). KHLL can be run on different possible combinations of fields in the data set at once to estimate how much data will be lost as a function of the technique. Together, the data custodian and data analysts can decide how to trade off the utility of the data projection and the reidentifiability risk (e.g., which fields to be included, suppressed, generalized or made disjoint in separate data sets, and whether the data needs stronger guarantees like differential privacy [7]).

Consider the Netflix prize data set, which contains the movie ids and ratings given by different users at different dates (year, month and day). Analyzing the data set using KHLL, we obtain results that mirror those of Narayanan and Shmatikov [8]. While no single field has high uniqueness (e.g., we observe that all movies included in the data set are rated by at least 50 users), the combination of movie ratings and dates are highly unique. An efficient analysis using the like of KHLL might have helped the Netflix team to measure the reidentifiability risks, explore alternatives for treating the data, or to potentially conclude that the risk was too high to share the data externally.

Regression Testing: In cases where data custodians regularly produce k-anonymous (or the like) data sets, KHLL can be further used as a regression test. KHLL analysis can be run on the output as part of the anonymization pipeline to expose any implementation bugs, or to alert on any unexpected changes to the characteristics of the input data.

Joinability Assessment: KHLL can also enable efficient joinability assessment to protect user privacy. If an organization collects data about users under multiple ID spaces in different contexts (e.g. signed in vs signed out), KHLL can be used to keep the IDs separate, respecting the choice of users to conduct certain activities in certain contexts. For example, KHLL analysis can be run on two data sets of different IDs, and be used to detect data fields in the two data sets that are similar (high containment in either direction) and that are highly unique. These data fields are potential join keys that can be used to trivially link the two ID spaces. To mitigate joinability risks, engineers can choose to suppress or generalize one of the fields, or use access controls to prevent someone from using the fields to join the two identifiers. The analysis can be run periodically and attached to an alerting system that notifies engineers if joinability exceeds pre-specified limits (e.g., to quickly detect whether any newly added fields increase joinability risks). Joinability assessment is highly intractable with pairwise comparisons of raw data, but KHLL enables joinability approximation based on its compact data structures (sketches).

Periodic KHLL-based joinability analyses have enabled us to uncover multiple logging mistakes that we were able to quickly resolve. One instance was the exact position of the volume slider on a media player, which was mistakenly stored as a 64-bit floating-point number. Such a high entropy value would potentially increase the joinability risk between signed-in and signed-out identifiers. We were able to mitigate the risk by greatly reducing the precision of the value we logged. In other cases, we have mitigated joinability risks by dropping certain fields entirely, or by ensuring that the access control lists of both data sets are disjoint.

Miscellaneous: If data custodians label their data sets with information about the semantics of certain fields, KHLL can be used to propagate labels through the system and find inconsistencies. If two fields have a high containment score (in either direction), they are likely to share the same semantics. If one of the fields is labelled but the other is not, then the label can be copied to the second field, and if the two fields have different labels then engineers can be alerted that one of the labels is likely incorrect. The scability of KHLL means that it can be used to propagate labels accross large data sets, and that the label correctness can be repeatedly checked by re-running analysis periodically.

Although not a primary purpose, an additional side effect of making a powerful analysis tool available to different roles in an organization is the increased awareness of anonymization and user privacy. Data custodians, engineers and analysts can discuss the analysis results with each other, gain a better understanding of reidentiability risks when working with user data, and understand why further anonymization may be necessary.

For all of these use cases one needs to keep in mind the estimation errors of KHLL (see Section VI-B and Section VII-B). It is possible that KHLL may underestimate reidentifiability or joinability risks (e.g., KHLL might miss values that are unique to a single user). In general, data custodians could use KHLL to estimate risks and impacts on data utility when exploring an appropriate data protection and anonymization strategy, but then use exact counting to execute the strategy. While the joinability analysis using KHLL might be sensitive to data formats and transformations, the efficiency of KHLL makes it the best regression test for data joinability that we are aware of.

IV. Approximate Counting Basics

Approximate counting is a technique to efficiently estimate the cardinality (number of distinct elements) of a set [9]–[11], typically using a small amount of memory. This technique can also be extended to compute other statistics such as quantiles [12], [13] and frequent values [14], [15]. Approximate counting algorithms use compact data structures, colloquially known as "sketches" that summarize certain observable properties of the elements in the analyzed data set.

In addition to being memory efficient, approximate counting sketches support additional operations such as merging (set union). Large-scale data sets are typically stored in multiple machines ("shards"), as the entire data set would not fit in a single machine. In such situations, one can compute the cardinality of the entire data set using a two-step approach:

1) Compute the sketches of individual data shards.
2) Merge all the sketches generated in step 1.

In this paper, we extend two approximate counting algorithms named K Minimum Values (KMV) [9] and HyperLogLog (HLL) [10] to build KHLL. We provide some intuition about KMV and HLL in the following.

A. K Minimum Values (KMV)

As implied by the name, KMV estimates the cardinality of a set by keeping the K smallest hash values of its elements. The intuition behind KMV is as follows. Suppose there exists a hash function that uniformly maps input values to its hash space. Note that this hash function does not need to be a cryptographic

hash function, and one-wayness is not required (i.e., it does not matter if the hash function can be reversed in polynomial time). If one computes the hash of each element in the analyzed data set, one can expect those hashes to be evenly distributed across the hash space. Then, one can estimate the cardinality of the analyzed data set by computing the density of the hashes (i.e., the average distance between any two consecutive hashes) and dividing the hash space by the density. Since storing all the hashes can incur a significant storage cost, one can store only the K smallest hash values and extrapolate the density of the entire hash space.

As a concrete example, say there is a hash function whose outputs are evenly distributed in the range $[1, 1000000]$. If $K = 100$, and the Kth smallest hash value is 1000, we can compute the density by simply dividing the Kth smallest hash value by K, i.e., density $= 1000/100 = 10$. Extrapolating to the range of $[1, 1000000]$, with the uniformity assumption but without bias correction, one can roughly estimate the number of unique values as $1000000/10 = 100000$.

Computing set union using KMV sketches is straightforward. Given two KMV sketches, S_1 and S_2, one can find the KMV sketch of the union of the two data sets by combining the two sketches and retaining only the K smallest hashes.

KMV sketches are efficient to produce. It requires a single pass over the data set, but only a space complexity of $O(K)$, as it consists of K unique hash values of fixed length. The cardinality estimated by a KMV sketch has a relative standard error of $\frac{1}{\sqrt{K}}$ with the assumption that the hash space is large enough to keep hash collisions to a minimum. As a concrete example, with $K = 1024$ and using a 64-bit uniformly distributed hashing function, one can estimate the cardinality with a relative standard error of 3% and KMV sketch size of 8 KB.

B. HyperLogLog (HLL)

Instead of keeping the K smallest hash values, HLL further reduces the space requirement by tracking the maximum number of trailing zeros of the hash values. The maximum number of trailing zeros increases as more unique values are added to HLL given the uniformity assumption of the hash function.

From the hash of an incoming value, HLL uses the first P bits to determine the bucket number, and uses the remaining bits to count the number of trailing zeros. HLL keeps track of the maximum number of trailing zeros at each of the $M = 2^P$ buckets. After processing all values in the analyzed data set, HLL estimates the cardinality of each bucket as 2^{m_i}, where m_i is the maximum number of trailing zeros seen in bucket i. Finally, HLL estimates the cardinality of the analyzed data set by combining the cardinalities of individual buckets by taking the harmonic mean.

HLL sketches are also efficient to compute (i.e., using a single pass over the analyzed data set) and provide cardinality estimates with a relative standard error of $\frac{1.04}{\sqrt{M}}$. Moreover, the space complexity of a HLL sketch is $O(M)$ since it consists of M counts of trailing zeros. As a concrete example, with M=1024 and using a 64-bit uniformly distributed hashing function, one can estimate the cardinality with a relative standard error of 3% and HLL sketch size of 768 B.

Heule et al. showed that HLL does not provide a good estimate for low cardinalities and proposed HLL++ [11] to accommodate such data sets. HLL++ maintains two different modes of sketches. When the cardinality is low, it remains in the sparse representation mode, which keeps almost the entire hash values. When the list of hash values kept grows, HLL++ switches to the conventional HLL mode which has a fixed memory footprint. The sparse representation allows HLL++ to use linear counting for estimating small cardinalities with negligible error while also keeping the sketch size small.

V. KHyperLogLog (KHLL)

While cardinality estimates are helpful, they are limited in many ways for reidentifiability and joinability analysis. While cardinality estimates can be used to estimate the average uniqueness when the total unique IDs in the data set is known, they do not estimate the uniqueness distribution. The average alone can be misleading as uniqueness distribution is more likely to be skewed in practice. The uniqueness distribution is also useful to inform about various data strategies, for example the feasibility of suppressing or generalizing a fraction of the unique values. The distribution could not be naively estimated as we could not assume the data sets to be structured in a way that every single row corresponds to a single user.

We present KHyperLogLog (KHLL) which builds on KMV and HLL to estimate uniqueness distribution and containment with a single pass over the data set and low memory requirements.

This algorithm uses a two-level data structure for analyzing tuples of field and ID values. KHLL contains K HLL sketches corresponding to K smallest hashes of field values. This is approximately equivalent to taking a uniform random sampling of size K over the field

values, each of which comes with a corresponding HLL sketch containing the hashes of IDs associated with the field value. We considered an alternative two-level KMV-based data structure, named K2MV, but concluded that KHLL is more memory-efficient and suitable for our needs. See Appendix XIII for a description of K2MV.

Consider a stream of pairs $(f_i, id_j) \in F \times ID$ and a hash function h. KHLL processes the incoming tuples as follows:

1) Calculate $h(f_i)$ and $h(id_j)$.
2) If $h(f_i)$ is present in the KHLL sketch, add $h(id_j)$ to the corresponding HLL sketch.
3) Else, if $h(f_i)$ is among the K smallest hashes:
   a) If there are more than K entries, purge the entry containing the largest hash of F.
   b) Add a new entry containing $h(f_i)$ and a HLL sketch with $h(id_j)$.
4) Else, do nothing.

As a specific example, consider a stream of User Agent (UA) and ID value pairs. Further consider an 8-bit hash function and a KHLL sketch of $K = 3$ and $M = 8$. The KHLL sketch contains at most 3 entries representing the 3 smallest values of h(UA) in the first level, each with a HLL sketch in the second level which has at most 8 counting buckets. For example, when the KHLL sketch processes the tuple (UA-4, ID-6) which hashes to `(00000011, 00011010)` as shown in Figure 1, the entry with with the largest h(UA) = `00011000` and its companion HLL sketch is purged to give way to h(UA-4) and a new HLL.

The memory signature of a KHLL sketch depends on the parameters K and M as well as the uniqueness distribution of the data. Per innovation in HLL++ [11], we design the HLL sketches in KHLL to start in the sparse representation mode which keeps a sparse list of the ID hash values. Once this representation exceeds the fixed size of a conventional HLL, it is converted to a normal representation with M counting buckets. Using a 64-bit hash function, individual counting buckets require less than a byte to count the maximum number of trailing zeros in the ID hash values. Improving over HLL++, we implemented HLL++ Half Byte which uses only half a byte for individual counting buckets (see Appendix XII).

Let $ID[f_i] = \{id_j : (f_i, id_j) \in D\}$ be the set of user IDs associated with a given field value f_i in data set D. The memory needed for a KHLL sketch considering both the sparse and conventional mode is thus $\min(8|ID[f_i]|, M)$ in bytes. Since the KMV approximates a K size uniform random sample over field values, the expected memory usage for the entire KHLL will be roughly K times the average HLL size i.e., $\frac{K}{|F|} \cdot \sum_i \min(8|ID[f_i]|, M)$.

This means that the memory usage of KHLL, while never above a strict upper bound, will be higher for data sets with low uniqueness in which most field values are associated with large user ID sets. Alternatively, when most field values correspond to only a few unique user IDs, the memory signature will be much smaller as the HLL sketches will be in sparse representations.

Note that KHLL does not dictate how the data is structured. To process a table of data, we simply read each row in the table to (1) extract the ID value and the values of fields (or combinations of fields) of interest, and (2) ingest the tuples of ID and field values into the corresponding KHLL sketches. This allows for tables that contain arbitrarily large number of fields, and even for tables where data about the same user can be repeated across multiple table rows.

VI. Estimating Reidentifiability using KHLL

From a KHLL sketch of (F, ID), one can estimate both the cardinality of the field and the number of unique IDs associated with individual field values i.e., the uniqueness distribution. The latter allows us to efficiently estimate the proportion of field values that are reidentifying as well as statistics such as the min, max, and median uniqueness.

A. Evaluating Data Loss and Reidentifiability Trade Off

One can plot the uniqueness distribution to visualize the percentage of field values or IDs meeting some k-anonymity thresholds. Figure 2 is an example histogram of how many field values are associated with each number of unique IDs. Tweaked slightly, Figure 3 plots the cumulative percentage of values not meeting varying k-anonymity thresholds. A relatively anonymous data set exhibits the curve on the left as most of the field values are expected to be associated with a large number of IDs. Conversely, a highly reidentifying data set will exhibit the curve on the right.

The cumulative distribution is particularly useful as it estimates how much data will be lost when applying a k-anonymity threshold. This allows one to determine a threshold that preserves some data utility while ensuring a reasonable privacy protection, especially when other risk mitigation measures are in place such as access control, audit trails, limited data lifetime or noising of released statistics. We can see that for the left curve, one can choose a high threshold with low data loss, while

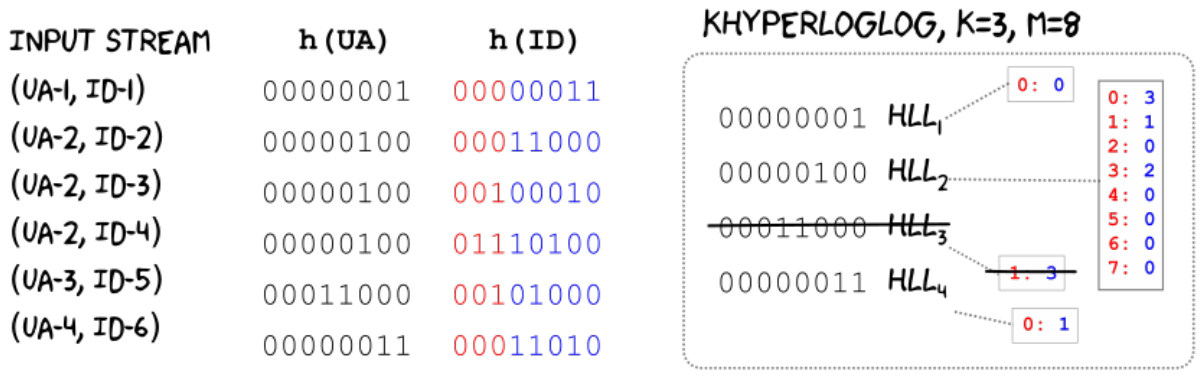


Fig. 1: A stream of User Agent (UA) and ID tuples processed by an example KHLL sketches with $K = 3$ and $M = 8$. When the tuple (UA-4, ID-6) is added, the entry with the largest $h(\text{UA}) = 00011000$ and its companion HLL sketch is purged to give way to h(UA-4) and a new HLL. Notice that the sketches HLL_1 and HLL_4 are in sparse representation, while HLL_2 is in the conventional table form.

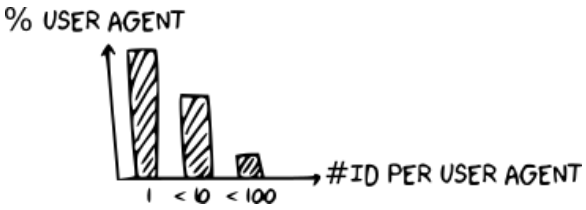


Fig. 2: Example uniqueness histogram. We expect User Agent (UA) to have a uniqueness distribution where a majority of UA strings are associated with only one or a few unique IDs.

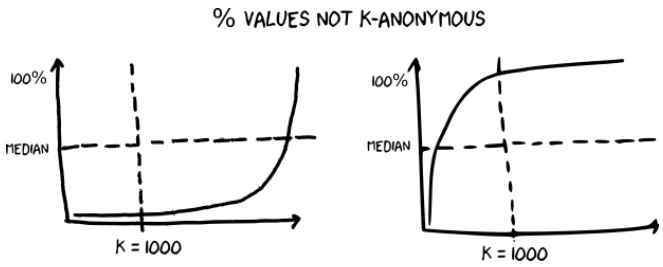


Fig. 3: Two possible shapes for cumulative uniqueness distributions. The left has low uniqueness, while the right contains values that are highly unique.

on the right even a moderate threshold will result in dropping a large portion of the data set.

Given the efficiency of KHLL analysis, one could set up periodic analyses to assure that the uniqueness distribution does not change over time, to monitor, for example, that no more than X% of values should have less than k-anonymity.

In addition to analyzing the uniqueness distribution of individual fields, KHLL can be used to analyze any combinations of fields, including a complete row. This can be done, for example, by simply concatenating the values of multiple fields, and treating the combination as a new field. The reidentifiability risk will grow as the number of dimensions increases. For example with movie recommendations, the combination of movie name, rating and date of recommendation can be highly unique [8].

B. Limitations and Mitigations

Using a KHLL sketching algorithm with $K = 2048$ and 512 HLL buckets would give us an estimated error rate of 2% for the value cardinality and about 4% error rate for ID cardinalities. A higher error rate of ID cardinality estimates is tolerable given that we are more concerned about field values that associate with low number of IDs. In those cases the algorithm will use HLL++ in sparse representation mode which gives good estimates with minimal errors. Note that the trade off between accuracy and efficiency is configurable.

Meanwhile as a KHLL sketch effectively maintains K uniform random samples of field values, the estimated distribution does come with sampling bias. Specifically, it is possible that the estimated distribution may miss some outlier field values that associate with a large or small number of IDs. One mitigation is to run multiple analyses using different hash functions (with random

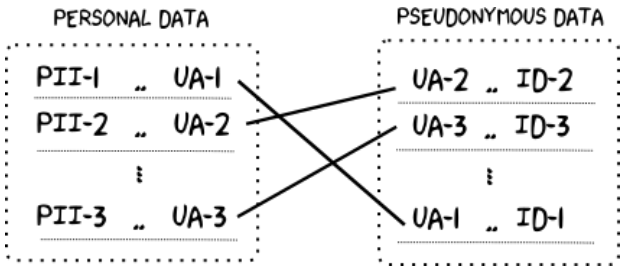


Fig. 4: Example illustration of joinability between Personally Identifiable Information (PII) and pseudonymous IDs with raw User Agent (UA) strings being the join key.

seeds) to reduce the chance of outliers being consistently left out of the analysis.

VII. Estimating Joinability using KHLL

Estimating the joinability of two data sets through a pair of fields say F_1 and F_2 from their KHLL sketches is also straightforward. Recall that containment of F_1 in F_2 is given by $|F_1 \cap F_2|/|F_2|$. To compute this, we need only the cardinality of F_1 and F_2 plus the cardinality of their intersection.

Using the inclusion-exclusion principle, we estimate $|F_1 \cap F_2| = |F_1| + |F_2| - |F_1 \cup F_2|$. The union of F_1 and F_2 can be easily estimated by merging the KHLL sketch of F_1 and that of F_2. An alternative approach for computing the set intersection is by identifying the hash values that exist in both the KHLL sketches of F_1 and F_2 and computing the minmax of the K smallest hashes on both sides. This would allow us to quantify the error rate of individual set intersections directly, but the error rate will vary as the minmax of the hashes will vary for different pairs of fields. We prefer the simpler inclusion-exclusion-based approach.

In addition to determining the joinability of fields, KHLL sketches can provide insights into the potential joinability of identities in two data sets. For example, pseudonymous IDs in one data set could be reidentified if the data set is joinable with another data set containing Personally Identifiable Information (PII), and if the join keys are associated with one pseudonymous ID and PII respectively (see Figure 4).

Specifically, given a pair of sketches of two fields F_1 and F_2 in data sets D_1 and D_2 respectively, we could estimate

- whether F_1 is highly contained in F_2 or vice-versa
- whether F_1 uniquely identifies ID_1 in D_1
- whether F_2 uniquely identifies ID_2 in D_2

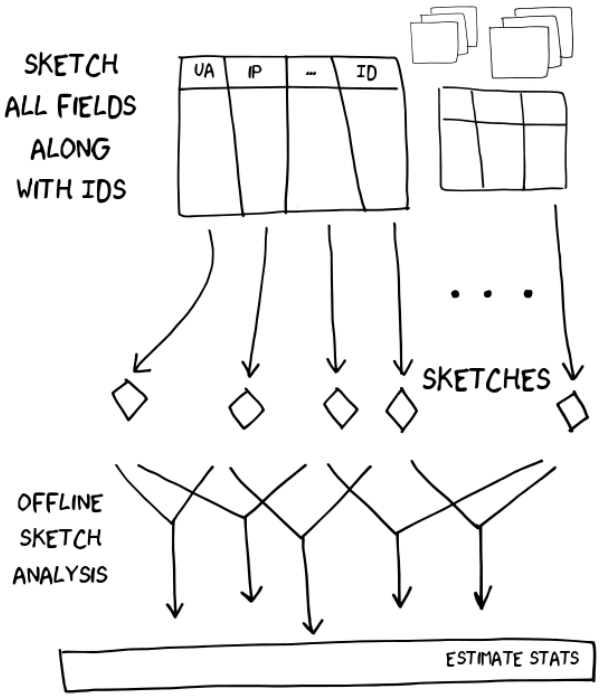


Fig. 5: Two-step approach of reidentifiability and joinability analysis: (i) distributed scanners read various data sets to produce a KHLL sketch for every (F, ID) tuple, (ii) various stats (including pairwise containment of data sets) are computed offline based on the sketches (rather than by comparing the raw data sets).

KHLL allows us to estimate all the above conditions. Specifically, the first level of KHLL which is essentially a KMV sketch can be used to estimate the cardinality of F_1 and F_2, the cardinality of the intersection, and thus containment. Meanwhile, the second level of KHLL, consisting of HLL sketches, gives the uniqueness distribution and thus the ratio of uniquely identifying values easily.

A. Practical Considerations for Joinability Analysis

Estimating the joinability of large data sets is a hard problem. Naively estimating the pairwise joinability of data sets involves a quadratic number of full-table scans. The number of scans needed can increase quickly, especially for large data sets with many data fields.

As shown in Figure 5 however, KHLL allows us to estimate joinability from the sketches alone. This is a huge saving in that it allows us to scan each data set only once and then pairwise compare the sketches rather than original data sets.

The sketching process can be agnostic to the underlying data when the schema of the data sets are well defined. For example when using protocol buffer messages [16] we can analyze new data sets without any need to configure information about fields, especially when the semantic types of data fields are properly annotated [17].

The sketching process can also be distributed. The respective data owners do not need to grant a central service access to the raw data, but simply to agree on the sketching method and to upload the sketches to a central repository. The sketches containing the hash values of potentially sensitive data including user IDs should still be treated with care such as by limiting the access and ensuring a short lifetime.

B. Limitations and Mitigations

Using sketches for joinability analysis comes with some risks of false positives and negatives.

False positives: The containment metric is agnostic to the semantics of the underlying data. Specifically, containment (or Jaccard) does not distinguish between fields that use a similar range of values but are semantically different. As an example, we could falsely determine `port_number` and `seconds_till_mignight` fields to be joinable since they both have an extensive overlap in the integer range of $[0, 86400)$. The rate of false positives could be mitigated by requiring a large cardinality threshold on the potential join keys.

False negatives: The containment metric will fail to detect similar fields that have been encoded differently (e.g., base64 versus raw string) or have undergone some slight transformations (e.g., a microsecond timestamp versus the coarsened millisecond version). This is a hard problem in practice. The system could potentially support some common transformations or encodings when the semantic type of a data field is known, but there is no way to handle all possibilities.

The containment metric can also be unreliable when set sizes are highly skewed. When the expected error rate of a set is larger than the cardinality of a much smaller set, the estimate for the set intersection computed using the inclusion- exclusion principle will be unreliable. One could potentially complement the containment metric with some other similarity scores like the similarity between the frequency distribution of the potential join keys.

While KHLL can evaluate the pairwise joinability of data sets based on individual fields, estimating the joinability of data sets through arbitrary combinations of fields remains practically infeasible given that intractable number of potential field combinations. One could however systematically approach this by testing the joinability between combinations of high-risk fields, for example, involving only those that have high uniqueness.

The pairwise joinability analysis does not readily detect multi-hop joinability. For example, when a data set D_1 is joinable with data set D_2, and D_2 is joinable with data set D_3 through two different pairs of join keys, we will not detect that D_1 is joinable to D_3. Such multi-hop joinability analysis could be similarly estimated using clustering and graph traversal algorithms such as label propagation [18].

VIII. Measuring Efficiency of KHLL

The KHLL algorithm and the metrics we estimate with it have been implemented in a proprietary production environment in Go using the MapReduce programing model [19].

For each field (or specified combinations of fields) in the data set the MapReduce outputs a KHLL sketch in one pass.

To reason about efficiency, we implemented two naive MapReduce algorithms: (i) Count Exact (CE) which computes the exact variants of the metrics that KHLL estimates, and (ii) Count Exact Single (CES) which computes the same set of metrics exactly, but analyzing only one single field during a given MapReduce run.

We designed the CE MapReduce to output, for each field, a dictionary of field values to ID sets (allowing the computation of various statistics that a KHLL sketch approximates). One would expect that emitting the entire field-value-to-ID-set dictionary will result in substantial memory overhead. The CES MapReduce is a more realistic simplification of CE which outputs the tuples of field values and ID sets of a only single specific field.

The test data set on which we ran our performance analyses represents the dataflow graph of various production operations at Google. Each row in this data set represents a node in the dataflow graph and has about 50 data fields describing the properties of the operations as well as the input and output data. One of the data fields specifies the globally unique name of the machine where the operation is run or controlled. We used this machine name as the ID field in our analyses. Note that this data set does not contain any user data and is not privacy sensitive as these are not necessary for performance measurements.

MapReduce type	Input size	Algorithm	CPU usage (vCPUs)	RAM usage (GBs)	Peak RAM (GB)	Output size (GB)	Runtime (s)
All fields	1 GB	CE	4.01e+3	1.14e+4	9.34e+0	1.04e+0	5.83e+2
		KHLL	9.78e+2	2.08e+3	9.45e-1	1.60e-3	1.43e+2
	100 GB	CE	3.53e+5	3.40e+6	1.10e+2	2.64e+0	1.93e+4
		KHLL	6.25e+4	3.11e+4	2.00e+0	3.46e-3	2.63e+2
	1 TB	CE	(n.a.)	(n.a.)	(n.a.)	(n.a.)	(n.a.)
		KHLL	9.92e+5	2.37e+6	2.52e+0	3.50e-3	1.13e+4
Single field	1 GB	CES	7.23e+2	4.76e+3	8.07e-1	1.57e-2	5.76e+2
	100 GB	CES	4.47e+4	1.30e+5	1.79e+0	2.35e-1	1.10e+3

TABLE I: Performance metrics of KHLL and exact counting algorithms. We configured KHLL to have K=2048 and M=1024. 1 GBs = 1 GB of RAM used for 1 second. Virtual CPU (vCPU) is a platform-neutral measurement for CPU resources. 1 vCPUs = 1 vCPU used for 1 second. The CE MapReduce for analyzing all data fields in a 1 TB data set was excessively expensive and was halted.

We ran KHLL, CE and CES on several subsets of the test data set in a shared computational cluster at Google. These analyses were provided computational resources at a priority class that is typically used for batch jobs. Measuring the performance metrics of jobs in a shared computational cluster is not straightforward since any given machine can host multiple unrelated jobs with varying priority classes that can consume machine resources in unpredictable ways. So we focused on the performance metrics which a typical customer of a commercial computational cluster (e.g., Amazon EC2, Google GCE) would pay for.

Table I shows the performance metrics of the MapReduce runs. As one can see, KHLL is consistently more efficient than CE across various metrics. Performance differs by 1 or 2 orders of magnitude even for the relatively small data sets in our experiment. In fact, the CE MapReduces for analyzing all data fields in an 1 TB data set became too expensive to be completed in the shared computational cluster. Interestingly, per our test data set, it is even more memory efficient (though slightly slower) to compute the KHLL sketches of all data fields in a single MapReduce run, than to analyze a single data field using CES. This performance disparity is critical in practice, as it is the difference between an analysis that is feasible to run, and one that is too slow to be worthwhile.

In various production setups at Google, KHLL scales to analyze hundreds of data sets, each containing potentially trillions of rows and tens of thousands of data fields measuring petabytes in sizes, to produce millions of sketches in each run.

IX. Measuring Accuracy of KHLL

To provide reproducible validation results, we have implemented a version of the KHLL algorithm in BigQuery Standard SQL, and simulated the computation of uniqueness distribution and joinability using publicly available data sets. The code for the experiments can likely be adapted for other SQL engines, and is shared on GitHub [20].

A. Accuracy of Uniqueness Distribution Estimation

We measured the accuracy of the estimated uniqueness distribution using three publicly available data sets. The first two are taken from the Netflix Prize data set which was released in 2006 and shown to be reidentifying for a significant fraction of the users [8]. We estimate the uniqueness distribution of (a) movies, and (b) tuples of movie and date. Note that we do not consider the entire list of movies (or respectively all pairs of movie and date) associated with individual pseudo-identifiers. We could analyze this easily but it will be less interesting for our validation purposes, as most of the values will be unique. The third data set is the 2010 US Census [21] through which we count the number of distinct individuals associated with a given (ZIP code, age) tuple. The corresponding uniqueness distribution gives an indication of the reidentifiability of these quasi-identifiers within the US population. As we will see, the three data sets present different uniqueness profiles, allowing us to test the accuracy of KHLL estimation in different situations.

We simulate the KHLL algorithm, with parameters $K = 2048$ and $M = 1024$. We learned from our production settings that $K = 2048$ gives a good tradeoff between precision and memory usage. $M = 1048$ was

chosen as the smallest possible parameter of the HLL++ library available in BigQuery Standard SQL. We use $M = 512$ in our production pipelines, which gives a comparable degree of precision in ID cardinality estimation (4% vs. 3%). As HLL++ counts elements exactly at lower cardinalities, this has a negligible influence on our estimations. For each data set, we compare the KHLL-based estimate to the true distribution, which is computed exactly (without approximation) using standard SQL operators.

Figure 6a plots the cumulative uniqueness distribution of movies in the Netflix Prize data set. It allows an analyst to quickly answer the question: how much data do we lose if we only release the movies which have been rated by more than k users, for all possible values of k. The uniqueness of movies is low: the median number of associated users per movie is larger than 500. Figure 6b plots the cumulative uniqueness distribution of the tuples of movie and date. The typical uniqueness of this setting is high: over 80% of the (movie, date) tuples are associated with 10 or less unique IDs.

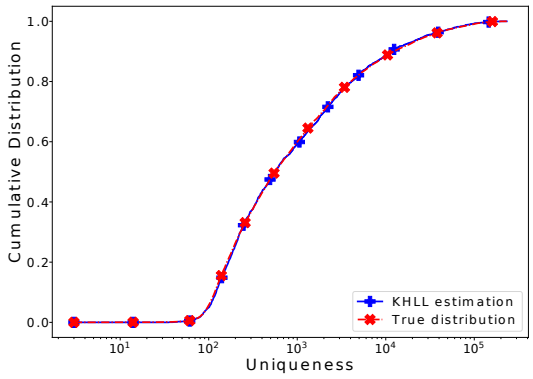
Figure 6c shows the cumulative uniqueness distribution of tuples (ZIP code, age) in the US census. Each individual in the census data set appears in only a single row, different from the case with Netflix data sets where ratings from the same user exist on several separate records. The uniqueness of (ZIP code, age) tuples is variable: a significant portion of possible values is associated to only a few individuals, but many of the (ZIP code, age) tuples associate with larger than 100 individuals.

Across all three experiments, we observe that the estimate of uniqueness distribution using KHLL is accurate.

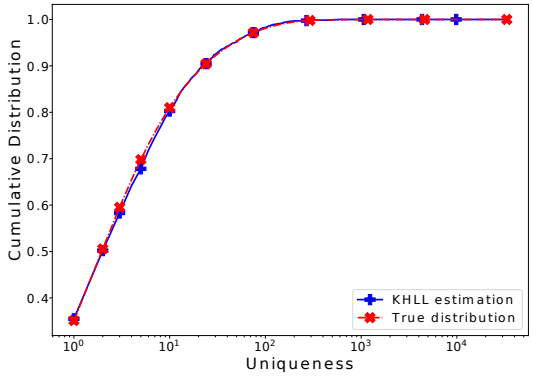
B. Accuracy of Joinability & Containment Estimation

As described in Section VII, joinability is most interesting from the privacy perspective when an pseudonymous ID space becomes joinable with PII. Specifically if fields F_1 and F_2 are joinable, and that F_1 uniquely identifies a pseudonymous ID space while F_2 uniquely identifies PII. Three conditions are important for estimating such risk: the ratio of F_1 values uniquely identifying ID_1, the ratio of F_2 values uniquely identifying ID_2, and the containment of F_1 in F_2 (or containment of F_2 in F_1).

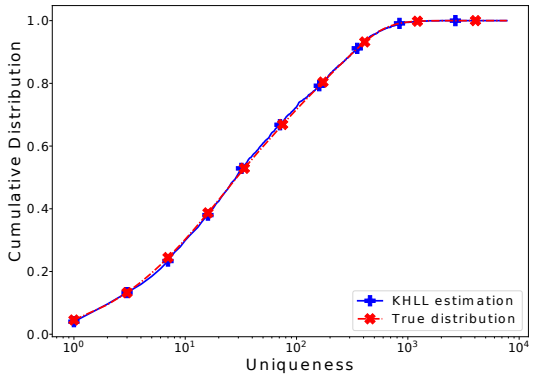
The estimate of ratio of field values uniquely identifying an ID can be seen as an estimator of the parameter p based on an observation of a binomial distribution of parameters K and p. It is well-known that a binomial distribution of parameters K and p has a variance of $p(1-p)K$, so the estimator which divides the result of the distribution by K has a variance of $p(1-p)/K$, or a standard distribution of $\sqrt{p(1-p)/K}$. Therefore,



(a) Netflix Prize: movie ID



(b) Netflix Prize: movie ID and date



(c) US Census: ZIP code and age

Fig. 6: Estimation of uniqueness distribution in different data sets using KHLL as compared to true distribution (computed exactly without approximation).

we focus our experiments on estimating the containment metric, as defined in Definition 3.

Using $K = 2048$ hashes, and assuming F_1 and F_2 have the same cardinality, the estimate of containment falls within $\pm 5\%$ of the true value over 90% of the time, and always stays within 10% of the true value. This is true regardless of the cardinality of the intersection of F_1 and F_2. Figure 7 shows the median as well as the 5th and 95th percentiles of the containment estimation, for cardinalities of 10,000 and 10,000,000.

When F_1 and F_2 have different cardinalities, however, precision can suffer. Figure 8 shows the median as well as the 5th and 95th percentiles of the estimation of $|F_1 \cap F_2|/|F_1|$, where true value is fixed at 50%, $|F_1| = 100,000$, and $|F_2|$ varies between 50,000 and 2,000,000 (so, the cardinality ratio $|F_2|/|F_1|$ ranges from 0.5 to 20).

We can observe that the larger the cardinality ratio gets, the worse the precision becomes. This is expected: since we compute $|F_1 \cap F_2|$ using the inclusion-exclusion principle, and the error of the estimation is proportional to the cardinality estimated, the estimation error of $|F_1 \cap F_2|$ should be roughly proportional to $\max(|F_1|, |F_2|)$. Since the value of $|F_1 \cap F_2|$ is roughly proportional to $\min(|F_1|, |F_2|)$, the error ratio of the containment estimation will grow linearly with the cardinality ratio. This is what we observe in practice.

X. Related Work

Using the frequency of (combinations of) field values as a proxy to measure reidentifiability is not new. A large body of research has emerged following the proposal of k-anonymity by Sweeney in 1997 [1]. Rather than just estimating k-anonymity, the KHLL algorithm estimates the entire uniqueness distribution, which is useful for evaluating the impact to data loss with k-anonymization or its variants (e.g. [2], [22]). While different from the notion of differential privacy [7], the reidentifiability and joinability risks as estimated using KHLL can serve to help determine a suitable anonymization strategy, particularly when considering the different contexts and use cases of anonymization.

The problem of gathering metadata and organizing data sets in large organizations has been described on several occasions. Halevy et al. [23] detail a search engine for data sets, which gathers metadata at data set level. Meanwhile, Sen et al. [24] explain how to detect and propagate field-level semantic annotations using assisted labeling and dataflow analysis. The tools we develop to automatically detect joinability of data sets could be used for a similar purpose. Inconsistent semantic annotations between data fields with high similarity scores (containment or Jaccard) can be automatically detected with the correct annotations propagated accordingly.

Approximate counting techniques have been the subject of a significant body of research e.g. [9]–[11], [25]. The techniques we propose are independent of the particular algorithm chosen to approximate ID cardinality. Specifically, the KHLL algorithm could use basic HyperLogLog [10], HyperLogLog++ [11] or our proprietary implementation of HyperLogLog++ Half Byte (see Appendix XII) to estimate uniqueness distribution and pairwise containment of data sets albeit with different memory efficiency.

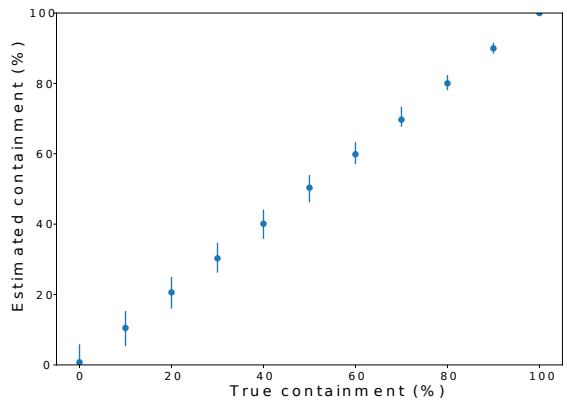
Beyond privacy, estimating value distribution is also essential to database query optimization and data stream processing. Most research in this domain focuses on sketching values of high frequency (i.e. statistical heavy hitters or top-k queries). The closest analogue of KHLL was presented by Cormode et al. [26]; it combines the Count-Min sketch [27] and the LogLog sketch [28] for value distribution estimation. However, the Count-Min algorithm biases towards values of high frequency, which is not helpful for evaluating the impact of k-anonymity given the typical choices of k are much smaller than the frequency of heavy hitters.

MinHash [4] and SimHash [29] are two popular algorithms for estimating the Jaccard similarity of data sets. The KHLL algorithm leverages the K Minimum Values in the first level of the two-level sketch for estimating Jaccard and containment scores, using a similar log n-space memory footprint. A possible improvement might be to adapt from HyperMinHash [30], capable of estimating Jaccard using only loglog n-space. Yet, given that the bulk of memory usage by KHLL actually comes from the second level of the sketch for estimating the uniqueness distribution, we have not explored the feasibility of adapting KHLL to have a HyperMinHash-like data structure in the first level.

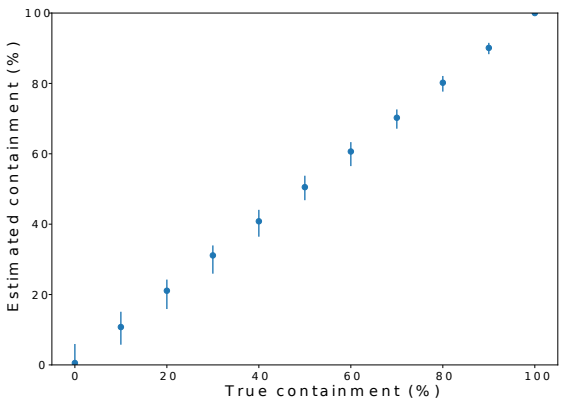
Finally, despite the extensive research for detecting data similarity, we have not seen any prior work tackling the problem of automatically detecting possible joinability between different ID spaces across data sets.

XI. Summary

The scale of data and systems in large organizations demands an efficient approach for reidentifiability and joinability analysis. The KHyperLogLog (KHLL) algorithm innovates on approximate counting techniques



(a) Low cardinality $= 10,000$



(b) High cardinality $= 10,000,000$

Fig. 7: Estimation of containment with fixed cardinalities. The bars indicate the 95th and 5th percentiles of the estimates.

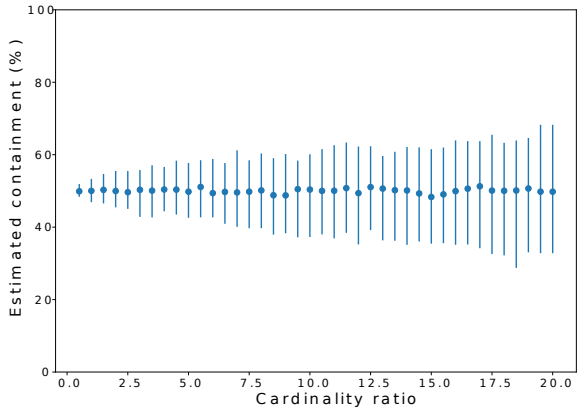


Fig. 8: Estimation of containment with varying cardinality ratio. The true containment is 50%. The bars indicate the 95th and 5th percentiles of the estimates.

to estimate the uniqueness distribution and pairwise containment of very large databases at scale.

The efficiency and scalability of risk analysis using KHLL represent a practical and useful tool for large organizations in protecting user privacy. It provides an objective, quantifiable and replicable measure of reidentifability of data sets. The KHLL algorithm also presents a novel and practical approach for tackling the joinability risks of data sets and ID spaces. The efficiency of KHLL further enables periodic analyses of complex production systems that evolve over time. We described the practical use of KHLL for protecting user privacy.

Future work: While KHyperLogLog is memory efficient, it still requires a linear pass over the data. Techniques to produce sketches suitable for joinability analysis without scanning the entire data set would be helpful. It would also be interesting to see more innovative use of approximate counting in privacy enhancing techniques (including data anonymization) rather than just for risk analysis purposes.

Acknowledgment

We thank Lea Kissner, Jessica Staddon, Rebecca Balebako, Lorenzo Martignoni, Nina Taft and the anonymous reviewers for the valuable feedback on the earlier version of the paper. We also thank all colleagues who have contributed directly or indirectly to the work.

References

[1] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002. [Online]. Available: http://dx.doi.org/10.1142/S0218488502001648

[2] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "L-diversity: Privacy beyond k-anonymity," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 1, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1217299.1217302

[3] P. Jaccard, "Lois de distribution florale dans la zone alpine," vol. 38, pp. 69–130, 01 1902.

[4] A. Broder, "On the resemblance and containment of documents," in *Proceedings of the Compression and Complexity of Sequences 1997*, ser. SEQUENCES '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 21–. [Online]. Available: http://dl.acm.org/citation.cfm?id=829502.830043

[5] P. Eckersley, "How unique is your web browser?" in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, ser. PETS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881151.1881152

[6] X. Xiao and Y. Tao, "Anatomy: Simple and effective privacy preservation," in *Proceedings of the 32Nd International Conference on Very Large Data Bases*, ser. VLDB '06. VLDB Endowment, 2006, pp. 139–150. [Online]. Available: http://dl.acm.org/citation.cfm?id=1182635.1164141

[7] C. Dwork, "Differential privacy," in *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ser. ICALP'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1007/11787006_1

[8] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse data sets," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, ser. SP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 111–125. [Online]. Available: https://doi.org/10.1109/SP.2008.33

[9] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, "On synopses for distinct-value estimation under multiset operations," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: ACM, 2007, pp. 199–210. [Online]. Available: http://doi.acm.org/10.1145/1247480.1247504

[10] P. Flajolet, Éric Fusy, O. Gandouet, and et al., "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *IN AOFA '07: Proceedings Of The 2007 International Conference On Analysis Of Algorithms*, 2007.

[11] S. Heule, M. Nunkesser, and A. Hall, "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*, ser. EDBT '13. New York, NY, USA: ACM, 2013, pp. 683–692. [Online]. Available: http://doi.acm.org/10.1145/2452376.2452456

[12] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '98. New York, NY, USA: ACM, 1998, pp. 426–435. [Online]. Available: http://doi.acm.org/10.1145/276304.276342

[13] Z. Karnin, K. Lang, and E. Liberty, "Optimal quantile approximation in streams," in *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, Oct 2016, pp. 71–78.

[14] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of the 10th International Conference on Database Theory*, ser. ICDT'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 398–412. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30570-5_27

[15] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1530–1541, Aug. 2008. [Online]. Available: http://dx.doi.org/10.14778/1454159.1454225

[16] "Protocol buffers," https://developers.google.com/protocol-buffers/, [Online, last accessed Oct 28, 2018].

[17] "Semantic types of protocol buffer fields can be annotated using custom options." https://developers.google.com/protocol-buffers/docs/proto#options, [Online, last accessed Oct 28, 2018].

[18] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Tech. Rep., 2002.

[19] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[20] "Evaluating khll accuracy with bigquery," https://github.com/google/khll-paper-experiments.

[21] "Decennial census of population and housing," https://factfinder.census.gov, 2010.

[22] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, April 2007, pp. 106–115.

[23] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang, "Goods: Organizing google's data sets," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 795–806. [Online]. Available: http://doi.acm.org/10.1145/2882903.2903730

[24] S. Sen, S. Guha, A. Datta, S. K. Rajamani, J. Tsai, and J. M. Wing, "Bootstrapping privacy compliance in big data systems," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 327–342. [Online]. Available: http://dx.doi.org/10.1109/SP.2014.28

[25] Q. Xiao, Y. Zhou, and S. Chen, "Better with fewer bits: Improving the performance of cardinality estimation of large data streams," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[26] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," in *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '05. New York, NY, USA: ACM, 2005, pp. 271–282. [Online]. Available: http://doi.acm.org/10.1145/1065167.1065201

[27] ——, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jalgor.2003.12.001

[28] P. Flajolet and G. N. Martin, "Probabilistic counting," in *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, ser. SFCS '83. Washington, DC, USA: IEEE Computer Society, 1983, pp. 76–82. [Online]. Available: https://doi.org/10.1109/SFCS.1983.46

[29] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '02. New York, NY, USA: ACM, 2002, pp. 380–388. [Online]. Available: http://doi.acm.org/10.1145/509907.509965

[30] Y. W. Yu and G. Weber, "Hyperminhash: Jaccard index sketching in loglog space," *CoRR*, vol. abs/1710.08436, 2017. [Online]. Available: http://arxiv.org/abs/1710.08436

XII. Appendix: HLL++ Half Byte

Standard HLL++ uses a 64-bit hash. Typically the counters of trailing zeros in hashes are stored in byte size values for simplicity and efficiency of implementation. This arrangement is convenient but memory inefficient because the counts are only in the range $[0, 64]$. We

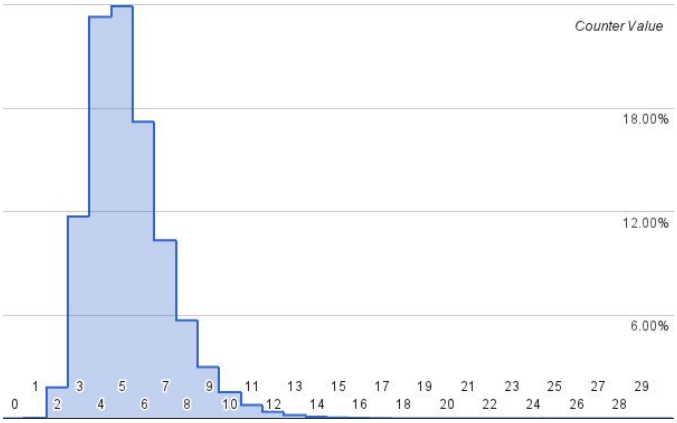


Fig. 9: Histogram of the counters of trailing zeros in hashes computed on our proprietary data sets.

improve the memory signature slightly with the following observation. Since hashes are distributed uniformly between all buckets, the counters tend to be clustered together. Rather than storing the actual count, we store count offsets instead. Specifically, we store a single value ρ and a table with offset values in the range $[0, 16)$. Real count values correspond to ρ + offset.

As we see more unique values, all the offsets in the table increase. When there are no more counters with offset 0 left, ρ is incremented and each offset in the table is decremented by 1. It is always possible there are outlier counters exceeding $\rho + 15$. We store all such additional values in an outlier list, merging them back into the table as ρ increases. As shown in Figure 9, we validated that the number of outliers is small in practice.

By storing offsets, we reduce the number of bits per HLL bucket from 8 to 4, allowing us to store 2 buckets in each byte. The additional elements (ρ and the outlier list) are small so this effectively allows us to have twice as many counters for roughly the same memory signature. Since we use the memory savings to increase the number of HLL buckets this improves the error rate by a factor of $\frac{1}{\sqrt{2}}$. This is comparable to the recent HLL-TailCut+ algorithm [25] which improves the memory efficiency of HLL by 45%.

XIII. Appendix: K2MV

At a high level KHLL stores a K Minimum Values (KMV) sketch with an HLL sketch for each distinct hash value (bucket). An alternative we considered and experimented was using KMV sketches instead of HLL for the individual buckets. We name this K2MV given the two-level data structure of minimum hash values. To improve the performance, we implemented this as a single table storing all hash values instead of in multiple KMV stores. The size of the table is governed by parameters K_1, K_2, corresponding to the number of hashes stored at each level. We use an amortized strategy where hash values are stored in sorted order. New hash values are written to a buffer which is periodically merged into the sorted list.