Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates

Michael Freitag, Thomas Neumann
Technische Universität München
{freitagm,neumann}@in.tum.de

ABSTRACT

Database systems heavily rely upon cardinality estimates for finding efficient execution plans, and estimation errors can easily affect query execution times by large factors. One particularly difficult problem is estimating the result size of a group-by operator, or, in general, the number of distinct combinations of a set of attributes. In contrast to, e. g., estimating the selectivity of simple filter predicates, the resulting number of groups cannot be predicted reliably without examining the complete input. As a consequence, most existing systems have poor estimates for the number of distinct groups.

However, scanning entire relations at optimization time is not feasible in practice. Also, precise group counts cannot be precomputed for every possible combination of attributes. For practical purposes, a cheap mechanism is thus required which can handle arbitrary attribute combinations efficiently and with high accuracy.

In this work, we present a novel estimation framework that combines sketched full information over individual columns with random sampling to correct for correlation bias between attributes. This combination can estimate group counts for individual columns nearly perfectly, and for arbitrary column combinations with high accuracy. Extensive experiments show that these excellent results hold for both synthetic and real-world data sets. We demonstrate how this mechanism can be integrated into existing systems with low overhead, and how estimation time can be kept negligible by means of an efficient algorithm for sample scans.

1. INTRODUCTION

Estimating the number of distinct values for a given set of attributes is one of the classical problems of query optimization. For example, the result cardinality of the following query fragment, which could be part of a larger query,

```
select    A, B, sum(C)
from      R
group by A, B
```


9th Biennial Conference on Innovative Data Systems Research (CIDR '19) January 13-16, 2019, Asilomar, California, USA.

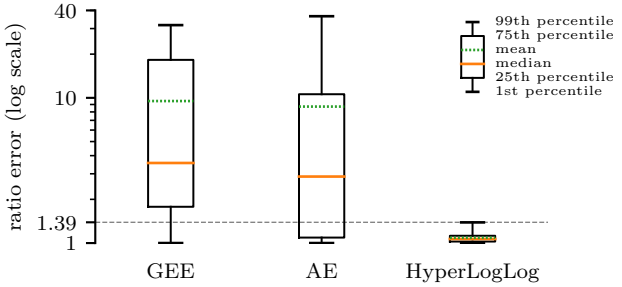


Figure 1: Multiplicative estimation error of existing sampling-based approaches GEE and AE in comparison to a 64 byte HyperLogLog sketch, over all individual columns of the IMDb data sets.

is determined by the number of unique pairs (A, B). Besides group-by clauses, these distinct value counts are also used in many other places like, e. g., hash table sizing or cardinality estimation for outer and multi-attribute joins. Getting these estimates wrong can lead to very poor performance [17].

Accordingly, the problem of estimating the number of distinct values has been extensively studied before, albeit largely with negative results [3, 11]. In their seminal paper, Charikar et al. showed that we cannot derive good estimates from reasonably sized samples [3]. Fundamentally, most of the input has to be examined to estimate the domain size accurately. Nevertheless, Charikar et al. proposed two sampling-based estimators, GEE and AE, for pragmatic reasons. One simply cannot read the complete input for estimation purposes, and sampling offers attractive performance.

A different family of approaches uses small fixed sized data sketches that allow for estimating the number of distinct values with little overhead. A prominent example is the HyperLogLog estimator that manages to get very accurate estimates using an astonishingly small state [7]. Figure 1 shows the estimation accuracy of GEE and AE, using a sampling fraction of 0.1%, compared to a 64 byte (!) HyperLogLog sketch using the improved estimator by Ertl [6]. The plot shows the error distribution over all individual columns of the Internet Movie Database (IMDb) data sets on a logarithmic scale. We can see that while the sampling based approaches often have very large estimation errors, the improved HyperLogLog (HLL) estimates are nearly perfect. The fundamental difference is that the HLL sketch has seen

Algorithm 1: Insert (traditional HLL sketch)

state : $m = 2^b$ zero-initialized buckets $M \in \mathbb{N}^m$
input: A 64-bit hash value $h = \langle h_{64}, \ldots, h_1 \rangle_2$

$i \leftarrow \langle h_{64}, \ldots, h_{64-b+1} \rangle_2$; // *bucket index*
$z \leftarrow$ `LeadingZeros(`$\langle h_{64-b}, \ldots, h_1 \rangle_2$`)` ;
$M_i \leftarrow \max(M_i,\ z)$;

every input value once during construction, while GEE and AE try to extrapolate from few samples to the full relation.

Nevertheless, most existing systems use sampling based approaches, often with very poor accuracy. PostgreSQL 10.3, for instance, which uses sampling, estimates the number of distinct `l_orderkey` values in TPC-H SF1 as 395 518. This estimate is off by a factor of 3.8, although the simple TPC-H data set exhibits convenient uniform distributions in its columns. Estimates on real-world data sets with skewed data distributions can be expected to be much worse. HLL based sketches promise dramatically better accuracy with very little state, but they are hard to use in general. First, one has to maintain the sketch during inserts, updates and deletions. Second, estimates must be supported for arbitrary combinations of attributes, but we cannot maintain an exponential number of HLL sketches.

In this work we overcome these difficulties, and introduce an estimation framework that combines the benefits of HLL sketches and sampling. Our contributions are 1) an efficient HLL sketch implementation that supports updates and deletions and that gives very accurate estimates for individual columns, 2) a sketch-based correction framework that allows for computing accurate multi-column estimates from a sample, and 3) a very fast frequency computation implementation within the sample using an efficient recursive algorithm. The combined framework allows for very accurate estimates with low overhead, as we demonstrate in a large-scale evaluation on synthetic and real-world data sets.

The rest of this paper is structured as follows: First, Section 2 introduces the updateable HLL sketches for individual columns. Then, Section 3 shows how these estimates can be combined with random sampling to derive estimates for multiple columns. An algorithm for efficient frequency computation is introduced in Section 4. Experimental results are shown in Section 5, and related work is discussed in Section 6.

2. SKETCHING INDIVIDUAL COLUMNS

Before addressing the general case of arbitrary attribute combinations, we first look at using sketches for individual columns. The goal is to maintain HLL sketches for all columns stored in the database, which allows us to obtain very accurate estimates for the number of distinct values within single columns. The main challenge lies in supporting arbitrary updates while keeping the overhead low. We first briefly review traditional HLL sketches, and then show how to generalize them to support updates and deletions.

2.1 Traditional HyperLogLog Sketches

HyperLogLog sketches are a greatly improved variation of the ground-breaking Flajolet-Martin sketches [7,8]. Given a high-quality hash function that maps values uniformly into the integer domain, the key idea is that the number of distinct values in a multiset can be deduced by making use of two properties of their hash values. First, two identical values will have the same hash value. Second, of the distinct hash values, roughly 50% will have a zero in the first bit of the hash value, roughly 25% will have only zeros in the first two bits, and a fraction of approximately $1/2^i$ will have only zeros in the first i bits.

Thus, we can compute a very rough estimate for the number of distinct values as follows: First, we hash all values in the multiset, and track the maximum number $\max(i)$ of leading zero bits i of all hash values. The number of distinct values can then be estimated as $2^{\max(i)}$, using just one small integer as state regardless of the size of the multiset.

In practice, using just one integer for estimation is too sensitive to outliers. Instead, hash values are assigned to $m = 2^b$ buckets based on their first b bits. The number of leading zeros is then computed on the remaining bits, and its maximum is tracked individually for each bucket (cf. Algorithm 1). The original HyperLogLog algorithm computes the harmonic mean of the resulting m individual estimates [7], but this can lead to biased results if the cardinality is small [14]. In the following, we will use an improved estimator that uses a Poisson model to handle the complete range of cardinalities [6]. The resulting algorithm executes only a handful of bit operations per hash value and is thus very cheap [6, 14].

Within each bucket the maximum number of leading zeroes is stored, which is at most $64 - b$ for 64 bit hash values. Each bucket thus fits into a single byte, leading to a very small state size of m bytes. The expected relative error is $1.04/\sqrt{m}$, which means that with just 64 bytes of state we expect a multiplicative error of 1.13, which is good enough for estimation purposes. During experiments with thousands of data sets from a commercial vendor, we found that, with 64 bytes of state, the improved estimator achieves a median multiplicative error of only 1.07, and an error of 1.24 in the 99% quantile. Based on these results, we choose a state size of 64 bytes in the following, which also happens to coincide with the cache line size on modern CPUs.

Algorithm 2: Insert (counting HLL sketch)

state : $m = 2^b$ buckets of $64 - b + 1$ zero-initialized counters $M \in \mathbb{N}^{m \times (64-b+1)}$
input: A 64-bit hash value $h = \langle h_{64}, \ldots, h_1 \rangle_2$

$i \leftarrow \langle h_{64}, \ldots, h_{64-b+1} \rangle_2$; // *bucket index*
$z \leftarrow$ `LeadingZeros(`$\langle h_{64-b}, \ldots, h_1 \rangle_2$`)` ;
if $M_{iz} \leq 128$ **then**
  increment M_{iz} ;
else
  increment M_{iz} with probability $1/2^{M_{iz}-128}$;
end

2.2 Updateable HyperLogLog Sketches

When using sketches inside a database system, we have to cope with the fact that values are both inserted and deleted. HLL sketches support inserts out of the box, but deleting a value whose leading zero count is equal to the current bucket value is problematic. We do not know if we have to decrease the bucket value, since other values could exist in

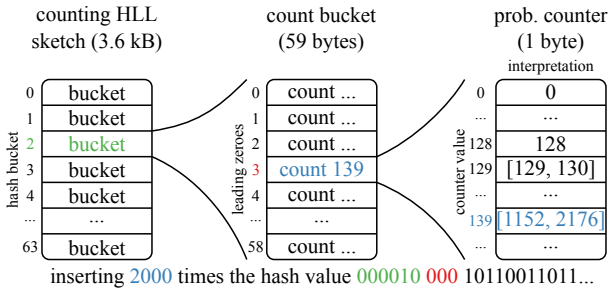


Figure 2: Using counting HLL sketches for updates.

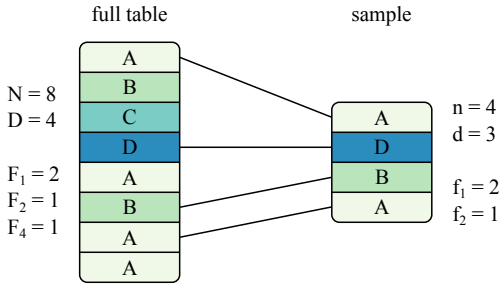


Figure 3: Example of a sample being drawn from a table with an unspecified number of columns. For an overview of the notation used, see Table 1.

that bucket with the same number of leading zeroes, and this information is not maintained by traditional HLL sketches.

Therefore, *counting* HyperLogLog sketches have been proposed that remember how many values had a certain number of leading zeroes [8, 21]. With this information we can support both insertion and deletion, increasing and decreasing the counters as needed. The estimation process itself remains unchanged, as we only maintain the sketched information in a different representation. Since we are using $m = 2^6 = 64$ buckets, there are 59 possible leading zero counts for 64 bit hash values. If we maintain counters for each of these values naively [21], using 8 byte integers, we end up with a sketch that requires nearly 30 kB of space. This can be prohibitively expensive if we sketch every column in a database, and we propose a more space-efficient variant of counting HLL sketches.

As outlined above, the probability that a hash value has exactly i leading zeros is $1/2^{i+1}$. That is, low values of i are exponentially more likely than high values, and the maximum observed value used for estimation likely occurs only a few times. This can be exploited to reduce storage space considerably, by using a one-byte *probabilistic counter* [8]. The first 128 occurrences of a value are counted exactly, and the remaining byte values $v > 128$ represent ranges of exponentially growing size $[128 + 2^{v-129}, 128 + 2^{v-128}]$. When incrementing a counter that is within these exponential ranges, we perform the increment with the probability that the current value is the largest value within the range (cf. Algorithm 2). This is a variant of the probabilistic counting approach by Flajolet and Martin [8], with the difference that we count the important small values exactly, while the less important large values are counted with some uncertainty, but expected correct behavior. The delete operation is symmetrical to Algorithm 2, decrementing instead of incrementing counters.

Figure 2 shows the effect of inserting one hash value 2 000 times into the sketch. The first 6 bits of the hash value indicate that bucket 2 needs to be updated. Within the remaining bits of the hash value, there are 3 leading zeroes, which means that we increase the corresponding counter 2 000 times. This is beyond the exact range of the counter, and we end up with an (expected) counter value of 139 which represents the interval [1 152, 2 176].

The proposed approach allows us to handle both deletion and insertion with a reasonable overhead. The state size is 3.6 kB, which is of course much larger than the original 64 bytes. Nevertheless, in many cases it is still much smaller than the space required to store a sample of a column, which requires 8 bytes per value in our implementation (i. e. 3.6 kB for only 450 rows). The update and delete operations require only few additional instructions compared to the original algorithm, and remain very cheap (cf. Section 5).

3. MULTI-COLUMN ESTIMATES

Counting HLL sketches offer excellent accuracy and performance on individual columns, but it is infeasible to maintain sketches on all combinations of attributes. For these cases, we propose a novel estimation approach which leverages the accurate single-column estimates of counting HLL sketches to correct multi-column estimates obtained through sampling. We apply our correction approach to the well-known estimators GEE and AE [3], and to a novel estimator based on improved estimation bounds.

3.1 Background

In the following, we consider a table with N rows and $C \geq 2$ attributes, which contains D distinct tuples. Let these distinct tuples be indexed by $k \in \{1, \ldots, D\}$, and suppose the k-th distinct tuple occurs N_k times in the table, i. e. $N = \sum_{k=1}^{D} N_k$. Furthermore, let $Q = \max(N_k)$, and define F_i to be the number of distinct tuples that occur exactly i times in the table, i. e. $N = \sum_{i=1}^{Q} i \cdot F_i$ and $D = \sum_{i=1}^{Q} F_i$. In the following, we will refer to a tuple which occurs exactly once as a *singleton* tuple, or simply *singleton*.

Our estimation approach examines a random sample containing $n \leq N$ rows, which are chosen uniformly at random from the table. For comparability with previous work [3], we consider sampling with replacement, however, our approach can be adapted easily to sampling without replacement. Suppose there are d distinct tuples in this sample, indexed by $k \in \{1, \ldots, d\}$, and the k-th distinct tuple occurs n_k times in the sample. Let f_i denote the number of distinct tuples which occur exactly i times in the sample, and $q = \max(n_k)$. Then, analogous as above, $n = \sum_{i=1}^{q} i \cdot f_i$ and $d = \sum_{i=1}^{q} f_i$.

For a clarification of this notation, consider the example shown in Figure 3. There is a table containing $N = 8$ rows, with $D = 4$ distinct tuples identified by distinct uppercase letters. Within the entire table, two tuples occur once ($F_1 = 2$), one tuple occurs twice ($F_2 = 1$), and one tuple occurs four times ($F_4 = 1$). We draw a sample containing $n = 4$ rows from this table, of which $d = 3$ are distinct tuples. Two tuples occur once in the sample ($f_1 = 2$), and one tuple occurs twice ($f_2 = 1$). An overview of our notation is also displayed in Table 1.

Table 1: Selected notation used throughout Section 3. Uppercase variables refer to the entire table, and lowercase variables refer to a sample of the table. A tuple refers to an entire row of the table.

notation table	notation sample	denotation
N	n	Number of rows
D	d	Number of distinct tuples over all columns
D_j	d_j	Number of distinct values in the j-th column
N_k	n_k	Absolute frequency of the k-th distinct tuple
$N_{k,j}$	$n_{k,j}$	Absolute frequency of the k-th distinct value in the j-th column
F_i	f_i	Number of distinct tuples over all columns which occur exactly i times
$F_{i,j}$	$f_{i,j}$	Number of distinct values in the j-th column which occur exactly i times

Following previous work on the subject [3], we evaluate an estimator $\hat{D}$ of the number of distinct tuples D in terms of its multiplicative *ratio error* which is defined as

$$error(\hat{D}) = \begin{cases} D/\hat{D} & \text{if } D \geq \hat{D} \\ \hat{D}/D & \text{if } D < \hat{D} \end{cases} . \quad (1)$$

A powerful negative result due to Charikar et al. states that any estimator which examines at most n rows of a table with N rows must incur an expected ratio error in $O(\sqrt{N/n})$ on some input [3]. They develop the Guaranteed Error Estimator (GEE) which is optimal with respect to this result, in the sense that its ratio error is bounded by $\sqrt{N/n}$ with high probability. This estimator is defined as

$$\hat{D}_{GEE} = \sqrt{\frac{N}{n}} f_1 + \sum_{i=2}^{q} f_i. \quad (2)$$

The key intuition underlying this approach is that any tuple which appears frequently in the entire table is also likely to be present in the sample. Thus, estimating the number of such tuples as $\sum_{i=2}^{q} f_i$ can be expected to be fairly accurate [3]. The total number of singleton tuples, on the other hand, can be much larger in the entire table than in the sample. Specifically, the f_1 singletons present in the sample could constitute up to a fraction N/n of the entire set of singletons, for a total of $Nf_1/n \leq N$ singletons. At the same time, however, there could be as few as f_1 singletons in the entire table. In order to minimize the expected ratio error, GEE estimates the true number of singletons as the geometric mean $\sqrt{N/n}f_1$ between the lower bound f_1 and upper bound Nf_1/n.

Despite its provable optimality, GEE provides only loose bounds on the ratio error for reasonable sampling fractions n/N. For example, for a sampling fraction of 1 % the ratio error of GEE can still be as large as 10. This renders its estimates unusable in many real-world scenarios (cf. Section 1). In particular, if f_1 is large relative to the number of distinct values in the sample, GEE will severely underestimate the actual number of singleton values [3]. Figure 4, for instance, shows a scatter plot of the true number of singletons F_1 in relation to the observed number of singletons f_1 in a sample of size $n/N = 1\,\%$ on the well-known Census data set. In most cases where f_1 is close to n, the true number of singletons F_1 is close to $N = 100n$. However, for $f_1 = n$, GEE would estimate the true number of singletons as $\sqrt{N/n}f_1 = 10n$, which differs from N by a *factor* of 10.

For this reason, Charikar et al. propose an adaptive estimator (AE), which attempts to derive some information about the data distribution from the sample in order to obtain more accurate estimates of the number of singleton values [3]. Nevertheless, our experimental results show that AE can still not produce satisfactory results in many cases (cf. Figure 1 and Section 5).

3.2 Improved Estimation Bounds

As outlined above, GEE incurs a high estimation error mainly when there is a large number of singleton tuples F_1 in the entire table. In these cases, GEE computes an overly conservative lower bound on F_1 from a given sample, which causes it to severely underestimate the true number of singletons. Figure 4 illustrates this problem on the well-known Census data set. There is a clear nonlinear relationship between the number of singletons observed in a sample f_1, and the number of singleton tuples F_1 in the entire table. However, as shown in Figure 4, GEE fails to exploit this relationship since it estimates the true number of singletons to be $\sqrt{N/n}f_1$ which scales linearly in f_1.

In the following, we thus derive improved bounds on the true number of singleton tuples based on quantities that can be observed in a sample of the relation. This allows us to subsequently derive a novel estimator with improved estimation accuracy in comparison to GEE and AE. In particular, we present an upper bound on the expected value $\mathbb{E}(f_1)$ of singleton tuples, and a lower bound on the expected value of distinct tuples $\mathbb{E}(d)$ in the sample. These inequalities link the number of distinct tuples D in the entire relation to these expected values, which can be estimated easily on a sample of the relation.

As shown in previous work [3], the expected number of singletons is given by

$$\mathbb{E}(f_1) = \sum_{k=1}^{D} nP_k(1-P_k)^{n-1}, \quad (3)$$

where $P_k = N_k/N$ denotes the relative frequency of the k-th distinct tuple in the entire table.

Intuitively, for a large number of distinct tuples, the expected value of f_1 is maximized when they are approximately uniformly distributed in the entire table. If some tuple occurred more frequently than others in the entire table, these would be more likely to be present frequently in the sample as well, reducing the expected number of singletons. This intuition is formalized as follows.

THEOREM 1. *Consider a table with N rows containing D distinct tuples. Suppose we draw a sample of n rows uniformly at random with replacement, and let f_1 denote the observed number of singleton tuples in this sample. Then, the following inequality holds*

$$\mathbb{E}(f_1) \leq \begin{cases} n \cdot (1-1/D)^{n-1} & \text{if } D \geq n, \\ D \cdot (1-1/n)^{n-1} & \text{otherwise.} \end{cases}$$

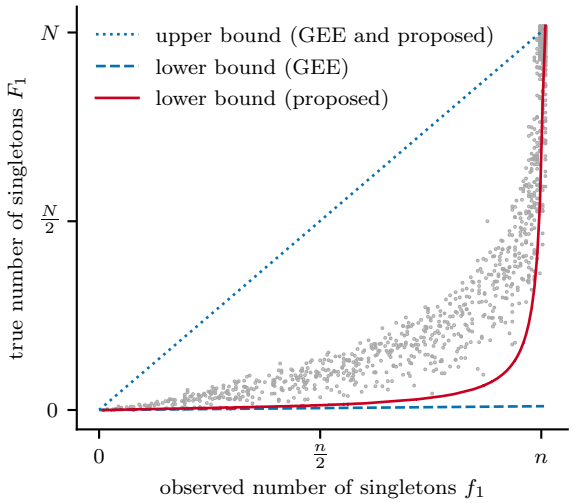


Figure 4: Scatter plot of the true number of singletons (y-axis) in relation to the number of singletons observed in a random sample (x-axis). The data points correspond to 1 500 randomly selected attribute combinations from the Census data set, and a sampling fraction of $n/N = 1\,\%$ is used.

A proof of this theorem is presented in Appendix A. On the other hand, as shown previously [3], the expected number of distinct tuples is given by

$$\mathbb{E}(d) = D - \sum_{k=1}^{D} (1 - P_k)^n. \tag{4}$$

Each distinct tuple must occur at least once in the table, i. e. $N_k \geq 1$ and consequently $P_k \leq 1/N$ for all k. Hence, a simple lower bound on $\mathbb{E}(d)$ can be derived as follows.

THEOREM 2. *Consider a table with N rows containing D distinct tuples. Suppose we draw a sample of n rows uniformly at random with replacement, and let d denote the observed number of distinct tuples in this sample. Then, the following inequality holds*

$$\mathbb{E}(d) \geq D - D \cdot (1 - 1/N)^n.$$

A formal proof of this theorem is presented in Appendix B. By rearranging the inequalities in Theorem 1 and Theorem 2 suitably, we obtain bounds L, U on the true number of distinct tuples D that depend on $\mathbb{E}(d)$ and $\mathbb{E}(f_1)$, where $L \leq D \leq U$. The observed quantities d and f_1 clearly constitute unbiased estimators for these expected values, allowing us to estimate the bounds on D as follows

$$\hat{L} = \begin{cases} 1/(1 - \sqrt[n-1]{f_1/n}) & \text{if } f_1 \geq n\,(1 - 1/n)^{n-1}, \\ f_1/(1 - 1/n)^{n-1} & \text{otherwise,} \end{cases} \tag{5}$$

as well as

$$\hat{U} = d/\left(1 - (1 - 1/N)^n\right). \tag{6}$$

Naturally, we apply sanity bounds to ensure that $d \leq \hat{L}, \hat{U} \leq N$. These estimated bounds can now be leveraged to define a novel estimator for the number of distinct tuples D. We adopt the assumption made by GEE that $\sum_{i=2}^{q} f_i$ accurately estimates the true number of tuples which occur more than once. Under this assumption, $\hat{L} - \sum_{i=2}^{q} f_i$ and $\hat{U} - \sum_{i=2}^{q} f_i$ provide approximate bounds on the true number of singletons F_1, allowing us to tighten the bounds originally used by GEE, i. e.

$$\hat{L}_{BC} = \max\left(f_1, \hat{L} - \sum_{i=2}^{q} f_i\right), \tag{7}$$

$$\hat{U}_{BC} = \min\left(\frac{N f_1}{n}, \hat{U} - \sum_{i=2}^{q} f_i\right). \tag{8}$$

Analogous to GEE, the true number of singletons is then estimated as the geometric mean between the adjusted lower and upper bounds, resulting in the bound-corrected estimator BC, specifically

$$\hat{D}_{BC} = \sqrt{\hat{L}_{BC}\hat{U}_{BC}} + \sum_{i=2}^{q} f_i. \tag{9}$$

As shown in Figure 4, the adjusted lower bound $\hat{L}_{BC}$ matches the data distribution much more accurately, especially for large cardinalities. In general, we observed that the adjusted upper bound $\hat{U}_{BC}$ frequently coincides with the original upper bound used by GEE, which is also evident in Figure 4. In practice, $\sum_{i=2}^{q} f_i$ will clearly underestimate the true number of tuples which occur more than once. Hence, $\hat{U} - \sum_{i=2}^{q} f_i$ will generally overestimate the upper bound on F_1, resulting in the observed behavior.

In case of sampling without replacement, one can follow a similar line of reasoning and develop an approximate lower bound based on the work of Goodman [10]. For space considerations, we only show the final result after applying standard numerical approximations, which yields

$$\hat{L} = \begin{cases} N/(\log(f_1/n)/\log(1 - r) + 1) & \text{if } f_1 \geq n(1 - r)^{1/r - 1}, \\ f_1/(1 - r)^{1/r - 1} & \text{otherwise,} \end{cases}$$

where $r = n/N$ is the sampling fraction.

3.3 Sketch-Corrected Estimators

As we will demonstrate in our experimental evaluation in Section 5, the BC estimator already exhibits a considerably improved ratio error in comparison to GEE and AE. Nevertheless, since it is based purely on a sample of the table, there are cases in which BC will incur a high ratio error in $O(\sqrt{N/n})$ as well [3]. For example, Figure 4 shows that both the proposed lower bounds and upper bounds are quite loose in many cases, which may lead to inaccurate estimates.

We overcome these problems by correcting the multi-column estimates using information about the value distribution of the individual attributes. Thus, let D_j denote the number of distinct values, and $F_{i,j}$ the number of distinct values which occur exactly i times in the j-th column of the table. Furthermore, suppose that the k-th distinct value in the j-th column occurs $N_{k,j}$ times, and define $Q_j = \max(N_{k,j})$, i. e. $\sum_{i=1}^{Q_j} F_{i,j} = D_j$ and $\sum_{i=1}^{Q_j} i \cdot F_{i,j} = N$. Finally, define d_j, $f_{i,j}$ and q_j analogously on a sample of the table (cf. Table 1).

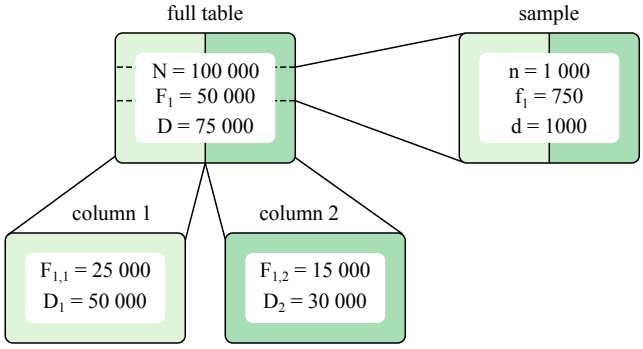


Figure 5: Sample value distributions in a table with two columns, from which a sample is drawn. For an overview of the notation used, see Table 1.

Assuming that D_j and $F_{1,j}$ are known for all individual attributes j, we can derive bounds on the true number of distinct tuples D and singleton tuples F_1, namely

$$\max(F_{1,j})_{j=1,\ldots,C} \leq F_1 \leq \Pi_{j=1}^{C} D_j, \tag{10}$$

$$\max(D_j)_{j=1,\ldots,C} \leq D \leq \Pi_{j=1}^{C} D_j. \tag{11}$$

The lower bound in Inequality 10 holds since any row which contains a singleton value in one column must be part of a singleton row when more columns are considered. Similarly, the lower bound in Inequality 11 applies because each distinct value in an individual column is part of at least one distinct tuple over multiple columns. For instance, the individual columns in Figure 5 contain up to $\max(F_{1,1}, F_{1,2}) = 25\,000$ singletons and $\max(D_1, D_2) = 50\,000$ distinct values. This implies that there are at least 25 000 singletons and 50 000 distinct values in the full table. In both cases, an upper bound is trivially given by the cardinality of the cross product of the distinct values in the individual columns. The latter bound is useful if the number of rows N is large, and there are few distinct values in the individual columns.

In practice, sketches can be employed to estimate D_j accurately and cheaply. Let these estimates be denoted by $\hat{D}_j$, and recall that GEE assumes $\sum_{i=2}^{q_j} f_{i,j}$ to fairly accurately estimate the number of non-singleton values in the j-th column. Hence, we can estimate the true number of singleton values in column j as

$$\hat{F}_{1,j} = \hat{D}_j - \sum_{i=2}^{q_j} f_{i,j}. \tag{12}$$

Substituting the exact values by these estimates in Inequalities 10 and 11 yields bounds on F_1 and D which can be used to correct the multi-column estimates of BC. For comparison purposes, we also correct the multi-column estimates of GEE and AE. In the following, we will refer to the corrected estimators as *sketch-corrected* estimators. In all cases, Inequality 11 is leveraged to provide sanity bounds on the estimates.

3.3.1 Sketch-Corrected GEE (SCGEE)

In case of GEE, we can tighten the original bounds on the true number of singleton tuples using Inequality 11, i. e.

$$\hat{L}_{SCGEE} = \max\left(f_1, \max(\hat{F}_{1,j})_{j=1,\ldots,C}\right), \tag{13}$$

$$\hat{U}_{SCGEE} = \min\left(\frac{N f_1}{n}, \Pi_{j=1}^{C} \hat{D}_j\right). \tag{14}$$

Analogous to GEE, the expected ratio error can be minimized by estimating F_1 as the geometric mean of the upper and lower bounds, and the corresponding sketch-corrected estimator is defined as

$$\hat{D}_{SCGEE} = \sqrt{\hat{L}_{SCGEE}\hat{U}_{SCGEE}} + \sum_{i=2}^{q} f_i. \tag{15}$$

In Figure 5, for example, GEE would estimate the number of singletons to be $\sqrt{N/n} f_1 = 7\,500$, far below the true value $F_1 = 50\,000$. Due to corrected bounds, on the other hand, SCGEE estimates the number of singletons much more accurately as $\sqrt{\hat{L}_{SCGEE}\hat{U}_{SCGEE}} \approx 43\,000$. Conveniently, the estimator inherits the worst-case error bound guarantee of GEE, since it only tightens the original bounds on F_1.

3.3.2 Sketch-Corrected AE (SCAE)

The adaptive estimator AE involves a complex numerical approximation of the estimated number of low-frequency elements in the table. It is beyond the scope of this paper to identify ways in which these approximations can be corrected directly. Hence, we only apply the sanity bounds provided by Inequality 11 to AE, resulting in the sketch-corrected adaptive estimator $\hat{D}_{SCAE}$.

3.3.3 Sketch-Corrected BC (SCBC)

Although the bound-corrected estimator BC already employs tightened estimation bounds, we conjecture that it can be improved further through sketch-correction. We correct BC in the same way as GEE, by adjusting the bounds on the true number of singleton tuples using Inequality 10. Therefore, we obtain

$$\hat{L}_{SCBC} = \max\left(\hat{L}_{BC}, \max(\hat{F}_{1,j})_{j=1,\ldots,C}\right), \tag{16}$$

$$\hat{U}_{SCBC} = \min\left(\hat{U}_{BC}, \Pi_{j=1}^{C} \hat{D}_j\right), \tag{17}$$

and the sketch-corrected estimator

$$\hat{D}_{SCBC} = \sqrt{\hat{L}_{SCBC}\hat{U}_{SCBC}} + \sum_{i=2}^{q} f_i. \tag{18}$$

Returning to the example displayed in Figure 5, BC would estimate the true number of singletons to be $\sqrt{\hat{L}_{BC}\hat{U}_{BC}} \approx 16\,000$, which already improves over the estimate by GEE. After sketch-correction, SCBC employs the same bounds as SCGEE in this case and estimates F_1 to be approximately $\sqrt{\hat{L}_{SCBC}\hat{U}_{SCBC}} \approx 43\,000$. In general SCBC produces more accurate estimates than SCGEE, as our experimental evaluation will demonstrate (cf. Section 5).

4. COMPUTING FREQUENCIES

The estimators presented in the previous section need to determine the number f_i of attribute combinations that occur exactly i times in a sample. This frequency vector f can be

Algorithm 3: ComputeFrequenciesR(ecursive)

input : A sample $S \in \mathbb{N}^{n \times C}$, a partially built frequency vector $f \in \mathbb{N}^n$, a set of row indices P, and a column index j

output: f updated by the multiplicities of rows in P

```
1 if j > C ∨ |P| = 1 then
      // base case
2     f_|P| ← f_|P| + 1;
3 else
      // partition j-th column and recurse
4     (P'_k)_{k=1,...,m} ← RefinePartition(S_{*,j}, P);
5     for partition index k = 1 to m do
6         f ← ComputeFrequenciesR(S, f, P'_k, j + 1);
7     end
8 end
9 return f;
```

computed in a straightforward way by using a hash table, but hashing or comparing entire rows can be expensive since each individual attribute has to be accessed. Moreover, the constants hidden in the $O(1)$ time complexity of the insertion and retrieval operations of a hash table can notably impact computation time even if the number of columns is small (cf. Section 5). Finally, as the number of distinct attribute combinations is not known beforehand, memory for the hash table has to be allocated pessimistically in order to avoid expensive rehashing during computation. Thus, the hash table will be unnecessarily large in many cases.

Instead, we propose a recursive approach for computing the frequency vector f based on an algorithm for string multiset discrimination first proposed by Cai and Paige [1]. Their algorithm scans strings in a multiset from left to right, and progressively splits the multiset into smaller partitions by examining the characters at the current position. A similar approach can be used to compute f if rows in the sample are interpreted as strings over a suitable alphabet, as the size of partitions then indicates how often a row occurs in the sample. For convenience, we will assume in the following that all values in the sample are integers. A high-level illustration of the proposed algorithm is displayed in Algorithm 3. It takes as input the sample $S \in \mathbb{N}^{n \times C}$, a partially built frequency vector $f \in \mathbb{N}^n$, a set of row indices P, and a column index $j \in \{1, \ldots, C+1\}$. The algorithm recursively computes the multiplicities of rows in P, and updates the corresponding entries of the frequency vector f.

The recursion terminates either if P contains only one row, or if there are no more columns to check, i. e. $j = C+1$. In these cases, we have found a row with multiplicity $|P|$, and the frequency vector is updated accordingly (lines 1–3). Otherwise, the given partitioning is refined based on the values in the j-th column, using the RefinePartition subroutine (line 4). It takes as input a column of the sample and a set of row indices, and splits the row indices into several sets so that the column values are equal for all rows within one set. Finally, the given frequency table is updated recursively on each of these refined partitions (lines 5–7). A frequency table for the entire sample can be computed by passing $f = 0$, $P = \{1, \ldots, n\}$, and $j = 1$ as parameters to Algorithm 3. The algorithm maintains the invariant that for a given j, all columns $j' < j$ have the same value within a partition, which implies correctness. It terminates since j is incremented in each recursive step and cannot exceed $C+1$.

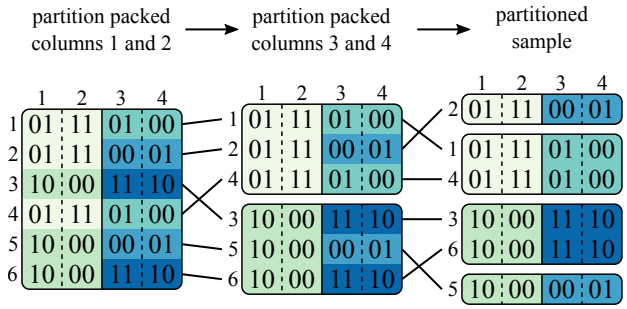


Figure 6: Recursively partitioning several columns at a time. Column values are encoded in 2 bits, and a machine word size of 4 bits is assumed. The computed frequencies are $f_1 = f_2 = 2$.

The proposed approach can be optimized further as follows. First, any row which contains a singleton value in at least one column must be a singleton attribute combination, and can be pruned in a preprocessing step, e. g. by maintaining suitable singleton bitmaps. Second, we encode the remaining column values as indices into a dictionary, which require at most $\lceil \log_2(n) \rceil$ bits for a sample of size n. This allows the algorithm to process multiple columns at once, by packing several column values into a single machine word (cf. Figure 6). Furthermore, all values in the sample can be converted to integers this way, justifying the corresponding assumption made above. Finally, we consider columns with many distinct values early, so that partition sizes decrease more quickly and the algorithm terminates faster.

The RefinePartition subroutine can be implemented in linear time using either hash tables or radix sort, at the cost of using some auxiliary memory. However, we have found that, in practice, simply sorting the rows in-place followed by a linear scan to determine the partition boundaries can perform better on realistic sample sizes. Since partitions never overlap, the recursive algorithm can be implemented using a single auxiliary array of row indices, which is progressively updated as partitions are refined (cf. Figure 6). When partitioning several columns at a time, another auxiliary array of the same size is required in order to compute the packed row values. These values cannot be precomputed because the algorithm must be able to compute frequency vectors for arbitrary subsets of attributes.

5. EXPERIMENTS

In the following, we evaluate the proposed approach with respect to its computational performance and estimation accuracy. First, we demonstrate that the proposed counting HLL sketch incurs a negligible performance overhead compared to traditional HLL sketches, while retaining similarly high estimation accuracy in the presence of deletions. Second, we show that the proposed sketch-corrected estimators exhibit superior estimation accuracy in comparison to previous estimators. Finally, our experiments yield that the proposed frequency computation algorithm offers excellent performance, providing low estimation latency even on large real-world data sets.

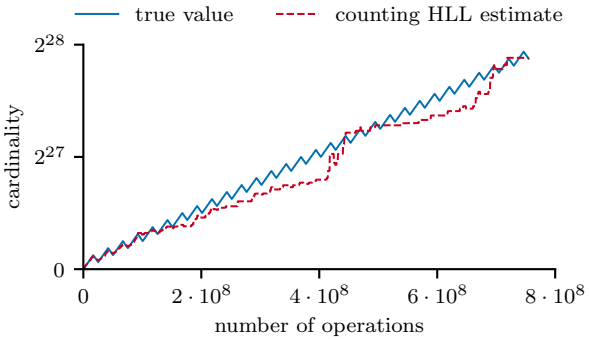


Figure 7: Sample workload used to evaluate the estimation accuracy of counting HLL sketches. After each $i = 2^{24}$ inserts, $r = 50\%$ of these operations are subsequently reverted by deleting the corresponding values from the sketch.

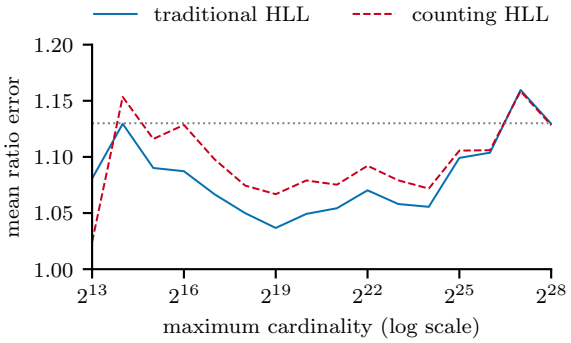


Figure 8: Mean ratio error (y-axis) of counting and traditional HLL sketches when inserting a given number of distinct values (x-axis). For counting HLL sketches, the ratio error is aggregated over all workload configurations. The dotted gray line indicates the theoretically expected ratio error of 1.13.

Table 2: CPU time required to sketch all values of a table with 10 million rows and 10 columns for the traditional HLL sketch and the proposed approach.

HLL variant	column-wise	row-wise
traditional	132 ms	111 ms
counting	340 ms	370 ms

Table 3: Ratio error incurred by counting HLL sketches, aggregated across all experiments. For comparison, the same values have been inserted into a traditional HLL sketch without any deletions.

HLL variant	mean	1 %	25 %	50 %	75 %	99 %
traditional	1.13	1.00	1.05	1.13	1.20	1.34
counting	1.13	1.00	1.05	1.12	1.20	1.34

5.1 Counting HyperLogLog Sketches

As discussed above, we propose to maintain a counting HLL sketch for each individual column in a database. Whenever values in a table are inserted, updated, or deleted, these sketches have to be updated. Therefore, it is critical that the counting HLL sketch incurs a low runtime overhead. At the same time, high estimation accuracy is required for the sketch-correction framework, even if values are deleted frequently.

5.1.1 *Computational Performance*

We only evaluate the runtime cost of inserting values into a counting HLL sketch, since the delete operation is symmetrical to the insert operation (cf. Section 2). The well-known MurmurHash64A[1] hash function is used throughout our experiments, and the traditional HLL sketch serves as a baseline for comparison. Table 2 shows the CPU time required to compute sketches for all 10 columns of a table with 10 million rows on an Intel i7 7820X CPU. All values in the table are 8 byte integers, and we differentiate between column-wise processing, i. e., sketching one column after the other, and row-wise processing, where values are inserted into their corresponding sketches row-by-row.

Unsurprisingly, counting sketches are more expensive, but only by a factor of 2.5. In absolute terms, both approaches are very fast, requiring at most 1.3 ns per value for the traditional approach, and 3.7 ns per value for the proposed approach. Correspondingly, the bulk-load time of TPC-H in a fast in-memory system increased only by about 5% when computing sketches of all columns on the fly.

[1]available at https://github.com/aappleby/smhasher

The counting sketch profits from column-wise processing due to better cache utilization. As the sketches are larger, row-wise sketching risks trashing the L1 cache. For the simple sketches we would expect the same behavior, but, surprisingly, row-wise processing is actually faster. We suspect the reason for this to be the good out-of-order execution engine of the CPU, which can execute multiple updates concurrently due to the low number of instructions. On an older Haswell CPU, column-wise processing is faster for simple sketches, too, as one would expect.

5.1.2 *Estimation Accuracy*

As long as there are no deletions, the improved estimator due to Ertl [6] will produce exactly the same estimates on counting and traditional HLL sketches, because it requires only the maximum leading zero count in each bucket. The probabilistic counters used in the proposed sketch count the first 128 values exactly, i. e., without deletions, the probabilistic counter for a certain leading zero count has a value greater than zero if and only if we have observed at least one value with that leading zero count. Thus, the maximum number of leading zeros in each bucket is tracked exactly, and matches the value maintained by a traditional HLL sketch.

For this reason, we present an evaluation of the estimation accuracy on a workload that involves frequent deletions. We generate $2^{28} \approx 268\,000\,000$ random 64-bit values which are successively inserted into the counting HLL sketch. After each i inserts, some fraction r of these inserts is reverted by deleting the corresponding values from the sketch (cf. Figure 7). In our experiments, we choose $2^8 \leq i \leq 2^{24}$ and $0.125 \leq r \leq 0.875$. As a baseline, we successively insert the same 2^{28} values into a traditional HLL sketch without

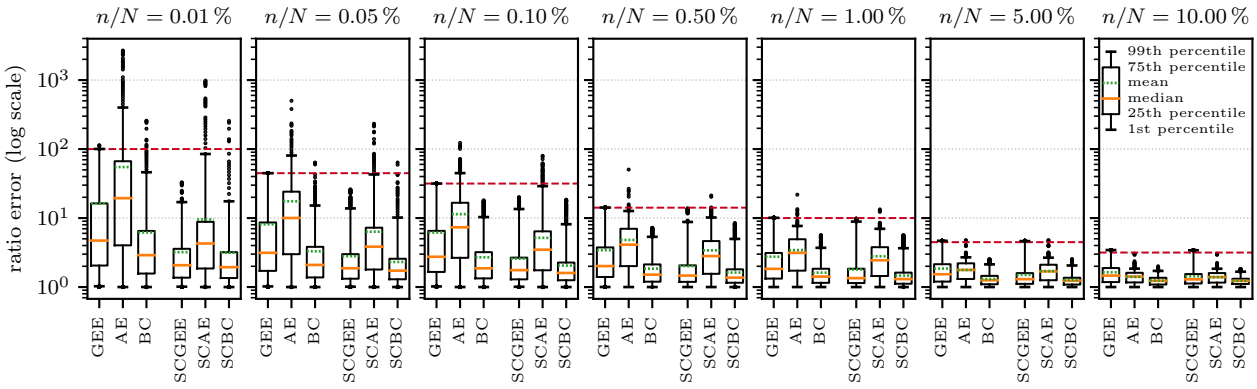


Figure 9: Distribution of the ratio error incurred by the estimators on synthetic data, for varying sampling fractions n/N. The dashed red line marks the theoretical error guarantee of GEE and SCGEE at $\sqrt{N/n}$.

Table 4: Mean and 99th percentile of the ratio error incurred by the estimators on synthetic data, for varying sampling fractions n/N. The best results for each sampling fraction are printed bold.

	GEE		AE		BC		SCGEE		SCAE		SCBC	
n/N	mean	99 %	mean	99 %	mean	99 %	mean	99 %	mean	99 %	mean	99 %
0.01 %	16.3	100.4	54.8	400.2	6.2	46.1	3.2	**17.0**	9.5	84.9	**3.1**	17.5
0.05 %	8.1	44.7	17.5	80.6	3.3	15.2	2.8	13.8	6.3	42.9	**2.3**	**10.2**
0.10 %	6.2	31.6	11.4	44.8	2.7	10.3	2.6	13.5	5.2	29.0	**2.0**	**8.1**
0.50 %	3.4	14.2	4.8	12.7	1.8	5.3	2.0	8.8	3.4	10.2	**1.6**	**5.0**
1.00 %	2.8	10.1	3.4	7.7	1.6	**3.7**	1.8	8.9	2.8	7.0	**1.5**	**3.7**
5.00 %	1.9	4.7	1.8	2.7	**1.3**	**2.1**	1.5	4.6	1.7	2.7	**1.3**	**2.1**
10.00 %	1.6	3.4	1.4	1.8	**1.2**	**1.7**	1.4	3.4	1.4	1.8	**1.2**	**1.7**

any deletions. The ratio error as defined in Section 3 is sampled in fixed intervals during the workload to obtain 2^{16} measurements per experiment.

As shown in Table 3, counting HLL sketches exhibit virtually identical estimation accuracy in comparison to the baseline. The displayed results are obtained by aggregating the ratio error measurements across all experiments. The mean ratio error of 1.13 matches the theoretically expected error of $1.04/\sqrt{m} = 13\,\%$ perfectly, and in the 99th percentile the ratio error is still only 1.34. The probabilistic counters employed by counting HLL sketches have expected correct behavior if the number of increment and decrement operations is sufficiently large. Therefore, we can indeed rely on counting and traditional HLL sketches to behave identically in terms of accuracy for a large number of operations.

Accordingly, we also investigate the mean ratio error for smaller cardinalities, by aggregating measurements with a true cardinality below a given value (cf. Figure 8). Our results show that the mean ratio error can be slightly greater for counting HLL sketches than for traditional HLL sketches. However, the difference is generally very small, and decreases as the maximum cardinality increases. Moreover, in most cases the mean ratio error actually lies below the theoretically expected value of 1.13 for both sketches.

We also experimented with repeating each insert or delete operation several times, in order to put additional strain on the probabilistic counters. However, we found that this has no visible impact on the overall estimation accuracy as the large number of inserts in our experiments causes many counters to take on values well beyond the range which is counted exactly anyway.

In summary, the proposed counting HLL sketches exhibit high estimation accuracy comparable to traditional HLL sketches. At the cost of negligible runtime overhead and moderately increased space consumption, the counting HLL sketch can retain high accuracy even in the presence of frequent deletions. Therefore we conclude that it is feasible to maintain a counting HLL sketch for each individual column in a database.

5.2 Multi-Column Estimators

A large-scale empirical evaluation of the proposed multi-column estimation framework is conducted on both real-world and synthetic data.

5.2.1 Data Sets

We chose four real-world data sets from the UCI Machine Learning repository[2], namely the census and forest cover type data sets used in the original evaluation of GEE and AE [3], as well as the poker hand and El Nino data sets. Moreover, we conduct experiments on the well-known IMDb data sets[3]. The forest cover type data set originally contains 55 attributes, of which 44 are binary. Since this results in an impractically high number of attribute combinations, we removed these binary attributes for our experiments. In addition, we generate 4 455 synthetic data sets with $N = 2^{20}$ rows and two columns that have varying correlation ($0 \leq \rho \leq 1$). The values in each individual column

[2]available at https://archive.ics.uci.edu/ml/
[3]available at https://www.imdb.com/interfaces/

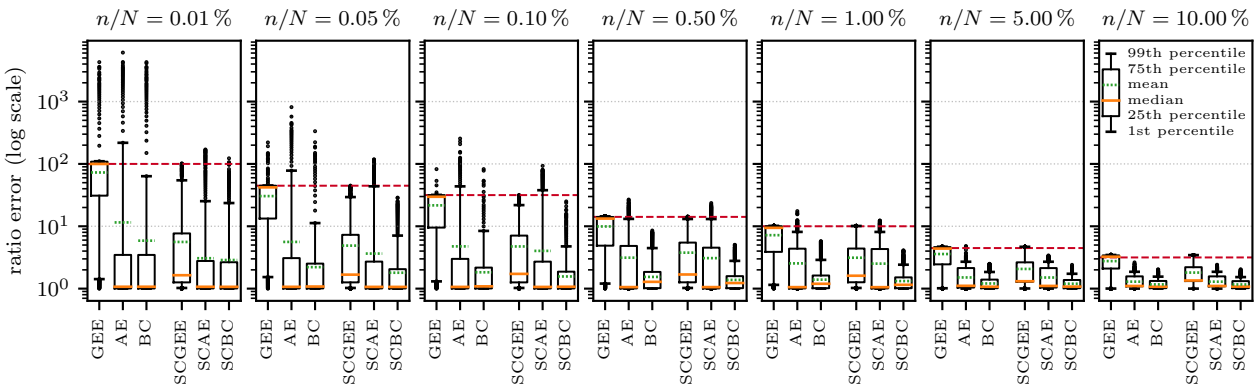


Figure 10: Distribution of the ratio error incurred by the estimators on real-world data, for varying sampling fractions n/N. The dashed red line marks the theoretical error guarantee of GEE and SCGEE at $\sqrt{N/n}$.

Table 5: Mean and 99th percentile of the ratio error incurred by the estimators on real-world data, for varying sampling fractions n/N. The best results for each sampling fraction are printed bold.

	GEE		AE		BC		SCGEE		SCAE		SCBC	
n/N	mean	99 %	mean	99 %	mean	99 %	mean	99 %	mean	99 %	mean	99 %
0.01 %	72.9	110.3	11.5	218.2	5.9	63.5	5.6	54.6	3.1	25.3	**2.9**	**23.6**
0.05 %	30.5	45.1	5.6	78.1	2.2	11.3	4.9	29.4	3.6	43.8	**1.8**	**7.1**
0.10 %	21.6	31.9	4.8	43.7	1.8	8.4	4.7	21.9	4.0	37.9	**1.6**	**4.8**
0.50 %	9.9	14.4	3.1	13.1	1.5	4.5	3.8	13.1	3.1	13.1	**1.4**	**2.8**
1.00 %	7.2	10.2	2.5	8.1	1.4	2.9	3.1	10.1	2.5	8.1	**1.3**	**2.4**
5.00 %	3.6	4.7	1.5	2.7	**1.2**	1.8	2.1	4.7	1.5	2.7	**1.2**	**1.7**
10.00 %	2.8	3.5	1.3	1.9	**1.2**	1.6	1.8	3.4	1.3	1.9	**1.2**	**1.5**

follow a generalized Zipfian distribution with varying population size ($2^4 \leq p \leq 2^{20}$) and skew coefficient ($0 \leq s \leq 4$).

The sample size n is selected per data set, so that a fixed sampling rate n/N is maintained ($0.01\,\% \leq n/N \leq 10.00\,\%$). For a given data set and sampling rate, ten different samples are drawn according to a uniform distribution on the rows, and the ratio error of the estimators is computed on all possible combinations of two or more attributes. By drawing ten different samples per data set, the impact of random fluctuations on our results is reduced. At the same time, it allows us to verify that the estimation approach is robust against small changes in the random sample.

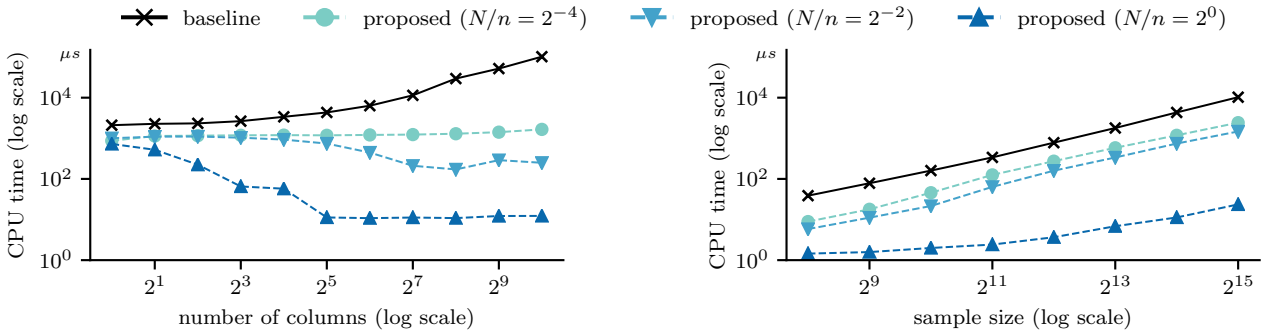
5.2.2 *Results*

Evaluation results are shown in Figure 9 and Table 4 for trials on the synthetic data sets, and in Figure 10 and Table 5 for trials on the real-world data sets. Note that the box plots use a logarithmic y-axis scale. Overall, the sketch-corrected variant SCBC of the proposed bound-corrected estimator BC consistently outperforms the other estimators, achieving the lowest mean ratio error in all cases. Furthermore, SCBC exhibits the lowest 99th percentile of the ratio error in all cases but one. Even with extremely small sampling rates, the estimates of SCBC remain sufficiently accurate in most cases to be useful in practice. In the following, we outline further key results in more detail.

First, we observe that GEE and AE generally provide rather poor estimates. In particular, AE struggles on the synthetic data sets, which is evident from the extremely high mean (up to 54.8) and 99th percentile (up to 400.2) of the ratio error (cf. Table 4). Upon closer inspection, we found that AE tends to widely underestimate the true cardinality when there is moderate skew in at least one column ($1 \leq s \leq 2$). In these cases, we can expect values to occur with a wide range of frequencies in the sample, which can cause the approximations employed by AE to become inaccurate [3]. At the same time, this leads to true cardinalities close to the value estimated by GEE, for which reason GEE performs better than AE on the synthetic data sets. The real-world data sets, on the other hand, seldom contain moderately skewed data, and AE consequently outperforms GEE in terms of the mean ratio error. However, the 99th percentile of its ratio error remains too large for practical purposes even for large sampling fractions (cf. Tables 4 and 5).

The proposed bound-corrected estimator BC can improve over GEE and AE substantially, even without sketch-correction. Especially for smaller sampling fractions, BC can provide much more accurate estimates, which underlines the robustness of the proposed approach. As shown in Figures 9 and 10, both the mean and the quantiles of its ratio error decrease sharply as the sampling fraction is increased. A mean ratio error below 2.0 can be achieved with a sampling fraction of only 0.05 % on the synthetic data sets, and only 0.01 % on the real-world data. Since the corrected bounds derived in Section 3 depend on possibly inaccurate estimates of expected values, there can be cases in which the maximum ratio error of BC exceeds that of GEE. However, this occurs only rarely in our experiments, indicating that the corrected bounds are usually sound.

Applying sketch-correction further improves the estimation accuracy of all estimators, and the best overall results are achieved by the sketch-corrected variant SCBC of the BC



(a) Computation time vs. column count for $n = 16384$

(b) Computation time vs. sample size for $C = 32$

Figure 11: CPU time (y-axis) required to compute frequency vectors in relation to the number of columns (a) and size (b) of samples (x-axis). The value of N/n has no impact on the performance of the baseline hash table implementation, and only a single graph is visible.

Table 6: Mean and percentiles of the absolute frequency vector computation time in milliseconds across all tested configurations (a). Additionally, the mean and percentiles of the speedup over the baseline approach are shown (b).

(a) Absolute computation time

	mean	1 %	25 %	50 %	75 %	99 %
baseline	9.84	0.02	0.17	0.95	5.28	200.68
proposed	0.38	0.00	0.01	0.05	0.35	2.81

(b) Speedup

	mean	1 %	25 %	50 %	75 %	99 %
speedup	418.5	1.6	3.4	9.3	58.8	8738.8

estimator. In particular, SCBC outperforms BC in all cases, allowing us to conclude that the proposed bound-correction and sketch-correction approaches are orthogonal to some degree. We observed that SCBC mainly improves over BC for small cardinalities, which is consistent with theoretical considerations. If all individual columns contain only few distinct values, sketch-correction can derive tight bounds on the true number of distinct values. In particular, the cardinality of the cross-product of the distinct values in the individual columns is small, i. e. the upper bound employed by SCBC is more accurate than the original upper bound used by BC. On the other hand, the improved estimation bounds employed by BC deviate from the bounds employed by GEE mainly for large cardinalities, where sketch-correction can not provide a useful upper bound.

The effectiveness of sketch-correction decreases for large sampling fractions, and there are cases in which no further improvement can be achieved. This is to be expected, however, since a larger sampling fraction allows the estimators to infer more accurate information about the data distribution themselves, without having to rely on sketch-correction. Depending on the cardinalities of the individual columns, it can even occur that all distinct values are present in a large sample of the relation, in which case sketch-correction cannot contribute any significant further information.

As outlined above, we generate the synthetic data sets with varying domain sizes and data skew in the individual attributes, and varying correlation between the attributes. We observed that all estimators produce similarly accurate estimates regardless of the correlation between attributes. As expected from our theoretical considerations (cf. Section 3), GEE and AE struggle if the domain size is large, while BC performs well across the entire tested range. Moderate data skew in at least one attribute causes accuracy to decrease for all estimators, although the effect is much less pronounced for the BC estimator than for GEE and AE.

Finally, we note that the maximum ratio error of GEE exceeds its theoretical error guarantee for low sampling fractions on the real-world data sets (cf. Figure 10). We determined that this is caused by exceedingly small samples, which can contain as few as 4 rows on the census data set, for example. Thus, a simple remedy in practice would be to set a sufficiently large minimum sample size. Apart from such edge cases, the ratio errors of GEE and SCGEE are bounded by $\sqrt{N/n}$ as expected from their theoretical analysis.

5.3 Frequency Vector Computation

The proposed approach for computing frequency vectors is evaluated only on synthetic data, so that its asymptotic behavior can be studied under controlled conditions. Samples are generated with $2^8 \leq n \leq 2^{15}$ rows and $2^0 \leq C \leq 2^{10}$ columns according to a uniform distribution on $\{1, \ldots, N\}$, where $2^{-8} \leq N/n \leq 2^2$ to simulate varying numbers of distinct values. We measure the CPU time required by the proposed approach to compute frequency vectors on the CPU introduced above, in comparison to a baseline hash table implementation as outlined in Section 4. We noticed that, as expected, the value of N/n has no visible influence on the performance of the baseline implementation, and we do not report separate baseline results for different values of N/n.

The proposed approach consistently improves over the baseline, with a minimum and median speedup of 1.4× and 9.3×, respectively. However, much larger speedups are possible depending on the data at hand, as illustrated by the 75th and 99th percentiles at approximately 59× and 8700×, respectively (cf. Table 6b). This large variability is caused by the different asymptotic behavior of the baseline and proposed approaches, as illustrated in Figure 11 (note again the logarithmic scale on the y-axis). While the figure displays

only selected results, they are representative for the behavior across all experiments.

Computation time scales approximately linearly in the sample size for all approaches, as well as in the column count for the baseline implementation. However, it remains constant or even decreases with increasing number of columns for the proposed approach, because more singleton rows can be pruned. For the same reason, computation time is reduced dramatically if there are many singleton values in each column, i. e. N/n is large. At the same time, Figure 11a shows that the recursive approach improves over the baseline even if there are few columns and singletons. Note that we resized the hash table suitably before taking our measurements, so that no rehashing was necessary during our experiments. This illustrates that despite the $O(1)$ complexity of inserting and retrieving values into a hash table, the constant overhead of these operations is large enough to negatively impact computation time (cf. Section 4).

In absolute terms, the proposed approach offers excellent performance across all tested configurations (cf. Table 6a), and requires at most 3.4 ms to compute a frequency vector even on an extremely large sample with 32 768 rows and 1 024 columns. The estimators themselves are very cheap to compute, typically taking less than 5 μs to produce an estimate for a given frequency vector.

6. RELATED WORK

Being a key problem of query optimization, cardinality estimation algorithms have been studied extensively in the literature [4, 12]. Broadly, such algorithms can be categorized into *sampling-based* and *sketch-based* approaches [19].

Algorithms in the first category examine only a small sample of a relation in order to produce an estimate. While this offers attractive performance and trivially allows for cardinality estimates over arbitrary attribute combinations, any purely sampling-based approach has provably poor accuracy [3]. Consequently, many approaches focus on improving the quality of the samples using auxiliary information obtained, for instance, from a full relation scan [5, 9], existing index structures [18], or query feedback [16]. Oracle has recently presented an adaptive scheme which iteratively builds a sample to provide confidence intervals around the estimated cardinalities [27]. The main drawback of these approaches is that they may produce different samples for different attribute combinations. Thus, an exponential number of samples has to be maintained in order to avoid expensive sample computations during query optimization.

Sketch-based approaches, on the other hand, hash each row in the relation once and build a small fixed-size synopsis from which the cardinality can then be estimated. Arguably the most prominent representative of this class of algorithms is HyperLogLog [7, 14], which provides much more accurate estimates than sampling-based approaches [12]. One can also sketch only a sample of a relation, which improves computation speed further without severely impacting accuracy [23]. However, sketches on individual attributes can not easily be combined, since by design there is no clear relationship between the hash values of multiple individual attribute values and of the corresponding attribute combination. Accordingly, an exponential number of sketches has to be stored in order to provide estimates for arbitrary attribute combinations.

Since it is obviously not feasible to maintain an exponential number of samples or sketches in practice, current systems frequently assume the individual attributes to be independent [17]. However, this assumption is often unfounded on real-world data which may lead to large estimation errors [26]. More accurate cardinality estimates could be derived from multi-dimensional histograms or wavelets [4, 24], as well as from information about soft functional dependencies [15]. Unfortunately, these synopses are prohibitively expensive to construct and maintain in the presence of updates and deletions [4, 22]. A recent approach estimates the inclusion coefficient between columns using only single-column sketches [21], which could be used to infer the number of distinct tuples if all attributes have equal domains. As we propose in this paper, sketches and sampling can be combined to provide accurate estimates for arbitrary attribute combinations with low overhead. A similar approach has been implemented successfully for selectivity estimation [20, 25], but to the best of our knowledge there is no previous work on combining sketches and sampling for cardinality estimation.

Traditional HyperLogLog sketches, however, are not suitable for this purpose since they do not support updates and deletions. Flajolet and Martin themselves point out that a possible solution is to maintain a counter for each possible bucket value [8], which has been adopted in recent work [21]. However, this results in an overly large memory footprint if sketches should be maintained for each individual column (cf. Section 2). Deletions are also inherently encountered in the sliding window model, where old observations have to be removed from the sketch when new observations arrive [2]. In these cases, it is known exactly at which time an element is going to be deleted, allowing for more specialized solutions which cannot be adopted in a general-purpose database scenario.

Finally, the proposed recursive algorithm for computing frequency vectors is based on a string partition refinement algorithm proposed by Cai and Paige [1]. Their algorithm is formulated without any recursion, and maintains auxiliary data structure instead. Henglein developed a generic discrimination framework which encompasses recursive partition refinement similar to the proposed approach [13].

7. CONCLUSION

Query optimizers require accurate cardinality estimates in order to find efficient execution plans. We showed that existing sketch-based approaches are highly accurate, but they require exponential space to produce estimates for arbitrary combinations of attributes. Furthermore, they do not support updates and deletions out of the box. Sample-based approaches, on the other hand, can produce such estimates but have provably poor accuracy. We presented a novel estimation framework, which employs highly accurate sketched estimates over individual columns to correct sample-based estimates over arbitrary combinations of attributes. We developed novel counting HyperLogLog sketches which support update and delete operations with little additional state, and an efficient algorithm for computing value frequencies in a sample, which are required for estimation. Our approach consistently improves over previous sample-based approaches, producing highly accurate estimates on synthetic and real-world data sets, while keeping the estimation overhead negligible.

8. REFERENCES

[1] J. Cai and R. Paige. Look ma, no hashing, and no arrays neither. In *POPL*, pages 143–154, 1991.

[2] Y. Chabchoub and G. Hébrail. Sliding HyperLogLog: Estimating cardinality in a data stream over a sliding window. In *ICDMW*, pages 1297–1303, 2010.

[3] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya. Towards estimation error guarantees for distinct values. In *SIGMOD*, pages 268–279, 2000.

[4] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[5] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample + Seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, pages 679–694, 2016.

[6] O. Ertl. New cardinality estimation algorithms for HyperLogLog sketches. *CoRR*, abs/1702.01284, 2017.

[7] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. In *Conference on Analysis of Algorithms*, pages 137–156, 2007.

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

[9] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, pages 541–550, 2001.

[10] L. A. Goodman. On the Estimation of the Number of Classes in a Population. *The Annals of Mathematical Statistics*, 20(4):572–579, April 1949.

[11] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB*, pages 311–322, 1995.

[12] H. Harmouch and F. Naumann. Cardinality estimation: An experimental survey. *PVLDB*, 11(4):499–512, 2017.

[13] F. Henglein. Generic top-down discrimination for sorting and partitioning in linear time. *J. Funct. Program.*, 22(3):300–374, 2012.

[14] S. Heule, M. Nunkesser, and A. Hall. HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *EDBT*, pages 683–692, 2013.

[15] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647–658, 2004.

[16] P. Larson, W. Lehner, J. Zhou, and P. Zabback. Cardinality estimation using sample views with quality assurance. In *SIGMOD*, pages 175–186, 2007.

[17] V. Leis, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *PVLDB*, 9(3):204–215, 2015.

[18] V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR Online Proceedings*, 2017.

[19] A. Metwally, D. Agrawal, and A. El Abbadi. Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic. In *EDBT*, pages 618–629, 2008.

[20] M. Müller, G. Moerkotte, and O. Kolb. Improved selectivity estimation by combining knowledge from sampling and synopses. *PVLDB*, 11(9):1016–1028, 2018.

[21] A. Nazi, B. Ding, V. R. Narasayya, and S. Chaudhuri. Efficient estimation of inclusion coefficient using HyperLogLog sketches. *PVLDB*, 11(10):1097–1109, 2018.

[22] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB*, 8(10):1082–1093, 2015.

[23] F. Rusu and A. Dobra. Sketching sampled data streams. In *ICDE*, pages 381–392, 2009.

[24] M. Shekelyan, A. Dignös, and J. Gamper. DigitHist: A histogram-based data summary with tight error bounds. *PVLDB*, 10(11):1514–1525, 2017.

[25] X. Yu, N. Koudas, and C. Zuzarte. HASE: A hybrid approach to selectivity estimation for conjunctive predicates. In *EDBT*, pages 460–477, 2006.

[26] X. Yu, C. Zuzarte, and K. C. Sevcik. Towards estimating the number of distinct value combinations for a set of attributes. In *CIKM*, pages 656–663, 2005.

[27] M. Zaït, S. Chakkappen, S. Budalakoti, S. R. Valluri, R. Krishnamachari, and A. Wood. Adaptive statistics in Oracle 12c. *PVLDB*, 10(12):1813–1824, 2017.

APPENDIX

A. PROOF OF THEOREM 1

In this appendix, we present a proof of

THEOREM 1 (REVISITED). *Consider a table with N rows containing D distinct tuples. Suppose we draw a sample of n rows uniformly at random with replacement, and let f_1 denote the observed number of singleton tuples in this sample. Then, the following inequality holds*

$$\mathbb{E}(f_1) \leq \begin{cases} n \cdot (1-1/D)^{n-1} & \text{if } D \geq n, \\ D \cdot (1-1/n)^{n-1} & \text{otherwise.} \end{cases}$$

PROOF. As stated in Section 3.2, the expected number of singleton tuples is given by

$$\mathbb{E}(f_1) = \sum_{k=1}^{D} nP_k(1-P_k)^{n-1}, \tag{3}$$

where $P_k = N_k/N$ denotes the relative frequency of the k-th distinct attribute combination in the entire table. It is useful to interpret this expected value as a function f of the vector $\mathbf{P} = (P_1, \ldots, P_D)$ of relative frequencies, i. e.

$$f(\mathbf{P}) = \sum_{k=1}^{D} g(P_k), \tag{19}$$

with $g(P_k) = nP_k(1-P_k)^{n-1}$. Note that we require $0 < P_k \leq 1$ for all k, since $1 \leq N_k \leq N$ by definition. The

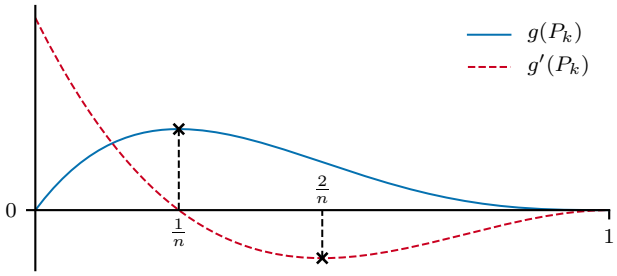


Figure 12: Qualitative behavior of the function g and its derivative g' used in the proof of Theorem 1.

behavior of g on this interval is essential in the remainder of this proof. Its first and second derivatives are given by

$$g'(P_k) = n(1 - nP_k)(1 - P_k)^{n-2}, \text{ and} \tag{20}$$

$$g''(P_k) = n(n-1)(nP_k - 2)(1 - P_k)^{n-3}. \tag{21}$$

From this, we can conclude that $g'(P_k)$ has a zero at $P_k = 1/n$ corresponding to a maximum of g. Furthermore, it can be verified that $g'(P_k) > 0$ for $P_k < 1/n$, and $g'(P_k) < 0$ for $P_k > 1/n$. Finally, one can confirm that $g'(P_k)$ is strictly monotonic decreasing for $P_k < 2/n$ and strictly monotonic increasing for $P_k > 2/n$ (cf. Figure 12). The remainder of the proof is now a case-by-case analysis of the proposition.

Case 1: $D < n$ In this case, recall that $g(P_k)$ has a maximum at $P_k = 1/n$. Hence, we can deduce

$$\mathbb{E}(f_1) \leq \sum_{k=1}^{D} g\left(\frac{1}{n}\right) = D\left(1 - \frac{1}{n}\right)^{n-1}. \tag{22}$$

While this upper bound trivially holds for $D \geq n$ as well, we aim to prove a tighter bound in that case.

Case 2: $D \geq n$ In order to derive this tighter bound, we maximize f subject to the constraint $\sum_{k=1}^{D} P_k = 1$. Note that the P_k can only take on discrete values in the original problem formulation. By relaxing this restriction and solving the corresponding continuous optimization problem, we obtain an upper bound on the solution of the discrete optimization problem. Thus, we introduce a Lagrange multiplier λ and define

$$\mathcal{L}(\mathbf{P}, \lambda) = \sum_{k=1}^{D} g(P_k) - \lambda \cdot \left(\sum_{k=1}^{D} P_k - 1\right). \tag{23}$$

Since a maximum of f must occur at a critical point of the Lagrange function $\mathcal{L}$, a necessary condition for optimality is $\nabla\mathcal{L}(\mathbf{P}, \lambda) = 0$, i. e.

$$\begin{pmatrix} g'(P_1) - \lambda \\ \vdots \\ g'(P_D) - \lambda \\ \sum_{k=1}^{D} P_k - 1 \end{pmatrix} = 0. \tag{24}$$

From this, one can immediately conclude that $\mathbf{P}$ is a critical point of $\mathcal{L}$ if and only if it satisfies the constraint $\sum_{k=1}^{D} P_k = 1$, and the derivatives $g'(P_k)$ are equal for all $k \in \{1, \ldots, D\}$. For $D \geq n$, the only such critical point which satisfies $0 < P_k \leq 1$ occurs at $P_1 = \ldots = P_D = 1/D$, which we prove by contradiction. Hence, let us assume that there exist $P_1, \ldots, P_D$ with $P_i \neq 1/D$ for some index i and $g'(P_1) = \ldots = g'(P_k)$. Furthermore, assume without loss of generality that $P_i > 1/D$.

Then, due to the constraint $\sum_{k=1}^{D} P_k = 1$, there exists an index j so that $P_j < 1/D \leq 1/n$. In Figure 12, P_i and P_j are thus two distinct points on the x axis, where P_j surely lies to the left of the zero of $g'(P_k)$ at $P_k = 1/n$. Intuitively, it is clear that this implies $g'(P_i) \neq g'(P_j)$. Formally, recall that $g'(P_j)$ is strictly monotonic decreasing for $P_j < 1/n$, hence $g'(P_j) > g'(1/D)$. On the other hand, $g'(P_i) \leq g'(1/D)$ surely holds, since $g'(P_i)$ is strictly monotonic decreasing for $1/D < P_i < 1/n$ and $g'(P_i) \leq 0 \leq g'(1/D)$ for $P_i \geq 1/n$. In summary, we obtain $g'(P_i) \neq g'(P_j)$ in contradiction to the assumption that $P_1, \ldots, P_D$ is a solution. Therefore, we conclude that $P_1 = \ldots = P_D = 1/D$ is indeed the only critical point of $\mathcal{L}$.

It can easily be verified that this critical point does not correspond to a minimum or saddle point of f. Thus, we obtain

$$\mathbb{E}(f_1) \leq \sum_{k=1}^{D} g\left(\frac{1}{D}\right) = n \cdot \left(1 - \frac{1}{D}\right)^{n-1}. \tag{25}$$

□

B. PROOF OF THEOREM 2

In the following, we present a proof of

THEOREM 2 (REVISITED). *Consider a table with N rows containing D distinct tuples. Suppose we draw a sample of n rows uniformly at random with replacement, and let d denote the observed number of distinct tuples in this sample. Then, the following inequality holds*

$$\mathbb{E}(d) \geq D - D \cdot (1 - 1/N)^n.$$

PROOF. As outlined in Section 3.2, the expected number of distinct tuples in the sample is given by

$$\mathbb{E}(d) = D - \sum_{k=1}^{D} (1 - P_k)^n. \tag{4}$$

Hence, $\mathbb{E}(d)$ is minimal if $\sum_{k=1}^{D} (1 - P_k)^n$ is maximized. Let

$$h(P_k) = (1 - P_k)^n, \tag{26}$$

and consider the derivative

$$h'(P_k) = -n(1 - P_k)^{n-1}. \tag{27}$$

For $0 \leq P_k \leq 1$ we have $h'(P_k) < 0$, thus $h(P_k)$ is strictly monotonic decreasing in this interval. As we require each of the distinct tuples to occur at least once in the table, we know that $P_k \geq 1/N$, and therefore

$$E(d) \geq D - \sum_{k=1}^{D} h\left(\frac{1}{N}\right) = D - D \cdot \left(1 - \frac{1}{N}\right)^n. \tag{28}$$

□