Workload-Driven Antijoin Cardinality Estimation

FLORIN RUSU and ZIXUAN ZHUANG, University of California Merced
MINGXI WU, GraphSQL
CHRIS JERMAINE, Rice University

Antijoin cardinality estimation is among a handful of problems that has eluded accurate efficient solutions amenable to implementation in relational query optimizers. Given the widespread use of antijoin and subset-based queries in analytical workloads and the extensive research targeted at join cardinality estimation – a seemingly related problem – the lack of adequate solutions for antijoin cardinality estimation is intriguing. In this paper, we introduce a novel *sampling-based estimator* for antijoin cardinality that – unlike existent estimators – provides sufficient accuracy and efficiency to be implemented in a query optimizer. The proposed estimator incorporates three novel ideas. First, we use prior workload information when learning a *mixture superpopulation model* of the data offline. Second, we design a *Bayesian statistics framework* that updates the superpopulation model according to the live queries, thus allowing the estimator to adapt dynamically to the online workload. Third, we develop an efficient algorithm to sample from a hypergeometric distribution in order to generate *Monte Carlo trials*, without explicitly instantiating either the population or the sample. When put together, these ideas form the basis of an efficient antijoin cardinality estimator satisfying the strict requirements of a query optimizer, as shown by the extensive experimental results over synthetically-generated as well as massive TPC-H data.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*query processing*; G.3 [**Probability and Statistics**]: *distribution functions, multivariate statistics, probabilistic algorithms*

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: antijoin operator; query optimization; sampling; Monte Carlo; Bayesian statistics

ACM Reference Format:
Rusu, F., Zhuang, Z., Wu, M., and Jermaine, C. 2015. Workload-Driven Antijoin Cardinality Estimation. *ACM Trans. Datab. Syst.* 0, 0, Article 0 (0), 37 pages.
DOI : http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Cost-based query optimization [Chaudhuri 1998] relies fundamentally on the ability to provide an accurate estimate for the number of tuples generated by a particular query plan operator. The cardinality estimate is derived using simple arithmetic formulae involving relation sizes and the number of distinct elements proposed more than three decades ago [Selinger et al. 1979]. This is often the case despite the tremendous amount of work on statistical synopses in databases [Cormode et al. 2012]. Out of the many techniques proposed, histograms [Ioannidis 2003] are the synopsis most widely adopted for cost-based query optimization, for several reasons. A single-column histogram requires very small space to maintain. Histograms are effective in cost estimation for single table selection. They are particularly accurate for highly selective queries, compared to other alternatives. Nonetheless, due to the implicit locality assumption, i.e., adjacent attribute values have similar frequency and thus can be grouped together, it has also long been acknowledged that histograms are

Authors' addresses: F. Rusu and Z. Zhuang, School of Engineering, University of California Merced, 5200 N Lake Rd., Merced, CA 95343; M. Wu, GraphSQL Inc., 883 N. Shoreline Blvd., Mountain View, CA 94043; C. Jermaine, Computer Science Department, Rice University, 6100 Main, Houston, TX 77005-1827. Email: frusu@ucmerced.edu; zzhuang@ucmerced.edu; send2mingxiwu@gmail.com; cmj4@cs.rice.edu.

DOI : http://dx.doi.org/10.1145/0000000.0000000

not very useful for join-size estimation since they cannot capture correlations between columns. The situation is particularly bad when there are multiple join levels since the errors propagate exponentially to the top of the query tree [Ioannidis and Christodoulakis 1993].

For this and other reasons, a significant amount of research effort has been directed at developing alternative synopsis structures for query optimization, chief among them being sampling [Cormode et al. 2012], due to its generality and applicability to a large class of SQL operators. One of the classical complaints against sampling – samples have to be large in order to produce low-variance estimates – is much less significant in the modern era when even the most inexpensive machines have gigabytes of memory. The common viewpoint that samples are useless for queries over infrequent attribute values is also less relevant for modern Big Data analytics applications, where most queries touch a large fraction of the data. Moreover, there are scenarios where sampling is the only alternative to compute estimates in the query optimizer. For example, Oracle temporary tables[1] store only volatile data. No statistics, e.g., number of distinct elements [Gibbons 2001] or histograms, are collected for temporary tables whatsoever. Nonetheless, temporary tables can be used in queries just like any other table. The Oracle query optimizer uses samples extracted dynamically at runtime in order to estimate the cardinality of operators applied to temporary tables.

While it is well understood how to use samples for join size estimation [Chaudhuri et al. 1999], there is little work on sampling-based antijoin cardinality estimation—a seemingly related problem. The classic example of an antijoin or subset-based query is a SQL nested statement involving two queries connected by `NOT IN` or `NOT EXISTS`. Given the dearth of applicable research on this topic in the literature, the assumption seems to be that having synopses that can be used to accurately place antijoin operators in a query plan has very limited applicability. However, this assumption makes little sense since subset-based queries are quite common, particularly in decision support applications, e.g., 10 of the 22 TPC-H[2] queries involve a subset-based operator. Furthermore, the placement of the antijoin operator can have a profound effect on the cost of executing a query plan, as the motivating example given below shows. With respect to generality, we point out that the antijoin operator – and the estimator proposed in this paper, in particular – is applicable to a fairly large class of problems. For example, `DISTINCT` queries and (anti)join similarity queries can be rewritten using the antijoin operator, as shown in Section 2.

Motivating example. Consider a standard TPC-H query "What is the total supply cost for parts less expensive than $700 supplied in 2007 by suppliers who did not supply any part in 2006?" having the corresponding SQL statement:

```
SELECT SUM(ps_supplycost*l1.l_quantity)
FROM partsupp ps, lineitem l1
WHERE ps_suppkey=l1.l_suppkey AND ps_partkey=l1.l_partkey AND
  year(l1.l_shipdate)=2007 AND ps_supplycost < 700 AND
  NOT EXISTS (                                                    (1)
    SELECT * FROM lineitem l2
    WHERE l2.l_suppkey=l1.l_suppkey AND
      year(l2.l_shipdate)=2006
  )
```

Figure 1 sketches three alternative left-deep evaluation plans for this query. The plan in Figure 1a is intuitive since it follows directly from the SQL statement. It first evaluates the outer query and for every generated tuple it iterates over the inner query to filter out only those tuples without any match. This requires a simple modification to a nested-loop join algorithm that breaks the inner loop as soon as a match is found. The same modification can be applied to other join algorithms, e.g., hash

[1] http://www.orafaq.com/wiki/Temporary_table
[2] www.tpc.org/tpch/

join or sort-merge join. The job of the query optimizer is to determine based on available synopses which of the three modified join algorithms, i.e., antijoin access paths [Colgan 2010; Schrag 2005], is the most efficient.

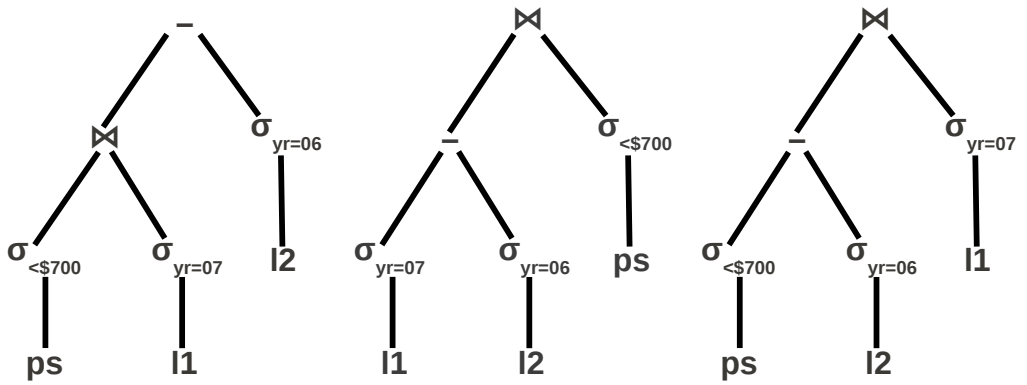


Fig. 1: Alternative left-deep execution plans for query (1). "–" represents the antijoin operator.

In many database servers, e.g., MySQL and PostgreSQL, plan 1a is the only available since plans 1b and 1c require sub-query unnesting for the optimizer to even consider them. Essentially, query (1) is rewritten as the following equivalent query with the sub-query in the `FROM` clause instead of `WHERE`:

```
SELECT SUM(ps_supplycost*l1.l_quantity)
FROM partsupp ps, lineitem l1,
   (SELECT * FROM lineitem WHERE year(l_shipdate)=2006) l2
WHERE ps_suppkey=l1.l_suppkey AND ps_partkey=l1.l_partkey AND
   year(l1.l_shipdate)=2007 AND ps_supplycost < 700 AND
   l1.l_suppkey antijoin= l2.l_suppkey
```
(2)

where `antijoin=` is the notation for the antijoin access path. In the unnested query, the optimizer can include the antijoin access path in the join ordering algorithm and generate plans that interchange the order of join and antijoin. Plan 1b is derived directly from the unnested query by switching the order of the antijoin and join predicates, while plan 1c is inferred from the predicate `ps_suppkey=l1.l_suppkey`. These plans have the potential to be considerably more efficient than the join in plan 1a because they can produce far fewer tuples. This is the case when the suppliers with line items in 2006 and 2007 are (almost) the same (plan 1b) or when the majority of the suppliers who sell parts cheaper than $700 have line items in 2006 (plan 1c). The query optimizer has to identify these situations based on data statistics. This requires antijoin cardinality estimation since the antijoin access path is not at the root of the plan anymore.

Antijoin cardinality estimation turns out to be extremely unreliable with existing synopses, e.g., histograms, because they aggregate data and lose tuple identity. For example, consider the antijoin in plan 1b, which has to identify suppliers with orders in 2007 but without any order in 2006. Assume that a histogram is available for `year(l_shipdate)`, having a separate bucket for each year. The optimizer makes the assumption that the line items in a bucket are uniformly distributed across all the suppliers. Moreover, it assumes that the suppliers in 2006 are a subset of the suppliers in 2007. Thus, the only time a strictly positive estimate is returned is when the number of 2006 line items is

smaller than the number of suppliers and there are more line items in 2007 than in 2006. Otherwise, the estimate is 0. There are at least two problems with this estimator. First, it does not provide any information in the usual case, i.e., when the number of tuples in `lineitem` is considerably larger than in `supplier`. Plan 1b is always assigned a lower cost than plan 1a in this situation. For this reason, many query optimizers do not even consider the unnested plans. Second, the 0 estimate might prove to be extremely inaccurate in certain scenarios, e.g., when the 2007 line items are uniformly distributed across suppliers, while the majority of the 2006 line items are provided by a small number of suppliers. Although end-biased multi-dimensional histograms [Poosala et al. 1996] can provide better accuracy, they are rarely implemented in query optimizers.

Problem statement & contributions. The problem we consider in this paper is sampling-based antijoin cardinality estimation. We explore *how a set of prior (anti)join queries and their results can be used to devise better sampling-based antijoin cardinality estimators for query optimization*. To this end, we introduce a novel sampling-based estimator for antijoin cardinality that – unlike existent estimators [Acharya et al. 1999; Jermaine et al. 2005; Joshi and Jermaine 2009; Yu et al. 2013] – provides the required generality and efficiency to be implemented in a query optimizer. The proposed estimator builds upon the biased estimator introduced in [Joshi and Jermaine 2009] by integrating query workload information using a Bayesian statistics framework [Gelman et al. 2003] and by designing an efficient algorithm to sample from a hypergeometric distribution. The use of prior workload information eliminates the time-consuming requirement to build a model of the data at query time – for each query – while the sampling algorithm replaces the naive procedure used in [Joshi and Jermaine 2009]. Our specific contributions can be summarized as follows:

— We develop methods that allow the proposed estimator to learn a superpopulation model of the data from prior workload information offline.
— We design a Bayesian statistics framework that updates the model according to the live queries, thus allowing the estimator to adapt dynamically to the online workload.
— We develop an efficient algorithm to sample from a hypergeometric distribution in order to generate Monte Carlo trials without instantiating either the population or the sample.
— We perform an extensive experimental study to evaluate the accuracy and efficiency of the proposed estimator over synthetic as well as massive TPC-H data.

Organization. We start by formalizing the problem in Section 2 and discussing existent antijoin cardinality estimators in Section 3. We provide a high-level description of the proposed estimator in Section 4, followed by a detailed presentation of the approach in Section 5, 6, 7, and 8, respectively. Section 9 contains the detailed experimental evaluation of the proposed estimator. Related work is presented in Section 10, while Section 11 concludes the paper.

2. THE ANTIJOIN SIZE-ESTIMATION PROBLEM

Antijoin cardinality has the following SQL form in the case of two relations R – the *outer* relation – and S—the *inner* relation:

```
SELECT COUNT(*) FROM R r
WHERE NOT EXISTS                                    (3)
   (SELECT * FROM S s WHERE f(r,s))
```

where f is an arbitrary boolean predicate applicable to a pair of records from R and S, respectively. If f(r, s) evaluates to true then r and s are said to match. Essentially, the antijoin cardinality problem asks for the number of records in R that do not have a match in S according to predicate f. When f is defined as equality between r and s, i.e., $f(r, s) \mapsto r = s$, antijoin cardinality reduces to computing the size of the set (bag) difference between R and S, i.e., $|R - S|$. Thus, we use the notation $|R -_f S|$ to indicate antijoin cardinality throughout the paper. For simplicity, we do not consider selection predicates over individual records from R and S explicitly because they can be encoded within the predicate f. Specifically, query (3) can handle an arbitrary boolean selection predicate $P_R(\mathtt{r})$ on

tuples $r \in R$ by having f(r, s) evaluate to true if $P_R(\mathtt{r})$ is false. Similarly, query (3) can handle an arbitrary selection predicate $P_S(\mathtt{s})$ on tuples $s \in S$ by having f(r, s) evaluate to false when $P_S(\mathtt{s})$ is false and $P_R(\mathtt{r})$ is true.

While we consider only antijoin queries as given in Eq. (3), it is important to notice that other types of SQL queries, e.g., `NOT IN`, `DISTINCT`, semijoin, (anti)join similarity, can be written in the form given in Eq. (3). For example, the number of distinct elements in R, i.e., COUNT DISTINCT, can be expressed as Eq. (3) by simply replacing S with R in the inner sub-query and setting f to $f(r, r') \mapsto (r.id < r'.id) \wedge (r = r')$, where *id* is a unique identifier, i.e., key, corresponding to each record. (Anti)join similarity requires only the definition of the appropriate similarity predicate in f. These examples prove that the antijoin query is fairly general and solutions for the antijoin operator are applicable to a large class of computations, beyond subset-based queries.

The specific problem we consider in this paper is sampling-based antijoin cardinality estimation. Given random samples $R' \subseteq R$ and $S' \subseteq S$, respectively, we are required to guess the answer to query (3) using only the samples. Moreover, we have to characterize the accuracy of the guess from the samples R' and S'. It is important to notice that the new estimators we introduce in this paper are applicable even when R and/or S are themselves the result of a SQL sub-query. For example, a sample R' can be extracted when R is the result of a join query using the well-known join synopsis technique [Acharya et al. 1999]. We do not consider this extended case further in the paper.

3. SAMPLING-BASED ESTIMATORS

We first discuss antijoin cardinality estimation when estimates are based on sampled data only. In this section, we discuss some of the difficulties associated with the problem, as well as existing sampling-based estimators and their benefits and drawbacks, respectively. We show that – due to the hardness of the problem – standard sampling-based strategies yield unsatisfactory results – both in terms of accuracy as well as execution time – while advanced sampling strategies, such as correlated samples, require a separate synopsis for every antijoin predicate. Nonetheless, these serve as background for the workload-aware estimator we propose in this paper.

3.1. How Do Query Optimizers Estimate Antijoin Cardinality?

Standard query optimizers do not use sampling for cardinality estimation. While they implement histograms, it is often the case that cardinality is estimated based solely on the size of the relations and the number of distinct elements. For example, the most common estimator for the size of join $|R \bowtie S|$ is $min(d_R, d_S) \cdot \frac{|R|}{d_R} \cdot \frac{|S|}{d_S}$, where d_R and d_S are the number of distinct elements in R and S, respectively [Swami and Schiefer 1994]. Two important assumptions are made in the case of this simple estimator—uniformity and containment. The values of the join attribute are assumed to be uniformly distributed over the domain of the attribute. And the set of values in the join column with the smaller column cardinality are assumed to be a subset of the set of values in the join column with the larger column cardinality. Although these are strong assumptions that rarely hold in practice, they are ubiquitously used in almost all the relational query optimizers. Consequently, the estimation error can be many orders of magnitude in some situations.

Following the same assumptions, we can immediately derive an estimator for antijoin cardinality $|R -_f S|$ as $(d_R - d_S) \cdot \frac{|R|}{d_R}$. We assume that $d_R \geq d_S$ in this case. The containment property tells us that all the distinct values in S are a subset of the distinct values in R, while uniformity tells us that each distinct value appears the same number of times. This is the standard non-sampling estimator to compute antijoin cardinality in query optimizers [Selinger et al. 1979]. When histograms are available, the formula is applied for every bucket and the results are summed-up in order to get the final estimator. As with the size of join estimator, the effect of the assumptions in practice can be catastrophic, with orders of magnitude errors being common.

3.2. A Naive Estimator

It is not hard to imagine what a naive sampling estimator for the antijoin cardinality problem might look like:

— Take a $p \cdot 100\%$ sample (denoted R') from R.
— Take a $q \cdot 100\%$ sample (denoted S') from S.
— Evaluate the antijoin cardinality query (3) over samples R' and S' and return $\frac{1}{p} \cdot |R' -_f S'|$ as the estimate for $|R -_f S|$.

Though intuitive, this estimator is severely biased – or incorrect on expectation – since it tends to overestimate the actual antijoin cardinality result. The reason is straightforward—simply because there is no match in S' for a given record $r' \in R'$ does not mean that there is no match for r' in S. Therefore, scaling up $|R' -_f S'|$ by the sampling ratio p incorrectly includes many records in R' that do not have any matches in S' but do have matches in the remainder $S - S'$. This is especially true when the sampling ratio q is small. The solution employed previously to counteract the bias – the concurrent estimator in [Jermaine et al. 2005] – requires an index, e.g., B+-Tree, on the entire relation S that permits fast identification of matching records s for any record $r' \in R'$. Unfortunately, this requirement limits the applicability of the concurrent estimator severely. If an index on the inner relation S is not available, the concurrent estimator cannot be used.

3.3. An Unbiased Estimator

Given the naive estimator, it is natural to ask if an unbiased estimator defined exclusively over samples R' and S' exists. As presented in [Joshi and Jermaine 2009], such an estimator can be defined theoretically. We present this unbiased estimator in detail in the following because it provides some intuition as to why sampling-based antijoin cardinality estimation is so difficult. Moreover, the estimator we propose in this paper is derived from the basic form of this theoretical unbiased estimator.

When each tuple in R has at most one match in S. It is useful to consider the form of the unbiased estimator in the simplest case first. Imagine that we know every record $r \in R$ has none or at most one match in S. R' can then be partitioned into two classes:

— R'_0: records $r' \in R'$ that have no matches in S'.
— R'_1: records $r' \in R'$ that have exactly one match in S'.

Likewise, records $r \in R$ can be separated into two corresponding classes R_0 and R_1 that contain records having no match and exactly one match in S, respectively. Throughout the paper, we refer to these records as *class 0* and *class 1* records, respectively. The antijoin cardinality problem consists in computing the number of class 0 records, i.e., $|R_0|$, based only on R'_0 and R'_1. If we can estimate how many records in R'_0 are contributed by those records that have one match in S, i.e., records $r_1 \in R_1$ that end up in R'_0 due to sampling, an unbiased estimate for $|R_0|$ can be obtained by subtracting this guess from $|R'_0|$ and scaling-up the result with the inverse of the sampling ratio p. We know that $|R_1| \cdot p \cdot (1-q)$ is an unbiased estimator for the number of records in R'_0 that are actually in R_1. The reason is straightforward: with probability p a record in R_1 appears in sample R' and with probability $1-q$ its match in S does not appear in sample S'. Given this quantity, the next step is to estimate $|R_1|$ from the samples R' and S'—the only observable data. The probability for a class 1 record to appear in R' is p and, independently, the probability for its match record to appear in S' is q. Thus, on expectation, $|R_1| \cdot p \cdot q$ records from R' have one match in S'. Then we can immediately infer that $\widehat{|R_1|}_1 = |R'_1| \cdot \frac{1}{p} \cdot \frac{1}{q}$ is an unbiased estimator for $|R_1|$. By replacing $|R_1|$ with $\widehat{|R_1|}_1$ and subtracting from $|R'_0|$, we obtain the unbiased estimator for antijoin cardinality $|R -_f S|_1$:

$$|R -_f S|_1 \approx \frac{1}{p} \cdot \left(|R'_0| - \widehat{|R_1|}_1 \cdot p \cdot (1-q)\right) = \frac{1}{p} \cdot \left(|R'_0| - |R'_1| \cdot \frac{1-q}{q}\right) \quad (4)$$

Extension to at most two matches. The unbiased estimator (4) for the simplified scenario of at most one matching record relies on not having any records in R that have two or more matches in S. If such records exist the estimator $\widehat{|R_1|}_1$ is not a good guess for $|R_1|$ since R'_1 inaccurately includes those records in R' that have only one match in S' but actually have additional matches in $S - S'$. So what happens if every record in R can have either zero, one, or two matches in S? Different from the simple at-most-one-match scenario this time there are two sources of bias in $|R'_0|$. One source is represented by class 1 records while class 2 records – records $r \in R$ with exactly two matches in S – are the other source. In order to define an unbiased estimator in this case we have to design estimators for each of the bias sources, subtract them from $|R'_0|$, and scale up the result by $1/p$. As before, $|R_1| \cdot p \cdot (1-q)$ is an unbiased estimator for the bias introduced by class 1 records. The bias introduced by class 2 records – records $r \in R$ with two matches in S that are part of R'_0 – is captured in $|R_2| \cdot p \cdot H\left(X = 0 \mid |S|, 2, |S'|\right)$, where $H\left(X = 0 \mid |S|, 2, |S'|\right)$ is the probability that none of the matches of a class 2 record $r_2 \in R_2$ appears in S'. This probability is given by the probability mass function (pmf)[3] of a hypergeometric distribution [Johnson et al. 1996] with parameters $|S|$, 2, and $|S'|$.

Given the bias formulae for at most one match and at most two matches, the next step in designing an unbiased estimator exclusively from the samples R' and S' is to define estimators for $|R_1|$ and $|R_2|$, respectively. In order to define an estimator for $|R_1|$ we cannot simply scale-up $|R'_1|$ by $1/p$ since part of R'_1 is coming from class 2 records—these are records $r \in R$ with two matches in S and only one match in S'. As a result, we have to estimate the bias in R'_1 caused by class 2 records $r_2 \in R_2$. Since the inclusion probability of a class 2 record in the sample R' is p and the probability of seeing exactly one of its matches in S' is $H\left(X = 1 \mid |S|, 2, |S'|\right)$, the average number of "disguised" class 2 records appearing in R'_1 is $|R_2| \cdot p \cdot H\left(X = 1 \mid |S|, 2, |S'|\right)$. By subtracting this quantity from $|R'_1|$ and multiplying the resulting difference by $1/p$, an unbiased estimator $\widehat{|R_1|}_2$ for $|R_1|$ is obtained:

$$\widehat{|R_1|}_2 = \left(|R'_1| - |R_2| \cdot p \cdot H\left(X = 1 \mid |S|, 2, |S'|\right)\right) \cdot \frac{1}{p} \cdot \frac{1}{q} \tag{5}$$

Notice though that this formula relies on $|R_2|$—information not available in the sample. Estimating $|R_2|$ turns out to be a much easier problem since the only source for R_2 records is R'_2:

$$\widehat{|R_2|}_2 = |R'_2| \cdot \frac{1}{p} \cdot \frac{1}{H\left(X = 2 \mid |S|, 2, |S'|\right)} \tag{6}$$

Putting everything together, we obtain the estimator for $|R -_f S|_2$ as a linear combination of R'_0, R'_1, and R'_2:

$$\begin{aligned} |R -_f S|_2 &\approx \frac{1}{p} \cdot \left[|R'_0| - \widehat{|R_1|}_2 \cdot p \cdot (1-q) - \widehat{|R_2|}_2 \cdot p \cdot H\left(X = 0 \mid |S|, 2, |S'|\right)\right] \\ &= \frac{1}{p} \cdot \left[|R'_0| - |R'_1| \cdot \frac{1-q}{q}\right. \\ &\qquad \left. -|R'_2| \cdot \left(-\frac{1-q}{q} \cdot \frac{H\left(X = 1 \mid |S|, 2, |S'|\right)}{H\left(X = 2 \mid |S|, 2, |S'|\right)} + \frac{H\left(X = 0 \mid |S|, 2, |S'|\right)}{H\left(X = 2 \mid |S|, 2, |S'|\right)}\right)\right] \end{aligned} \tag{7}$$

General estimator form. At this point, we have designed unbiased estimators for two simple cases—at most one match in S and at most two matches in S for every record $r \in R$, respectively. In both cases, the unbiased estimator is a linear combination of $|R'_0|, |R'_1|, \ldots, |R'_m|$, where m is the maximum number of matches a record $r \in R$ can have in S. In fact, it can be shown that the

[3]The probability mass function is the equivalent of the probability density function (pdf) for discrete distributions.

estimator is a linear combination of these quantities for any m:

$$|R -_f S|_m \approx \sum_{i=0}^{m} b_i \cdot |R'_i| \tag{8}$$

The generalization as well as a recursion to compute the coefficients b_i are given in [Joshi and Jermaine 2009]. Moreover, the variance of the estimator is also computed in the general case in [Joshi and Jermaine 2009]. While theoretically sound, the unbiased estimator has several drawbacks that limit its applicability in practice. First, the algorithm to compute the coefficients b_i is factorial in m. Second, the probabilities returned by the hypergeometric distribution pmf are infinitesimal—more so for large values of m. Handling extremely small/large values introduces numerical instability in computation which results in high errors both in the estimate as well as in the variance—known to be prohibitively large in many situations. Third, the assumption under which the unbiased estimator is theoretically sound is that each record $r \in R$ has its own class of matching records in S—without any overlap. If this assumption does not hold, the correlations between records in R sharing records in S cannot be accurately captured by the hypergeometric distribution and bias is introduced in the estimator. Quantifying the bias analytically is virtually impossible and practically infeasible. In summary, the unbiased estimator is best suited for key-foreign key antijoins over relations with a small number of matches m.

3.4. A Biased Estimator

A standard method to deal with an unbiased estimator with high variance is to drop it in favor of a biased estimator having low error. Since error is the result of a combination of both bias and variance, it is often possible to exchange a highly erroneous unbiased estimator – which has massive variance – for a far more accurate estimator with some bias and much lower variance. In our particular case, designing a low error biased estimator for anitjoin cardinality requires picking an alternative set of b_i coefficients that provide for an optimal error level.

A solution to devise such a biased estimator exclusively from the samples R' and S' is introduced in [Joshi and Jermaine 2009]. There are several aspects of this solution that we present in detail since they are also used in the workload-aware estimator we propose in this paper. Since this is a parametric model-based estimator, an assumption is made about the population – relations R and S – samples R' and S' are drawn from. The class – number of matches in S – of each record $r \in R$ is assumed to be extracted from a multinomial distribution [Johnson et al. 1996] with m bins. The parameters of the multinomial distribution, i.e., the probability of each bin, are not known in advance. They are inferred from samples R' and S' using the maximum-likelihood estimator (MLE) technique. An iterative expectation maximization (EM) algorithm [Dempster et al. 1977] which tries to determine optimal values for $|R_i|$ from known $|R'_i|$ values, $0 \leq i \leq m$, is designed to this end. The inferred model is subsequently used to draw a specified number of random populations R_{d_i} and their corresponding samples R'_{d_i}. These are used to compute the optimal value for the coefficients b_i that minimize the resulting estimator's sum-squared error across the drawn population instances. The optimal b_i coefficients are finally plugged in Eq. (8) together with the original sample to obtain the final query estimate. While there is no guarantee on the estimator value, a good indicator of its accuracy is the mean-squared error across the generated populations.

Although more scalable and – in general – more accurate than the unbiased estimator, the biased estimator has a series of problems that make it infeasible for query optimization. The first issue is computation time. Drawing multiple populations from the parametric model and then a sample from each population is highly time-consuming when the size of the population is in the order of millions of records—not to mention when it is larger. Moreover, the entire process has to be repeated for every query. No efficient solutions are discussed in [Joshi and Jermaine 2009] for any of these problems. A second issue is poor estimator accuracy, especially when the size of the sample is relatively small.

3.5. A Correlated Estimator

Instead of drawing independent samples from both R and S, respectively, an alternative is to build sample S' based on R' [Acharya et al. 1999; Dobra et al. 2009; Yu et al. 2013]. R' is obtained as before, by drawing a simple random sample from R. S', on the other hand, includes only those tuples s in S that are matches for sampled tuples from R. Thus, S' is correlated with R'. While R' is still a random sample from R, S' is not a sample from S anymore.

Two estimators can be defined for $|R -_f S|$. The *direct estimator* scales-up the number of tuples $r' \in R'$ without a match in S' accordingly, i.e., $|R -_f S| \approx \frac{|R|}{|R'|} \cdot |R' -_f S'|$. The difference with respect to the naive estimator is that tuples $r' \in R'$ without a match in S' are guaranteed to not have a match in S. This results in a more accurate estimator. An *indirect estimator* for the antijoin can be defined based on the semijoin, i.e., $|R -_f S| \equiv |R| - |R \ltimes_f S|$. The semijoin $R \ltimes_f S$ contains those tuples $r \in R$ that have matches in S. The indirect estimator is defined as $|R -_f S| \approx |R| - \frac{|R||R''|}{|R'|}$, where $|R''|$ is the number of samples with at least one match in S, out of the $|R'|$ drawn samples, i.e., $|R''| \leq |R'|$. It holds under both set and bag semantics. The indirect estimator is useful when the correlated sample stores only tuples $r \in R$ that have matches in S, i.e., non-dangling tuples. This is a standard procedure to improve the accuracy for size of join estimation and make better usage of the synopsis storage space. In general, though, the accuracy of the indirect estimator is likely worse than the accuracy of the direct estimator.

The main drawback of correlated samples is that they are specific to a particular instance of the antijoin problem. A correlated sample from R to S can provide estimates only for $|R -_f S|$. It cannot provide estimates for $|S -_f R|$ because – unlike join cardinality – antijoin is not commutative. The only solution is to build two correlated samples—one from R to S and another from S to R. This has to be done for all possible antijoin predicates ever to appear in queries. Otherwise, no estimate can be produced. A similar problem exists in the case of the concurrent estimator [Jermaine et al. 2005], which requires an index on the inner relation. Essentially, we can think of the correlated estimator as a materialized version of the concurrent estimator. Both of them are unbiased estimators. And both of them require the construction of a specialized synopsis for each antijoin attribute pair.

3.6. Estimation Example

In order to clarify how each estimator works and to emphasize their differences, we provide an illustrative example. Consider $R = \{1, 2, 3, \ldots, 14, 15\}$ and $S = \{11, 12, 13, \ldots, 24, 25\}$, each containing 15 integers, i.e., $|R| = |S| = 15$. We want to compute $|R - S| = 10$ using only the samples $R' = \{2, 6, 8, 10, 13\}$ and $S' = \{13, 14, 19, 20, 25\}$. Using our notation, we have $p = q = 1/3$, $|R'_0| = 4$, and $|R'_1| = 1$. *The naive estimator simply returns 12*: $1/p \cdot |R'_0| = 12$—an overestimate computed as efficiently as possible. The unbiased estimator used in this case is $|R - S|_1$ since each record $r \in R$ has at most one match in S. *The unbiased estimator returns 6*: $3 \cdot (4 - 1 \cdot 2) = 6$—a considerably erroneous underestimate. This is expected given the high variance of the unbiased estimator.

In the case of the biased estimator, the multinomial distribution corresponding to the original population, i.e., $\{2/3, 1/3\}$, has to be approximated as accurately as possible by an MLE process that takes as input the multinomial distribution corresponding to the sample, i.e., $\{4/5, 1/5\}$, and the sampling ratios p and q, respectively. Once the multinomial distribution is determined, a population R_d consisting of 15 records and their corresponding number of matches in S_d is randomly drawn. These are equivalent to our original relations R and S with the only difference that we actually know the value of $|R_d - S_d|$. A sample of 5 records is then taken from R_d and S_d and $|R'_{d_0}|$ and $|R'_{d_1}|$ are computed. The entire process of drawing a random population from the multinomial distribution and sampling from it is repeated several times. The resulting $|R'_{d_0}|$, $|R'_{d_1}|$, and $|R_d - S_d|$ are used to compute optimal values for b_0 and b_1 in Eq. (8), e.g., $b_0 = 2.25$ and $b_1 = 0.8$. With these values, *the biased estimate is 9.8*: $b_0|R'_0| + b_1|R'_1| = 2.25 \cdot 4 + 0.8 \cdot 1 = 9.8$.

The correlated samples derived from the simple random samples R' and S' are $R'' = \{2, 6, 8, 10, 13\}$ and $S'' = \{13\}$. The corresponding *direct correlated estimator returns 12*: $\frac{15}{5} \cdot 4 = 12$. The *indirect estimator returns 12* as well: $15 - \frac{15 \cdot 1}{5} = 12$. Since the overall sample size is only 6 in this case – compared to 10 for simple random samples – the accuracy of the correlated estimator can be improved further by increasing the overall sample size to 10. This requires taking more samples from R and including their matches from S, if any. It is important to emphasize that, while R' and S' can be used to estimate $|S - R|$ directly, this is not the case for R'' and S''. The estimation of $|S - R|$ requires a new set of correlated samples built from scratch.

The standard non-sampling estimator used by many query optimizers has the most inaccurate result. *The non-sampling estimator returns 0*: $(15 - 15) \cdot \frac{15}{15} = 0$—which is far away from the correct result. And this uses exact values for the number of distinct elements in the two relations, not inaccurate synopses. The cause for this inaccuracy is the containment assumption. While uniformity holds perfectly, containment is not true. Only 5 of the 15 distinct elements in S are contained in R. In practice, it might easily be the case that none of the assumptions hold, case in which the estimate error can skyrocket. This is exactly the reason why many query optimizers do not even attempt to estimate antijoin cardinality and default to the intuitive execution plan that positions the antijoin operator at the top of the query tree. In many situations, this is not the optimal choice.

3.7. Summary

None of the sampling-based estimators discussed in this section is directly applicable in the time-constrained environment specific to a query optimizer. The naive estimator is severely biased. The unbiased estimator is factorial in the maximum number of matches m, thus prohibitively slow for $m > 10$. The biased estimator requires parametric sampling from a multinomial distribution followed by random sampling without replacement to generate points used in the optimization problem that computes the coefficients b_i. Repeating this process multiple times for sizable populations, e.g., in the order of millions, renders the biased estimator inapplicable inside a query optimizer. Moreover, the accuracy of both the unbiased and the biased estimator can be unacceptably low – especially for small sampling ratios – due to the fact that they are computed exclusively from the samples. The concurrent and the correlated estimators, while more accurate, require the existence of auxiliary data structures, beyond simple random samples. The concurrent estimator requires an index for every attribute ever to be part of the antijoin predicate in the inner relation. The correlated estimator requires separate samples for every antijoin predicate. Whenever these auxiliary structures are not available, no estimator can be produced. This is a drastic limitation on the generality of a query optimizer.

4. WORKLOAD-DRIVEN ESTIMATION

In this paper, we introduce a completely different approach starting from the biased estimator presented in Section 3.4. While we use the same intuition of determining optimal values for the coefficients b_i in the linear combination (8), we propose several novel ideas that eliminate the inefficiencies of the biased estimator and make it suitable for query optimization. First, we separate the model learning phase from the estimation process by defining a mixture model exclusively over past workload queries. These can be either join or antijoin predicates as long as they provide a match frequency histogram, i.e., the values $|R_i|$. Second, whereas model learning is an offline process, the learned model is not static. It is updated dynamically at runtime with data extracted from the current running query using a principled Bayesian statistics strategy. This results in improved estimation for similar queries executed in sequence. Third, we design an efficient algorithm for extracting samples without replacement from a population generated itself from a multinomial parametric distribution without even materializing the population. This is the major technical contribution that makes our approach suitable for query optimization.

4.1. High-Level Description

Recall that we classify records $r \in R$ with respect to the number of matches they have in S. $|R_0|$ is the number of such records without any match in S—class 0 records. $|R_1|$ is the number of such records with exactly one match in S—class 1 records. And so on. We define the *match frequency histogram* or *histogram pattern* as the array h containing the number of records $r \in R$ in each class: $h = [|R_0|, |R_1|, \ldots, |R_m|]$. For example, for $R = \{a, b, c, d, e, f\}$ and $S = \{d, d, d, g, f\}$, $h = [4, 1, 0, 1]$ when $m = 3$ since a, b, c, and e have no matches in S, f has one match, and d has three matches. It is important to emphasize that the match frequency histogram can be computed for any predicate, not only equality. Similarity-based predicates using any distance function are a good representative in this sense. The estimator we propose in this paper is applicable to any such predicate as long as the match frequency histogram is provided.

We can define $h' = [|R'_0|, |R'_1|, \ldots, |R'_m|]$ in a similar way. Abstractly, the sampling-based antijoin cardinality estimation problem consists in guessing $h[0] = |R_0|$ when only h' is known. The approach taken in the biased estimator (Section 3.4) is to first infer h from h' and then return $h[0]$ as the estimate. Although this is a logical solution given the correlation between h and h', it still remains extremely difficult to accurately infer h due to the large number of configurations of such histograms that can generate h' through sampling. The experimental evaluation in [Joshi and Jermaine 2009] confirms the mixed results of the approach—the only approach known though.

The approach we propose in this paper follows the same basic idea—infer h from h' and then return $h[0]$ as the estimate. Our strategy is completely different though. We start with the assumption that a large number of match frequency histograms h, from which the given h' is drawn, exist. This allows us to capture multiple histogram patterns, e.g., uniform, skewed to the left, skewed to the right, that produce very different estimates for the given h'. The fundamental question is which histogram h is h' drawn from? We assign a weight or probability to each histogram h that specifies the extent to which we believe this is the histogram h' is drawn from—representing such a belief with a probability is the hallmark of Bayesian statistics [Gelman et al. 2003]. The initial set of weights, that we start out with before any data, i.e., query, have been encountered, are known in Bayesian terminology as the prior distribution. While there are many strategies to design a reasonable prior distribution, we choose to learn the prior from the *historical query workload* since it is common in a database setting to have similar queries executed repeatedly. Considering a set, i.e., mixture, of histograms h and including the query workload in the inference process is what differentiates our approach from the biased estimator discussed in Section 3.4. The sampled match frequency histogram h' corresponding to a runtime query is used to update the mixture weights in a statistically rigorous fashion that takes into account the new query. The updated weights represent the posterior distribution in Bayesian terminology. They are used to find an optimal estimator following a similar process to the biased estimator.

Errors in the prior distribution. At first glance, assuming the existence of a prior distribution may seem error prone. Since we learn the prior from previous queries, we are assuming that new queries are never totally different from all of the queries in the training workload. In the case where we see a "new" query, the histogram corresponding to the new query necessarily has a zero prior weight, since the query was totally unanticipated. This is a problem with all methods that rely on learning from the past and not specific to our approach. However, robustness to errors in the prior with an adequate sample size is a widely recognized merit of the Bayesian approach. As more and more samples are taken, the posterior distribution that we use to do inference becomes less dependent on the prior distribution. In most circumstances, after a few hundred query samples (i.e., histograms h') the prior carries little – if any – weight and the query samples are used almost exclusively for guessing the true histogram h. Overall, the proposed approach performs best when the workload is stable and may suffer in the case of ad-hoc queries never seen before.

Algorithm 1 Bayesian inference framework for workload-driven antijoin cardinality estimation

OFFLINE PROCESSING

Learning phase

Input: Set of match frequency histograms extracted from previous queries
Output: Query superpopulation model
Method: EM algorithm for mixture of multinomial distributions

ONLINE PROCESSING DURING QUERY OPTIMIZATION

1. Characterization phase

Input: Sample match frequency histogram for new query
Output: Likelihood distribution of sample match frequency histogram to superpopulation model
Method: CD sampling algorithm from hypergeometric distribution

2. Inference phase

Input: Query superpopulation model and new query likelihood distribution
Output: Updated query superpopulation model
Method: Bayesian inference to update mixture of multinomial distributions

3. Optimization phase

Input: Updated query superpopulation model
Output: Cardinality estimate and confidence bounds for new query
Method: Linear optimization over Monte Carlo trials sampled from superpopulation model

4.2. Bayesian Inference Framework

Given the high-level description of our approach, we first introduce the four steps of the proposed Bayesian inference framework, depicted in Algorithm 1. Then we present the details of each step in a separate section.

The ***learning phase*** (Section 5) uses statistical methods to build a prior histogram model composed of a number of candidate match frequency histogram patterns. Each histogram pattern represents a class of (anti)join queries. A weight is assigned to each pattern, indicating how likely a future histogram query matches that histogram pattern. Both the weights and the histogram patterns are learned *offline* from the historical query workload using an EM algorithm.

In the ***characterization phase*** (Section 6), a likelihood distribution of the current sample histogram h' corresponding to each learned histogram pattern is generated. This is done using a novel Monte Carlo [Robert and Casella 2005] sampling algorithm that extracts a sample h' from the parametric representation of h without materializing the relations R and S. This is the main technical contribution we introduce in this paper—contribution that makes the proposed Bayesian inference framework suited for query optimization.

The ***inference phase*** (Section 7) uses the likelihood distribution computed in the characterization phase and the observed sample histogram h' to infer the posterior histogram probabilities. The prior weights of the histogram model are updated based upon the evidence seen in the sample histogram h'. The updated histogram model is then deemed as a superpopulation from which the query result "hidden" histogram h is generated. It is important to notice that the characterization phase applies solely to the learned histogram patterns and not to the sample histogram h' observed at runtime.

The ***optimization phase*** (Section 8) generates a large number MCo of histograms from the superpopulation computed in the inference phase. Each histogram is then used to instantiate a pair of outer and inner relations $P_i = (R_i, S_i)$, respectively. For each relation pair, a sample pair $T_i = (R'_i, S'_i)$ is subsequently generated. As expected, this process is executed using the same Monte Carlo algorithm introduced in the characterization phase. Let $C_{P_i}[0]$ be the class 0 count, i.e., the correct result, in the i$^{\text{th}}$ relation pair. Let $W(T_i)$ denote the estimator function. The optimal estimator is computed such that it minimizes the sum-squared error (SSE) criterion, i.e., $\sum_{i=1}^{d} \left(C_{P_i}[0] - W(T_i)\right)^2$, across all

the generated populations. It is important to clarify that characterization, inference, and optimization are executed *online* during query optimization.

5. LEARNING PHASE

Any Bayesian method requires a probabilistic model of how data are generated. In our case, each individual "data point" produced by the generative process is a match frequency histogram containing the count for each record class in the outer relation R of an antijoin operator. Informally, we assume the following generative process for producing each histogram. First, a biased die is rolled to determine by which pattern the match frequency histogram is generated. We assume the existence of a set of c different histogram patterns and the die roll selects one of them. Second, we repeatedly sample $|R|$ times from the selected histogram pattern. Each sample is generated with a class label i meaning we see a record $r \in R$ that has i matching records in the inner relation S. The resulting histogram is composed by the sampled count of each class. The assumption that we make here – which need not be strictly accurate – is that any two records $r_i, r_j \in R$ do not share matches in S.

Given this intuitive generative process, the next step is to formalize it rigorously. There are three parts to consider in the formalization. First, a sound parametric model has to be hypothesized for the histogram pattern. Second, a probability mass function (pmf) that returns the probability of a given histogram to be produced by the generative parametric model has to be defined. Third, learning the model consists in finding optimal values for the parameters such that the learning error over the training workload is minimized.

Multinomial parametric model. It is necessary to choose a parametric distribution to model the histogram pattern. The parametric distribution has to be both general – in the sense that it allows any observable histogram to be generated – and specific—in the sense that it captures the typical histogram shape of the underlying data and is tailored for the historical workload. Following the reasoning in [Joshi and Jermaine 2009], we settle for the multinomial distribution [Johnson et al. 1996] with $(m + 1)$ bins – m is the maximum number of matches a record $r \in R$ can have in S – as the histogram pattern parametric model. As discussed in [Joshi and Jermaine 2009], the multinomial distribution is the most general choice since it does not impose any restriction on the probability of a match record class. What differentiates our approach from [Joshi and Jermaine 2009], though – and makes it considerably more complex – is that we use a family or mixture of c multinomial distributions rather than a single one as the generative model. The increased complexity is beneficial in this case since it reduces the chances of over-fitting. This is because the mixture components compete for the shared weight in the overall model. While any component might overfit the training data, the others will limit the degree of over-fitting. In general, mixture models are less prone to over-fitting because they can be seen as multiple instances of the model, instantiated with different parameters. This is a standard method to handle over-fitting [Joshi and Jermaine 2009].

Generative model pmf. The generative pmf function takes as input a histogram and returns the probability that it is produced by the two-step generative process. Formally, let $\{C_0, C_1, \ldots, C_m\}$ denote the m possible classes and $h = [h_0, h_1, \ldots, h_m]$ denote the match frequency histogram, where $h_i, 0 \leq i \leq m$ is the cardinality of class C_i. Also, let $\vec{p} = [p_0, p_1, \ldots, p_m]$, with $\sum_{i=0}^{m} p_i = 1$, denote the multinomial "success" rate for each class. Then, the likelihood of observing a histogram h given $|R|$ and $\vec{p}$ is given by the pmf of the multinomial distribution $\vec{p}$ applied to histogram h:

$$pmf_{mult}(h \mid \vec{p}) = \frac{|R|!}{h_0!h_1!\ldots h_m!} p_0^{h_0} p_1^{h_1} \ldots p_m^{h_m} \tag{9}$$

Notice that $|R|$ is not a model parameter – it is a data-dependent parameter – since it can be derived from the input histogram h, i.e., $|R| = \sum_{i=0}^{m} h_i$. Remember though that our generative model assumes that a histogram is chosen at random from a weighted set of c patterns $\{\vec{p}_1, \ldots, \vec{p}_c\}$ where the weight of each pattern is $0 < w_j < 1, 1 \leq j \leq c$. Moreover, $\sum_{j=1}^{c} w_j = 1$. Thus, the pmf f^*

of the generative model is given by:

$$f^* (h \mid \Theta) = \sum_{j=1}^{c} w_j \cdot \mathit{pmf}_{\mathit{mult}} (h \mid \vec{p}_j) \tag{10}$$

where $\Theta = \{c, \vec{p}_1, \ldots, \vec{p}_c, w_1, \ldots, w_c\}$ are the model parameters.

Learn the model parameters. While the biased estimator in Section 3.4 uses exclusively a query sample histogram h' to learn the model parameters, we use prior workload information. This takes the form of a set $H = \{h^{(1)}, h^{(2)}, \ldots, h^{(k)}\}$ of match frequency histograms obtained as a result of processing prior antijoin queries. We divide the learning phase into two separate stages. First, determine the structure of the model, i.e., the number of mixtures c and the number of bins m in the multinomial distribution. And second, find optimal values for $\vec{p}_i[j], 1 \leq i \leq c, 0 \leq j \leq m$ and $w_i, 1 \leq i \leq c$, respectively, for the configuration chosen in the first stage. In order to guarantee that an overall optimal solution is found even though we do not solve the complete optimization problem at once, we execute the second stage for multiple model configurations of increasing complexity until we do not observe considerable improvement in the objective function. The second stage can be formulated as a sound optimization problem with a well-defined solution for a given model configuration, whereas the best choice for c and m is not that clear even for reduced workloads that produce relatively small-sized histograms. It is also important to notice that larger values for c and m increase the execution time of the online phases accordingly in the Bayesian inference framework. As a result, a tradeoff has to be made between estimation accuracy and execution time. Since this tradeoff is specific to every problem instance, it is best to be determined experimentally. Nonetheless, in the experimental evaluation (Section 9), we present such a tradeoff analysis that gives clear guidelines on how to choose c and m independent of the problem.

Algorithm 2 EM parametric superpopulation extraction

Input: $H = \{h^{(1)}, h^{(2)}, \ldots, h^{(k)}\}$
Output: Θ'

1. Initialize model parameters $\Theta' = \{\vec{p}_1, \ldots, \vec{p}_c, w_1, \ldots, w_c\}$
2. **while** (**true**) **do**
3. $L_{\mathit{curr}} = \Lambda (\Theta' \mid H)$
4. **if** (convergence (L_{curr}) == **true**) **break**
5. **E-step**: Compute posterior probabilities for each histogram bin count $h^{(i)}[j]$ over all the (model mixture, multinomial bin) pairs. Let $\mathit{post}_{ijl} = \frac{\mathit{Prob}\left(h^{(i)}[j] | \vec{p}_l[j]\right)}{\sum_{l'=1}^{c} \mathit{Prob}\left(h^{(i)}[j] | \vec{p}_{l'}[j]\right)}$ be the posterior probability of bin j in input histogram $h^{(i)}$ drawn from the multinomial mixture component $l, 1 \leq i \leq k, 0 \leq j \leq m, 1 \leq l \leq c$.
6. **M-step**: Recompute the model parameters in Θ' using the update rules:
7. $w'_l = \frac{1}{k} \sum_{i=1}^{k} w_l \cdot \mathit{post}_{i*l} = \frac{1}{k} \sum_{i=1}^{k} w_l \cdot \mathit{pmf}_{\mathit{mult}} \left(h^{(i)} \mid \vec{p}_l\right)$
8. $\vec{p'}_l[j] = \frac{\sum_{i=1}^{k} h^{(i)}[j] \cdot \mathit{post}_{ijl}}{\sum_{j'=0}^{m} \sum_{i=1}^{k} h^{(i)}[j'] \cdot \mathit{post}_{ij'l}}$
9. **end while**
10. **return** Θ'

EM algorithm. Given a model configuration, i.e., fixed c and m, and a query workload histogram set $H = \{h^{(1)}, h^{(2)}, \ldots, h^{(k)}\}$, we have to determine the optimal values for the parameters $\vec{p}_i$ and $w_i, 1 \leq i \leq c$ that maximize the likelihood that H is drawn from $\Theta' = \{\vec{p}_1, \ldots, \vec{p}_c, w_1, \ldots, w_c\}$.

This can be expressed as the following log-likelihood optimization function:

$$\Lambda\left(\Theta' \mid H\right) = argmax_{\Theta'} \sum_{i=1}^{k} log f^* \left(h^{(i)} \mid \Theta'\right) \tag{11}$$

The EM algorithm is a general method for finding the maximum likelihood estimate of the parameters of an underlying distribution from a given dataset, when the available data are incomplete or have missing values. In our case, the histogram set H is incomplete. There are two pieces of information missing for every instance in H. We do not know from which superpopulation mixture component it is generated. And, for every bin in a particular instance in H, we do not know which bin in the mixture component generates it. EM starts out with an initial assignment of values to the unknown parameters and, at each step, computes new values for each of the parameters via a set of update rules. EM continues this process until the likelihood function stops increasing. An alternative is to let the algorithm run for a fixed number of iterations.

Without diving into the derivation details, we present the EM algorithm for the mixture of multinomial distributions in Algorithm 2. Every iteration of the EM algorithm performs an expectation (E) step and a maximization (M) step. In the E-step, a set of posterior probabilities are computed for each histogram bin $h^{(i)}[j]$ over all the (model mixture, multinomial bin) pairs. These probabilities can be computed directly from the multinomial distribution pmf in Eq. (9) restricted to a single bin j. The posterior probabilities are then used in the M-step to update the values of the model parameters in such a way that the expected value of the likelihood function associated with the model is maximized with respect to the posterior probabilities. The derivation of these formulae for any parametric distribution can be found elsewhere [Bilmes 1998; Benaglia et al. 2009]. Since the particular case of a multinomial distribution requires only to plug-in the corresponding pmf formula, we do not include the complete derivation here.

While the number of components c in the mixture and the number of bins m in the multinomial distribution have to be passed as input parameters to the EM algorithm using the two-step optimization procedure described previously, the final set of model parameters Θ' returned by the EM algorithm contains valuable information that can be used to reduce the search space of c and m, respectively. Very small weights, i.e., $w_l < 10^{-10}$, are a clear indication that the number of components c is too large and some can be dropped. Similarly, very small probabilities corresponding to bins in $\vec{p}_l$ point to a reduction in m. A correct implementation of the EM algorithm is guaranteed to detect this behavior.

Model construction. There are two separate stages in the computation of the query superpopulation model. First, the match frequency histograms for the query workload H have to be computed. In order to generate one such histogram $h^{(i)}$, the SQL query (12) has to be executed in addition to the antijoin query (3)—which returns only $h^{(i)}[0]$. The main idea is to count how many matches each record $r \in R$ has in S and then group on the number of matches to obtain the match frequency histogram.

```
SELECT Q.cnt AS bin, COUNT(*) AS freq
FROM (SELECT r, COUNT(*) AS cnt
      FROM R r, S s WHERE f(r,s)
      GROUP BY r) AS Q
GROUP BY Q.cnt
```
(12)

If taken separately, this query can take a significant amount of time. Thus, it can only be executed during query optimizer's synopses maintenance. Otherwise, it can impact drastically the execution of the original query. An alternative is to extend existing join algorithms, e.g., nested loops, hybrid hash, sort-merge, such that they produce the histogram as a side effect. While the changes are minimal – for every tuple in R count how many tuples it generates in the join result – the integration with the query optimizer requires care since the histogram is built and stored at query time.

The second stage comprises the execution of the EM algorithm. Since it can take place only after all the workload histograms are extracted, it is executed during query optimizer's synopses maintenance. The complexity of the EM algorithm is $\mathcal{O}(k \cdot c \cdot m)$ for every iteration, with convergence typically achieved in a few thousand iterations. Although this might seem prohibitive, the `R` implementation [Benaglia et al. 2009] we used in the experiments computes a parametric model with 20 components and 100 bins over a workload of 100 queries in less than one second. When the computation of the match frequency histograms is included, the overall execution time is less than 5 minutes—acceptable for synopses maintenance even in the most constrained query optimizers.

Model maintenance. Two situations have to be considered. First, the set of histograms H is kept constant, while modifications to the base relations are allowed. These have to be reflected in the corresponding match frequency histogram, though. The naive approach is to recompute the histogram from scratch after each modification operation—or a number of operations. Incremental maintenance is the more optimal approach in which the histogram is updated minimally. For example, when a new record is inserted into R, the number of matching records in S is found and the count of the corresponding histogram bin is incremented. In the case of a deletion from R, the count is decremented. Insertions and deletions into/from S modify two counts—one is decremented, while the other is incremented. Updates are executed as a delete followed by an insert. In order to have the modifications reflected in the parametric model, the EM algorithm has to be re-evaluated. The previous model can be used as a promising starting solution to speed-up convergence. In the second situation, the set of histograms H is modified through the addition/deletion of one or more histograms. This can be the result of a significant change in the query workload. Since a complete re-execution of the EM algorithm with different parameters is required in this case, this type of modification is recommended only as part of the query optimizer's synopses maintenance—executed at fixed time intervals, e.g., daily, or at the request of the database administrator.

6. CHARACTERIZATION PHASE

The learning phase provides us with a set of weighted histogram patterns that describe the historical workload. As a preparation for the inference phase that makes use of these patterns, we have to derive the distribution of the query sample histogram h' corresponding to a model component $\vec{p}_l$. In this section, we first show how to do this when we know h' is sampled from a match frequency histogram h generated by a given pattern $\vec{p}_l$. Then we present the extension to the distribution of an unknown sample histogram h'.

Recall that we treat a given match frequency histogram h as a sample from a superpopulation modeled as a mixture of multinomial distributions. Thus, a sample match frequency histogram h' can be viewed as the result of the following two-stage sampling process when the multinomial distribution $\vec{p}$ is fixed. First, a match frequency histogram h is produced by drawing a sample of size $|R|$ from $\vec{p}$. h contains in each bin the number of records $r \in R$ corresponding to each class. We use h to instantiate a pair of outer and inner relations R and S, respectively. For each $r \in R$, we assign a unique id to it and add i matching records with the same id to the inner relation S, where r is a class i record, i.e., r has i matching records in S. Second, two samples R' and S' of size $|R'|$ and $|S'|$ are drawn without replacement from R and S, respectively. h' is obtained by executing query (3) and (12) over R' and S'. Given this process, it is clear that h and h' are correlated. However, it is infeasible to analytically obtain the distribution of h' from the distribution of h in the general case of a multinomial distribution. Therefore, we resort to Monte Carlo methods [Robert and Casella 2005] to obtain the distribution of h'—represented (approximately) as a multinomial distribution as well. In order to obtain the distribution of h' given $\vec{p}$, the Monte Carlo approach obtains a large number of i.i.d. samples $h'_{(i)}$ directly from the underlying generative process described above. The samples are then summarized as a set of sufficient statistics to represent the final distribution of h'.

Class-dependent Monte Carlo sampling. It is not hard to imagine how a naive Monte Carlo algorithm for generating the distribution of h' works. The algorithm simply repeats the above two-stage sampling process MCc times, where MCc is the number of Monte Carlo samples to produce.

Though simple, the direct implementation of the naive algorithm is very inefficient. It is essentially impractical in query optimization when the size of R is in the order of millions of records since the cost per Monte Carlo iteration is $\mathcal{O}(|R| + |S|)$. Thus, a more efficient algorithm is required.

Algorithm 3 CD sampling

Input: $\vec{p}$, $|R'|$, $|S|$, $|S'|$
Output: h'

1. Draw a sample R' from *Multinomial*$(\vec{p}, |R'|)$. Let $h_{R'} = [r'_0, r'_1, \ldots, r'_m]$ with $r'_i =$ *Multinomial*$(X = i \mid \vec{p}, |R'|)$, $0 \leq i \leq m$, represent the match frequency histogram of R' corresponding to S.
2. Initialize h' to a zero vector with $(m + 1)$ elements $[0, 0, \ldots, 0]$
3. $N_S = |S|$; $N_{S'} = |S'|$
4. **for** $i = m$ **downto** 0 **do**
5. Let $\vec{q} = [q_0, q_1, \ldots, q_i]$, where $q_j =$ *Hypergeometric*$(X = j \mid N_S, i, N_{S'})$, $0 \leq j \leq i$
6. Draw sample $Y = \{y_0, y_1, \ldots, y_i\}$ from *Multinomial*$(\vec{q}, r'_i)$
7. **for** $j = 0$ **to** i **do**
8. $h'[j] = h'[j] + y_j$; $N_{S'} = N_{S'} - j \cdot y_j$
9. **end for**
10. $N_S = N_S - i \cdot r'_i$
11. **end for**
12. **return** h'

We propose the class-dependent sampling algorithm – or *CD sampling* – to generate h' directly from $\vec{p}$ without even instantiating R and S or executing the naive sampling procedure. CD sampling is the principal technical contribution we introduce in this paper, required in order to apply the proposed method in a query optimization setting. Essentially, CD sampling simulates the naive Monte Carlo procedure using a considerably more efficient distributional sampling process that generates statistically indistinguishable results when compared to the naive Monte Carlo procedure. The central idea in CD sampling is the observation that sampling without replacement in the second stage of the naive Monte Carlo process follows a hypergeometric distribution for every bin in h. In order to determine the contribution to h' for a specific bin in h, we have to sample from the multinomial distribution composed by the hypergeometric probabilities corresponding to bin h. This is similar to the generative process for h with the difference that the probabilities in the multinomial distribution have to be computed from the given $\vec{p}$ and the relation sizes.

The CD sampling pseudocode is given in Algorithm 3. It takes as input parameters the multinomial probabilities $\vec{p}$, the size of S, and the sample sizes $|R'|$ and $|S'|$, respectively. It returns h', the sample match frequency histogram. The first step is to generate R'. This is nothing else than drawing a sample of size $|R'|$ from the multinomial distribution given by $\vec{p}$, since we consider the records $r \in R$ to be i.i.d. (line 1). The only difference between generating R and R' is the sample size. The records $r' \in R'$ are grouped in histogram $h_{R'}$ on the record class, i.e., number of matches, they have in S—the equivalent of h with fewer records. CD sampling is computing h' iteratively from $h_{R'}$ (the `for` loop in lines 4-11). Starting from bin m corresponding to records $r' \in R'$ having m matches in S, the distribution of the number of matches r' has in S' is computed by drawing samples from a hypergeometric distribution with parameters determined by the size of S and S', respectively (vector $\vec{q}$ in line 5). We use the hypergeometric distribution since it is the pmf [Johnson et al. 1996] corresponding to sampling without replacement—the generative process for h'. In order to determine the number of matches a particular record $r' \in R'$ has in S', we draw a single sample from the multinomial distribution governed by the probabilities $\vec{q}$. Line 6 groups the drawn values corresponding to all records r' in the same class into a single multinomial invocation with size r'_i. This assumes that two records in the same class i are independent, which is approximately true when $m \ll |S'|$. The inner `for` loop in lines 7-9 updates histogram h' and the size of S' with

the counts drawn from the multinomial distribution, while line 10 updates the size of S. It is necessary to update $|S|$ and $|S'|$ in each iteration in order to guarantee that the correct sample match frequency histogram h' is generated. Otherwise it is possible to generate a larger sample. Moreover, the execution of the outer loop in descending order guarantees that large number of matches in h' are likely and they are determined exclusively by the sample size.

CD sampling is a quadratic algorithm in the number of bins m. There are $\mathcal{O}(m^2)$ calls to the hypergeometric distribution pmf and $\mathcal{O}(m^2)$ updates to h' and $N_{S'}$. Considering that CD sampling is invoked MCc times for every component in the mixture, the overall complexity is $\mathcal{O}(MCc \cdot c \cdot m^2)$. Thus, it is important to remark that, unlike the naive Monte Carlo procedure, CD sampling is independent of the relation and sample sizes. This results in increased scalability and applicability to query optimization.

THEOREM 6.1. *Denote by $\mathcal{H}' \subset [0 \dots |R'|]^{m+1}$ the set of all possible realizations of h'. For fixed m, $\vec{p}$, and $|R'|$ we have:*

$$\lim_{|S'|,|S|\to\infty} \sup_{v \in \mathcal{H}'} |\textit{Prob}\,(h'_{\textit{naive}} = v) - \textit{Prob}\,(h'_{\textit{CD}} = v)| = 0$$

where $|S'|, |S| \to \infty$ in a manner such that $\frac{|S'|}{|S|} \to p \in (0,1)$.[4]

Thus, if S' is a $100 \cdot p\%$ sample of S and both $|S|$ and $|S'|$ are sufficiently large, then the distributions of h'_{naive} and h'_{CD} are approximately equal. (As shown in the experiments, the deviation between the distributions is negligible for values of $|S|$ and $|S'|$ encountered in practice.)

PROOF. First, observe that, in the naive procedure, R'_{naive} is obtained by drawing an i.i.d. sample of size $|R|$ from $\vec{p}$ and then drawing a simple random sub-sample of size $|R'|$. This is probabilistically equivalent to directly drawing an i.i.d. random sample of size $|R'|$ from $\vec{p}$, which is how R'_{CD} is obtained in the CD algorithm. Now fix v and write

$$\textit{Prob}\,(h'_{\textit{naive}} = v) = \sum_{U \in \mathcal{R}'} \textit{Prob}\,(h'_{\textit{naive}} = v | R'_{\textit{naive}} = U) \cdot \textit{Prob}\,(R'_{\textit{naive}} = U)$$

$$\textit{Prob}\,(h'_{\textit{CD}} = v) = \sum_{U \in \mathcal{R}'} \textit{Prob}\,(h'_{\textit{CD}} = v | R'_{\textit{CD}} = U) \cdot \textit{Prob}\,(R'_{\textit{CD}} = U)$$

where $\mathcal{R}'$ is the set of all possible values of R'. Because $\textit{Prob}\,(R'_{\textit{naive}} = U) = \textit{Prob}\,(R'_{\textit{CD}} = U)$ for all U as discussed above, it suffices to show that $\lim_{|S'|,|S|\to\infty} |\textit{Prob}\,(h'_{\textit{naive}} = v | R') - \textit{Prob}\,(h'_{\textit{CD}} = v | R')| = 0$, for all $R' \in \mathcal{R}'$.

To this end, fix R' and, for $i \in [0 \dots m]$, denote by $n_i = |\{r' \in R' : r' = i\}|$ the number of class i records in R' (where class is with respect to matches in S). Denote by $r'_{i,j}$ the j^{th} record of class i, so that $R' = \bigcup_{i=0}^{m} \bigcup_{j=1}^{n_i} \{r'_{i,j}\}$. Denote by $X_{i,j}$ the class of $r'_{i,j}$ with respect to S' and set $\vec{Y}_i^* = \sum_j I\,(X_{i,j})$, where $I(l)$ is the unit vector of length $m+1$ whose l^{th} component equals 1, with all other components equal to 0. Thus, $h'_{\text{naive}} = \sum_i Y_i$, so that, for each $i = m, m-1, \dots, 0$, an exact sequential implementation of the Naive method algorithm generates $\vec{X}_i = (X_{i,1}, \dots, X_{i,n_i})$ (conditional on the observed value $\vec{x}_j$ of $\vec{X}_j$ for each $j > i$) and then computes Y_i^* and adds it to the current value of h'_{naive}. (That is, only line 6 of the CD algorithm is modified.) The actual CD algorithm computes $Y_i = \sum_j I\,(Z_{i,j})$, where the components of $\vec{Z}_i = (Z_{i,1}, \dots, Z_{i,n_i})$ are i.i.d. with $\textit{Prob}(Z_{i,l} = k) = H\left(k; |S| - \sum_{j=i+1}^{m} j \cdot n_j, i, |S'| - \sum_{j=i+1}^{m} ||\vec{z}_j||\right)$. Here, "$H$" denotes the hypergeometric probability mass function (pmf), $\vec{z}_j$ denotes the observed value of $\vec{Z}_j$, and

[4]The proof of the theorem is largely provided by one of the anonymous reviewers. We are grateful for the time spent on this paper and the insightful comments that increased the quality of this work significantly.

$||\vec{w}|| = \sum_j w_j$ is the L_1-norm for a vector $\vec{w}$. Set $\vec{X} = (\vec{X}_1, \ldots, \vec{X}_m)$ and $\vec{Z} = (\vec{Z}_1, \ldots, \vec{Z}_m)$. Assuming that:

$$\lim_{|S'|,|S|\to\infty} \sup_{\vec{v}} \left| \textit{Prob}\left(\vec{X} = \vec{v}\right) - \textit{Prob}\left(\vec{Z} = \vec{v}\right) \right| = 0, \tag{13}$$

the desired result follows by a straightforward argument which uses the fact that m and $|R'|$ are fixed.

To establish Eq. (13), first observe that $\left(X_{1,1}, \ldots, X_{m,n_m}, |S'| - \sum_j ||\vec{X}_j||\right)$ has a multivariate hypergeometric distribution [Johnson et al. 1996] denoted by $\textit{MH}\left(|S|, \vec{b}, |S'|\right)$, where $\vec{b} = \left(r'_{1,1}, \ldots, r'_{m,n_m}, |S| - \sum_{i,j} r'_{i,j}\right)$. Fixing $\vec{u} = (u_1, \ldots, u_m)$ and using standard grouping and conditioning properties of the *MH* distribution, we find that:

$$\frac{\textit{Prob}(Z_{m,1} = u_1)}{\textit{Prob}(X_{m,1} = u_1)} = \frac{H(u_1; |S|, m, |S'|)}{H(u_1; |S|, m, |S'|)} = 1.$$

We also have:

$$\frac{\textit{Prob}(Z_{m,2} = u_2 | Z_{m,1} = u_1)}{\textit{Prob}(X_{m,2} = u_2 | X_{m,1} = u_1)} = \frac{\textit{Prob}(Z_{m,2} = u_2)}{\textit{Prob}(X_{m,2} = u_2 | X_{m,1} = u_1)} = \frac{H(u_2; |S|, m, |S'|)}{H(u_2; |S| - m, m, |S'| - u_1)}.$$

It follows from the definition of the hypergeometric pmf that:

$$H(u_2; |S|, m, |S'|) = \binom{m}{u_2} \times \frac{\prod_{i=0}^{m-u_2-1} (|S| - |S'| - i) \prod_{i=0}^{u_2-1} (|S'| - i)}{\prod_{i=0}^{m-1} (|S| - i)} \to \binom{m}{u_2} (1-p)^{m-u_2} p^{u_2}$$

as $|S'|, |S| \to \infty$. We have used the fact that, e.g., $\frac{\prod_{i=0}^{m-u_2-1} (|S|-|S'|-i)}{(|S|-|S'|)^{m-u_2}} \to 1$ as $|S'|, |S| \to \infty$ since $u_2 \leq m \ll |S|, |S'|$. A similar calculation shows that $H(u_2; |S| - m, m, |S'| - u_1) \to \binom{m}{u_2} (1-p)^{m-u_2} p^{u_2}$, so that:

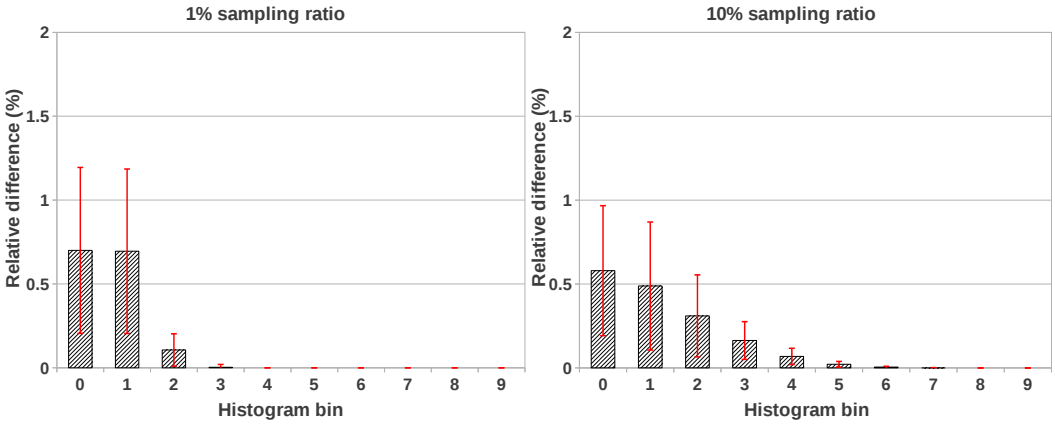
$$\frac{\textit{Prob}(Z_{m,2} = u_2 | Z_{m,1} = u_1)}{\textit{Prob}(X_{m,2} = u_2 | X_{m,1} = u_1)} \to 1$$

and hence, by our previous calculation:

$$\begin{aligned} \frac{\textit{Prob}(Z_{m,2} = u_2, Z_{m,1} = u_1)}{\textit{Prob}(X_{m,2} = u_2, X_{m,1} = u_1)} &= \frac{\textit{Prob}(Z_{m,2} = u_2 | Z_{m,1} = u_1)}{\textit{Prob}(X_{m,2} = u_2 | X_{m,1} = u_1)} \times \frac{\textit{Prob}(Z_{m,1} = u_1)}{\textit{Prob}(X_{m,1} = u_1)} \\ &= \frac{\textit{Prob}(Z_{m,2} = u_2 | Z_{m,1} = u_1)}{\textit{Prob}(X_{m,2} = u_2 | X_{m,1} = u_1)} \times 1 \to 1 \times 1 = 1. \end{aligned}$$

We can iterate the foregoing argument to show that $\frac{\textit{Prob}(\vec{Z}_m = \vec{u})}{\textit{Prob}(\vec{X}_m = \vec{u})} \to 1$ and then that $\frac{\textit{Prob}(\vec{Z} = \vec{v})}{\textit{Prob}(\vec{X} = \vec{v})} \to 1$, for an arbitrary $\vec{v}$. The limit in Eq. (13) follows directly. □

In order to confirm empirically the correctness of Theorem 6.1, we compare the match frequency histograms of the two sampling processes on a multitude of synthetically-generated datasets. The details are presented in Section 9. Figure 2 depicts the relative difference between the bin frequencies with respect to the sample size for a histogram with 10 bins. Two different sampling ratios are used—1% and 10%. The relative difference and corresponding standard deviation are averaged over 1000 independent runs. In both cases, the relative difference between the histogram generated by CD sampling and the histogram obtained through the naive Monte Carlo sampling process is below 1% of the sample size across all the bins. Essentially, the two histograms are virtually identical. Since we observe the same behavior across all the datasets, we provide convincing empirical evidence that



(a) (b)

Fig. 2: Comparison between the sample match frequency histograms generated by CD sampling and naive Monte Carlo sampling. The percentage relative difference with respect to the sample size is depicted for each bin across the two sampling processes.

CD sampling is producing a sample match frequency histogram with the same characteristics as the naive Monte Carlo sampling, but in a tiny fraction of the time.

Parametric representation for Monte Carlo samples. The MCc match frequency histograms produced by CD sampling represent the empirical distribution of h'. However, using the histograms directly is inconvenient and requires significant space. Ideally, we want to summarize these histograms using a parametric distribution from which any particular histogram instance can be sampled. Thus, we first have to choose a distribution family and then learn its parameters such that they fit the generated histograms best. Since histograms h' are the result of a sampling without replacement process, the hypergeometric distribution $\mathit{Hypergeometric}(|S|, (m+1), |S'|)$ would be the clear choice. In the specific setting of a query optimizer, though, the size of the sample S' is considerably smaller than the size of S, i.e., $|S'| \ll |S|$. The multinomial distribution with $(m+1)$ bins $\mathit{Multinomial}(m+1)$ – corresponding to sampling with replacement – is asymptotically equivalent to the hypergeometric distribution in this case. Moreover, $|S|$ and $|S'|$ are not parameters of the multinomial distribution. Thus, we model histograms h' as samples from a multinomial distribution. Bin probabilities – the single parameters of the distribution – can be determined by a straightforward application of the MLE technique over the generated histograms h'. With standard arithmetic manipulations, it can be shown that the multinomial maximum likelihood estimator h' computed over a set $MC = \{h'_{(1)}, h'_{(2)}, \ldots, h'_{(MCc)}\}$ of MCc sample histograms is given by:

$$h'[j] = \frac{\sum_{i=1}^{MCc} h'_{(i)}[j]}{\sum_{j'=0}^{m} \sum_{i=1}^{MCc} h'_{(i)}[j']}, 0 \leq j \leq m \tag{14}$$

The complete derivation of this formula is available at many places online, e.g., [Murphy 2006]. Thus, we do not include it here.

7. INFERENCE PHASE

At this point, we have most of the tools necessary to compute the posterior model probabilities. The superpopulation model is computed offline in the learning phase from the prior workload. Given a runtime sampling scenario, the corresponding empirical histogram is computed efficiently using the CD sampling algorithm and represented as a parametric multinomial distribution in the character-

ization phase. There is one such multinomial for every component in the superpopulation mixture model. The next step is to compute the likelihood of a query sample match frequency histogram corresponding to every multinomial in the mixture model. The resulting posterior probabilities identify the most likely generative model the query sample histogram is drawn from. We then use this posterior model in the estimation phase to compute the estimator for the query sample histogram.

The formal problem we solve in the inference phase is the following. Given a mixture model $\Theta = \{c, \vec{p}_1, \ldots, \vec{p}_c, w_1, \ldots, w_c\}$ and a query sample histogram h'_{query}, compute the posterior weights of the most likely generative model from which the query sample histogram is drawn. It is important to notice that we compute posterior probabilities only for the weights $\{w_1, \ldots, w_c\}$. We update the mixture model components $\{\vec{p}_1, \ldots, \vec{p}_c\}$ only at periodic model-maintenance times, using the EM learning algorithm described in Section 5. In order to accomplish this task, we use the parametric distributions $\{\vec{h}'_{(1)}, \vec{h}'_{(2)}, \ldots, \vec{h}'_{(c)}\}$ computed in the characterization phase for each mixture component. Specifically, we compute the likelihood of h'_{query} being drawn from each of the distributions $\{\vec{h}'_{(1)}, \vec{h}'_{(2)}, \ldots, \vec{h}'_{(c)}\}$ and then use Bayes' rule to obtain the posterior weights of the mixture model as follows:

$$w'_l = \frac{w_l \cdot pmf_{mult}\left(h'_{query} \mid \vec{h}'_{(l)}\right)}{\sum_{l'=1}^{c} w_{l'} \cdot pmf_{mult}\left(h'_{query} \mid \vec{h}'_{(l')}\right)}, 0 \leq l \leq c \tag{15}$$

How exactly to use the posterior weights w'_l requires further discussion. While it is clear that they have to be used directly for computing the estimator corresponding to the current query sample histogram h'_{query}, it is not so clear if they should be used as they are in model Θ. This is because the values returned by the multinomial pmf tend to be infinitesimal, e.g., it is common for the GSL[5] implementation of pmf_{mult} to return values in the order of 10^{-10}. Moreover, there can be orders of magnitude difference between these values across components. This results in posterior weights w'_l becoming virtually zero – no matter what the prior weight is – even with the afferent normalization—on the values returned by the pmf, or on the posterior weights. Consequently, if we use the posterior weights in the model as they are computed in Eq. (15), the effect of a single query might be too skewed. A single query does not provide sufficient evidence that the superpopulation model changes so dramatically. The solution we use in the implementation is to limit the posterior weight change for every query. The maximum change allowed for a query determines how fast the model transitions between states. This way, it takes a larger number of queries to bring a particular weight from 0.99 to 0.01, for example.

8. OPTIMIZATION PHASE

At the end of the inference phase, we have a posterior model adapted to the current query sample histogram. This is the most probable model from which relations R and S and their corresponding samples R' and S' are drawn. In the optimization phase, we define, optimize, and compute the estimator for our query starting from the posterior model, the sampling parameters, and the sample match frequency histogram. We resort to Monte Carlo techniques again to accomplish this task. Specifically, we generate a large number of histograms MCo from the posterior model. Each histogram is then used to instantiate a pair of outer and inner relations $P_i = (R_i, S_i)$, respectively. For each relation pair, a sample pair $T_i = (R'_i, S'_i)$ is subsequently generated. Let $C_{P_i}[0]$ be the class 0 count, i.e., the correct result, in the i$^{\text{th}}$ relation pair. Let $W(T_i)$ denote the estimator function. The optimal estimator is computed such that it minimizes the sum-squared error (SSE), i.e., $\sum_{i=1}^{MCo} \left(C_{P_i}[0] - W(T_i)\right)^2$, across all the generated histograms.

Generating population-sample pairs. Recall that the superpopulation model is composed of c weighted histogram patterns. To generate a population-sample pair, we first use a die-roll to select

[5] www.gnu.org/software/gsl/

Algorithm 4 Generate population-sample pairs

Input: $\Theta = \{c, \vec{p}_1, \ldots, \vec{p}_c, w'_1, \ldots, w'_c\}$, $|R|$, $|R'|$, $|S|$, $|S'|$
Output: $C_P[0]$, h'

1. Let $\vec{w'} = [w'_1, w'_2, \ldots, w'_c]$
2. Draw a sample *comp* from *Multinomial*$(\vec{w'}, 1)$
3. Draw a sample $Ys = \{ys_0, ys_1, \ldots, ys_m\}$ where $ys_i = Multinomial(X = i \mid \vec{p}_{comp}, |R'|)$, $0 \leq i \leq m$
4. Call `CD sampling` ($\vec{p}_{comp}$, $|R'|$, $|S|$, $|S'|$). Store result in h'.
5. Draw a sample $Ya = \{ya_0, ya_1, \ldots, ya_m\}$ where $ya_i = Multinomial(X = i \mid \vec{p}_{comp}, |R| - |R'|)$, $0 \leq i \leq m$
6. $C_P[0] = ys_0 + ya_0$
7. **return** $C_P[0]$, h'

one of the patterns $\vec{p}_i$. The probability of each pattern is given by its weight in the posterior model. Then we generate the pair (P, T) by applying the CD sampling algorithm presented in Section 6 to the drawn pattern $\vec{p}_i$. Remember though that CD sampling generates only the sample match frequency histogram h' corresponding to T. We still have to generate $C_P[0]$ – the correct query result – in order to compute the estimator. This is not a problem. All we have to do is draw $|R| - |R'|$ additional samples from the multinomial distribution defined over $\vec{p}_i$ and count the class 0 records. $C_P[0]$ is the number of class 0 records drawn in $|R|$ trials. The overall process to generate a single population-sample pair is summarized in Algorithm 4. The complexity of this algorithm is dominated by the call to CD sampling which has complexity $\mathcal{O}(m^2)$. Unlike the characterization phase, where MCc Monte Carlo trials are executed for every mixture component, MCo trials are executed overall in the optimization phase. Their distribution across components is dictated by the posterior model probabilities. Thus, the overall complexity is $\mathcal{O}(MCo \cdot m^2)$.

Constructing the estimator. We define the antijoin cardinality estimator as a linear combination of the bin values in the sample match frequency histogram, i.e., $W(T) = \sum_{i=0}^{m} b_i \cdot h'[i]$ (Eq. (8)). Algorithm 4 provides us with MCo sample histograms $\{h'_1, \ldots, h'_{MCo}\}$ and correct query results $\{C_{P_1}[0], \ldots, C_{P_{MCo}}[0]\}$. We still have to determine the coefficients b_i, though, in order to complete the definition of the estimator. The strategy we implement is to compute the coefficients b_i as the solution to the following sum-squared error (SSE) minimization problem:

$$SSE = \sum_{i=1}^{MCo} \left(C_{P_i}[0] - W(T_i)\right)^2 = \sum_{i=1}^{MCo} \left(C_{P_i}[0] - \sum_{j=0}^{m} b_j \cdot h'_i[j]\right)^2 \tag{16}$$

After applying the Lagrange multipliers optimization method, this reduces to solving the following linear equations system:

$$\begin{bmatrix} \sum_i^{MCo} (h'_i[0])^2 & \ldots & \sum_i^{MCo} h'_i[m]h'_i[0] \\ \sum_i^{MCo} h'_i[0]h'_i[1] & \ldots & \sum_i^{MCo} h'_i[m]h'_i[1] \\ \vdots & \vdots & \vdots \\ \sum_i^{MCo} h'_i[0]h'_i[m] & \ldots & \sum_i^{MCo} (h'_i[m])^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} \sum_i^{MCo} C_{P_i}[0]h'_i[0] \\ \sum_i^{MCo} C_{P_i}[0]h'_i[1] \\ \vdots \\ \sum_i^{MCo} C_{P_i}[0]h'_i[m] \end{bmatrix} \tag{17}$$

The optimal coefficients b_i are easily computed using a solver to solve the above linear equations system. Then they are plugged-in W together with the query histogram h'_{query} to get the final estimate. It is important to emphasize that W is a biased estimator. Moreover, the variance of the estimator cannot be derived in a closed-form formula. A good indicator of model accuracy is the mean-squared error (MSE) over the MCo sample histograms $\{h'_1, \ldots, h'_{MCo}\}$.

9. EXPERIMENTAL EVALUATION

The objective of the experimental evaluation is to investigate the accuracy and execution time of the proposed estimator across a variety of datasets and workloads—including a single query as well as a sequence of queries. Additionally, the sensitivity of the estimator is quantified with respect to many configuration parameters. Specifically, the experiments we design are targeted to answer the following questions:

— What is the sensitivity of the estimator with respect to the number of mixture components, the number of historical queries, the number of bins in the sample match frequency histogram, the number of Monte Carlo trials, and the sampling ratio, respectively?
— How is the execution time distributed across the online stages?
— How does the proposed estimator compare to the previous sampling-based estimators in terms of accuracy and execution time?
— How well does the estimator adapt to the query workload?

Implementation. We implemented the proposed estimator as well as the other subset-based query estimators in `C++`. Statistical operations such as sampling from parametric distributions and computing the pmf of a distribution are implemented as function calls to the `GSL` library. Linear algebra computations are implemented using the `armadillo` library[6] which proved to be more stable than `gsl`. In addition to the estimation code, we implemented a complete suite of data generation routines that allow the creation of datasets exhibiting specific patterns required in the sensitivity micro-benchmarks.

System. We execute the experiments on a standard server with 2 AMD Opteron 6128 series 8-core processors – a total of 16 cores – 40 GB of memory, and four 2 TB 7200 RPM SAS hard-drives configured RAID-0 in software. Each processor has 12 MB of L3 cache while each core has 128 KB L1 and 512 KB L2 local caches. The storage system supports 240, 436 and 1600 MB/second minimum, average, and maximum read rates, respectively—based on the Ubuntu disk utility. The cached and buffered read rates are 3 GB/second and 565 MB/second, respectively. Ubuntu 12.04.3 SMP 64-bit with Linux kernel 3.2.0-56 is the operating system.

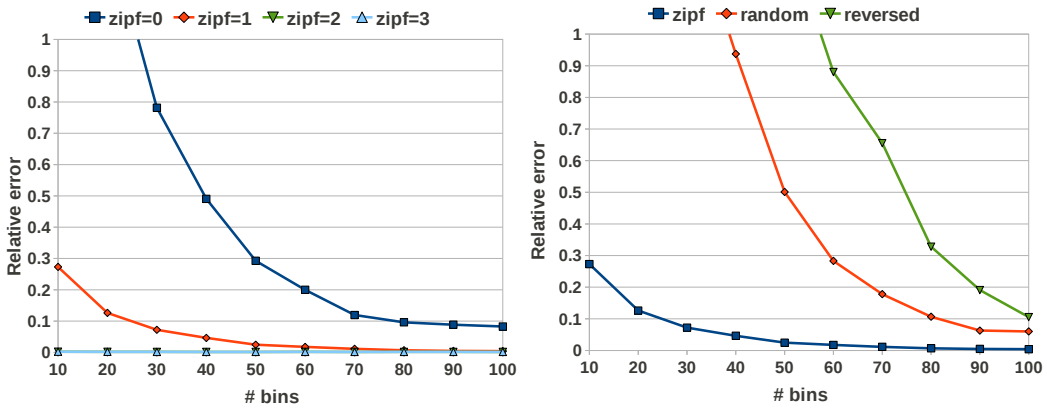
Methodology. We measure estimator accuracy using relative error defined as $\frac{|estimate - true\ result|}{true\ result}$. We perform all the experiments 100 times and report the average value as the result. If the experiment consists of a single query, the same initial model is used across all the iterations. In experiments over a sequence of queries, the model weights are updated after each query. Thus, the model taken as input by subsequent queries takes into account workload knowledge.

9.1. Micro-Benchmarks

In order to evaluate the sensitivity of the proposed estimator with respect to the various configuration parameters, we design a complete suite of experiments over synthetically generated data. To this end, we develop a versatile data generator for the match frequency histogram h that takes as parameters the size of the outer relation $|R|$, the number of bins m, the type of the distribution and parameters used to generate the bin values, and the amount of noise in the inner relation S. Histogram h plays the role of superpopulation model as well as of a particular population instance, i.e., the mixture model consists of a single component in this case. As a result, the only phase that is executed is optimization. In the experiments we execute, $|R|$ takes values in $\{10^6, 10^7, 10^8, 10^9\}$, m is in $\{10, 100, 1000\}$, while the amount of noise in S, i.e., tuples that do not have any match in R, is chosen from 0%, 0.1%, 1%, and 100% the size of S, as generated by a given distribution. The values in the histogram bins are generated using a zipfian distribution[7] with skew parameters ranging from 0 (uniform distribution) to 3 (highly skewed distribution). The shape of the histogram can be normal zipfian (skewed to the left), random, or reversed zipfian (skewed to the right). Query sample match frequency histograms h' are extracted from a histogram h by applying the `CD sampling` algo-

[6] www.arma.sourceforge.net/
[7] www.en.wikipedia.org/wiki/Zipf's_law

rithm without materializing the two relations. We do not include results for all the configurations in the paper. We use $< |R| = 10^6, m = 10, noise_S = 0\% >$, coupled with 1% sampling on both R and S, as the canonical configuration without further mentioning in the paper, except where necessary. Notice, though, that we observed similar – if not better – results for all the other configurations.



(a) (b)

Fig. 3: Accuracy as a function of the number of bins in the sample match frequency histogram.

Number of histogram bins. In this experiment, we study the effect the number of bins in the sample match frequency histogram has on the proposed estimator. We consider sample histograms with number of bins varying from 10 to 100, where 100 is the number of bins in the original population. The histogram with reduced size is generated by summing the values in the right-most bins – bins that do not appear in the histogram anymore – and storing them in the last, i.e., right-most, bin of the reduced histogram. There are two methods to do this. We can compact the original histogram and then execute the normal sampling procedure to get h'. Or we can compact only the sample histogram h'. We found from experiments that the probability of generating tuples with large number of matches in the sample histogram is quite low even when such tuples exist in h. Thus, we use the first alternative in our implementation since it also simulates the superpopulation sampling process. The effect of the number of bins on estimator accuracy and execution time are depicted in Figure 3. The number of Monte Carlo trials used in these experiments is 100—a rather modest number. Figure 3a depicts the relative error of the estimator as a function of skew in the normal zipfian distribution used to generate the original population. As expected, the more bins in the sample histogram, the more accurate the estimator is. For highly-skewed histograms, i.e., zipf=2 and zipf=3, a large part of the population mass is already concentrated in the few left-most bins. Thus, reducing the number of bins in the representation does not degrade accuracy, which is always above 99%. The effect of the number of bins is best illustrated for the uniform distribution, i.e., zipf=0. It is clear that reducing the dimensionality of the histogram degrades the accuracy. Nonetheless, with 70 or more bins the relative error converges to 10% and does not improve further. The reason is the large number of false positives, i.e., tuples with zero matches, in the sample histogram h'. The accuracy becomes even worse when the shape of the original histogram is random or reversed zipfian. Figure 3b depicts the relative error for these distributions when derived from zipf=1. In these situations, it is required to accurately represent the original distribution – using the same number of bins – in order to achieve acceptable relative error below 10%. Another observation that can be drawn from these results is that the proposed estimator is sensitive to the number of bins in the histogram representation only when the result is relatively low. This insight is confirmed by further experiments.

While a large number of bins improves accuracy, it also increases the execution time of the estimator. As depicted in Figure 5a, the time taken for 80 bins or more is over 1 second, when a single execution thread is used. If we perform the Monte Carlo trials in parallel, a linear speedup is obtained since the trials are independent. For 16 threads, the maximum on the test system, the estimation always takes less than 100 milliseconds. There is an interesting tradeoff to consider between the number of Monte Carlo trials and the width of the sample histogram with respect to execution time. Narrow histograms and a high number of trials are effective for skewed distributions, while wide histograms and a low number of trials work well for uniform and reversed zipfian distributions. These are the two parameters that have to be carefully adjusted in order to guarantee a desired execution time for the estimation process.

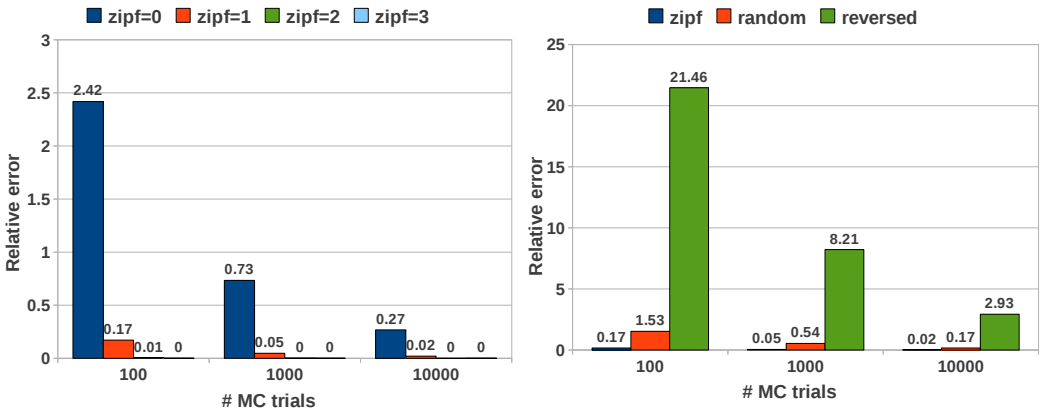
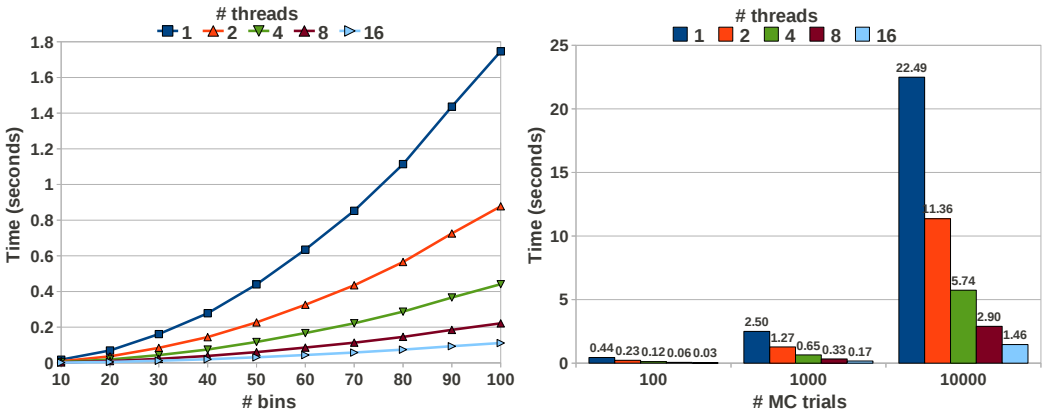


Fig. 4: Accuracy as a function of the number of Monte Carlo trials in the optimization phase.

Number of Monte Carlo trials in the optimization phase. In this experiment, we vary the number of Monte Carlo trials in the optimization phase – the only phase when the number of components is one – while we keep all the other parameters as defined in the canonical configuration. The number of bins in the sample histogram is set to 50. Figure 4a depicts the accuracy of the proposed estimator as a function of the skew parameter in the zipfian distribution. As expected, an increase in the number of trials translates into lower relative error, independent of the skew. For distributions with relatively modest skew, however, the decrease is more pronounced, since the error is quite high when the number of trials is low. Relative error of 0 on the bars for zipf=2 and zipf=3 stands for error smaller than 1%. We observe similar results for distributions having other shapes with zipf=1 in Figure 4b. The errors are of course higher for random and reversed distributions, respectively. The careful reader observes immediately that the results in Figure 3 and Figure 4 are not identical even though there is overlap in the experiments run, i.e., 50 bins and 100 Monte Carlo trials. The reason is that we generate different datasets for each experiment. Moreover, the sample histogram is also randomly generated in each case. From the results in the two sets of figures it is clear that the sample extracted in the number of bins experiment provides more accurate estimates.

The execution time results as a function of the number of Monte Carlo trials are depicted in Figure 5b. The most important observation is that a ten-fold increase in the number of trials translates in a factor smaller than ten increase in execution time. With straightforward parallelization across trials, even configurations with thousands of Monte Carlo trials can achieve sub-second execution times, as the results in Figure 5b show. However, it is important to notice that thousands of trials provide a significant improvement in accuracy only for adversarial distributions. Hundreds of trials

are more than sufficient to obtain satisfactory accuracy in a matter of milliseconds. This is ideal for estimation in query optimization.



(a) (b)

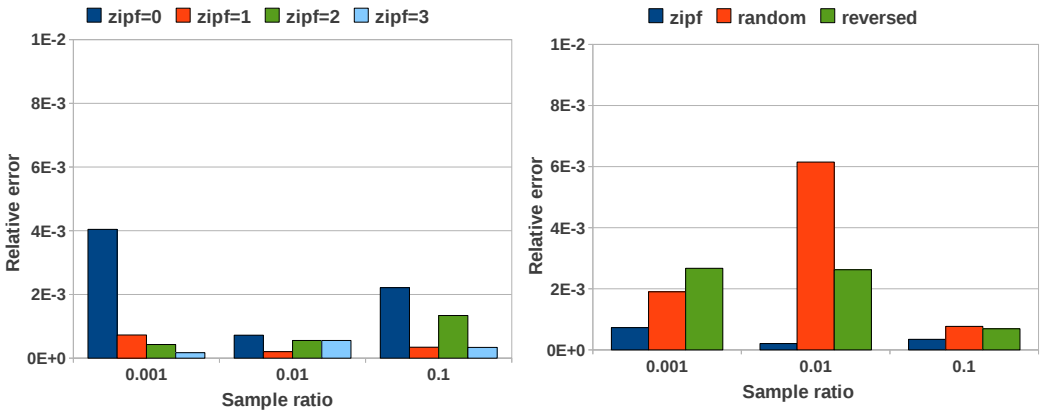
Fig. 5: Execution time as a function of the number of bins in the sample match frequency histogram (a) and the number of Monte Carlo trials in the optimization phase (b).

Sampling ratio. The last set of experiments we execute over synthetically generated zipfian data are to determine the effect of the sampling ratio on the accuracy of the proposed estimator. It is important to notice that the sampling ratio does not impact the execution time. The results for zipfian distributions with different parameters and for different distribution shapes with zipf=1 are presented in Figure 6. The most important observation that can be drawn from these results is that larger samples do not automatically translate into higher accuracy. In order to confirm that this is indeed the case, we execute these experiments over a much larger set of configuration parameters. The results depicted in Figure 6 are for ten bins both in the original population as well as in the sample histogram and with 10000 Monte Carlo trials, thus the extremely low relative error. After careful investigation, we found an explanation for this unexpected behavior. While larger samples translate into match frequency histograms with non-zero values for larger number of matches, i.e., further to the right, this does not guarantee a more accurate estimator because more coefficients b_i in the linear combination have to be learned. This process is highly sensitive to the samples generated in the Monte Carlo trials and to the numerical stability of the linear system solver due to rank-deficiency—many cells in the matrix in Eq. (17) are zero. These are specific to every sample histogram and do not necessarily depend on the sampling ratio. To conclude, although sampling ratio is not entirely correlated with the estimator accuracy it is important to notice that the absolute difference across ratios is insignificantly small, i.e., less than 0.0001%.

9.2. TPC-H Data

In order to illustrate the power of the proposed estimator in query workload scenarios we devise a query generator based on the TPC-H benchmark. We use the generator to create a training workload of 100 query pairs as given in Eq. (3) and (12), respectively. We execute these queries over a TPC-H scale 1 (1 GB) instance in `MonetDB`[8]. The resulting 100 match frequency histograms are used to learn a superpopulation mixture model using the `R multmix` package [Benaglia et al. 2009].

[8] www.monetdb.org/



(a) (b)

Fig. 6: Accuracy as a function of the sampling ratio.

The entire process takes less than five minutes on the test machine. We use the same query generator to create a different test workload consisting also of 100 queries. Instead of running the test queries over the training TPC-H scale 1 instance, we use a scale 50 (50 GB) instance. This makes the estimation considerably harder because both the queries and data are different from training. Since the input to the estimator consists of sample match frequency histograms, we execute the test queries over a 1% sample of the TPC-H scale 50 instance, i.e., a 1% sample is drawn from each table using the MonetDB `SAMPLE` operator. We also execute the antijoin query in Eq. (3) over the entire database to get the correct query result required to determine the estimator accuracy. We use the learned model to compute estimates for the test queries. In the following, we present a series of experiments that assess various properties of the proposed estimator in this workload scenario. Before that, though, we give more details on the query generator.

Query workload generator. The generator selects each query randomly from a set of ten predefined patterns. The patterns involve common (anti)join predicates – not necessarily key-foreign key – in TPC-H queries. We use the following (anti)join pairs:

(`p_partkey`, `ps_partkey`); (`s_suppkey`, `ps_suppkey`)
(`c_custkey`, `o_custkey`); (`o_orderkey`, `l_orderkey`)
(`s_suppkey`, `l_suppkey`); (`ps_suppkey`, `l_suppkey`)
(`l_suppkey`, `ps_suppkey`); (`p_partkey`, `l_partkey`)
(`l_partkey`, `ps_partkey`); (`ps_partkey`, `l_partkey`)

Using only these ten pairs alone, it is impossible to generate 100 different queries. We achieve this by adding selection predicates with random and correlated selectivity on each of the two relations that appear in the (anti)join query. The selection predicates involve at most two attributes from each relation, not necessarily the antijoin attribute. This guarantees a large mix of match frequency histograms both over the complete data as well as over the samples.

Number of mixture components. In this experiment, we investigate the accuracy of the estimator as a function of the number of mixture components. For this, we execute Algorithm 2 in the learning phase with four c values, i.e., 5, 10, 15, and 20. Each execution results in a superpopulation model with the corresponding number of components. Finding the optimal solution for a given configuration requires a large number of reinitializations since the optimization problem is not convex, thus there exist multiple local minima. Given that each algorithm run takes less than 10 seconds, this procedure can be executed relatively fast. Figure 7 depicts the accuracy of the estimator as a function of the cardinality size for the four numbers of components considered. The other estimator

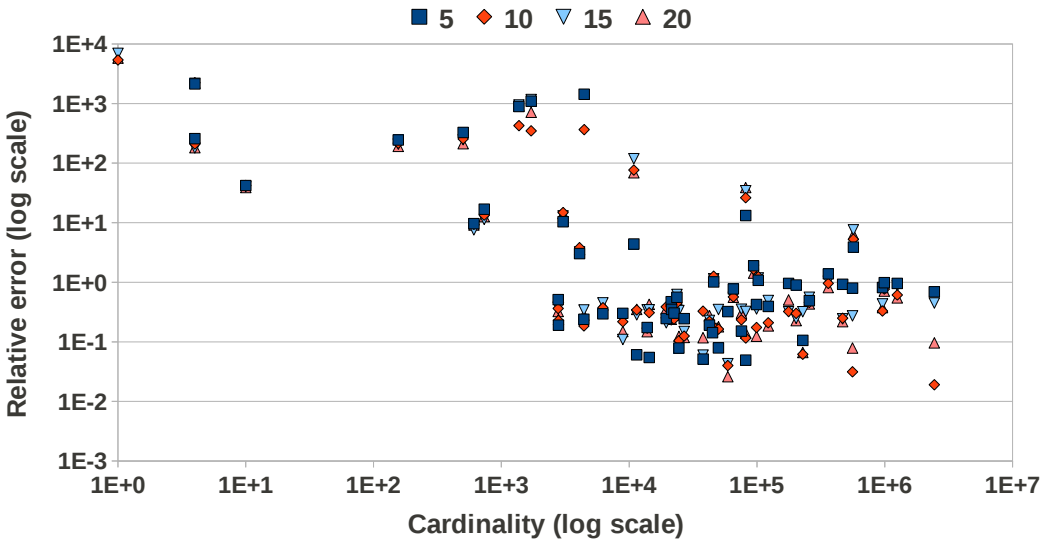


Fig. 7: Accuracy as a function of the number of mixtures.

parameters are set as follows: $m = 10$, $MCc = 100$, and $MCo = 100$. Since the relative error formula goes to infinity when the true result is zero, we eliminate the queries with result zero from the figure. Thus, 32 queries are not displayed even though estimators are computed for each of them. Although not obvious for all the queries, models with different number of components provide different accuracy levels. It is not always the case that more components provide higher accuracy even though the log likelihood function in the EM learning algorithm is larger—the log likelihood increases monotonically from 5 to 20 components. In these particular results, 10 components provide the lowest relative error for multiple queries. Overall, the 10 and 20 mixture models outperform the 5 and 15 components slightly. After a closer investigation, we discovered that in models with more than 13 components the remaining components have extremely small weights, i.e., $< 10^{-10}$, thus they can be discarded without degrading accuracy. An orthogonal observation that can be drawn from Figure 7 is that the proposed estimator provides higher accuracy for cardinality results above 5000. This is expected since no sampling estimator excels when the result is in the order of tens or hundreds.

In order to determine the optimal number of components, we use two criteria. First, if we do not see significant improvement in the log likelihood objective function of the EM algorithm as we increase c this is a clear sign that increasing the number of components further does not provide real benefits. Moreover, remember that the execution time is linear in the number of components. And second, in the case of too many components, i.e., over-fitting, the weight associated with some of the components becomes insignificantly small, thus the probability to generate samples from them is close to zero. Essentially, the EM algorithm identifies only the important components if we pass in a too large c value.

Number of historical queries. The impact of the number of historical queries used to build the superpopulation model in the learning phase is depicted in Figure 8. All the model parameters in the EM learning algorithm are kept the same, while the number of historical match frequency histograms is increased from 20 to 100, in increments of 20. The histograms in each batch are a random sample without replacement from the overall query workload, with the additional constraint that each smaller set is a subset of the larger sets. It is clear from the figure that the results corresponding to each query are mostly clustered together—the accuracy is not highly sensitive to the

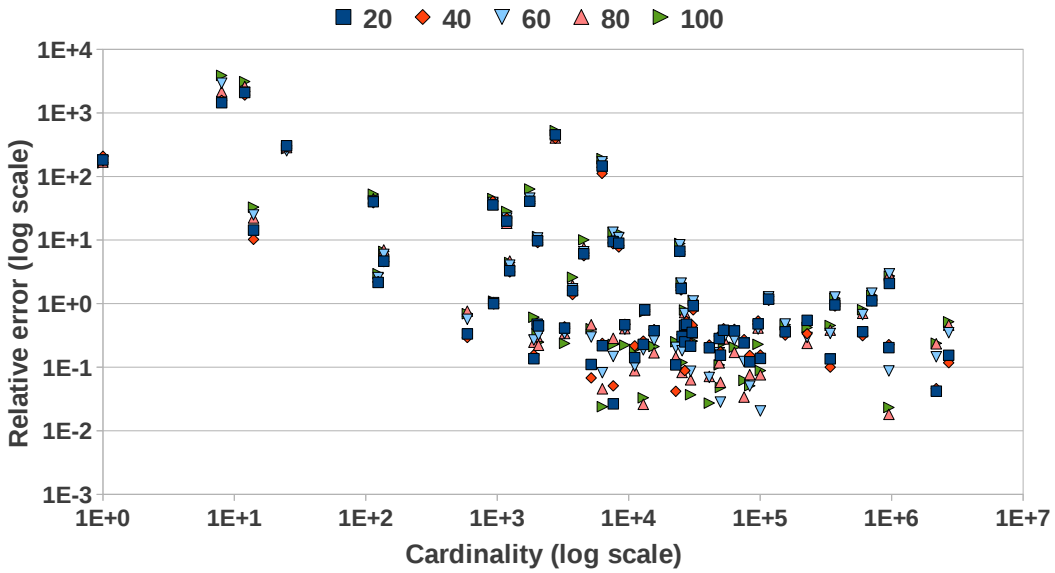


Fig. 8: Accuracy as a function of the number of historical queries used to learn the model.

size of the workload. Random sampling guarantees that a representative set of queries is generated even when the size is small, e.g., 20. The results also confirm that the model does not over-fit as more queries are included in the workload.

Comparison with existent sampling-based estimators. Figure 9 depicts a comparison between the proposed estimator and the other sampling-based estimators for subset queries. These estimators are discussed in Section 3. The same number of bins in the histogram representation is used across all the estimators, i.e., $m = 10$. The number of iterations in the biased estimator is set to 1000, while the number of Monte Carlo trials in the proposed estimator is relatively small, i.e., $MCc = 100$ and $MCo = 100$, respectively. While all the other estimators require the existence of a random sample for every relation, the correlated estimator demands one synopsis for every antijoin predicate. In our particular case, this equates to twenty samples vs. six samples. For every additional antijoin predicate, two samples have to be built. Otherwise, no estimator can be produced. Even if all the antijoin predicates are known beforehand, e.g., in a production system, the construction and maintenance of such a large number of samples demands significantly more resources than a simple random sample per relation. The overall size of the samples used by the correlated estimator is kept close to the size of the random samples used by the other estimators. However, in order to obtain accurate estimates, the total size of the correlated samples on `lineitem` is 50% larger.

Figure 9 depicts the ratio of the average relative error between the previous estimators and the proposed estimator for all the non-zero result queries in the workload. Values above the horizontal line at position 1 on y axis indicate that the proposed estimator is more accurate. This is the case for a large majority of the queries. As expected, the correlated estimator is the most accurate whenever an estimator can be produced—the direct correlated estimator is used throughout these experiments. This requires the existence of a synopsis for every antijoin predicate and a relatively uniform distribution of the matching tuples. When this is not the case, no estimator can be produced, e.g., the points at the top of the y axis. Overall, the proposed estimator always outperforms the other estimators in the case of hard estimation problems with low cardinality results. Recall that none of the previous estimators can be implemented in a query optimizer due to the large execution time – factorial in the number of histogram bins for the unbiased estimator and linear in the size of the relations for the biased estimator – or the unbounded number of synopses—one for every possible

antijoin predicate in the case of the correlated estimator. It is also important to reiterate that the proposed estimator is trained neither on the test dataset nor on the test queries. Thus, it has practically no advantage over the other estimators, derived exclusively from the samples.

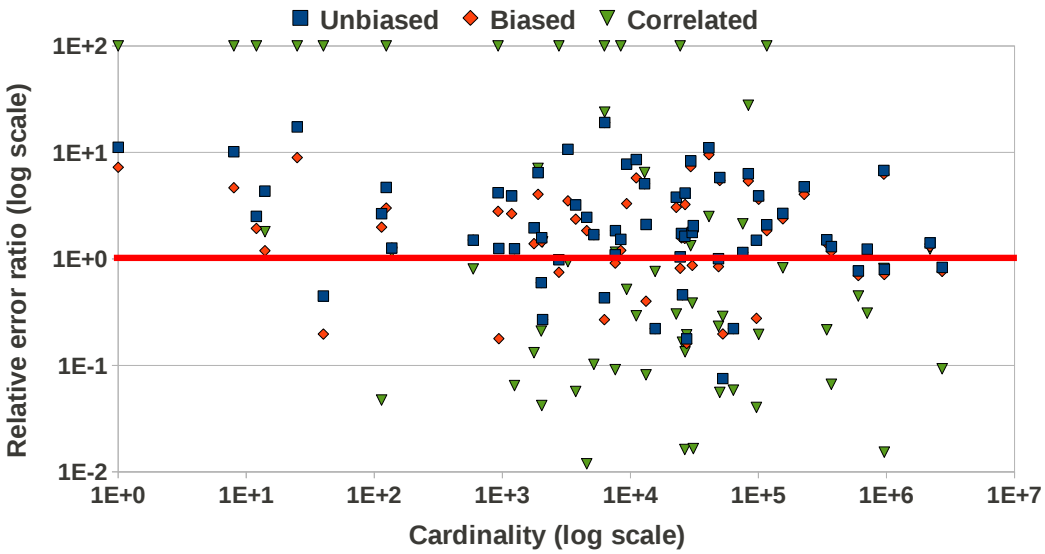


Fig. 9: Accuracy comparison with antijoin sampling-based estimators.

Table I contains the estimation time for the three match frequency histogram-based estimators—Bayes is the proposed estimator. The correlated estimator is not included in the comparison since it can be computed by simply executing the query over the samples in the synopsis. This is also the case for the naive estimator. For a small number of bins in the histogram, the unbiased estimator is the fastest. When the number of bins is large, e.g., 100, the unbiased estimator does not produce an estimate even after 10 hours—we stopped the execution at that stage. The execution of the biased estimator is dominated by the EM algorithm that infers the superpopulation model from the samples. Since the EM algorithm is executed offline in the proposed estimator, we observe a sharp decrease in the execution time. To speed-up the execution further, we execute the characterization phase in parallel across all the mixture components. A factor of 10 speedup is achieved since the components are independent. Two conclusions can be drawn. First, when the number of bins is small, there is no significant difference between the unbiased and the proposed estimator in terms of estimation time. Second, the proposed estimator is the only one that achieves satisfactory performance when the number of bins is large.

Estimator	**10 bins**	**100 bins**
Unbiased	2.3	∞
Biased	177	10,657
Bayes	9.4	769

Table I: Execution time (milliseconds) comparison with antijoin sampling-based estimators.

Table II shows how the estimation time is distributed across the phases of the proposed estimator. Parallelization is disabled in the characterization phase. The results confirm the time complexity.

Inference can be ignored. Characterization is by far the most time-consuming stage. Since the computation across the mixture components is independent, parallelization can reduce the time spent in characterization by a linear factor—as shown in Table I.

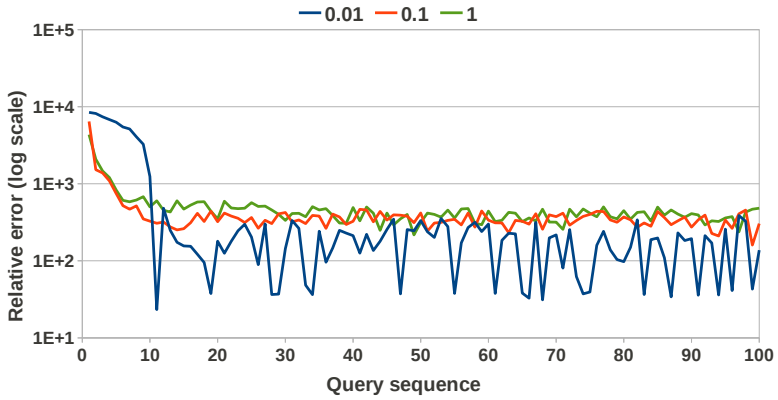
Phase	10 bins	100 bins	Time complexity
Characterization	42	3,496	$\mathcal{O}\left(MCc \cdot c \cdot m^2\right)$
Inference	0.03	0.07	$\mathcal{O}(c)$
Optimization	5.84	345	$\mathcal{O}\left(MCo \cdot m^2\right)$

Table II: Execution time (milliseconds) distribution across framework phases.

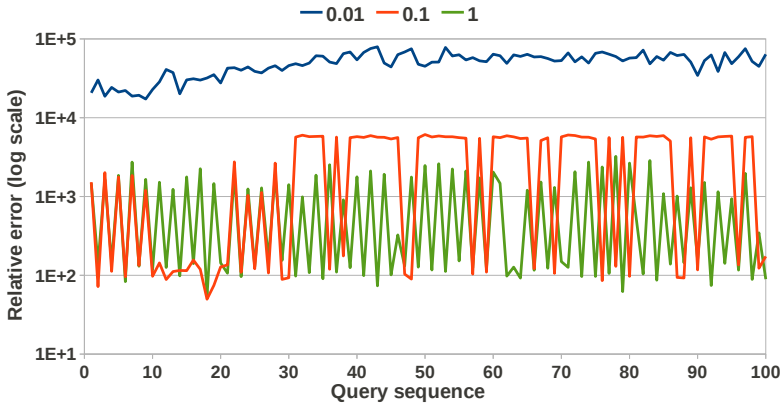
We do not include the results generated by the standard query optimizer estimator given in Section 3.1 in the comparison because they are considerably worse than for any of the sampling-based estimators. Although the uniformity and containment assumptions hold for most of the antijoin predicates – they are key-foreign key joins – the selection predicates applied to each relation make the accurate cardinality estimation after the selection predicates almost impossible. Not to mention the estimate for the number of distinct elements. These two inaccurate values propagate with an exponential factor in the error of the antijoin operator, resulting in unusable estimates. This confirms the difficulty in estimating antijoin cardinality with the standard query optimizer estimator and provides support for the default query execution plan. The estimator we propose, however, is accurate and fast enough to be used inside a query optimizer, as the experimental results in this section prove.

Estimation for a query sequence. In all the experiments presented up to this point, we kept the superpopulation model static, i.e., we did not update the mixture weights after a query is executed. In this experiment, we study the behavior of the proposed estimator for a query sequence when the model adapts dynamically to the workload, i.e., updates the weights based on the processed queries. Notice that none of the other estimators proposed in the literature is dynamic. To properly study the estimator in such a complex environment, we extract 10 queries with high error from the test workload and create three sequences consisting of 1000 queries each. In the first sequence, we execute each query 100 times consecutively. In the second sequence, we execute 10 instances of each query consecutively. We repeat this in the same order for 10 times, e.g., query 1 executes at positions 1-10, 101-110, ..., 901-910. And finally, in the third sequence, we execute one instance of each query round-robin for 100 times, e.g., query 1 executes at positions 1, 11, 21, 31, ..., 991.

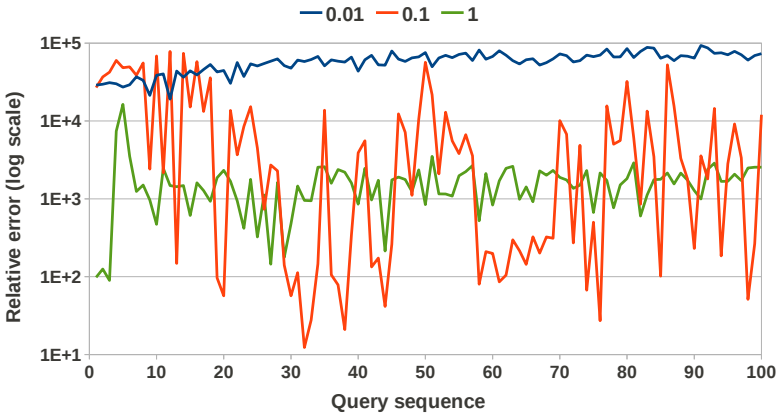
Figure 10 presents the accuracy as a function of the position in the sequence for a selected query class out of the ten executed. We present results for three different amounts of maximum weight change between queries, e.g., 0.01, 0.1, and 1. The larger this value, the larger the influence of the last query on the superpopulation model. The initial model is the same for all the three query sequences as well as the order of the query classes. A common trend in the three scenarios depicted in Figure 10 is the periodic behavior of the estimator resulting from the repetitions in the query sequence. Moreover, in all the three cases and for any of the three weight changes the estimator converges to some value. This value is highly dependent on the sequence and the weight change though. Nonetheless, some clear patterns emerge. First, the more queries from the same class are executed together, the higher the accuracy and the lower the estimator variance. This trend is very clear for weight changes of 0.1 and 1, respectively. Second, when the impact of a single query on the model is limited, e.g., weight change of 0.01, it takes a large number of consecutive queries until the model learns the workload and accurate estimates are produced (Figure 10a). And third, the careful reader remarks immediately the oscillations of the estimator even when the same query is executed repetitively with the same weight change (Figure 10a with 0.01 and Figure 10b with 0.1 and 1). After carefully investigating, we determined that two mixture components in the model – out of the total of 10 – are used alternatively to make predictions. Their accuracy is quite different though. Whenever the weights are updated – or after a number of updates – the dominant mixture changes. As a result, the model enters a steady state consisting of two mixtures – there can be more than two –



(a)



(b)



(c)

Fig. 10: Accuracy for a query executed in sequences of 100(a) 10(b), and 1(c) in a workload of 1000 queries of 10 different classes.

and jumps between them, thus the oscillating behavior. Overall, it is important to emphasize that in the best situation the proposed estimator achieves a two orders of magnitude accuracy improvement by dynamically learning the query workload. The example query has a cardinality of only 40, thus it is extremely difficult to estimate. For comparison, the two existent estimators provide accuracy in the order of 10^5 which is 10 times worse than the proposed estimator without model update and three orders of magnitude inferior when updating the model.

9.3. Discussion

We conclude this section by summarizing the main results and providing answers to the three questions driving the experimental evaluation. Out of the four configuration parameters – mixture components, histogram bins, Monte Carlo trials, and sampling ratio – the proposed estimator is most sensitive to the number of histogram bins and the Monte Carlo trials. Increasing the number of mixture components or the sampling ratio does not necessarily result in better accuracy. With respect to the execution time, the estimator has quadratic complexity in the number of bins and linear in the number of mixtures and Monte Carlo trials. When compared to the other sampling-based estimators for subset-based queries, the proposed estimator is more accurate than the unbiased and biased estimators in most cases. The correlated estimator is, as expected, the most accurate whenever an estimate can be produced. However, this requires a synopsis for every antijoin predicate—a rather unrealistic condition. The proposed estimator is also the only estimator that scales to match frequency histograms with a large number of bins. The capacity to adapt to the query workload results in considerable accuracy improvement, e.g., two orders of magnitude, over the static estimators. We found a configuration with 10 components, 10 histogram bins, 1% sampling ratio, and 100 Monte Carlo trials both in characterization as well as in optimization to perform satisfactorily well across all the experiments. The execution time for such a configuration is a few milliseconds. This allows for hundreds of alternative query execution plans to be inspected in less than a second and confirms the applicability of the proposed estimator to query optimization.

10. RELATED WORK

Query optimization in relational database systems has been studied for over thirty five years, starting with the seminal paper on cost-based access path selection [Selinger et al. 1979]. Many survey articles [Ioannidis 1996; Chaudhuri 1998; Kaushik et al. 2005; Haas et al. 2009] provide a thorough view on cost-based query optimization both from a system perspective, as well as a theoretical and statistical standpoint. Other articles [Graefe 1995] discuss rule-based query optimization. Without exception, all this work focuses on select-project-join queries and ignores the antijoin operator. To date, the antijoin cardinality estimator used in standard query optimizers is still the simplified estimator introduced in [Selinger et al. 1979] (Section 3.1). This is a good example of an optimization problem that did not see any progress since the beginning of the relational systems. A SIGMOD Blog post by G. Lohman [Lohman 2014] provides other such examples. The reason for the lack of innovation is not the satisfactory performance of the standard estimator, but rather the difficulty of the problem, as pointed out in [Cormode et al. 2012].

Progressive optimization. Two strategies to handle inaccurate cardinality estimates for select-project-join queries are progressive optimization [Kabra and DeWitt 1998; Stillger et al. 2001; Markl et al. 2003; Markl et al. 2004] and query re-optimization [Babu et al. 2005]. The idea is to monitor the cardinality of intermediate results during query processing, compare these live cardinalities with the cardinalities used by the query optimizer when choosing the plan, and re-optimizing, i.e., changing the execution plan, whenever there is sufficient evidence that the current plan is not optimal. The decision to change the plan during processing has to factor-in the work that has already been done and might be lost. As an additional step, the statistics stored in the query optimizer can be reconciled with the statistics gathered at runtime. [Moerkotte et al. 2009] introduces a q-error measure that directly relates cardinality estimates to query plan costs. The q-error improves the detection accuracy of bad plans in progressive optimization. The estimator we propose in this paper is similar in spirit to progressive optimization in using workload information to improve estimation

accuracy. However, histograms are the synopsis targeted in progressive optimization. Antijoin cardinality is still estimated with the standard inaccurate estimator. A possible research direction is to integrate sampling – and the proposed antijoin estimator – in progressive optimization.

Workload-driven histogram construction. The idea of using workload information to build better statistics is pushed to extreme in [Aboulnaga and Chaudhuri 1999; Bruno et al. 2001]. STHoles histograms and their subsequent variants [Srivastava et al. 2006; Kaushik and Suciu 2009] are built entirely from the query workload, without scanning the data. Initially, the histogram does not contain any data. When a query is processed, exact cardinality information is gathered for the predicates appearing in the query and an initial histogram is built. The histogram is refined accordingly as more queries are executed. Since STHoles histograms contain only range frequency data, they are not applicable to antijoin queries. Nonetheless, the same principle can be used to extract the workload match frequency histograms required when building the superpopulation model.

Sampling synopses. There is a plethora of work on sampling-based estimation published in the database literature. We point the interested reader to the book [Cormode et al. 2012] which surveys synopses for approximate query processing over massive data. Sampling is introduced as a synopsis for query optimization in [Haas and Swami 1992]. With almost no exception, all the work on sampling focuses on size of join [Ganguly et al. 1996; Chaudhuri et al. 1999] and distinct value [Haas et al. 1995; Gibbons 2001] estimation. There is no work on antijoin cardinality estimation in the context of query optimization. It is important to notice that, while `DISTINCT` can be expressed as an antijoin query (Section 2), the reverse is not true. Thus, sampling-based solutions for distinct value estimation are not applicable to antijoin cardinality estimation.

Join [Acharya et al. 1999] and CS2 [Yu et al. 2013] synopses are representative implementations of the correlated estimator. They are identical for the case of two relations. In the case of multiple relations, instead of drawing an independent sample from each individual relation, the sample is extracted from the join, following key-foreign key relationships. In a join synopsis, a sample is built starting from each relation. A single CS2 synopsis is built for an entire database starting from a designated source relation. The CS2 synopsis is subsequently enriched with supplement samples, i.e., join synopses, at other relations, ultimately degenerating into a join synopsis. Join – and CS2 – synopses can be used for antijoin cardinality estimation between two relations connected by a key-foreign key relationship, only for a predicate from the foreign key to the key—the direction in which the synopsis is built. Of course, without additional selection predicates, the result of such an antijoin is zero. Estimating the antijoin cardinality of any other predicate requires the construction of a separate synopsis. This is a rather demanding constraint for a query optimizer, impossible to achieve in practice due to the unlimited number of possible antijoin predicates.

TuG synopses [Spiegel and Polyzotis 2009] are often classified in the same category with join and CS2 synopses because they are constructed over the relational schema graph. However, they are not sampling-based synopses and they do not impose a direction on the graph. They are tuple-based holistic data summaries that combine histograms with exact join cardinality computation. A TuG synopsis consists of multiple such data summaries built over separate partitions of the relational schema graph. Estimation is done using the value and join independence assumptions, specific to histograms. These can be extended to antijoin cardinality estimation following the same principles discussed in Section 3.1 and with corresponding degradation in accuracy. We do not include TuG synopses in the experimental comparison with the other estimators because of two reasons. First, they are not sampling-based synopses. Second, join and CS2 synopses are shown to be more accurate and considerably faster to construct than the TuG synopsis in [Yu et al. 2013]. These are included in our comparison with the correlated estimator.

Antijoin cardinality estimation. Essentially, there are only two papers that address subset-based query estimation as a generalization of antijoin cardinality estimation. The first work that introduces an estimator for subset-based queries is the online estimation paper [Jermaine et al. 2005]. The authors acknowledge the difficulty of designing an unbiased estimator and propose an alternative estimator for a simplified version of the problem in which a sample is taken only from the outer relation while the inner relation is always inspected in full. In order to make this process efficient,

an index is built over the inner relation that allows for the fast identification of the matching records corresponding to any record $r \in R$. Due to this strict requirement, the applicability of the concurrent estimator [Jermaine et al. 2005] is limited to scenarios where such an index exists.

The paper by Joshi and Jermaine on sampling-based estimators for subset-based queries [Joshi and Jermaine 2009] is the closest to the work we propose in this paper. The authors introduce two estimators – unbiased and biased – derived exclusively from the samples R' and S', respectively. The unbiased estimator has large variance due to the numerical stability issues arising in the computation of the hypergeometric pmf. The real problem though is the factorial algorithm in the number of histogram bins which practically renders the unbiased estimator unusable for m values larger than 10. The biased estimator proposed in [Joshi and Jermaine 2009] follows the same superpopulation approach as the estimator we introduce in this paper. However, we use a more complex mixture superpopulation model that captures the reality more accurately and is less prone to over-fitting. There are two important differences that render our estimator applicable to query optimization, even though it is more complex. The first difference is the learning phase. We use prior workload information to learn the superpopulation model offline, once, while the biased estimator in [Joshi and Jermaine 2009] learns a different model online, for every query, using only the samples. Moreover, we use a Bayesian statistics framework to update the model dynamically based on the runtime query workload. The second difference is the `CD sampling` algorithm. This is our main technical contribution that makes the proposed estimator amenable for query optimization. The biased estimator in [Joshi and Jermaine 2009] implements the naive Monte Carlo sampling procedure. Notice, however, that `CD sampling` can replace naive sampling in the biased estimator.

Bootstrap. Bootstrap is a statistical Monte Carlo method to derive the probability distribution of an estimator computed over samples. It is widely applied to derive confidence bounds for complex aggregate queries [Pol and Jermaine 2005]. Following the presentation in this paper, bootstrap consists in taking a sample from a superpopulation and then resampling the sample repetitively. The query at hand is evaluated on each resample to produce an estimate. The empirical distribution is obtained by collecting all the estimates together—without defining a parametric distribution, though. Recent work [Agarwal et al. 2014; Zeng et al. 2014] proposes bootstrap algorithms that avoid the inefficiencies required by resampling—similar in nature to `CD sampling`. In [Agarwal et al. 2014], the authors use a Poisson distribution with mean 1 to estimate the number of times each tuple in the sample is part of the resample—without executing the resampling. The constraint on the resample size is dropped in favor of ease of parallelization. In [Zeng et al. 2014], while the authors enforce the resample size constraint, they approximate the multinomial distribution corresponding to a resample with a multivariate normal distribution having a more concise representation. This results in reduced storage overhead. The main difference between these efficient bootstrap algorithms and `CD sampling` is that they provide concise representations for a single relation, while `CD sampling` operates over the match frequency histogram of two relations connected by an antijoin operator. Moreover, the dimensionality of the distributions used in bootstrap is given by the size of the relation, while in our case it is given by the number of bins in the match frequency histogram. As a result, the execution time for the bootstrap algorithms is considerably higher. It is not clear how this can be reduced for the analytical bootstrap algorithms to be integrated into a query optimizer.

11. CONCLUSIONS AND FUTURE WORK

In this paper, we present a novel sampling-based estimator for antijoin cardinality that provides the required efficiency to be implemented in a query optimizer. The proposed estimator incorporates three novel ideas. First, it uses prior workload information when learning a mixture superpopulation model of the data offline. Second, we design a Bayesian statistics framework that updates the model according to the live queries, thus allowing the estimator to adapt dynamically to the online workload. And third, we develop an efficient algorithm to sample from a hypergeometric distribution in order to generate Monte Carlo trials without explicitly instantiating either the population or the sample. When put together, these ideas form the basis of an efficient antijoin cardinality estimator satisfying the strict requirements of a query optimizer as shown by the extensive experimental re-

sults over synthetically-generated as well as massive TPC-H data. We plan to extend the proposed estimator to parallel settings where samples are distributed across computing nodes in future work.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments that improved the quality of this article. In particular, the proof of Theorem 6.1 is largely provided by one of the anonymous reviewers. We are grateful for the time spent on this article and the insightful comments that increased the quality of this work significantly.

REFERENCES

A. Aboulnaga and S. Chaudhuri. 1999. Self-Tuning Histograms: Building Histograms Without Looking at Data. In *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data*. 181–192. DOI: http://dx.doi.org/10.1145/304182.304198

S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. 1999. Join Synopses for Approximate Query Answering. In *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data*. 275–286. DOI: http://dx.doi.org/10.1145/304182.304207

S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. 2014. Knowing when You'Re Wrong: Building Fast and Reliable Approximate Query Processing Systems. In *Proceedings of 2014 ACM SIGMOD International Conference on Management of Data*. 481–492. DOI: http://dx.doi.org/10.1145/2588555.2593667

S. Babu, P. Bizarro, and D. DeWitt. 2005. Proactive Re-Optimization. In *Proceedings of 2005 ACM SIGMOD International Conference on Management of Data*. 107–118. DOI: http://dx.doi.org/10.1145/1066157.1066171

T. Benaglia, D. Chauveau, D. R. Hunter, and D. Young. 2009. mixtools: An R Package for Analyzing Finite Mixture Models. *Journal of Statistical Software* 32, 6 (2009), 1–29.

J. A. Bilmes. 1998. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. (1998). [Online at: ftp://ftp.icsi.berkeley.edu/pub/techreports/1997/tr-97-021.pdf; accessed September 2014].

N. Bruno, S. Chaudhuri, and L. Gravano. 2001. STHoles: A Multidimensional Workload-Aware Histogram. In *Proceedings of 2001 ACM SIGMOD International Conference on Management of Data*. 211–222. DOI: http://dx.doi.org/10.1145/375663.375686

S. Chaudhuri. 1998. An Overview of Query Optimization in Relational Systems. In *Proceedings of 1998 ACM PODS Symposium on Principles of Database Systems*. 34–43. DOI: http://dx.doi.org/10.1145/275487.275492

S. Chaudhuri, R. Motwani, and V. Narasayya. 1999. On Random Sampling over Joins. In *Proceedings of 1999 ACM SIGMOD International Conference on Management of Data*. 263–274.

M. Colgan. 2010. Optimizer Transformations: Subquery Unesting, Part 2. (2010). [Online at: https://blogs.oracle.com/optimizer/entry/optimizer_transformations_subquery_unesting_part_2; accessed September 2014].

G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends in Databases* 4, 1–3 (2012), 1–294.

A.P. Dempster, N.M. Laird, and D.B. Rubin. 1977. Maximum-Likelihood from Incomplete Data via the EM Algorithm. *J. Royal Statist.* 39 (1977).

A. Dobra, C. Jermaine, F. Rusu, and F. Xu. 2009. Turbo-Charging Estimate Convergence in DBO. *PVLDB* 2, 1 (2009), 419–430. DOI: http://dx.doi.org/10.14778/1687627.1687675

S. Ganguly, P. B. Gibbons, Y. Matias, and A. Silberschatz. 1996. Bifocal Sampling for Skew-Resistant Join Size Estimation. *SIGMOD Record* 25, 2 (1996), 271–281. DOI: http://dx.doi.org/10.1145/235968.233340

A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. 2003. *Bayesian Data Analysis*. Chapman & Hall/CRC.

P. B. Gibbons. 2001. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *Proceedings of 2001 VLDB International Conference on Very Large Data Bases*. 541–550.

G. Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data Engineering Bulletin* 18, 3 (1995), 19–29.

P. J. Haas, I. F. Ilyas, G. M. Lohman, and V. Markl. 2009. Discovering and Exploiting Statistical Properties for Query Optimization in Relational Databases: A Survey. *Statistical Analysis and Data Mining* 1, 4 (2009), 223–250.

P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. 1995. Sampling-Based Estimation of the Number of Distinct Values of an Attribute. In *Proceedings of 1995 VLDB International Conference on Very Large Data Bases*. 311–322.

P. J. Haas and A. N. Swami. 1992. Sequential Sampling Procedures for Query Size Estimation. In *Proceedings of 1992 ACM SIGMOD International Conference on Management of Data*. 341–350. DOI: http://dx.doi.org/10.1145/130283.130335

Y. E. Ioannidis. 1996. Query Optimization. *ACM Computing Surveys* 28, 1 (1996), 121–123. DOI: http://dx.doi.org/10.1145/234313.234367

Y. E. Ioannidis. 2003. The History of Histograms (abridged). In *Proceedings of 2003 VLDB International Conference on Very Large Data Bases*. 19–30.

Y. E. Ioannidis and S. Christodoulakis. 1993. Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results. *Transactions on Database Systems (TODS)* 18, 4 (1993), 709–748. DOI: http://dx.doi.org/10.1145/169725.169708

C. Jermaine, A. Dobra, A. Pol, and S. Joshi. 2005. Online Estimation for Subset-Based SQL Queries. In *Proceedings of 2005 VLDB International Conference on Very Large Data Bases*. 745–756.

N. L. Johnson, S. Kotz, and N. Balakrishnan. 1996. *Discrete Multivariate Distributions*. John Wiley & Sons, Inc.

S. Joshi and C. Jermaine. 2009. Sampling-Based Estimators for Subset-Based Queries. *VLDB Journal* 18, 1 (2009), 181–202. DOI: http://dx.doi.org/10.1007/s00778-008-0095-0

N. Kabra and D. DeWitt. 1998. Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. In *Proceedings of 1998 ACM SIGMOD International Conference on Management of Data*. 106–117. DOI: http://dx.doi.org/10.1145/276304.276315

R. Kaushik, J. F. Naughton, R. Ramakrishnan, and V. T. Chakaravarthy. 2005. Synopses for Query Optimization: A Space-Complexity Perspective. *Transactions on Database Systems (TODS)* 30, 4 (2005), 1102–1127. DOI: http://dx.doi.org/10.1145/1114244.1114251

R. Kaushik and D. Suciu. 2009. Consistent Histograms in the Presence of Distinct Value Counts. *PVLDB* 2, 1 (2009), 850–861. DOI: http://dx.doi.org/10.14778/1687627.1687723

G. M. Lohman. 2014. Is Query Optimization a "Solved" Problem? (2014). [Online at: http://wp.sigmod.org/?p=1075; accessed May 2014].

V. Markl, G. Lohman, and V. Raman. 2003. LEO: An Automatic Query Optimizer for DB2. *IBM Systems Journal* 42, 1 (2003), 98–106. DOI: http://dx.doi.org/10.1147/sj.421.0098

V. Markl, V. Raman, D. E. Simmen, G. M. Lohman, and H. Pirahesh. 2004. Robust Query Processing through Progressive Optimization. In *Proceedings of 2004 ACM SIGMOD International Conference on Management of Data*. 659–670. DOI: http://dx.doi.org/10.1145/1007568.1007642

G. Moerkotte, T. Neumann, and G. Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *PVLDB* 2, 1 (2009), 982–993. DOI: http://dx.doi.org/10.14778/1687627.1687738

K. P. Murphy. 2006. Binomial and Multinomial Distributions. (2006). [Online at: www.cs.ubc.ca/~murphyk/Teaching/CSE40-Fall06/reading/bernoulli.pdf; accessed September 2014].

A. Pol and C. Jermaine. 2005. Relational Confidence Bounds Are Easy with the Bootstrap. In *Proceedings of 2005 ACM SIGMOD International Conference on Management of Data*. 587–598. DOI: http://dx.doi.org/10.1145/1066157.1066224

V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. 1996. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proceedings of 1996 ACM SIGMOD International Conference on Management of Data*. 294–305. DOI: http://dx.doi.org/10.1145/233269.233342

C. P. Robert and G. Casella. 2005. *Monte Carlo Statistical Methods*. Springer.

R. Schrag. 2005. Speeding Up Queries with Semi-Joins and Anti-Joins: How Oracle Evaluates EXISTS, NOT EXISTS, IN, and NOT IN. (2005). [Online at: http://www.dbspecialists.com/files/presentations/semijoins.html; accessed September 2014].

P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. 1979. Access Path Selection in a Relational Database Management System. In *Proceedings of 1979 ACM SIGMOD International Conference on Management of Data*. 23–34. DOI: http://dx.doi.org/10.1145/582095.582099

J. Spiegel and N. Polyzotis. 2009. TuG Synopses for Approximate Query Answering. *Transactions on Database Systems (TODS)* 34, 1 (2009). DOI: http://dx.doi.org/10.1145/1508857.1508860

U. Srivastava, P. J. Haas, V. Markl, and N. Megiddo. 2006. ISOMER: Consistent Histogram Construction Using Query Feedback. In *Proceedings of 2006 IEEE ICDE International Conference on Data Engineering*. 39. DOI: http://dx.doi.org/10.1109/ICDE.2006.84

M. Stillger, G. Lohman, V. Markl, and M. Kandil. 2001. LEO—DB2's Learning Optimizer. In *Proceedings of 2001 VLDB International Conference on Very Large Data Bases*. 19–28.

A. Swami and K. B. Schiefer. 1994. On the Estimation of Join Result Sizes. In *Proceedings of 1994 EDBT Extended Database Technology Conference*. 287–300.

F. Yu, W. Hou, C. Luo, D. Che, and M. Zhu. 2013. CS2: A New Database Synopsis for Query Estimation. In *Proceedings of 2013 ACM SIGMOD International Conference on Management of Data*. 469–480. DOI: http://dx.doi.org/10.1145/2463676.2463701

K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. 2014. The Analytical Bootstrap: A New Method for Fast Error Estimation in Approximate Query Processing. In *Proceedings of 2014 ACM SIGMOD International Conference on Management of Data*. 277–288. DOI: http://dx.doi.org/10.1145/2588555.2588579

Received ; revised ; accepted