Efficient Estimation of Inclusion Coefficient using HyperLogLog Sketches

Azade Nazi, Bolin Ding, Vivek Narasayya, Surajit Chaudhuri
Microsoft Research
{aznazi, bolind, viveknar, surajitc}@microsoft.com

ABSTRACT

Efficiently estimating the *inclusion coefficient* – the fraction of values of one column that are contained in another column – is useful for tasks such as data profiling and foreign-key detection. We present a new estimator, BML, for inclusion coefficient based on Hyperloglog sketches that results in significantly lower error compared to the state-of-the art approach that uses Bottom-k sketches. We evaluate the error of the BML estimator using experiments on industry benchmarks such as TPC-H and TPC-DS, and several real-world databases. As an independent contribution, we show how Hyperloglog sketches can be maintained incrementally with data deletions using only a constant amount of additional memory.

PVLDB Reference Format:
Azade Nazi, Bolin Ding, Vivek Narasayya, Surajit Chaudhuri. Efficient Estimation of Inclusion Coefficient using HyperLogLog Sketches. *PVLDB*, 11(10): 1097-1109, 2018.
DOI: https://doi.org/10.14778/3231751.3231759

1. INTRODUCTION

The discovery of all inclusion dependencies in a dataset is an important part of data profiling efforts. It is useful for tasks such as foreign-key detection and data integration [17, 28, 4, 22, 21]. However, due to issues in data quality such as missing values or multiple representations of the same value, it becomes important to relax the requirement of exact containment. Thus, computing the fraction of values of one column that are contained in another column –*inclusion coefficient* is of interest to these applications.

When the database schema and data sizes are large, computing the inclusion coefficient for many pair of columns in the database can be both computationally expensive and memory intensive. One approach for addressing this challenge is to *estimate* the inclusion coefficient using only bounded-memory *sketches* of the data. Given a fixed budget of memory per column, these techniques scan the data once, and compute a data sketch that fits within the memory budget. For a given pair of columns X and Y, the inclusion coefficient $\Phi(X, Y)$ is then estimated using only the sketches on columns X and Y.

One such approach, adopted in Zhang et al. [31] is to build a Bottom-k sketch [12] on each column, and develop an inclusion coefficient estimator using these sketches. Intuitively, Bottom-k sketches have high accuracy for inclusion coefficient estimation when the number of distinct values in both X and Y are smaller than k since the sketches effectively behave like hash tables of the respective columns. However, as we show empirically in this paper, one of the limitations of using Bottom-k sketches for estimating inclusion coefficient is that for given memory budget (of k values), as the cardinality (i.e. number of distinct values) of column X or Y increases beyond k, the estimation error becomes large. Additionally, Bottom-k sketches are not amenable to incremental maintenance in situations where data is deleted. For instance, in data warehousing scenarios, it is not uncommon for recent data to be added and older data to be removed from the database.

Developing an estimator with low error for inclusion coefficient using bounded-memory sketches is challenging. We first establish a hardness result showing that any estimator that relies only on sketches with bounded memory must incur unbounded error in certain cases. Intuitively, the difficult cases are when column X has small cardinality and column Y has large cardinality or vice-versa. Furthermore, as we report in our empirical evaluation, these difficult cases appear to be quite common in the real-world databases that we have studied.

We develop a new estimator for inclusion coefficient BML (Binomial Mean Lookup) based on *Hyperloglog* sketches [14]. Hyperloglog (HLL) sketches can be computed efficiently and within a given memory budget, requiring invocation of only a single hash function for each value in the column. HLL sketches are becoming increasingly popular for estimating cardinality in different applications, and are even being adopted in commercial database engines – e.g., [16] uses Hyperloglog to support approximate distinct count functionality. The main idea behind the BML estimator is a theoretical result that establishes a mapping from the inclusion coefficient $\Phi(X, Y)$ to the probability that the value of a bucket in the HLL sketch of column Y is greater than the value of the corresponding bucket in the HLL sketch of column X. BML, is based on *maximum likelihood estimation* (MLE) method. It observes the number of buckets in the HLL sketch of column Y whose HLL value is greater than the HLL value of the corresponding bucket in column X, and it returns the value of $\Phi(X, Y)$ that maximizes the likelihood of this observation. As a by-product of our estimation technique, we are also able to provide a *bound* on the error. This error bound is data-dependent, i.e. it is specific to the pair of columns X and Y. Such an error bound can be valuable to applications that consume the estimates, and is not provided by prior techniques.

Hyperloglog sketches can be maintained incrementally in the presence of insertions. An independent contribution of this paper is a technique for incrementally maintaining a Hyperloglog sketch in the presence of data *deletions* with a constant memory overhead.

Proceedings of the VLDB Endowment, Vol. 11, No. 10

DOI: https://doi.org/10.14778/3231751.3231759

The key idea is that each bucket in an HLL sketch always holds an integer value less than a constant ℓ, where ℓ is the number of bits of the hash value. By maintaining a max-heap of constant size (at most ℓ) for each bucket we are able to support incremental deletion.

We show through experiments on real-world databases and industry benchmark TPC-H and TPC-DS databases that BML has significantly lower overall error than the approach using Bottom-k sketches [31], particularly in cases where at least one of the columns has large cardinality. For cases where both columns have small cardinality, the accuracy of both estimators are similar. For instance, in two real-world databases where there are many columns with small and large cardinality, the average error using Bottom-k sketches is 0.30 and 0.59 respectively, whereas the corresponding errors for BML are only 0.10 and 0.14. For the other two real-world databases where most columns have small cardinality the average error using Bottom-k sketches is 0.05 and 0.02 respectively, whereas the corresponding errors for BML are 0.06 and 0.04.

Finally, we consider one important application of inclusion coefficients, namely the problem of foreign-key (FK) detection in a database. All prior work on FK detection [29, 31, 11] relies on exact inclusion coefficients to prune the FK candidates. We show empirically on several real-world and benchmark databases that the estimation error of BML is acceptable for these FK detection techniques. In other word, replacing the exact inclusion coefficient with an estimate obtained via the BML estimator has no noticeable impact on the precision and recall of these FK detection algorithms.

In summary, this paper makes the following contributions:

- We establish a hardness result for inclusion coefficient estimation using bounded-memory sketches.
- We develop an MLE-based estimation algorithm called BML (Binomial Mean Lookup) for inclusion coefficient estimation based on Hyperloglog sketches [14]. We prove the correctness of our algorithm and the error bound of our estimator.
- We show how, with a constant memory overhead, Hyperloglog sketches can be extended to support incremental deletion.
- We implement our inclusion coefficient estimator (BML) on Cosmos [10], a distributed, Big Data engine that is extensively used within Microsoft for analyzing large data sets; and evaluate the effectiveness of BML using several synthetic and real-world datasets.
- We measure the precision and recall of two existing foreign-key detection techniques when using inclusion coefficient estimates rather than the exact inclusion coefficients.

The rest of the paper is organized as follows. We present a hardness result for inclusion coefficient estimation using sketches and introduce background of Hyperloglog sketch construction in §2. We describe the BML estimator for inclusion coefficient and its error analysis in §3. The extension to support incremental deletion is presented in §4. We describe the results of our experimental evaluation of inclusion coefficient estimation and its application to FK detection in §5. We discuss related work in §6 and conclude in §7.

2. PROBLEM DEFINITION AND BACKGROUND

Let database $\mathcal{D}$ be the collection of tables T, where the set of all columns in tables T are $\mathcal{C}$. Let n be the total number of columns ($|\mathcal{C}| = n$). For each column $X \in \mathcal{C}$, the set of all its possible values is called the domain of X, denoted by $\mathsf{dom}(X)$. We use $X[i]$ as the value of column X for tuple i. We formally define the inclusion coefficient estimation problem and we discuss the hardness result.

2.1 Inclusion Coefficient Estimation and Hardness Result

Inclusion coefficient is defined between two sets of values, and is used to measure the fraction of values of one set that are contained in the other set.

DEFINITION 1. (Inclusion Coefficient) *Given two columns/sets, X and Y, the inclusion coefficient of X and Y is defined as:*

$$\Phi(X,Y) = \frac{|X \cap Y|}{|X|}, |X| \neq 0 \quad (1)$$

where $|\cdot|$ represents the number of distinct values in a set.

If X is fully covered by Y ($X \subseteq Y$), $\Phi(X,Y) = 1$; otherwise $0 \leq \Phi(X,Y) < 1$. Note that $\Phi(X,Y)$ is *asymmetric*: in general, $\Phi(X,Y)$ is not equal to $\Phi(Y,X)$.

In tasks such as foreign key detection and data profiling, we need to calculate inclusion coefficients for many pairs of columns, which could be too expensive (in terms of both time and memory) for large datasets. As a trade-off between accuracy and performance, the idea is to construct a *sketch* (or a compact data structure) for each column $C \in \mathcal{C}$ by scanning the data once, and *estimate* the inclusion coefficient between any pair of columns using their sketches.

Such sketching and estimation techniques are useful in two scenarios: i) columns are too large to fit into memory; and ii) we want to compute inclusion coefficients for many pairs of columns (e.g., there are n columns and inclusion coefficients need to be calculated for all the $n(n-1)/2$ pairs).

PROBLEM 1. (Estimating Inclusion Coefficient using sketches) *For each column $C \in \mathcal{C}$, we construct a sketch $\mathcal{S}_C$ by scanning C once. Then for any two columns X and Y, we want to derive an estimator $\hat{\Phi}(\mathcal{S}_X, \mathcal{S}_Y)$ to $\Phi(X,Y)$, by accessing only the two sketches.*

In the rest part, we write $\hat{\Phi}(\mathcal{S}_X, \mathcal{S}_Y)$ as $\hat{\Phi}$ if X and Y are clear from the context. Table 1 shows frequently used notations.

2.1.1 Estimation Error and Sketch Size

Suppose we estimate the inclusion coefficient $\Phi(X,Y)$ of two columns X and Y as $\hat{\Phi}$, using their sketches. The *estimation error* $|\Phi(X,Y) - \hat{\Phi}|$ ranges from 0 to 1. Considering the randomness in the sketch construction, ideally, we would like the estimation error to be bounded with high probability, i.e., $|\Phi(X,Y) - \hat{\Phi}| \leq \epsilon$ with probability at least $1-\delta$, for any given two columns X and Y. Unfortunately, we can show that, unless the sketch size is linear in the number of distinct values (which could be equal to the number of rows), there is no sketch based on which the worst-case estimation error can be bounded with $\epsilon < 1$ and $\delta < 1$.

A lower bound of sketch size: The hardness can be observed even in a very simple case when $X = \{x\}$ contains only one element, and Y is large. In this case, $\Phi(X,Y)$ takes value either 0 (if $x \notin Y$) or 1 (otherwise). Therefore, to bound the estimation error below any constant less than 1, from the sketches of Y, we need to distinguish two cases, i) $x \notin Y$ or ii) $x \in Y$, with high probability—this is exactly the approximate membership problem [9]. More formally, we can prove the following hardness result.

THEOREM 1. *We pre-compute a sketch for each column in a database to estimate the inclusion coefficient for pairs of columns. If we require that, for any two given columns X and $Y \in \mathcal{C}$, $|\Phi(X,Y) - \hat{\Phi}| \leq \epsilon < 1$ with probability at least $1-\delta > 0$, then any sketch must use space at least $\Omega(n_C \log(1/\delta))$ bits per column $C \in \mathcal{C}$, where n_C is the number of distinct values in C.*

PROOF. From the above discussion, we can reduce *approximate membership problem* to our inclusion coefficient estimation problem: for each column Y of size n, we want to construct a sketch

to support membership queries, i.e., check whether $x \in Y$ for any given x, with a false positive rate at most δ.

Suppose we want to estimate the inclusion coefficient $\Phi(X, Y)$ using sketches of X and Y. Consider the case when $X = \{x\}$. Indeed, $\Phi(X, Y) = 1$ if $x \in Y$, or 0 if $x \notin Y$. Therefore, to ensure that the estimation error is less than 1, there must be no false positive to the membership query $x \in Y$ (with probability less than δ)—[9] shows that, for this purpose, any sketch must use space at least $\Omega(n \log(1/\delta))$ bits. So the proof is completed.

Remark: [25] gives an even tighter lower bound of sketch sizes when the number of distinct values in each column is unknown. □

2.2 HLL Sketch Construction

As shown in Theorem 1, the worst-case error cannot be bounded based on sublinear-size sketches. However, the hope is that, for particular instances of X and Y, the estimation errors still could be much better than the worst-case error. Hyperloglog (HLL) sketch [14, 15] provides a near-optimal way to estimate cardinality (i.e. the number of distinct values in a set). In §3, we will show how to use HLL sketches to estimate inclusion coefficients. We first review how to construct the HLL sketch of a column.

Let $h : \mathsf{dom}(X) \rightarrow \{0,1\}^\ell$ be a hash function that returns ℓ bits for each value $X[i] \in \mathsf{dom}(X)$. For each hash value $s_i = h(X[i])$, we find the position of the leftmost 1 represented by $\rho(s_i)$ and the maximum of $\{\rho(s_i) : s_i = h(X[i])\}$ is the HLL sketch of the column X which is used for cardinality estimation [14, 15].

One way to reduce the variance of the estimation is to use multiple hash functions [15]; however, [15] proposed *stochastic averaging* that employing only a single hash function emulates the effect of using different hash functions. They use one hash function but take the first m bits to bucketize hash values to mimic 2^m hash functions and reduce the variance. Then the remaining $\ell - m$ bits of each hash value $s_i = h(X[i])$ are used to find the position of the leftmost 1 as $\rho(s_i)$. Algorithm 3 in Appendix A shows the steps to construct such sketches for a set of columns $\mathcal{C}$. In §3.3, we discuss how to choose the parameter m for given two columns X, and Y.

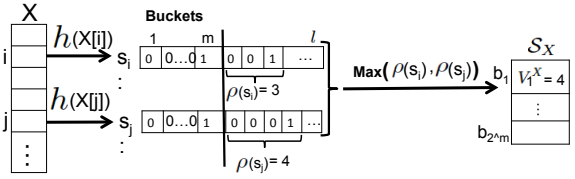

Figure 1: Constructing the HLL sketch of the column X as $\mathcal{S}_X$.

For example, as shown in Figure 1, one hash function h is used for all values in column X, and the first m bits are used to bucketize hash values. As a result there are 2^m buckets $(b_1, b_2, \cdots, b_{2^m})$ in HLL sketch of column X represented by $\mathcal{S}_X$. Specifically, in this figure $s_i = h(X[i])$ and $s_j = h(X[j])$ are assigned to bucket b_1 because their first m bits are the binary representation of the value one. Moreover, $\rho(s_i) = 3$ and $\rho(s_j) = 4$ because the position of the leftmost 1 in the remaining $\ell - m$ bits of the hash values s_i and s_j are 3, and 4 respectively. If the s_i and s_j are the only hash values assigned to b_1, the final value in b_1 is $V_1^X = max(3, 4) = 4$. The formal definition of V_i^X is given by Definition 2.

DEFINITION 2. (V_i^X: HLL value of column X in bucket b_i). *Let s_j be the hash value of the tuple j in X whose first m bits ($s_j[1, ..., m]$) indicate it belongs to bucket b_i, i.e., $s_j[1, ..., m]$ is the binary representation of the value i ($(s_j[1...m])_2 = i$), and $\rho(s_j)$ be the leftmost one in the remaining $\ell - m$ bits of the hash value s_j ($s_j[m + 1, ..., \ell]$). The HLL value of column X for the bucket b_i is defined as:*

Table 1: Summary of frequently used notations

n	Total number of columns
n_X	Number of distinct values in column X
$\mathcal{S}_X$	Sketch of column X
$\Phi(X, Y)$	Inclusion coefficient of columns X and Y
$\hat{\Phi}$	Estimation of $\Phi(X, Y)$
h	ℓ bits Hash function
m	Number of bits assigned for bucketization
2^m	Number of buckets in HLL sketch
V_i^X	HLL value of column X in bucket b_i
V^X	HLL value of column X when there is only one bucket
$\hat{\mathcal{P}}$	Estimated value of $pr(V^X \leq V^Y)$
e_p	Estimation error of $pr(V^X \leq V^Y)$
e_Φ	Estimation error of $\Phi(X, Y)$

$$V_i^X = \max_{s_j : (s_j[1...m])_2 = i} \rho(s_j) \quad (2)$$

Note that when there is only one bucket we ignore index i and denote the HLL value of column X by V^X.

Space complexity: In the HLL sketch of the column X ($\mathcal{S}_X$) the HLL value of each bucket is an integer number ($1 \leq V_i^X \leq \ell - m$). Thus each bucket needs $\log(\ell - m)$ bits to store V_i^X, i.e., total memory to store the sketch $\mathcal{S}_X$ is $\mathcal{O}(2^m \log(\ell - m))$.

3. EFFICIENT ESTIMATION OF INCLUSION COEFFICIENT

In this section, we describe a technique to estimate the inclusion coefficient using Hyperloglog (HLL) sketches. In a preprocessing step, a HLL sketch [14, 15] is constructed for all columns by scanning the data only once—in practice these sketches are compact enough to be able to fit into memory. Our algorithm aim to estimate the inclusion coefficient of two columns X and Y using pre-computed HLL sketches of X and Y. We also produce an error bound for the estimate.

3.1 Overview of our Approach

The main idea of our algorithm is to produce an estimate of the inclusion coefficient between two columns X and Y by comparing their HLL values (Definition 2). To develop intuition on why comparing the HLL values of X and Y can help in finding the inclusion coefficient, we start with the case of a HLL sketch with a *single bucket*, and suppose that X and Y have the same *number* of distinct values. We examine the following two extreme cases. i) if $X = Y$ or $\Phi(X, Y) = 1$, we have $pr(V^X \leq V^Y) = pr(V^X = V^Y) = 1$, because the hash function h in HLL is applied on the same set of values for both X and Y; and ii) if $X \cap Y = \varnothing$ or $\Phi(X, Y) = 0$, $pr(V^X \leq V^Y)$ is at least 0.5. A useful observation here is that $pr(V^X \leq V^Y)$ increases monotonically as $\Phi(X, Y)$ increases (we prove it formally later). For example, in Figure 2, we plot $pr(V^X \leq V^Y)$ as a function of $\Phi(X, Y)$ when $|X| = |Y| = 10^4$. §3.2.1 introduces how to derive this function for general cases.

Clearly given bucket i for each columns X and Y, the event $V_i^X <= V_i^Y$ is a Bernoulli trial. When there are multiple buckets since buckets are independent, the events $V_i^X <= V_i^Y$ are independent Bernoulli trials [15]. The reason that for a given column the buckets are independent is the result of the HLL construction 2.2. from property of the universal hashing, first m bits of a hash value is independent of the rest $\ell - m$ bits.

The intuition behind our algorithm is that, using multiple buckets in the HLL sketches of X and Y, we first estimate $pr(V^X \leq V^Y)$ given the fact that the event $V_i^X \leq V_i^Y$ is an independent Bernoulli trial for each bucket b_i (e.g., estimating $pr(V^X \leq V^Y) \approx 0.8$ in Figure 2). Then, since $pr(V^X \leq V^Y)$ can be written as a function of the inclusion coefficient, we effectively "lookup" the value of

$\Phi(X,Y)$ that produces the estimated $pr(V^X \leq V^Y)$. For example, in Figure 2 by looking up we get $\Phi(X,Y) \approx 0.62$).

Maximizing the likelihood. Our algorithm is based on the *maximum likelihood estimation* (MLE). More formally, let $\mathcal{Z} = |\{i \mid V_i^X \leq V_i^Y\}|$ be the number of buckets (among all 2^m buckets) where $V_i^X \leq V_i^Y$. The random variable $\mathcal{Z}$ follows a distribution parameterized by $|X|$, $|Y|$, and $\Phi(X,Y)$. We observe $\mathcal{Z} = z$ from HLL sketches of X and Y, and we want to choose $\Phi(X,Y)$ to maximize the likelihood of our observation.

$$\Phi_{mle} = \text{argmax}_{\phi} pr\left(\mathcal{Z} = z \mid \Phi(X,Y) = \phi\right). \tag{3}$$

Next we introduce our MLE-based estimation algorithm called *BML* (*Binomial Mean-Lookup*). Suppose $|X|$ and $|Y|$ are known (or estimated from their HLL sketches), there are still two remaining questions. First, we need to characterize the distribution of $\mathcal{Z}$ with $\Phi(X,Y)$ as a parameter (§3.2.1). Second, we need an efficient algorithm to maximize the likelihood as in Equation 3 (§3.2.2). We prove the correctness of our algorithm, i.e., show that our algorithm indeed gives a solution to Equation 3, and analyze its estimation error in §3.2.3. Finally, we briefly discuss how BML can leverage additional memory if available to improve accuracy (§3.4).

3.2 BML: Binomial Mean-Lookup Estimator

As shown in Figure 2, $pr(V^X \leq V^Y)$ increases monotonically as $\Phi(X,Y)$ increases. We first show how to derive the closed form of $pr(V^X \leq V^Y)$ as a function of $\Phi(X,Y)$ then we discuss the detail of our inclusion coefficient estimator BML.

3.2.1 Calculating $pr(V^X \leq V^Y)$

Given columns X and Y, V^X and V^Y are both random variables. Let us first consider a simpler case where there is only one random variable V^X and discuss how to find $pr(V^X \geq k)$, where k is constant. Then we show how to use this simple case to derive the general case $pr(V^X \leq V^Y)$.

Given column X with n_x distinct values, when there is only one bucket ($m = 0$), based on the Definition 2, the HLL value of the column X is in fact the maximum over n_x independent random variables. As we discussed in §2.2, for each hash value s_i, we find the position of the leftmost 1 represented by $\rho(s_i)$ and the maximum of $\{\rho(s_i) : s_i = h(X[i])\}$ is the HLL sketch of the column X. Obviously every bit in the s_i is a Bernoulli trial given that it is a random experiment with exactly two possible outcomes, "0" and "1" and from property of the universal hashing every bit of a hash value is independent from each other. Thus, the bits in s_i are independent Bernoulli trials. Each random variable $\rho(s_i)$ represents the leftmost one in s_i, i.e, first one after $\rho(s_i) - 1$ zeros. Thus as also pointed out in [14] each random variable is geometrically distributed and $pr(V^X \leq k)$ is:

$$pr(V^X \leq k) = \Pi_{i=1}^{n_X}(1 - Pr(first\ k\ bits\ are\ zero)) = (1 - \frac{1}{2^k})^{n_X} \tag{4}$$

By Equation 4, $pr(V^X = k)$ is $pr(V^X \leq k) - pr(V^X \leq k-1)$.

$$pr(V^X = k) = (1 - \frac{1}{2^k})^{n_X} - (1 - \frac{1}{2^{k-1}})^{n_X} \tag{5}$$

Next we show how to use these equations to derive $pr(V^X \leq V^Y)$, where both V^X and V^Y are random variables. When the intersection of X and Y is non empty V^X and V^Y are not independent. Let $\mathcal{T}$ be $X \cap Y$. To resolve the dependency of X and Y, we consider three disjoint sets: $\mathcal{T}$, $\mathcal{X} = X \setminus \mathcal{T}$, and $\mathcal{Y} = Y \setminus \mathcal{T}$. As shown in Figure 3 based on the cardinality of these sets there are three different cases. (1) X and Y are disjoint, i.e. $n_\mathcal{T} = 0$, $n_\mathcal{X} = n_X$, $n_\mathcal{Y} = n_Y$. (2) Y is subset of X, i.e., $\mathcal{Y} = Y \setminus \mathcal{T}$ is empty ($n_\mathcal{T} = n_Y$, $n_\mathcal{X} = n_X - n_Y$, $n_\mathcal{Y} = 0$). (3) X and Y are partially overlapped $\mathcal{Y} = Y \setminus \mathcal{T}$ is not empty ($n_\mathcal{T} \neq 0$,$n_\mathcal{X} \neq 0$, $n_\mathcal{Y} \neq 0$). Next we show how to use Equations 4, and 5 to derive the $pr(V^X \leq V^Y)$ for each case.

Case1: Clearly, if $\mathcal{T} = X \cap Y$ is empty ($n_\mathcal{T} = 0$), then V^X and V^Y are independent. As shown in Figure 3, since we are interested in $V^X \leq V^Y$ if X and Y are disjoint and $V^\mathcal{Y} = k$, then $V^\mathcal{X}$ should be at most k. Based on Definition 2 we have $0 \leq k \leq \ell - m$. Thus the $pr(V^X \leq V^Y)$ with the help of Equations 4 and 5 can be calculated by Equation 6. Note that since $n_\mathcal{T} = 0$, $n_\mathcal{X} = n_X$ and $n_\mathcal{Y} = n_Y$.

$$\begin{aligned} pr(V^X \leq V^Y) &= \sum_{k=0}^{\ell-m} pr(V^\mathcal{X} \leq k) \wedge pr(V^\mathcal{Y} = k) \\ &= \sum_{k=0}^{\ell-m} (1 - \frac{1}{2^k})^{n_\mathcal{X}} \left((1 - \frac{1}{2^k})^{n_\mathcal{Y}} - (1 - \frac{1}{2^{k-1}})^{n_\mathcal{Y}} \right) \end{aligned} \tag{6}$$

For example, by Equation 6 when $|X| = |Y| = 10^4$, $n_\mathcal{T} = 0$, $\Phi(X,Y)$ is 0 the $pr(V^X \leq V^Y) \approx 0.58$ (Figure 2).

Case2: If $Y \subset X$, then $\mathcal{T} = Y$ and $\mathcal{Y}$ is empty ($n_\mathcal{Y} = 0$). Similar to case 1, if $V^\mathcal{T} = k$, then $V^\mathcal{X}$ should be at most k, where k can be any value in $[0, \ell - m]$. So $pr(V^X \leq V^Y)$ is as follows:

$$\begin{aligned} pr(V^X \leq V^Y) &= \sum_{k=0}^{\ell-m} pr(V^\mathcal{X} \leq k) \wedge pr(V^\mathcal{T} = k) \\ &= \sum_{k=0}^{\ell-m} (1 - \frac{1}{2^k})^{n_\mathcal{X}} \left((1 - \frac{1}{2^k})^{n_\mathcal{T}} - (1 - \frac{1}{2^{k-1}})^{n_\mathcal{T}} \right) \end{aligned} \tag{7}$$

For example, when $|X| = |Y| = 10^4$, and $n_\mathcal{T} = 10^4$ the $\Phi(X,Y)$ is 1 and the $pr(V^X \leq V^Y) \approx 1$ (Figure 2).

Case3: Finally, if X and Y are partially overlapped, as shown in Figure 3, given k there three scenarios 1) $V^\mathcal{X} \leq k, V^\mathcal{T} \leq k - 1, V^\mathcal{Y} = k$, 2) $V^\mathcal{X} \leq k, V^\mathcal{T} = k, V^\mathcal{Y} \leq k - 1$, and , and 3) $V^\mathcal{X} \leq k, V^\mathcal{T} = k, V^\mathcal{Y} = k - 1$. Thus the $pr(V^X \leq V^Y)$ can be derived by Equation 8.

$$\begin{aligned} pr(V^X \leq V^Y) &= \sum_{k=0}^{\ell-m} pr(V^\mathcal{X} \leq k) \wedge pr(V^\mathcal{T} \leq k-1) \wedge pr(V^\mathcal{Y} = k) \\ &\quad + pr(V^\mathcal{X} \leq k) \wedge pr(V^\mathcal{T} = k) \wedge pr(V^\mathcal{Y} \leq k-1) \\ &\quad + pr(V^\mathcal{X} \leq k) \wedge pr(V^\mathcal{T} = k) \wedge pr(V^\mathcal{Y} = k) \\ &= \sum_{k=0}^{\ell-m} (1 - \frac{1}{2^k})^{n_\mathcal{X}} (1 - \frac{1}{2^{k-1}})^{n_\mathcal{T}} \left((1 - \frac{1}{2^k})^{n_\mathcal{Y}} - (1 - \frac{1}{2^{k-1}})^{n_\mathcal{Y}} \right) \\ &\quad + (1 - \frac{1}{2^k})^{n_\mathcal{X}} \left((1 - \frac{1}{2^k})^{n_\mathcal{T}} - (1 - \frac{1}{2^{k-1}})^{n_\mathcal{T}} \right) (1 - \frac{1}{2^{k-1}})^{n_\mathcal{Y}} \\ &\quad + (1 - \frac{1}{2^k})^{n_\mathcal{X}} \left((1 - \frac{1}{2^k})^{n_\mathcal{T}} - (1 - \frac{1}{2^{k-1}})^{n_\mathcal{T}} \right) \\ &\qquad \left((1 - \frac{1}{2^k})^{n_\mathcal{Y}} - (1 - \frac{1}{2^{k-1}})^{n_\mathcal{Y}} \right) \end{aligned} \tag{8}$$

For example, when $|X| = |Y| = 10^4$, and $n_\mathcal{T} = 6200$ the $\Phi(X,Y)$ is 0.62 and the $pr(V^X \leq V^Y) \approx 0.8$ (Figure 2).

Multiple buckets: When there are 2^m buckets ($m > 0$), inspired by stochastic averaging [15] we consider the average case where the HLL value of column X in Definition 2 will be the maximum over on average $\frac{n_X}{2^m}$ independent random variables for each bucket i and Equations 4 and 5 are updated as:

$$pr(V_i^X \leq k) = (1 - \frac{1}{2^k})^{\frac{n_X}{2^m}} \tag{9}$$

$$pr(V_i^X = k) = (1 - \frac{1}{2^k})^{\frac{n_X}{2^m}} - (1 - \frac{1}{2^{k-1}})^{\frac{n_X}{2^m}} \tag{10}$$

Later in §3.3, we discuss how considering the average case affects the number of buckets.

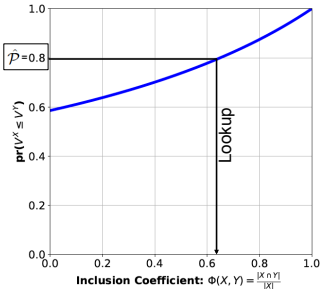

Figure 2: $|X| = |Y| = 10^4$**, Lookup approach**

	$V^{\mathcal{X}}$	$V^{\mathcal{T}}$	$V^{\mathcal{Y}}$
	V^X=max($V^{\mathcal{X}}$, $V^{\mathcal{T}}$)		V^Y=max($V^{\mathcal{Y}}$, $V^{\mathcal{T}}$)
(1) $\mathcal{X}$ $\mathcal{Y}$	$\leq k$	Empty	$= k$
(2) $\mathcal{X}$ $\mathcal{T}$	$\leq k$	$= k$	Empty
(3) $\mathcal{X}$ $\mathcal{T}$ $\mathcal{Y}$	$\leq k$	$\leq k-1$	$= k$
		$= k$	$\leq k-1$
		$= k$	$= k$

Figure 3: For columns X and Y ($|X| \geq |Y|$) cases are (1) disjoint, (2) overlapped and (3) partially overlapped, where $\mathcal{T} = X \cap Y$, $\mathcal{X} = X \setminus \mathcal{T}$, $\mathcal{Y} = Y \setminus \mathcal{T}$

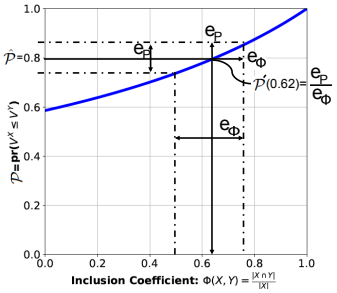

Figure 4: e_p and e_Φ are the error of estimating $pr(V^X \leq V^Y)$ and inclusion coefficient $\Phi(X,Y)$.

3.2.2 *Efficient Algorithm to Maximize the Likelihood*

Recall from Equation (3) that we formulate the problem of estimating inclusion coefficient as a maximum likelihood estimation problem, where we observe the number of buckets (among all 2^m buckets) where $V_i^X \leq V_i^Y$ is z and the goal is to choose $\Phi(X,Y)$ to maximize the likelihood of our observation. Here we propose our estimator, BML, to efficiently solve the MLE. BML has two main steps. Step one is to estimate $pr(V^X \leq V^Y)$ using only $\mathcal{S}_X$ and $\mathcal{S}_Y$. Step two is to use lookup approach to map the estimated probability into the inclusion coefficient.

Before describing the details of these two steps, we first provide an overview of how BML works. As shown in Figure 2, $pr(V^X \leq V^Y)$ increases by increasing the $\Phi(X,Y)$ (proof in theorem 2). Clearly, if BML estimates $pr(V^X \leq V^Y)$, it can be used to find the $\Phi(X,Y)$. For example, let us assume the estimated value of $pr(V^X \leq V^Y)$ be 0.8 ($\hat{\mathcal{P}} = 0.8$). As shown in Figure 2, after projection to blue line (§3.2.1) BML estimates $\Phi(X,Y)$ to be $\approx$ 0.62. Algorithm 1 shows the detail of the two main steps of BML. The input to this algorithm is the HLL sketches of column X and Y ($\mathcal{S}_X, \mathcal{S}_Y$). In §3.3, we discuss how to choose number of buckets for these sketches.

In the first step, in order to calculate $\hat{\mathcal{P}}$, since the event $V_i^X \leq V_i^Y$ in each bucket b_i is an independent Bernoulli trial (§3.1). Thus $\hat{\mathcal{P}}$ is the ratio of the number of buckets where $V_i^X \leq V_i^Y$ to the total number of buckets (2^m) (lines 3-7). Later in Theorem 4, we use Hoeffding inequality to provide the error bound of the $\hat{\mathcal{P}}$.

Algorithm 1: BML// inclusion coefficient estimator

1: **Input:** $\mathcal{S}_X, \mathcal{S}_Y$
2: //Step 1: Estimate $pr(V^X \leq V^Y)$ using $\mathcal{S}_X, \mathcal{S}_Y$
3: $z = 0$
4: **for** b_i, where $i \in [1, 2^m]$
5: **if** $V_i^X \leq V_i^Y$ then $z = z + 1$
6: $\hat{\mathcal{P}} = \frac{z}{2^m}$
7: //Step 2: Lookup step
8: n_X = Estimated number of distinct values from $\mathcal{S}_X$
9: n_Y = Estimated number of distinct values from $\mathcal{S}_Y$
10: **return** $\hat{\Phi}(X,Y) =$ **Lookup**$(\hat{\mathcal{P}}, 0, \min(n_X, n_Y), n_X, n_Y)$

In the second step, BML (Algorithm 1) calls *Lookup* function (Algorithm 2) to estimate inclusion coefficient $\Phi(X,Y)$ as the one that produces the $\hat{\mathcal{P}}$. The key idea is that since $pr(V^X \leq V^Y)$ in an increasing function of $\Phi(X,Y)$ (Theorem 2), given an estimated probability $\hat{\mathcal{P}}$ we can use binary search to estimate $\Phi(X,Y)$. Algorithm 2 shows the pseudo code of the lookup approach. The input to this algorithm are $\hat{\mathcal{P}}$, minInc, maxInc, n_X, n_Y, and ϵ where $\hat{\mathcal{P}}$ is the estimation of the $pr(V^X \leq V^Y)$, minInc and maxInc are the boundary of the search, n_X, n_Y are the estimated cardinality of the X and Y ([14]), and ϵ is the error or tolerance. Algorithm 2 is doing binary search over the possible intersection size $n_\mathcal{T}$ and at each iteration based on the value of $n_\mathcal{T}$ depends on which case it is, it uses the suitable Equation from §3.2.1 (lines 5-8). For example, if $X \cap Y = \emptyset$, it uses Equation 6. Such simple bisection procedure for iteratively converging on a solution which is known to lie inside some interval $[a,b]$ has been introduced in [8] for root finding. They showed the number of iterations required to obtain an error smaller than ϵ is $\frac{\ln(a-b)-\ln\epsilon}{\ln 2}$. In our problem since $0 \leq \Phi(X,Y) \leq 1$, the number of iteration would be $\frac{-\ln\epsilon}{\ln 2}$. For example, Algorithm 2 only needs 17 iterations to obtain an error smaller than $\epsilon = 10^{-5}$. It worth mentioning that the cost of each iteration is very cheap (analysis in §3.2.3).

Algorithm 2: Lookup // Map $\hat{\mathcal{P}}$ into $\hat{\Phi}(X,Y)$

1: **Input:** $\hat{\mathcal{P}}$, minInc, maxInc, n_X, n_Y, ϵ
2: **Outputs:** $\hat{\Phi}(X,Y)$
3: $n_\mathcal{T} = \frac{minInc+maxInc}{2}$; $\hat{\Phi} = \frac{n_\mathcal{T}}{n_X}$
4: **if** ($n_\mathcal{T} = n_X$ & $n_\mathcal{T} = n_Y$) **then** Prob = 1.0 //$X = Y$
5: **else if** ($n_\mathcal{T} = 0$) **then** Prob = Equation 6 //$X \cap Y = \emptyset$
6: **else if** ($n_X > n_Y$ & $n_\mathcal{T} = n_Y$) **then** Prob = Equation 7
7: **else** $Prob$ = Equation 8
8: **if** $|Prob - \hat{\mathcal{P}}| \leq \epsilon$ **return** $\hat{\Phi}$
9: **if** $Prob > \hat{\mathcal{P}}$ **return Lookup**$(\hat{\mathcal{P}}, n_\mathcal{T}, maxInc, n_X, n_Y)$
10: **if** $Prob < \hat{\mathcal{P}}$ **return Lookup**$(\hat{\mathcal{P}}, minInc, n_\mathcal{T}, n_X, n_Y)$

3.2.3 *Algorithm Correctness and Error Analysis*

We provide the following analysis to show the correctness, efficiency, and the error bound of BML.

- We prove $pr(V^X \leq V^Y)$ is an increasing function of $\Phi(X,Y)$, i.e., in lookup step there is a one to one mapping between the probability $pr(V^X \leq V^Y)$ and the inclusion coefficient.
- In (3), we formulated the problem of estimating inclusion coefficient as maximum likelihood problem. We prove that the results of BML and the MLE formulation (3) are identical.
- We also provide the error bound of the BML for estimation of the $pr(V^X \leq V^Y)$ and inclusion coefficient.
- Finally, we show the time complexity of the BML Algorithm.

Algorithm Correctness: BML uses binary search in order to find the mapping between the probability $\hat{\mathcal{P}}$ and the inclusion coefficient (Algorithm 2). This approach only works if there is a one to one mapping from $pr(V^X \leq V^Y)$ into $\Phi(X,Y)$. Theorem 2 shows that probability $pr(V^X \leq V^Y)$ (§3.2.1) is an increasing function of $\Phi(X,Y)$. Clearly an increasing function is a one to one function, hence is invertible.

THEOREM 2. *Given two columns X, and Y with intersection $\mathcal{T}$, where $|\mathcal{T}| = n_\tau$, probability $pr(V^X \leq V^Y)$ is an increasing function of $\Phi(X, Y)$.*

PROOF. Please find the proof in Appendix B. □

In Theorem 3 we prove that the results of Algorithm 1 and the MLE formulation (3) are identical.

THEOREM 3. *The inclusion coefficient estimate $\hat{\Phi}$ from Algorithm 1 and the inclusion coefficient estimate Φ_{mle} from MLE formulation* (3) *are identical.*

PROOF. Please find the proof in Appendix C. □

Error Bound of the Probability Estimation: Let $\mathcal{P}$ and $\hat{\mathcal{P}}$ be the exact and estimated value of the $pr(V^X \leq V^Y)$ respectively. BML (Algorithm 1) calculates $\hat{\mathcal{P}}$ as the ratio of the number of buckets where $V_i^X \leq V_i^Y$ to the total number of buckets 2^m. As shown in Figure 4, this estimation may produce error e_p, e.g., $\hat{\mathcal{P}} = 0.8$ while $\mathcal{P} = 0.8 \pm e_p$. This figure also show that when $\hat{\mathcal{P}} = 0.8$, the lookup step in BML returns 0.62 as the estimated inclusion coefficient while the actual one is $0.62 \pm e_\Phi$ (Figure 4). Thus, estimation error e_p result in e_Φ, i.e., estimation error of the inclusion coefficient $\Phi(X, Y)$. Here we first show that given an error e_p ($0 \leq e_p \leq 1$), what is the probability that the estimation error of Algorithm 1 be at most e_p. Next we show how to use the e_p in order to bound estimation error of the inclusion coefficient.

THEOREM 4. *Probability that estimation error of $\hat{\mathcal{P}}$ be at most e_p is: $pr\left(|\hat{\mathcal{P}} - \mathcal{P}| \leq e_p\right) \geq 1 - 2\exp\left(-2^{m+1}e_p^2\right)$*

PROOF. Please find the proof in Appendix D. □

For example, when $m = 7$ and estimation error e_p is 0.014, the $pr(|\hat{\mathcal{P}} - \mathcal{P}| \leq e_p)$ is at least 0.95, i.e., with 95% confidence the estimation error is at most 1.4%.

Error Bound of the Inclusion Coefficient Estimation: From Figure 4 one can see that the slope of $\mathcal{P}$ (the blue line) at any point represent the ratio of e_p to e_Φ, e.g., when $\hat{\mathcal{P}} = 0.8$, the estimated inclusion coefficient is 0.62 and the slope of $\mathcal{P}$ at 0.62, $\mathcal{P}'(0.62)$, is $\frac{e_p}{e_\Phi}$. One can numerically find this slope of $\mathcal{P}$ for a given a point α, e.g., $\mathcal{P}'(0.62) = 0.7$. Moreover, as we discussed in theorems 4, with 95% confidence the estimation error e_p is 0.014. Thus e_Φ at this point can be calculated as $\frac{0.014}{0.7} = 0.02$. More formally at any point α we have:

$$e_\Phi = \frac{e_p}{\mathcal{P}'(\alpha)} \tag{11}$$

So far we have shown how to find e_Φ for a given point. Clearly in Figure 4 the slope of $\mathcal{P}$ at different points are different. Thus in order to bound e_Φ of two columns X and Y the key observation is to first find the minimum and maximum slopes of $\mathcal{P}$ and then the ratio of e_p to those slopes will be the bound of e_Φ. Table 2 shows the minimum and maximum slopes of the $\mathcal{P}$ and the e_p bound for columns X and Y with different cardinalities. For example, when the cardinality of both X and Y is 10^4, the minimum and maximum slopes of $\mathcal{P}$, in Figure 4 are 0.241 and 0.719 and the minimum and maximum e_Φ are $\frac{0.014}{0.719} = 0.02$ and $\frac{0.014}{0.241} = 0.05$, where the e_p is 0.014. Figures 6 to 8 shows how the slope of $\mathcal{P}$ varies for the columns with different cardinalities. One can see that the slope of $\mathcal{P}$ reduces as the difference of the cardinalities of the X and Y increases e.g., when $|X| = 1000, |Y| = 10^4$ the minimum and maximum slopes of $\mathcal{P}$ are 0.062 and 0.072 respectively. Clearly, when the slope of $\mathcal{P}$ reduces, e_Φ will increase (Table 2).

Time complexity of the BML Algorithm: In BML Algorithm 1, the estimation of $pr(V^X \leq V^Y)$ using $\mathcal{S}_X, \mathcal{S}_Y$ takes 2^m steps (lines 4-6), where 2^m is the number of buckets. As we discussed in §3.2.2 the binary search in the lookup step takes $\frac{-\ln \epsilon}{\ln 2}$ iterations to obtain an error smaller than ϵ. The cost of each iteration is in $\mathcal{O}(\ell - m)$ due to the fact that the Equations 6, 7, 8 used in Algorithm 2 (lines 6-8) is the sum of $\ell - m$ values. Thus, the time complexity of BML Algorithm is $\mathcal{O}(2^m + \frac{-\ln \epsilon}{\ln 2}(\ell - m))$ which is linear in the number of buckets.

Table 2: The minimum and maximum e_Φ for synthetic data

		$\Phi(X,Y)$			
$\|X\|$	$\|Y\|$	$\min(\frac{e_p}{e_\Phi})$	$\max(\frac{e_p}{e_\Phi})$	$\max(e_\Phi)$	$\min(e_\Phi)$
10000	10000	0.241	0.719	0.05	0.02
9000	10000	0.235	0.646	0.06	0.02
7000	10000	0.219	0.503	0.06	0.03
5000	10000	0.192	0.359	0.07	0.04
3000	10000	0.145	0.216	0.09	0.06
1000	10000	0.062	0.072	0.23	0.2

3.3 Choosing Number of Buckets

Algorithm 3 (Appendix A) shows the steps to construct a HLL sketch for a fixed m. In this section, we first discuss given two columns X, and Y how to choose the parameter m (number of bits for the buckets). Next, we show how this algorithm changes when we consider all pair of columns since parameter m needs to be varied.

Parameter m for a given column pair: The HLL construction of column X can be viewed as bins and balls problem [2], where buckets are the bins and distinct values in column X are balls. As we discussed in §3.2.1, given column X with n_X distinct values, when there is only one bucket the HLL value of column X is in fact the maximum over n_X independent random variables (Definition 2). When there are 2^m buckets ($m > 0$), the HLL value of column X will be the maximum over, on average, $\frac{n_X}{2^m}$ independent random variables for each bucket, i.e, *balanced load* in each bucket. In bins and balls problem [2], it is shown that as the number of bins increases the probability of balanced load will decrease. In other words, given n_X balls and 2^m bins the probability that all bins contains exactly $\frac{n_X}{2^m}$ balls reduces as the number of bins increases. Thus in HLL construction of column X, we should also expect that as the number of buckets increases the probability that all buckets have the same load decreases. However, it is also shown in [15, 14] that having large number of buckets reduces the *variance* of cardinality estimation using a HLL sketch.

Thus in one hand, having less number of buckets increases the probability of *balanced load* ($\frac{n_X}{2^m}$), but in the other hand more number of buckets reduces the *variance* of estimation. For the perfect *balanced load*, there should be only one bucket, and for the lowest *variance* number of buckets should be n_X ($m = \log(n_X)$). Given two columns X, and Y, in this paper, as a trade-off between the *variance* and the *balanced load* we heuristically pick a mid point between the best variance and the best balanced load by considering the number of buckets as equation 12. Note to say that it is a heuristic and it is not the optimal choice.

$$m = \frac{\log(max(n_X, n_Y))}{2} \tag{12}$$

Parameter m for multiple pairs of columns: Our goal is to efficiently estimate inclusion coefficient for all column pairs in a database. When we consider multiple pairs of columns Equation 12 might returns different m for a given column. For example, consider three columns X, Y, and Z with cardinality n_x, n_Y, and n_z where cardinality of X is smaller than Y and Z ($n_X < n_Y$, $n_X < n_Z$) and cardinality of Y and Z are not equal ($n_Y \neq n_Z$). In this example, for column pair X and Y, parameter m is $\frac{\log(n_Y)}{2}$,

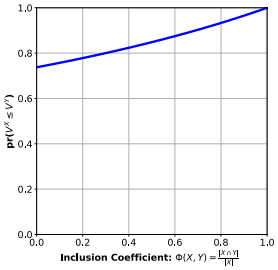

Figure 5: $|X| = 5000, |Y| = 10^4$

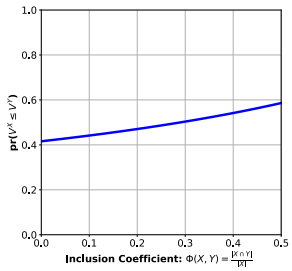

Figure 6: $|X| = 10^4, |Y| = 5000$

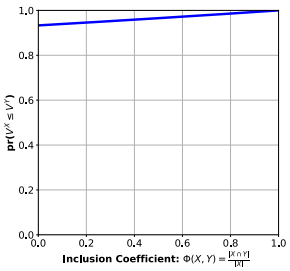

Figure 7: $|X| = 10^3, |Y| = 10^4$

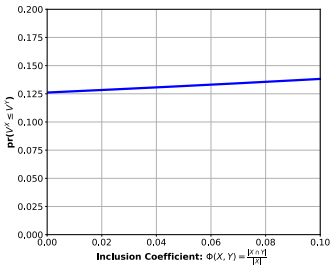

Figure 8: $|X| = 10^4, |Y| = 10^3$

while for column pair X and Z parameter m is $\frac{\log(n_Z)}{2}$. Thus, for column X the m has two values $\frac{\log(n_Y)}{2}$ and $\frac{\log(n_Z)}{2}$.

To address this problem, we change Algorithm 3 such that it still reads data only once but it keeps all the sketches for different $m \in \{0, \cdots, l\}$ (value of k can be determined by Theorem 5). More specifically, after reading the data in line 4, we iterate over different values of m from 0 to l and the rest of the algorithm is the same. Observe that even if we keep all sketches for m from 0 to l, the memory required only doubles because $\sum_{m=0}^{k} 2^m \log(\ell) = 2^{l+1} \log(\ell)$, where $2^m \log(\ell)$ is the size of sketch for each column (§2.2). After constructing the sketches, given two columns X and Y, we decide which m produces better estimation of inclusion coefficient (Equation 12) and we pass those sketches to Algorithm 1.

Finally, we discuss how to set the parameter l. Suppose the memory bound for each column is $\mathcal{M}$. In Theorem 5, we prove that given column X, bounded memory $\mathcal{M}$, and ℓ bits hash function h, the parameter l should be at most $\ln(\frac{\mathcal{M}}{\log(\ell)}) - 1$.

THEOREM 5. *Given column X, bounded memory $\mathcal{M}$, and ℓ bits hash function h, $l \leq \ln(\frac{\mathcal{M}}{\log(\ell)}) - 1$.*

PROOF. We hash each value $X[i] \in X$ only once but we generate $l + 1$ sketches h for each $m \in \{0, \cdots, l\}$. Since we only keep the maximum ρ for each bucket b_j, the amount of memory for each bucket is $\log(\ell)$. Each sketch has 2^m buckets so in total it requires $\sum_{i=0}^{l} 2^i \times \log(\ell) = 2^{l+1} \times \log(\ell)$ bits. Since $\mathcal{M}$ is the bounded memory, 2^{l+1} should be at most $\frac{\mathcal{M}}{\log(\ell)}$,i.e. $(2^{l+1} \leq \frac{\mathcal{M}}{\log(\ell)})$ which gives us $l \leq \ln(\frac{\mathcal{M}}{\log(\ell)}) - 1$. □

3.4 Discussion: Leveraging more memory

We briefly discuss how BML can leverage additional memory, when available, to improve its accuracy. It is shown in [14] that increasing the number of buckets for HLL sketches reduces the *variance* of cardinality estimation. In other words, the bucketization (with stochastic averaging) emulates the effect of n hash functions with only one single hash function [15]. However, as discussed in §3.3, increasing the number of buckets reduces the probability of *balanced load* ($\frac{n_X}{2^m}$) which can ultimately reduce the accuracy BML. Thus, leveraging additional memory by increasing the number of buckets is not adequate for our problem.

One approach is to combine the use of multiple hash functions and stochastic averaging in order to take advantage of additional memory. In other words, given two columns X, and Y we fix number of buckets to 2^m, where m can be find by Equation 12. Then we use multiple hash functions in HLL construction. BML uses the sketches built by each hash function in order to estimate inclusion coefficient (Algorithm 1), and the final estimation of inclusion coefficient is the average of those results. A thorough empirical study of the trade-offs between estimation error and the above method of leveraging additional memory is an interesting area of future work.

4. EXTENSION OF HLL CONSTRUCTION TO SUPPORT DELETION

HLL sketches are becoming increasingly popular for estimating cardinality in different applications, and are even being adopted in commercial database engines, e.g., [16] uses Hyperloglog to support approximate distinct count functionality. However, in data warehousing scenarios, it is not uncommon for recent data to be added and older data to be removed from the database. Clearly HLL sketches can be maintained incrementally in the presence of insertions. When a new data item $X[k]$ inserted to column X, the same Algorithm 3 finds the hash value of the $X[k]$ ($s_k = h(X[k])$) and the affected bucket b_j can be identified by the leftmost m bits of s_k. It then finds $\rho(s_k)$ which is the position of the leftmost 1 in the $l - m$ bits of s_k and it updates the value of bucket j as $V_j^X = \max(\mathcal{S}_X[b_j], \rho(s_k))$ (Definition 2). For example, Figure 1 shows the HLL sketch of the column X, i.e., $\mathcal{S}_X$. let us assume a new value $X[k]$ is added to column X such that the first m bits of the $s_k = h(X[k])$ represents the first bucket (b_1) in $\mathcal{S}_X$ and $\rho(s_k)$ be 5. Thus the value of the b_1 will be updated to $\max(4, 5) = 5$. On the other hand, when $X[i]$ is deleted, similar to insertion we can find which bucket is affected. Lets b_j be the affected bucket. Since $X[i]$ exists in database, $\rho(s_i) \leq \mathcal{S}_X[b_j]$. If $\rho(s_i) < \mathcal{S}_X[b_j]$, no update is required but if $\rho(s_i) = \mathcal{S}_X[b_j]$, it means $\rho(s_i)$ is the largest and should be deleted. Since we do not know what the second largest value for that bucket, we cannot handle deletion.

We can modify Algorithm 3 such that for each bucket we keep track of all $\rho(s_i)$s in order to support deletion. Since V_j^X is the max over all those value (Definition 2), and when a deletion happens knowing the second largest for each bucket is crucial, as shown in Figure 9, rather than only keeping the maximum value in each bucket (Figure 1), we keep all $\rho(s_i)$s in a max-heap. Interestingly, we can show that by maintaining a heap of constant size (at most ℓ) for each bucket we are able to support incremental deletion. Recall that during HLL sketch construction, we apply a hash function $h : \mathsf{dom}(X) \rightarrow \{0,1\}^\ell$ on each $X[i] \in \mathsf{dom}(X)$ which returns ℓ bits s_i, and if m is the number of bits for the buckets, $\rho(s_i)$ is always an integer number smaller than equal $\ell - m$ ($1 \leq \rho(s_i) \leq \ell - m$). For example, if it uses 64 bits hash function and $m = 0$, then $1 \leq \rho(s_i) \leq 64$. In the worst case, if we keep all distinct $\rho(s_i)$s, we are required to keep only $\ell - m$ values, which requires $(\ell - m)\log(\ell - m)$ bits. This explains why with only a heap of constant size (at most $\ell - m$) incremental deletion can be supported.

One issue we need to consider is that it is possible that $X[i]$ and $X[k]$ are assigned to same bucket b_j and $\rho(s_i)$ is equal to $\rho(s_j)$. In this case, if $V_j^X = \rho(s_i)$ and $X[i]$ is deleted, the value of bucket b_j (V_j^X) should still be $\rho(s_i)$ because $X[k] \in \mathsf{dom}(X)$ still exists and $\rho(s_j) = \rho(s_i)$. To handle this scenario and keep the max heap size still limited to $\ell - m$, we keep a *counter* for each node in heap. For

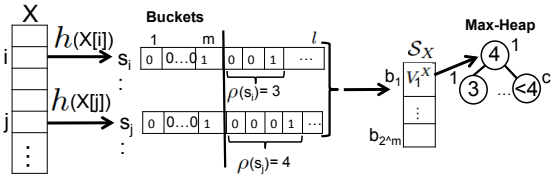

Figure 9: HLL sketch of the column X for deletion support.

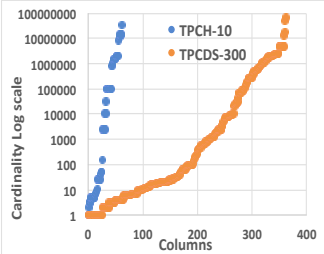

Figure 10: Cardinality of the columns in TPCH-10 and TPCDS-300

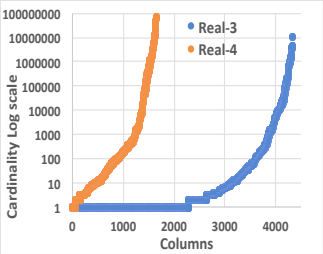

Figure 11: Cardinality of the columns in Real-3 and Real-4

example in Figure 9, since there is only one X[i] that its $\rho(s_i) = 4$, value 1 is attached to node 4. So, a node in heap is deleted if the counter is one; otherwise we just reduce the counter by one.

Space complexity: For each bucket b_i the heap size is at most $\ell - m$ and each value in the heap is an integer number at most $\ell - m$. Thus each bucket only needs $(\ell - m)\log(\ell - m)$ bits. Thus total memory to store the sketch $\mathcal{S}_X$ is $\mathcal{O}(2^m(\ell - m)\log(\ell - m))$. Since the space complexity of the original HLL is $\mathcal{O}(2^m \log(\ell - m))$ (§2.2), with a constant overhead we are able to support deletion.

5. EXPERIMENTS

The main goals of our experimental evaluation are: (a) Evaluate the accuracy of the BML estimator for inclusion coefficient against the existing approach using Bottom-k sketches [31]. (b) Evaluate the impact of inclusion coefficient estimates instead of exact inclusion coefficient on precision and recall of two existing foreign detection techniques [11, 29]. The key takeaways from the experiments are:

- On databases containing columns with large cardinality BML results in substantially smaller estimation error (ranging from 0.1 to 0.15) compared to the Bottom-k sketch approach (where errors range from 0.3 to 0.59).
- As expected, Bottom-k performs well when the cardinality of the columns are small (less than k). In databases where most columns have cardinality smaller than k, the estimation error of the BML range between 0.04 and 0.07 and the error of Bottom-k ranges between 0.03 and 0.06.
- The running time of the BML estimator is comparable to the approach that uses Bottom-k sketches.
- When we use BML instead of the exact inclusion coefficients, it has only a small impact on the F1 measure of the two foreign-key algorithms. In contrast, using the estimator based on Bottom-k sketches significantly reduces F1 measure in some databases.

5.1 Setup

Hardware and Platform: We implemented algorithm for constructing the sketches on Microsoft's big data system Cosmos [10]. Cosmos is designed to run on large clusters consisting of thousands of commodity servers and it is based on map-reduce model to achieve parallelism. The rest of our experiments, including the estimation of inclusion coefficient between pairs of columns using sketches, were performed on a 6-core, 2.2 GHz, Intel Xeon E-5-2620 machine with 384 GB of RAM.

Datasets: Table 3 shows the details of the datasets we used in our experiments. Specifically, we used TPC-H with 10GB scaling factor and TPCDS with 300GB, 500GB, and 2000GB(2TB) scaling factors as benchmark databases. We also evaluated our technique on four real-world databases, Real-1 and Real-2 are publicly available in [23] and Real-3 and Real-4 are real-world customer databases. Figures 10 and 11 show the distribution over the cardinality of the columns over the synthetic (TPCH-10, TPCDS-300) and the largest real databases (Real-3, Real-4). As these figures show, in TPCDS-300, Real-3 and Real-4 there are many columns with small and large cardinality.

Table 3: Databases used in experimental evaluation.

Database	#of tables	# of columns	# of FKs	Size
TPCH-10	9	61	9	10GB
TPCDS-300	25	362	98	300GB
TPCDS-500	25	362	98	500GB
TPCDS-2000	25	362	98	2TB
Geeea(Real-1)	19	128	20	61.4MB
Mondial(Real-2)	34	167	78	3.2MB
Real-3	612	4331	unknown	80GB
Real-4	339	1648	unknown	700GB

Bottom-k: Since the inclusion coefficient estimator in [31] used Bottom-k sketches, we refer to it as Bottom-k in the rest of this section. In brief, the inclusion coefficient of X and Y ($\phi(X,Y) = \frac{|X \cap Y|}{|X|}$) can be calculated using the Jaccard coefficient, i.e., $\phi(X,Y) = \frac{\varphi(X,Y)}{\varphi(\{X \cup Y\},Y)}$. They use Bottom-k sketches to estimate Jaccard coefficient, $\varphi(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$. Another approach to estimate $\phi(X,Y)$ is to divide the estimated value of $|X \cap Y|$ [5] by the estimated value of $|X|$. The authors in [31] have shown that the Bottom-k approach is more accurate than the estimator based on the [5] so we only compare the accuracy of BML against the estimator in [31].

Setting k in Bottom-k: We used a 64-bit hash function (MurmurHash [1]) for the sketch construction of both HLL and Bottom-k. For the fair comparison we fixed the amount of memory per column used by both estimators (BML and Bottom-k). In §3.3 we discussed how to pick number of buckets m. For example, in TPCDS-300 the maximum number of buckets used by our algorithm is 2^{13}, and since we used 64-bits hash function and HLL keeps the position of the leftmost 1, so each bucket only requires $\log 64 = 6$ bits. Since we keep all the sketches for $m \in \{0, 1, ..., 13\}$, total memory is $2^{14} * 6$ bits (12.28 KB). On the other hand, Bottom-k keeps the k smallest hash values, i.e, $k * 64$ bits. Thus, to ensure equal memory, we configure the k in Bottom-k as $k = \frac{2^{14} * 6}{64} = 1536$. Table 4 shows the m and k parameters for each database.

5.2 Comparing BML and Bottom-k

5.2.1 Execution Time

Figure 12 shows the execution time of estimating $\Phi(X,Y)$ for all pair of columns belonging to different tables using BML and Bottom-k. The total execution time depends on the number of such column-pairs, and hence varies across the different databases. For example, BML computes the inclusion coefficient of the 17,635,800 column-pairs in Real-3 which takes roughly 600 seconds, i.e., 0.03 ms per column-pair. Not surprisingly, for each database, the execution times of the BML and Bottom-k are quite close to each other (BML is slightly more expensive computationally).

[1]http://code.google.com/p/smhasher/

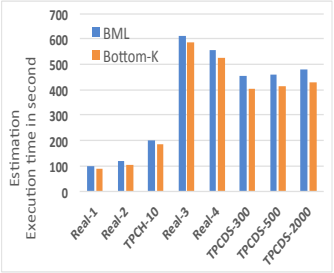

Figure 12: Execution time of estimating $\Phi(X,Y)$ for all pair of columns

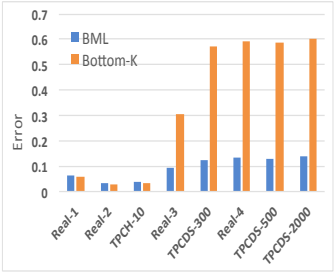

Figure 13: Average Error of estimated $\Phi(X,Y)$

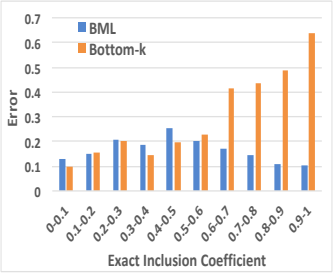

Figure 14: Average Error of estimated $\Phi(X,Y)$ in TPCDS-300: all pair of columns, k = 1536

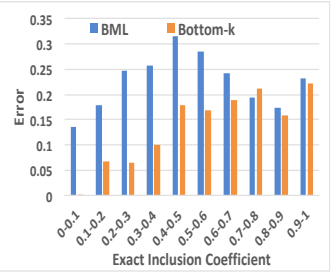

Figure 15: Average Error of estimated $\Phi(X,Y)$ in TPCDS-300: $|X| \leq t$, $|Y| \leq t$, t = 5000, k = 1536

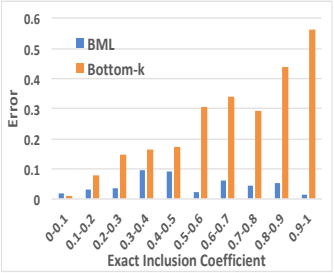

Figure 16: Average Error of estimated $\Phi(X,Y)$ in TPCDS-300: $|X| > t$, $|Y| > t$, t = 5000, k = 1536

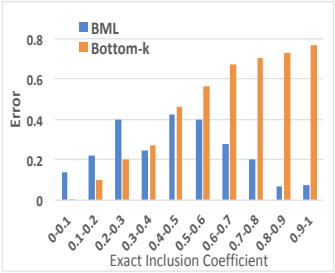

Figure 17: Average Error of estimated $\Phi(X,Y)$ in TPCDS-300: $|X| > t$, $|Y| \leq t$ or $|X| \leq t$, $|Y| > t$, t = 5000, k = 1536

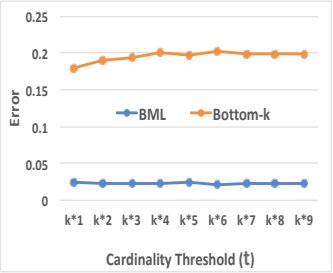

Figure 18: Average Error of estimated $\Phi(X,Y)$ in TPCDS-300: $|X| > t$, $|Y| > t$, k = 1536

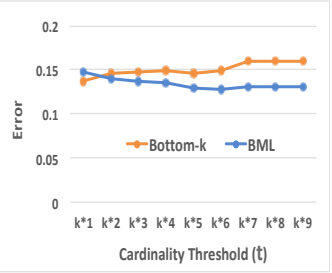

Figure 19: Average Error of estimated $\Phi(X,Y)$ in TPCDS-300: $|X| > t$, $|Y| \leq t$ or $|X| \leq t$, $|Y| > t$, k = 1536

Table 4: Parameter m is the number of bits that define a bucket in HLL, and k and number hash values kept for Bottom-k.

	TPCH-10	TPCDS-300	TPCDS-500	TPCDS-2000	Real-1	Real-2	Real-3	Real-4
m	9	13	15	18	6	5	11	16
k	96	1536	3072	24576	12	6	384	6144

5.2.2 *Estimation Error*

Next we compare the average error of the two estimators. The estimation error is the difference between the estimated inclusion coefficient $\hat{\Phi}$ and its actual value $\Phi(X,Y)$, i.e.,$|\Phi(X,Y) - \hat{\Phi}|$. Recall that both estimators rely on sketches that use an equal amount of memory (§5.1). As shown in Figure 13, the estimation error in the databases where most columns have small cardinality (Real-1, Real-2, and TPCH-10) is almost similar for the BML and Bottom-k. However, BML results in substantially smaller error for the other databases (TPCDS-300, Real-3, and Real-4) where many columns have large cardinality (significantly larger than k).

To better understand these results, Figures 14-17 breakdown the result for TPCDS-300 database. In Figure 14, the x-axis shows the buckets that each bucket (lets say $a - b$) contains those column pairs whose *exact* inclusion coefficients are in $[a, b)$, and the y-axis represents the absolute error of BML and Bottom-k for those column pairs. For example, $0.2 - 0.3$ in x-axis is the bucket contains those columns pairs whose their exact inclusion coefficients are in $[0.2,0.3)$ and the error of both BML and Bottom-k (k=1536 in Table 4) is around 0.2. The results show that in TPCDS-300 when exact inclusion coefficient is smaller than 0.5, Bottom-k is slightly better than BML, however, BML performs better when inclusion coefficient is large. In applications like foreign-key detection where the high containment (e.g., inclusion coefficient ≥ 0.8) is an important signal an estimator with low error for large coefficients is preferable. Next, we show that, the reason is the cardinality of the columns. As we discussed in §2.1.1, the error of the inclusion coefficient estimators depends on the cardinality of the columns.

Analyzing error based on cardinality threshold t: In Figures 15-17, we evaluate the error of the estimators over three different sets of column-pairs in TPCDS-300 based on a cardinality threshold $t = 5000$ (we later vary t in Figures 18, 19). In particularly, Figure 15 only considers the column-pairs (X,Y) where *both* columns have cardinality *less* than 5000. Figure 16 considers the column-pairs where both columns have cardinality *more* than 5000. Figure 17 considers pairs of columns where the cardinality of one column is smaller than 5000 and the cardinality of the other column is more than 5000. When the cardinality of the columns are small, Bottom-K performs better than BML (Figure 15). In effect, for columns with cardinality less than k, if a good hash function is used, Bottom-k keeps the hashes of all distinct values, thereby behaving similar to full hash table of the column. On the other hand, for large columns (Figure 16) and the one that one column is large and the other column is small (Figure 17), BML outperforms Bottom-k significantly.

Figures 18, and 19 show the error of BML against Bottom-K when the cardinality threshold t in above experiments is varied. We show the cardinality threshold t as a factor of k in Bottom-k. Figures 18 shows the error for those columns whose cardinalities are more than t, and Figure 19 shows the error when the cardinality of one column is smaller than t and the cardinality of the other column is more than t. Figure 18 shows that for the large columns BML significantly outperforms Bottom-K. In the unbalanced case (Figure 19), when t is exactly k, Bottom-k performs slightly better than BML but as t grows BML outperforms Bottom-K.

Effect of deletion: Next we evaluate the error and scalability of the version of BML that uses a HLL sketch that supports data deletions (discussed in §4) vs. Bottom-k . Recall, that we keep a constant size

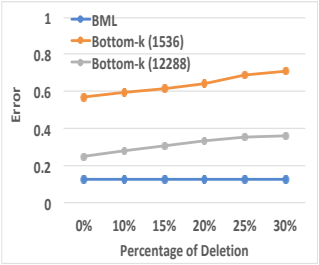

Figure 20: Deletion: Average Error of estimated Φ in TPCDS-300 when data is read once

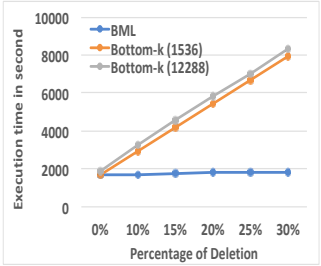

Figure 21: Deletion: Execution time in TPCDS-300 when Bottom-k reads data multiple times

heap for each HLL bucket (at most 64-m) to support deletion. Thus, for the fair comparison we again fix the memory and increased the parameter k for Bottom-k. For example, in TPCDS-300, k is increased to 12288 to ensure equal memory for both estimators. We show the results for both k=1536 and k=12288. Bottom-k can not support deletion because it only keeps the k smallest hashed values and if one of them deleted as a consequence of data deletion, it needs to access the data again to find a replacement for the deleted hash value. We consider two scenarios. In the first scenario, both algorithms only read data once and after constructing the sketches we evaluate how the error varies when deletion occurs. Note that in this scenario, if i values in the Bottom-k sketch are deleted, it effectively behaves like a Bottom-(k-i) sketch. In the second scenario, we allow Bottom-k to read the data more than once in order to support deletion and always keep k entries. Figures 20 and 21 shows the result of these two scenarios respectively. As one can see, as more data is deleted the error of the Bottom-k increases while, the error of BML remains the same. On the other hand, when Bottom-k is allowed to read the data more than once its total execution time is increased. The execution time of the BML increased only slightly because when deletion occurs it needs to modify the max-heaps in the affected buckets.

5.3 Foreign-Key detection: an application of inclusion coefficient estimation

Foreign-key relationships (FKs) in relational databases are valuable for a variety of tasks such as data integration, join path enumeration, and query optimization. Despite the importance, many application designers do not declare FKs in the database due to performance considerations and data inconsistencies. For $X \rightarrow Y$ to be a FK, Y must be unique, and X must be contained in Y. Although a strict definition of FK requires uniqueness and containment properties to hold *exactly*, in practice some tolerance is crucial to accommodate for the fact that real-world data can be dirty. Prior work, e.g. [29, 11] use inclusion coefficients to prune candidate FKs. In other words, they prune any pair of columns whose inclusion coefficients is less than a threshold. Since the computation of exact inclusion coefficients are expensive so having an estimator with fairly small error that reduces the cost is important. In this set of experiments, we empirically investigate how the precision and recall of the two FK detection algorithms [29, 11], change when we use inclusion coefficient estimates rather than the exact inclusion coefficients.

To evaluate how our estimator performs for these algorithms we have considered three versions of each algorithm in [29, 11]. Rostin-Exact, and Chen-Exact prune column pairs whose *exact* inclusion coefficients are smaller than 1.0. While Rostin-BottomK and Chen-BottomK use the inclusion coefficient estimator in [31]; Rostin-BML and Chen-BML use BML. Note that when we use estimated inclusion coefficients, we prune those columns whose estimated inclusion coefficients are smaller than 0.95 (rather than one). We focus on the databases where the true FKs are known (Table 3), so that we can compute precision and recall. Since the FKs in Real-3 and Real-4 are not known, we do not use them in our experiments.

5.3.1 Results

The FK detection algorithm in [11] ranks the column pairs by their similarity of column names and table names and it only considers those pairs whose similarity score is larger than a threshold t. It then iteratively adds the high ranked pair as FK which does not conflict with the existing FKs. To figure this threshold, we ran the Chen-Exact algorithm with different t. Figure 22, shows the precision(P), Recall(R), and F1-measure(F1) of [11] when exact inclusion coefficients are used for pruning (Chen-Exact) and threshold are $t = 0.6$, $t = 0.7$, $t = 0.8$. As one can see the accuracy of this algorithm depends on the threshold, e.g., for Real-2 when the threshold is greater than 0.6, F1-measure is zero. Moreover, we observed that the iterative approach may generate false positive result which increase the false negative. For example, at step i it may fixes a pair as FK which is false positive and in step $j > i$ there is a pair which should be FK but the algorithm prunes it because it conflicts with the one it fixed as FK in step i. For the rest of experiments for [11] we set $t = 0.6$, since based on Figure 22 it performs better in almost all of our datasets. Moreover, [29] used state of the art classifiers like naive bayes classifier, SVM, and decision tree for FK detection problem. Here we show the results for the SVM.

Figures 23 and 24 show the result of the Chen-Exact and Rostin-Exact against the algorithms with estimated inclusion coefficients, i.e., Chen-BottomK, Chen-BML (Figures 23) and Rostin-BottomK, Rostin-BML (Figures 24). The accuracy of both Chen-BottomK and Chen-BML for the TPCH-10, Real-1, and Real-2 are almost similar to Chen-Exact. Similarly for those datasets the accuracy of Rostin-BottomK and Rostin-BML are almost similar to Rostin-Exact (Figures 24). The reason is that, as shown in Figure 13, for these small datasets the error of both Bottom-k and BML are small. However, in TPCDS-300, (see Figures 23 and 24) Chen-BML performs better than Chen-BottomK and Rostin-BML performs better than Rostin-BottomK. The reason is that, as shown in Figure 14, the error of the Bottom-k approach is high when the exact inclusion coefficient is greater than 0.9 and these algorithms prune those columns that their inclusion coefficients are smaller than 0.95. So Chen-BottomK prunes those column-pairs whose actual inclusion coefficients are high; which lead to false negatives and thereby reduces recall of FK detection. Conversely, there are some column-pairs whose exact inclusion coefficients are in fact small but due to overestimation by Bottom-k, they are not pruned, which leads to false positives and reduces precision of FK detection.

In summary, our experimental results show that when we use our inclusion coefficient estimates the accuracy of the [29, 11] is almost identical to when the *exact* inclusion coefficients are used. Moreover, for the large databases, if we use the existing inclusion coefficient estimator (Bottom-k) proposed in [31], the precision and recall are significantly reduced.

6. RELATED WORK

Inclusion Dependency Discovery: In this paper, we propose an efficient general approach to estimate inclusion coefficient. Inclusion dependency (IND) is an special case where the inclusion coefficient is exactly one. Finding all INDs in a dataset has been extensively studied in the literature [17, 28, 4, 22, 21, 24]. The authors in [22]

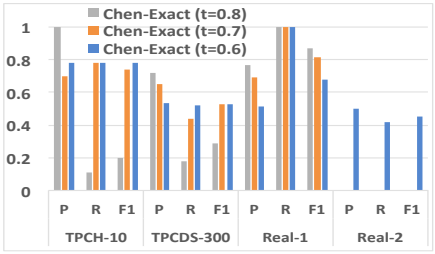

Figure 22: Result of Chen-Exact with threshold of 0.8, 0.7 and 0.6.

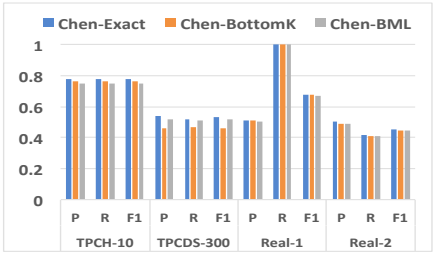

Figure 23: Result of Chen-Exact, Chen-BottomK, and Chen-BML with threshold of 0.6

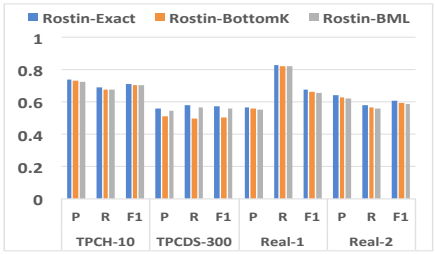

Figure 24: Result of Rostin-Exact, Rostin-BottomK, and Rostin-BML using SVM.

propose an algorithm based on association rule mining and clique-finding to discover INDs. SPIDER [4] needs two passes over the data, i.e., one pass for the sort and second pass for the containment check. It has large sorting overhead and if a column does not fit into main memory, external sorting is required. BINDER on the other hand uses divide and conquer approach, which allows to handle large datasets [28]. SPIDER and BINDER requires to copy all or part of the data to the disk which has I/O overhead and they pass over the data more than once. An approximate algorithm for the n-ary IND discovery problem studied in [17]. Note that partial inclusion is not addressed in these works. Partial inclusion dependency is formally defined in [21]. They reduce the search space to only those columns in query workloads. However, [21] uses full join to find inclusion coefficients, which can be expensive. Conditional dependencies have been studied in [3, 24, 1]. A conditional dependency is a dependency with a limited scope defined by conditions over one or more attributes. Our proposed BML estimator is more general than IND discovery algorithms [17, 28, 22, 21] because for every pair of columns it returns an estimate of their *inclusion coefficient*. In order to find exact INDs, BML can be used to prune those columns whose inclusion coefficients are smaller than a threshold and a IND discovery algorithms is required to validate whether the exact containment requirement is satisfied or not. We used Metanome [27] to evaluate the memory consumption and execution time of the BINDER vs. BML (see Appendix E for results).

Inclusion Coefficient Estimation: There has been prior work on estimating the inclusion coefficient efficiently using the Jaccard coefficient or set intersection estimators. Jaccard coefficient $\varphi(X, Y)$ has been used for inclusion coefficient estimation, i.e., $\phi(X, Y) = \frac{\varphi(X,Y)}{\varphi(X \cup Y, X)}$ [31]. There has been a considerable amount of work for estimating the Jaccard coefficients between the two sets [7, 20, 19, 30, 26, 12]. The authors in [31] use the Jacard coefficient estimator based on Bottom-k sketches [12] for the inclusion coefficient estimation. Bottom-k sketch uses one hash function and it keeps the k smallest hash values. Morover, [31] uses set intersection estimation in [5] for inclusion coefficient estimation as well. There exist several prior work in set intersection problem over large datasets where the memory and the pass over the data is limited [18, 6, 5, 13]. For example [18] proposes an algorithm for efficiently computing associations, for example, word associations (co-occurrences, or joint frequencies) which can be interpreted as a set intersection problem. The authors in [6] propose a sampling technique to check the resemblance and containment of the two documents. The authors in [13] propose an algorithm based on the Maximum Likelihood method. They use n different hash functions and they rely on other sketches like Hyperloglog and bottom-k sketches for the cardinality and union cardinality estimation. Among these works [31] uses the set intersection estimator [5] for inclusion coefficient estimation and they already show that the inclusion coefficient estimator based on Jaccard coefficient estimator based on Bottom-k [12] is more accurate than the one based on set intersection estimator in [5]. Thus, in our experiment, we evaluate the accuracy of our inclusion coefficient estimator against the estimator in [31] which uses Bottom-k for the Jaccard coefficient estimator.

Foreign-key Detection: There has been much prior work in foreign-key (FK) detection, e.g. [29, 31, 11]. A machine learning technique to find the single-column FKs from the INDs is proposed by [29]. They first use the SPIDER [4] to find INDs. Then, they define a set of characteristics/features, which are fed into a binary classification to discover the FKs. They use different classifiers but based on their result no classifier is consistently the best across all databases. Discovering FKs in PowerPivot is studied in [11], where the data fits in the memory and it has in-memory dictionaries of each column that allows efficient lookup the cardinality of each column and execute random sample based probing. The authors in [31] integrate the IND detection in multi-column FK discovery using two passes over the data. However, they assume all primary keys are defined and in most cases the values in a FK column form a nearly uniform random sample of values in the key column.

7. CONCLUSION

In this paper, we present a new estimator, BML, for inclusion coefficient based on Hyperloglog sketches that results in significantly lower error compared to existing approaches. Our experimental results confirms the effectiveness of our algorithms on industry benchmarks such as TPC-H and TPC-DS as well as several real-world databases. We show how to incrementally maintain HyperLogLog sketches with data deletions; this broadens the applicability of these sketches to additional scenarios. In the future, it is also interesting to consider if hybrid approaches that combine HLL and Bottom-k sketches can further improve accuracy. One challenge in such a hybrid approach is determining how much memory to assign to each of the sketches to obtain the greatest accuracy.

8. ACKNOWLEDGMENT

We thank Arnd Christian König for his insightful comments.

APPENDIX

A. HLL CONSTRUCTION ALGORITHM

Algorithm 3 shows the steps to construct HLL sketches for a set of columns $\mathcal{C}$. This algorithm reads data only once and for each column $X \in \mathcal{C}$ it generates the HLL sketch with 2^m buckets.

B. PROOF OF THEOREM 2

PROOF. We first show $pr(V^X \leq V^Y)$ is an increasing function of n_τ. Since the inclusion coefficient $\Phi(X, Y)$ is $\frac{n_\tau}{n_X}$, $pr(V^X \leq$

Algorithm 3: ConstructHLLSketch

1: **Input:** Columns: $\mathcal{C}$, Number of buckets: 2^m, Hash function: h
2: **Output:** $\mathcal{S}_X = \{b_1 : V_1^X, \cdots, b_{2^m} : V_{2^m}^X\}$
3: **for** column $X \in \mathcal{C}$
4: **for** value $X[i] \in X$
5: $s_i = h(X[i])$
6: j = bucket index determined by the leftmost m bits of s_i
7: $\rho(s_i)$ = position of the leftmost 1 in the $l-m$ bits of s_i
8: **if** b_j in $\mathcal{S}_X$ **then** $V_j^X = Max(\mathcal{S}_X[b_j], \rho(s_i))$ **else** $V_j^X = \rho(s_i)$
9: $\mathcal{S}_X.Add(b_j, V_j^X)$

V^Y) is also an increasing function of $\Phi(X,Y)$. Let us assume $n_\tau = t$. Equation 8 can be written as:

$$\begin{aligned} pr(V^X \le V^Y) = \sum_{k=0}^{\ell-m} &(1-\frac{1}{2^{k-1}})^t(1-\frac{1}{2^k})^{n_\mathcal{X}} \\ &\Bigg[\left((1-\frac{1}{2^k})^{n_\mathcal{Y}} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}}\right) \\ &+ \left(\frac{(1-\frac{1}{2^k})^t - (1-\frac{1}{2^{k-1}})^t}{(1-\frac{1}{2^{k-1}})^t}\right)(1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}} \\ &+ \left(\frac{(1-\frac{1}{2^k})^t - (1-\frac{1}{2^{k-1}})^t}{(1-\frac{1}{2^{k-1}})^t}\right) \\ &\qquad \left((1-\frac{1}{2^k})^{n_\mathcal{Y}} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}}\right)\Bigg] \end{aligned} \tag{13}$$

for $n_\tau = t+1$, this equation will be update to:

$$\begin{aligned} pr(V^X \le V^Y) = \sum_{k=0}^{\ell-m} &(1-\frac{1}{2^{k-1}})^{t+1}(1-\frac{1}{2^k})^{n_\mathcal{X}-1} \\ &\Bigg[\left((1-\frac{1}{2^k})^{n_\mathcal{Y}-1} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}-1}\right) \\ &+ \left(\frac{(1-\frac{1}{2^k})^{t+1} - (1-\frac{1}{2^{k-1}})^{t+1}}{(1-\frac{1}{2^{k-1}})^{t+1}}\right)(1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}-1} \\ &+ \left(\frac{(1-\frac{1}{2^k})^{t+1} - (1-\frac{1}{2^{k-1}})^{t+1}}{(1-\frac{1}{2^{k-1}})^{t+1}}\right) \\ &\qquad \left((1-\frac{1}{2^k})^{n_\mathcal{Y}-1} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}-1}\right)\Bigg] \end{aligned} \tag{14}$$

We show that for a given k the term in Equation 14 is larger than the term in Equation 13.

$$\begin{aligned} &(1-\frac{1}{2^{k-1}})^{t+1}(1-\frac{1}{2^k})^{n_\mathcal{X}-1}\Bigg[\left((1-\frac{1}{2^k})^{n_\mathcal{Y}-1} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}-1}\right) \\ &+ \left(\frac{(1-\frac{1}{2^k})^{t+1} - (1-\frac{1}{2^{k-1}})^{t+1}}{(1-\frac{1}{2^{k-1}})^{t+1}}\right)(1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}-1} \\ &+ \left(\frac{(1-\frac{1}{2^k})^{t+1} - (1-\frac{1}{2^{k-1}})^{t+1}}{(1-\frac{1}{2^{k-1}})^{t+1}}\right) \\ &\qquad \left((1-\frac{1}{2^k})^{n_\mathcal{Y}-1} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}-1}\right)\Bigg] \\ &- (1-\frac{1}{2^{k-1}})^t(1-\frac{1}{2^k})^{n_\mathcal{X}}\Bigg[\left((1-\frac{1}{2^k})^{n_\mathcal{Y}} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}}\right) \\ &+ \left(\frac{(1-\frac{1}{2^k})^t - (1-\frac{1}{2^{k-1}})^t}{(1-\frac{1}{2^{k-1}})^t}\right)(1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}} \\ &+ \left(\frac{(1-\frac{1}{2^k})^t - (1-\frac{1}{2^{k-1}})^t}{(1-\frac{1}{2^{k-1}})^t}\right)\left((1-\frac{1}{2^k})^{n_\mathcal{Y}} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}}\right)\Bigg] \\ &= \left((1-\frac{1}{2^k})^{n_\mathcal{Y}} - (1-\frac{1}{2^{k-1}})^{n_\mathcal{Y}}\right)\frac{1}{2^k-1} > 0 \end{aligned} \tag{15}$$

C. PROOF OF THEOREM 3

PROOF. As we proved in Theorem 2 given columns X and Y the $pr(V^X \le V^Y)$ is an increasing function of $\Phi(X,Y)$. Let $pr(V^X \le V^Y)$ be $\alpha(\Phi)$, ($\alpha(\Phi) \in [0,1]$). For the 2^m buckets of HLL sketches of X and Y let z_1,z_2, $\cdots$, z_{2^m} be the independent bernoulli random variables where z_i is one when $V_i^X \le V_i^Y$ and zero otherwise. Thus in 2^m buckets, the probability of getting exactly z buckets where $V_i^X \le V_i^Y$ is given by:

$$g(\phi) = B(z; 2^m, \alpha(\Phi)) = \binom{2^m}{z}\alpha(\Phi)^z(1-\alpha(\Phi))^{2^m-z} = f(\alpha(\Phi)) \tag{16}$$

Where $f(x) = \binom{2^m}{z}x^z(1-x)^{2^m-z}$ for $x \in [0,1]$ and obviously, $\max_\phi g(\phi) = \max_x f(x)$. Based on MLE we have $\Phi_{mle} = argmax_\phi g(\phi)$. Let $\hat{\Phi}$ be the inclusion coefficient estimate from our Algorithm 1. In fact to show that $\hat{\Phi} = \Phi_{mle}$ we should prove $g(\hat{\Phi}) = \max_\phi g(\phi)$. Our Algorithm 1 has two important steps, it first estimates the $\alpha(\Phi)$ as $\frac{\mathcal{Z}}{2^m}$ (lines 3-7), then it uses binary search to estimate $\hat{\Phi}$, where $\alpha(\Phi) = \frac{z}{2^m}$ (line 8). Here are two observations that we used in this proof.

1. Clearly for bionomial distribution function $f(x)$, its mean is the $argmax_x f(x)$. Thus in the first step of our algorithm we have:
$$x^* = argmax_x f(x) = \frac{z}{2^m} \tag{17}$$
2. The binary search in our algorithm is equivalent to $\alpha^{-1}(\frac{z}{2^m})$.
$$\hat{\Phi} = \alpha^{-1}(\frac{z}{2^m}) \tag{18}$$

Using these observations we now prove $g(\hat{\Phi}) = \max_\phi g(\phi)$.

$$\begin{aligned} g(\hat{\Phi}) &= f(\alpha(\hat{\Phi})) = f(\alpha(\alpha^{-1}(x^*))) \quad //by\ Equation\ 18 \\ &= f(x^*) \quad //by\ Equation\ 17 \\ &= max_x f(x) \quad //by\ Equation\ 16 \\ &= max_\phi g(\phi) = \Phi_{mle} \quad //by\ Equation\ 3 \end{aligned} \tag{19}$$

D. PROOF OF THEOREM 4

PROOF. For 2^m buckets let z_1,z_2, $\cdots$, z_{2^m} be the independent random variables where z_i is one when $V_i^X \le V_i^Y$ and zero otherwise, i.e., each z_i is an independent Bernoulli trial. In Algorithm 1, $\hat{\mathcal{P}} = \frac{\mathcal{Z}}{2^m}$, where $\mathcal{Z} = \sum_{i=1}^{2^m} z_i$. Since $\mathcal{Z}$ is sum of independent random variables, using the Hoeffding inequality we have:

$$\begin{aligned} pr\left(|\hat{\mathcal{P}} - \mathcal{P}| \le e_p\right) &= pr\left(\mathcal{P} - e_p \le \hat{\mathcal{P}} \le \mathcal{P} + e_p\right) \\ &= pr\left(2^m(\mathcal{P} - e_p) \le \mathcal{Z} \le 2^m(\mathcal{P} + e_p)\right) \\ &\ge 1 - 2\exp\left(-2^{m+1}e_p^2\right) \end{aligned} \tag{20}$$

E. BINDER vs. BML

We evaluated the memory consumption and the execution time of the BINDER vs BML over TPCDS-300. BINDER uses file inputs or database inputs. However [28], already showed that BINDER performs worse on database inputs than on file inputs. Thus we used BINDER on file inputs and we ran it on the same machine we ran BML (§5.1). We imported the TPCDS-300 data as csv files (240GB) in the Metanome [27]. We varied amount of memory in BINDER. BINDER was able to find 3861 unary INDs in 371, 403, and 460 minutes, where the memory sets to 384GB, 250GB, and 128GB respectively. Similar to BINDER our BML estimator read data only once but it only keeps HLL sketches. As we discussed in §5.1, total memory of sketches per column in TPCDS-300 is $2^{14} * 6$ bits and total number of columns are 362 (Table 3). Thus total memory consumption of BML is 4.4GB and as shown in Figure 12 BML takes 454 seconds to estimate the inclusion coefficients of all column pairs. Thus, BML is one to two orders of magnitude faster, and can work with relatively small amounts of memory.

F. REFERENCES

[1] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *PVLDB*, 24(4):557–581, 2015.

[2] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM*, 29(1):180–200, 1999.

[3] J. Bauckmann, Z. Abedjan, U. Leser, H. Müller, and F. Naumann. Discovering conditional inclusion dependencies. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 2094–2098, New York, NY, USA, 2012. ACM.

[4] J. Bauckmann, U. Leser, and F. Naumann. Efficiently computing inclusion dependencies for schema discovery. In *Workshop-Proceedings of the ICDE*, 2006.

[5] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, SIGMOD '07, pages 199–210, New York, NY, USA, 2007. ACM.

[6] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[7] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 327–336, New York, NY, USA, 1998. ACM.

[8] R. L. Burden and J. D. Faires. Numerical analysis. 2011.

[9] L. Carter, R. W. Floyd, J. Gill, G. Markowsky, and M. N. Wegman. Exact and approximate membership testers. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 59–65, 1978.

[10] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. *PVLDB*, 1(2):1265–1276, 2008.

[11] Z. Chen, V. Narasayya, and S. Chaudhuri. Fast foreign-key detection in microsoft sql server powerpivot for excel. *PVLDB*, 7(13):1417–1428, 2014.

[12] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 225–234. ACM, 2007.

[13] R. Cohen, L. Katzir, and A. Yehezkel. 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*, pages 95–103. ACM, 2017.

[14] P. Flajolet, E. Fusy, O. Gandouet, and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 2007 international conference on analysis of algorithms*, 2007.

[15] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[16] 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*, pages 683–692. ACM, 2013.

[17] S. Kruse, T. Papenbrock, C. Dullweber, M. Finke, M. Hegner, M. Zabel, C. Zollner, and F. Naumann. Fast approximate discovery of inclusion dependencies. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)*, 2017.

[18] P. Li and K. W. Church. A sketch algorithm for estimating two-way and multi-way associations. *Comput. Linguist.*, 33(3):305–354, 2007.

[19] P. Li, A. Owen, and C.-H. Zhang. One Permutation Hashing for Efficient Search and Learning. *ArXiv e-prints*, 2012.

[20] P. Li, A. Shrivastava, and A. C. König. b-bit minwise hashing in. *CoRR*, abs/1205.2958, 2012.

[21] S. Lopes, J.-M. Petit, and F. Toumani. Discovering interesting inclusion dependencies: Application to logical database tuning. *Inf. Syst.*, 27(1):1–19, 2002.

[22] F. D. Marchi and J.-M. Petit. Zigzag: A new algorithm for mining large inclusion dependencies in databases. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 27–, Washington, DC, USA, 2003. IEEE Computer Society.

[23] J. Motl and O. Schulte. The CTU prague relational learning repository. *CoRR*, abs/1511.03086, 2015.

[24] F. Naumann. Data profiling revisited. *SIGMOD Rec.*, 42(4):40–49, 2014.

[25] R. Pagh, G. Segev, and U. Wieder. How to approximate a set without knowing its size in advance. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 80–89, 2013.

[26] R. Pagh, M. Stöckel, and D. P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 109–120, New York, NY, USA, 2014. ACM.

[27] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome. *PVLDB*, 8(12):1860–1863, 2015.

[28] T. Papenbrock, S. Kruse, J.-A. Quiané-Ruiz, and F. Naumann. Divide and conquer-based inclusion dependency discovery. *PVLDB*, 8(7):774–785, 2015.

[29] A. Rostin, O. Albrecht, J. Bauckmann, F. Naumann, and U. Leser. A machine learning approach to foreign key discovery. In *WebDB*, 2009.

[30] A. Shrivastava and P. Li. Improved Densification of One Permutation Hashing. *ArXiv e-prints*, 2014.

[31] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *PVLDB*, 3(1-2):805–814, 2010.