A Sampling Algebra for Aggregate Estimation

Supriya Nirkhiwale
University of Florida
supriyan@ufl.edu

Alin Dobra
University of Florida
adobra@cise.ufl.edu

Christopher Jermaine
Rice University
cmj4@cs.rice.edu

ABSTRACT

As of 2005, sampling has been incorporated in all major database systems. While efficient sampling techniques are realizable, determining the accuracy of an estimate obtained from the sample is still an unresolved problem. In this paper, we present a theoretical framework that allows an elegant treatment of the problem. We base our work on generalized uniform sampling (GUS), a class of sampling methods that subsumes a wide variety of sampling techniques. We introduce a key notion of equivalence that allows GUS sampling operators to *commute* with selection and join, and derivation of confidence intervals. We illustrate the theory through extensive examples and give indications on how to use it to provide meaningful estimates in database systems.

1. INTRODUCTION

Sampling has long been used by database practitioners to speed up query evaluation, especially over very large data sets. For many years it was common to see SQL code of the form "`WHERE RAND() > 0.99`". Widespread use of this sort of code lead to the inclusion of the `TABLESAMPLE` clause in the SQL-2003 standard [1]. Since then, all major databases have incorporated native support for sampling over relations. One such query, using the TPC-H schema, is:

```
SELECT SUM(l_discount*(1.0-l_tax))
FROM lineitem TABLESAMPLE (10 PERCENT),
     orders TABLESAMPLE (1000 ROWS)
WHERE l_orderkey  = o_orderkey AND
      l_extendedprice > 100.0;
```

The result of this query is obtained by taking a Bernoulli sample with $p = .1$ over `lineitem` and joining it with a sample of size 1000 obtained without replacement (WOR), from `orders` and *evaluating* the `SUM` aggregate.

In practice, there are two main reasons practitioners write such code. One is that sampling is useful for debugging expensive queries. The query can be quickly evaluated over a sample as a sanity check, before it is unleashed upon the full database.

The second reason is that the practitioner is interested in obtaining an idea as to what the actual answer to the query would be, in less time than would be required to run the query over the entire database. This might be useful as a prelude to running the query "for real"—the user might want to see if the result is potentially interesting—or else the estimate might be used in place of the actual answer. Often, this situation arises when the query in question performs an aggregation, since it is fairly intuitive to most users that sampling can be used to obtain a number that is a reasonable approximation of the actual answer.

The problem we consider in this paper comes from the desire to use sampling as an approximation methodology. In this case, the user is not actually interested in computing an aggregate such as "`SUM(l_discount*`(1.0-`l_tax))`" over a sample of the database. Rather, s/he is interested in estimating the *answer* to such a query over the entire database using the sample. This presents two obvious problems:

- First, what SQL code should the practitioner write in order to compute an estimate for a particular aggregate?
- Second, how does the practitioner have any idea how accurate that estimate is?

Ideally, a database system would have built-in mechanisms that automatically provide estimators for user supplied aggregate queries, and that automatically provide users with accuracy guarantees. Along those lines, in this paper we study how to automatically support SQL of the form:

```
CREATE VIEW APPROX (lo, hi) AS
SELECT QUANTILE(SUM(l_discount*(1.0-l_tax)), 0.05)
       QUANTILE(SUM(l_discount*(1.0-l_tax)), 0.95)
FROM lineitem TABLESAMPLE (10 PERCENT),
       orders TABLESAMPLE(1000 ROWS)
WHERE l_orderkey  = o_orderkey AND
        l_extendedprice > 100.0;
```

Presented with such a query, the database engine will use the user-specified sampling to automatically compute two values `lo` and `hi` that can be used as a $[0.05, 0.95]$ confidence bound on the true answer to the query. That is, the user has asked the system to compute values `lo` and `hi` such that there is a 5% chance that the true answer is less than `lo`, and there is a 95% chance that the true answer is less than `hi`. In the general case, the user should be able to specify any aggregate over any number of sampled base tables using any sampling scheme, and the system would automatically figure out how to compute an estimate of the desired quantile. A database practitioner need have no idea how to compute an estimate for the answer, nor does s/he need to have any idea how to compute confidence bounds; the user only specifies the desired quantiles, and the system does the rest.

Existing Work on Database Sampling. This is not an easy problem to solve. While there has been a lot of research on implementing efficient sampling algorithms [25, 24], providing confidence intervals for the sample estimate is understood only for a few restricted cases. The simplest is when only a single relation is sampled. A slightly more complicated case was handled by the AQUA system developed at Bell labs [13, 4, 5, 3]. AQUA considered *correlated sampling* where a fact table in a star schema is sampled. These cases are relatively simple because when a single table is sampled, classical sampling theory applies with a few easy modifications. Simultaneous work on ripple joins and online aggregation [18, 16, 15, 20, 17] extended the class of queries amenable to analysis to include those queries where multiple tables are sampled with replacement and then joined.

Unfortunately, the extension to other types of sampling is not straightforward, and to date new formulas have been derived every time a new sampling is considered (for example, two-table without-replacement sampling [23]). Our goal is to provide a simple theory that makes it possible to handle very general types of queries over virtually any uniform sampling scheme: with replacement sampling, fixed-size without replacement sampling, Bernoulli sampling, or whatever other sampling scheme is used. The ability to easily handle arbitrary types of sampling is especially important given that the current SQL standard allows for a somewhat mysterious `SYSTEM` sampling specification, whose exact implementation (and hence its statistical properties) are left up to the database designers. Ideally, it should be easy for a database designer to apply our theory to an arbitrary `SYSTEM` sampling implementation.

Generalized Uniform Sampling. One major reason that new theory and derivations were previously required for each new type of sampling is that the usual analysis is tuple-based, where the inclusion probability of each tuple in the output set is used as the basic building block; computing expected values and variances requires intricate algebraic manipulations of complicated summations. In previous work, we defined a notion that we called *Generalized Uniform Sampling* (GUS) (ref Section 3) that subsumes many different sampling schemes (including all of the aforementioned ones, as well as block-based variants thereof). In this paper, we develop an algebra over many common relational operators, as well as the GUS operator. This makes it possible to take any query plan that contains one or more GUS operators and the supported relational operators, and perform a statistical analysis of the accuracy of the result in an algebraic fashion, working from the leaves up to the top of the plan.

No complicated algebraic manipulations over nested summations are required. This algebra can form the basis for a lightweight tool for providing estimates and quantiles, that should be easily integrable into any database system. The database need only feed the tool the user-specified quantiles, the set of tuples returned by the query, some simple lineage information over those result tuples, and the query plan, and the tool can automatically compute the desired quantiles.

Our Contributions. The specific contributions we make in this paper are:

- We define the notion of Second Order Analytical equivalence (SOA equivalence), a key equivalence relationship between query plans that is strong enough to allow quantile analysis but weak enough to ensure commutativity of sampling and relational operators.
- We define the GUS operator that emulates a wide class of sampling methods. This operator commutes with most relational operators under SOA-equivalence.
- We develop an algebra over GUS and relational operators that allows derivation of SOA-equivalent plans. These plans easily allow moment calculations that can be used to estimate quantiles.
- We describe how our theory can be used to add estimation capabilities to existing databases so that the required changes to the query optimizer and execution engine are minimal. Alternatively, the estimator can be implemented as an external tool.

Our work provides a straightforward analysis for the SUM aggregate. It can be easily extended for COUNT by substituting the aggregated attribute to 1 and applying the analysis for SUM on this attribute. Though the analysis for AVERAGE presents a slightly non-linear case, the analyses for SUM and COUNT lay a foundation for it. The confidence intervals can be derived using a method for approximating probability distribution/variance such as the delta method. The analysis for MIN, MAX and DISTINCT are extremely hard problems to solve due to their non-linearity. For example DISTINCT requires an estimate of all the distinct values in the data and the number of such values. It is thus beyond the scope of this paper.

While selections and joins are the highlight of our paper, we show that SOA-equivalence allows analysis for other database operators like cross-product (compaction), intersection (concatenation) and union. Due to space constraints, we are unable to include all technical proofs, deeper implementation details and discussions. These are available in the extended version of this paper[2].

The rest of the paper is organized as follows. In Section 2, we provide a brief overview of related work in this area. In Section 3, we introduce GUS methods and give details on how to get estimates and confidence intervals for them. In Section 4, we introduce the notion of SOA-equivalence between query plans and prove that GUS operators commute with a variety of relational operators in the SOA sense. In Section 5, we investigate interactions between GUS operators when applied to the same data and explore more possibilities in using them. In Section 6, we provide insights on how our theory can be used to implement a separate add-on tool and how the performance of the variance estimation can be enhanced. In Section 7, we test our implementation thoroughly, and provide accuracy and runtime analysis. We explore some possible applications and conclude with a discussion in Section 8.

2. RELATED WORK

The idea of using sampling in databases for deriving estimates for a single relation was first studied by Shapiro et al. [26]. Since then, much research has focused on implementing efficient sampling algorithms in databases [25, 24]. Providing confidence intervals on estimates for SQL aggregate queries is difficult, which is why there has been limited progress in this area. Olken [25] studied the problem for specific sampling methods for a single relation. This line of work ended abruptly when Chaudhuri et al. [9, 10] proved that extracting IID samples from a join of two relations is infeasible.

Another line of research was the extension to the *correlated sampling* pioneered by the AQUA system [13, 3, 4]. AQUA is applicable to a star schema, where the goal is sampling from the fact table, and including all tuples in dimension tables that match selected fact table tuples. The AQUA type of sampling has been incorporated in DB2 [14].

The reason confidence intervals can be provided for AQUA type sampling is the fact that *independent identically distributed* (IID) samples are obtained from the set over which the aggregate is computed. A straightforward use of the *central limit theorem* readily allows computation of good estimates and confidence intervals. Indeed, it is widely believed [9, 10, 25, 13, 3, 4] that IID samples at the top of the query plan are required to provide *any* confidence interval. This idea leads to the search for a *sampling operator* that commutes with database operators. This endeavor proved to be very difficult from the beginning [9] when joins are involved. To see why this is the case, consider a tuple $t \in$ `orders` and two tuples u_1, u_2 in `lineitem` that join with t (i.e. they have the same value for `orderkey`). Random selection of tuples t, u_1, u_2 in the sample does not guarantee random selection of result tuples (t, u_1) and (t, u_2). If t is not selected, neither tuple can exist, and thus sampling is correlated. A lot of effort [9, 10] has been spent in finding practical ways to de-correlate the result tuples with only limited success.

Progress has been made using a different line of thought by Hellerstein and Hass [20] and the generalization in [19] for the special case of sampling with replacement. The problem of producing IID result samples is avoided by developing central limit theorem-like results for the combination of relation level sampling with replacement. The theory was generalized first to *sampling without replacement* for single join queries [23], then further generalized to arbitrary uniform sampling over base directions and arbitrary `SELECT-FROM-WHERE` queries without duplicate elimination in DBO [21], and finally to allow sampling across multiple relations in Turbo-DBO [12]. Even though some simplification occurred through these theoretical developments, they are mathematically heavy and hard to understand/interpret. Moreover, the theory, especially DBO and Turbo-DBO, is tightly coupled with the systems developed to exploit it.

Technically, one major problem in all the mathematics used to analyze sampling schemes is the fact the analyses use functions and summations over tuple domains, and not the operators and algebras that the database community is used to. This makes the theory hard to comprehend and apply. The fact that no database system picked up these ideas to provide a confidence interval facility is a direct testament of these difficulties.

3. GENERALIZED UNIFORM SAMPLING

Previous attempts at accommodating a sampling operator in a query plan were limited to specific sampling methods. In previous work [12], we analyzed a large class of sampling methods for which the analysis can be unified: Generalized Uniform Sampling (GUS). Sampling methods such as uniform sampling with/without replacement, Bernoulli sampling and more elaborate strategies like the chaining in [12] are members of the GUS family. Moreover, the variance of any GUS method can be efficiently estimated. We briefly introduce GUS methods in this section and investigate them further in this paper.

DEFINITION 1 (GUS SAMPLING [12]). *A randomized selection process $\mathcal{G}_{(a,\bar{b})}$ which gives a sample $\mathcal{R}$ from $\mathbf{R} = R_1 \times R_2 \times \cdots \times R_n$ is called Generalized Uniform Sampling (GUS) method, if, for any given tuples $t = (t_1, \ldots, t_n), t' = (t'_1, \ldots, t'_n) \in \mathbf{R}$, $P(t \in \mathcal{R})$ is independent of t, and $P(t, t' \in \mathcal{R})$ depends only on $\{i : t_i = t'_i\}$. In such a case, the GUS parameters a, $\bar{b} = \{b_T | T \subset \{1 : n\}\}$ are defined as:*

$$a = P[t \in \mathcal{R}]$$

$$b_T = P[t \in \mathcal{R} \wedge t' \in \mathcal{R} | \forall i \in T, t_i = t'_i, \forall j \in T^C, t_j \neq t'_j].$$

This definition requires GUS sampling to behave like a *randomized filter*. In particular, any GUS operator can be viewed as a selection process from the underlying data, a process that can introduce correlations. The uniformity of GUS requires that the randomized filtering is performed on *lineage* of tuples and not on the content. As simple as the idea is, expressing any sampling process in the form of GUS is a non-trivial task. Example 1 shows the calculation of GUS parameters for a simple case.

EXAMPLE 1. *In this example, we show how the GUS definition above can be used to characterize the estimation necessary for the query from the paper's introduction. We denote by `l_s` the Bernoulli sample with $p = 0.1$ from `lineitem` and by `o_s` the WOR sample of size 1000 from `orders`. We assume that cardinality of `orders` is 150000. Henceforth, for ease of exposition, we will denote all base relations involved by their first letters. For example, `lineitem` will be denoted by `l`.*

Applying the definition above and the independence between sampling processes, we can derive the parameters for this GUS as follows: For any tuple $t \in$ `lineitem` and tuple $u \in$ `orders`:

$$a = P[(t \in \texttt{l\_s}) \wedge (u \in \texttt{u\_s})] = 0.1 \times \frac{1000}{150000} = 6.667 \times 10^{-4}$$

since the base relations are sampled independently from each other. For any tuples $t, t' \in$ `lineitem` and $u, u' \in$ `orders`:

$$\begin{aligned} b_\varnothing &= P[(t, t' \in \texttt{l\_s}) \wedge (u, u' \in \texttt{o\_s})] \\ &= 0.1 \times 0.1 \times \frac{1000}{150000} \times \frac{999}{149999} \\ &= 4.44 \times 10^{-7}, \end{aligned}$$

and

$$\begin{aligned} b_o &= P[t \in \texttt{l\_s}] \times P[t' \in \texttt{l\_s} | t \in \texttt{l\_s}] \times P[u \in \texttt{o\_s}] \\ &= 0.1 \times 0.1 \times \frac{1000}{150000} = 6.667 \times 10^{-5}. \end{aligned}$$

Similarly,

$$\begin{aligned} b_l &= P[(t \in \texttt{l\_s}) \wedge (u, u' \in \texttt{o\_s})] \\ &= P[t \in \texttt{l\_s}] \times P[u \in \texttt{o\_s}] \times P[u' \in \texttt{o\_s} | u \in \texttt{o\_s}] \\ &= 0.1 \times \frac{1000}{150000} \times \frac{999}{149999} \\ &= 4.44 \times 10^{-6}. \end{aligned}$$

The last term is

$$b_{l,o} = P[(t \in \texttt{l\_s}) \wedge (u \in \texttt{o\_s})] = 0.1 \times \frac{1000}{150000} = 6.667 \times 10^{-4}.$$

Notice that the GUS **captures** *the entire estimation process, not only the two individual sampling methods. The above analysis dealt with a simple join consisting of two base relations. For more complex query plans, the derivation of GUS parameters would involve consideration of all possible interactions between participating tuples. This will make the analysis highly complex.*

The analysis of any GUS sampling method for a SUM-like aggregate is given as follows.

THEOREM 1. *[12] Let $f(t)$ be a function/property of $t \in R$, and $\mathcal{R}$ be the sample obtained by a GUS method $\mathcal{G}_{(a,\bar{b})}$. Then, the aggregate $\mathcal{A} = \sum_{\mathbf{t} \in R} f(\mathbf{t})$ and the sampling estimate $X = \frac{1}{a} \sum_{\mathbf{t} \in \mathcal{R}} f(\mathbf{t})$ have the property:*

$$\begin{aligned} E[X] &= \mathcal{A} \\ \sigma^2(X) &= \sum_{S \subset \{1:n\}} \frac{c_S}{a^2} y_S - y_\phi \end{aligned} \qquad (1)$$

with

$$\begin{aligned} y_S &= \sum_{t_i \in R_i | i \in S} \left(\sum_{t_j \in R_j | j \in S^C} f(t_i, t_j) \right)^2 \\ c_S &= \sum_{T \in \mathcal{P}(n)} (-1)^{|T|+|S|} b_T. \end{aligned}$$

The above theorem indicates that the GUS estimates of SUM-like aggregates are unbiased and that the variance is simply a linear combination of properties of the data, terms y_S and properties of the GUS sampling method c_S. Moreover, y_S can be estimated from samples of *any* GUS [12]. This result is not asymptotic; it gives the exact analysis even for very small samples. Once the estimate and the variance are computed, confidence intervals can be readily provided using either the normality assumption or the more conservative *Chebychev* bound [12].

In the rest of the paper, we will study GUS sampling methods in detail.

4. ANALYSIS OF SAMPLING QUERY PLANS

The high-level goal of this paper, is to introduce a tool that computes the confidence bounds of estimates based on sampling. Given a query plan with sampling operators interspersed at various points, our tool transforms it to an *analytically equivalent* query plan that has a particular structure: all relational operators except the final aggregate form a subtree that is the input to a single GUS sampling operator. The GUS operator feeds the aggregate operator that produces the final result. Note that this transformation is done solely for the purpose of computing the confidence bounds of the result; it does not provide a better alternative to the execution plan used as input. Once this transformation is accomplished, Theorem 1 readily gives the desired analysis – the equivalence ensures that the analysis for the special plan coincides with the analysis for the original plan.

A natural strategy to obtain the desired structure is to perform multiple local transformations on the original query plan. These local transformations are based on a notion of analytical equivalence, that we call Second Order Analytical (SOA) equivalence. They allow both commutativity of relational and GUS operators, and consolidation of GUS operators. Effectively, these local transformations allow a plan to be put in the special form in which there is a single GUS operator just before the aggregate.

In this section, we first define the SOA-equivalence and then use it to provide *equivalence* relationships that allow the plan transformations mentioned above. A more elaborate example showcases the theory in the latter part of the section.

4.1 SOA-Equivalence

The main reason the previous attempts to design a *sampling operator* were not fully successful is the requirement to ensure IID samples at the top of the plan. Having IID samples makes the analysis easy since Central Limit Theorem readily provides confidence intervals. However it is too restrictive to allow plans with multiple joins to be dealt with. It is important to notice that the difficulty is not in *executing* query plans containing sampling but in *analyzing* such query plans.

The fundamental question we ask in this section is: What is the least restrictive requirement we can have and still produce useful estimates? Our main interest is in how the requirement can be transformed into a notion of *equivalence*. This will enable us to talk about *equivalent plans*, initially, but more usefully about *equivalent expressions*. The key insight comes from the observation that it is enough to compute the expected value and variance for any query plan. Then either the conservative Chebychev bounds or the optimistic[1] normal-distribution based bounds can be used to produce confidence intervals. Note that confidence intervals are the end goal, and, preserving expected value and variance is enough to guarantee the same confidence interval using both CLT and Chebychev methods.

Thus, for our purposes, *two query plans are equivalent if their result has the same expected value and variance*. This equivalence relation between plans already allows significant progress. It is an extension of the classic plan equivalence based on obtaining the same answer to *randomized* plans. From an operational sense, though, the plan equivalence is not sufficient to provide interesting characterizations. The main problem is the fact that the equivalence exists only between complete plans that compute aggregates. It is not clear what can be said about intermediate results–the equivalent of non-aggregate relational algebra expressions.

The key to extend the equivalence of plans to equivalence of expressions is to first design such an extension for the clas-

[1]While the CLT does not apply due to the lack of IID samples, the distribution of most complex random variables made out of many loosely interacting parts tends to be normal.

sic relational algebra. To this end, assume that we can only use equality on numbers that are results of SUM-like aggregates but we cannot directly compare sets. To ensure that two expressions are equivalent, we could require that they produce the same answer using **any** SUM-aggregate. Indeed, if the expressions produce the same relation/set, they *must* agree on *any* aggregate computation using these sets since aggregates are deterministic and, more importantly, do not depend on the order in which the computation is performed. The SUM-aggregates are crucial for this definition since they form a vector space. Aggregates A_t that sum function $f_t(u) = \delta_{tu}$ are the basis of this vector space; agreement on these aggregates ensures set agreement. Extending these ideas to randomized estimation, we obtain the following.

Definition 2 (SOA-equivalence). *Given (possibly randomized) expressions $\mathcal{E}(R)$ and $\mathcal{F}(R)$, we say*

$$\mathcal{E}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{F}(R)$$

*if for **any arbitrary** SUM-aggregate $\mathcal{A}_f(S) = \sum_{t \in S} f(t)$,*

$$E[\mathcal{A}_f(\mathcal{E}(R))] = E[\mathcal{A}_f(\mathcal{F}(R))]$$
$$Var[\mathcal{A}_f(\mathcal{E}(R))] = Var[\mathcal{A}_f(\mathcal{F}(R))].$$

From the above discussion, it immediately follows that SOA-equivalence is a generalization and implies set equivalence for non-randomized expressions, as stated in the following proposition.

Proposition 1. *Given two relational algebra expressions $E(R)$ and $F(R)$ we have:*

$$E(R) = F(R) \Leftrightarrow E(R) \overset{SOA}{\Longleftrightarrow} F(R).$$

The next proposition establishes that SOA-equivalence is indeed an *equivalence relation* and can be manipulated like relational equivalence.

Proposition 2. *SOA-equivalence is an equivalence relation, i.e., for any expressions $\mathcal{E}, \mathcal{F}, \mathcal{H}$ and relation R:*

$$\mathcal{E}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{E}(R)$$
$$\mathcal{E}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{F}(R) \Rightarrow \mathcal{F}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{E}(R)$$
$$\mathcal{E}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{F}(R) \wedge \mathcal{F}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{H}(R) \Rightarrow \mathcal{E}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{H}(R).$$

SOA-equivalence subsumes relational algebra equivalence. The strength of SOA-equivalence is the fact that it does not depend on a notion of randomized set equivalence, an equivalence that would be hard to define especially if it has to preserve aggregates.

Proposition 3. *Given two relational algebra expressions $\mathcal{E}(R)$ and $\mathcal{F}(R)$ we have:*

$$\mathcal{E}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{F}(R)$$
$$\Leftrightarrow$$
$$\forall t \in R, \quad P[t \in \mathcal{E}(R)] = P[t \in \mathcal{F}(R)] \quad and$$
$$\forall t, u \in R, \quad P[t, u \in \mathcal{E}(R)] = P[t, u \in \mathcal{F}(R)].$$

Proposition 3 provides a powerful alternative to SOA-equivalence. This equivalence is in terms of first and second

Sampling method	GUS parameters
Bernoulli(p)	$a = p$, $b_\varnothing = p^2$, $b_{\mathtt{R}} = p$
WOR (n, N)	$a = \frac{n}{N}, b_\varnothing = \frac{n(n-1)}{N(N-1)}, b_{\mathtt{R}} = \frac{n}{N}$

Figure 1: GUS parameters for known sampling methods on a single relation

order probabilities, and we refer to it as *SOA-set equivalence.* Another way to interpret the result above is that SOA-set equivalence is the same as agreement on all SUM-like aggregates. More importantly for this paper, SOA-set equivalence provides an alternative proof technique to show SOA-equivalence. Often, proofs based on SOA-set equivalence are simpler and more compact.

Section 6 contains a recipe for expected value and variance computation for a specific situation, when there is a single overall GUS sampling on top. Starting with the given query plan that contains both sampling and relational operators, if we find a SOA-equivalent plan that is equivalent and has no sampling except a GUS at the top, we readily have a way to compute the expected value and variance of the original plan. In the rest of this section we pursue this idea further and show how SOA-equivalent plans with the desired structure can be obtained from a general query plan.

4.2 GUS Quasi-Operators

Except under restrictive circumstances, the sampling operators will not commute with relational operators. This, as we mentioned is the main reason previous work made limited progress on the issue. As we will see later in this section, GUS sampling does commute in a SOA-equivalence sense with most relational operators. The reason we can commute GUS (but not specific sampling methods) is that, due to its generality, it can *capture* the correlations induced by the relational operators. The first step in our analysis has to be a *translation* from specific sampling to GUS-sampling.

Before we talk about the translation from sampling to GUS operators, we need to clarify and refine the Definition 1 of GUS sampling. As part of the definition, terms of the form $t_i = t'_i$ or $t_j \neq t'_j$ are used. Intuitively, they capture the idea that tuples (or parts) are the same or different. Since in this paper we will have multiple GUS operators involved, it is important to make the meaning of such terms very clear. We do this through a notion that proved useful in probabilistic databases (among other uses): *lineage*[11]. Lineage allows dissociation of the ID of a tuple from the content of the tuple, for base relation, and tracking the composition of derived tuples. With this, $t_i = t'_i$ means that the two tuples are the same – have the same ID/lineage – not that they have the same content.

Representing and manipulating lineage is a complex subject. In this work, since we only accommodate selection and joins the issue is significantly simpler. The selection leaves lineage unchanged, the lineage of the result of the join is the union of the lineage of the matching tuples. Thus, lineage can be represented in relational form with one attribute for each base relation participating in the expression. We can thus talk about *lineage schema* $\mathcal{L}(R)$, a synonym of the set of base relations participating in the expression of R. The lineage of a specific tuple $t \in R$ will have values for the lineage of all base relations constituting R. A particularly useful notation related to lineage is:

$\mathcal{T}(t,t') = \{R_k | t_k = t_k', k \in \mathcal{L}(R)\}$, the common part of the lineage of tuples t and t', i.e. the base relations on which the lineage of t and t' agree.

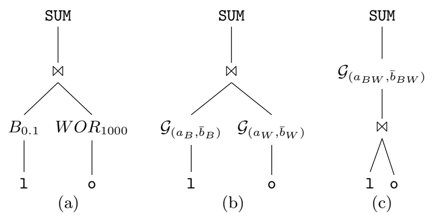

SUM
⋈
$B_{0.1}$
WOR_{1000}
l
o
(a)
SUM
⋈
$\mathcal{G}_{(a_B,\bar{b}_B)}$
$\mathcal{G}_{(a_W,\bar{b}_W)}$
l
o
(b)
SUM
$\mathcal{G}_{(a_{BW},\bar{b}_{BW})}$
⋈
l
o
(c)

Figure 2: Query 1

Example 2. *The query from the introduction uses two sampling methods: Bernoulli sampling with* $p = 0.1$ *on* `lineitem` *and sampling 1K tuples without replacement from* `orders`*(150K tuples). These methods can be expressed in terms of GUS as* $\mathcal{G}_{(a_B,\bar{b}_B)}$ *and* $\mathcal{G}_{(a_W,\bar{b}_W)}$ *as follows: For* $\mathcal{G}_{(a_B,\bar{b}_B)}$*:* $a_B = 0.1$ *and* $\bar{b}_B = \{b_{B,\varnothing}, b_{B,\imath}; b_{B,\varnothing} = 0.01, b_{B,\imath} = 0.1\}$*. For* $\mathcal{G}_{(a_W,\bar{b}_W)}$*:* $a_W = 6.667 \times 10^{-3}$ *and* $\bar{b}_W = \{b_{W,\varnothing}, b_{W,o}; b_{W,\varnothing} = 4.44 \times 10^{-5}, b_{W,o} = 6.667 \times 10^{-3}\}$

It is important to note that the GUS is not an operator but a quasi-operator. While it corresponds to a real operator when the translation from specific sampling to GUS happens, it will not correspond to an operator after transformations. There is no need to provide or even to consider an implementation of a general GUS operator since GUS will only be used for the purpose of analysis.

In the rest of this section, we will assume that all specific sampling operators were replaced by GUS quasi-operators, thus will not be encountered by the re-writing algorithm. We designate by $\mathcal{G}_{(a,\bar{b})}(R)$, a GUS method applied to a relation R, and the resulting sample by $\mathcal{R}$. When multiple GUS methods are used, the i'th GUS method and its resulting sample will be denoted by $\mathcal{G}_{(a_i,\bar{b}_i)}$ and $\mathcal{R}_i$ respectively.

4.3 Interaction Between GUS and Rel Ops

As we stated in Section 4.1, SOA-equivalence is the key for deriving an *analyzable* plan that is *equivalent* to the one provided by the user. The results in this section provide equivalences that allow such transformations that lead to a single, top, GUS operator. The results in this section make use of the notation in Table 4.2.

Proposition 4 (Identity GUS). *The quasi-operator* $\mathcal{G}_{(1,\bar{1})}$*, i.e. a GUS operator with* $a = 1$*,* $b_T = 1$*, can be inserted at any point in a query plan without changing the result.*

Proof. Since $a = 1$, all input tuples are allowed with probability 1, i.e., no filtering happens. □

Notation	Meaning
$\mathcal{R}$	Random subset of R
a	$P[t \in \mathcal{R}]$
$\mathcal{L}(R)$	Lineage schema of R
$\mathcal{L}(t)$	Lineage of tuple t
T	Subset of $\mathcal{L}(R)$
$\mathcal{T}(t,t')$	$\{R_k \| t_k = t_k', k \in \mathcal{L}(R)\}$
b_T	$P[t, t' \in \mathcal{R} \| T = \mathcal{T}(t,t')]$
$\bar{b}$	$\{b_T \| T \in \mathcal{P}(n)\}$
$\mathcal{G}_{(a,\bar{b})}$	GUS method with parameters a and $\bar{b}$
$\mathcal{G}_{(a,\bar{b})}(R)$	$\mathcal{G}_{(a,\bar{b})}$ applied to relation R

Figure 3: Notation used in this section

Proposition 5 (Selection-GUS Commutativity). *For any* R*, selection* σ_C *and GUS* $\mathcal{G}_{(a,\bar{b})}$*,*

$$\sigma_C(\mathcal{G}_{(a,\bar{b})})(R) \overset{SOA}{\Longleftrightarrow} \mathcal{G}_{(a,\bar{b})}(\sigma_C(R)).$$

Proof. Let $R' = \sigma_C(R)$. On computing $\mathcal{R} \cap R'$ we see that

$$\forall (t \in R'), P[t \in \mathcal{R} \cap R'] = P[t \in \mathcal{R}] I_{\{t \in R'\}} = a.$$

$$\begin{aligned}\forall (t, t' \in R'), & P[t, t' \in \mathcal{R} \cap R' | T = \mathcal{T}(t,t')] \\ &= P[t, t' \in \mathcal{R} | T = \mathcal{T}(t,t')] = b_T \quad \square\end{aligned}$$

The above results are somewhat expected and have been covered for particular cases in previous literature. The following result, though, overcomes the difficulties in [10].

Proposition 6 (Join-GUS Commutativity). *For any* R*,* S*, join* $\bowtie_\theta$ *and GUS methods* $\mathcal{G}_{(a_1,\bar{b}_1)}, \mathcal{G}_{(a_2,\bar{b}_2)}$*, if* $\mathcal{L}(R_1) \cap \mathcal{L}(R_2) = \varnothing$

$$\begin{aligned}\mathcal{G}_{(a_1,\bar{b}_1)}(R_1) \bowtie_\theta \mathcal{G}_{(a_2,\bar{b}_2)}(R_2) &\overset{SOA}{\Longleftrightarrow} \mathcal{G}_{(a,\bar{b})}(R_1 \bowtie_\theta R_2), \\ \textit{where}, \qquad a = a_1 a_2, &\qquad b_T = b_{1,T_1} b_{2,T_2}\end{aligned}$$

with $T_1 = T \cap \mathcal{L}(R_1)$ *and* $T_2 = T \cap \mathcal{L}(R_2)$*.*

Proof. We proved in Proposition 5 that a GUS method commutes with selection. Thus, it is enough to prove commutativity of a GUS method with cross product. Let $R = R_1 \times R_2$ and $t = (t_1, t_2), t' = (t_1', t_2') \in R$. Thus, $\mathcal{L}(R) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$. We have:

$$\begin{aligned} a = P[t \in \mathcal{R}] &= P[t_1 \in \mathcal{R}_1 \wedge t_2 \in \mathcal{R}_2] \\ &= P[t_1 \in \mathcal{R}_1 \wedge t_2 \in \mathcal{R}_2] = a_1 a_2.\end{aligned}$$

Since $\mathcal{L}(R_1) \cap \mathcal{L}(R_2) = \varnothing$, for an arbitrary $T \in \mathcal{L}(R)$, $T_1 = T \cap \mathcal{L}(R_1)$ and $T_2 = T \cap \mathcal{L}(R_2)$ we have, $T_1 \cap T_2 = \varnothing$ (disjunct lineage). With this, we first get:

$$\mathcal{T}(t,t') = T \Leftrightarrow E_1 \cap E_2,$$

where $E_1 = \{\mathcal{T}(t_1, t_1') = T_1\}$ and $E_2 = \{\mathcal{T}(t_2, t_2') = T_2\}$. Using the above and independence of GUS methods,

$$\begin{aligned} b_T &= P[t \in \mathcal{R} \wedge t' \in \mathcal{R} | \mathcal{T}(t,t') = T] \\ &= P[t_1, t_1' \in \mathcal{R}_1 \wedge t_2, t_2' \in \mathcal{R}_2 | E_1, E_2] \\ &= P[t_1, t_1' \in \mathcal{R}_1 | E_1] P[t_2, t_2' \in \mathcal{R}_2 | E_2] \\ &= b_{1,T_1} b_{2,T_2}. \quad \square \end{aligned}$$

Example 3. *Applying the above results to the GUS coefficients obtained in Example 2, we can derive the following co-efficients for* $\mathcal{G}_{(a,\bar{b})}$ *in Fig 4.2:*

$$a = a_1 a_2 = 0.1 \times 6.667 \times 10^{-3} = 6.667 \times 10^{-4}.$$

$$b_\varnothing = b_{1,\varnothing} b_{2,\varnothing} = 0.01 \times 4.44 \times 10^{-5} = 4.44 \times 10^{-7}.$$

$$b_o = b_{1,\varnothing} b_{2,o} = 0.01 \times 6.667 \times 10^{-3} = 6.667 \times 10^{-5}.$$

$$b_\imath = b_{1,\imath} b_{2,\varnothing} = 0.1 \times 4.44 \times 10^{-5} = 4.44 \times 10^{-6}.$$

$$b_{\imath o} = b_{1,\imath} b_{2,o} = 0.1 \times 6.667 \times 10^{-3} = 6.667 \times 10^{-4}.$$

EXAMPLE 4. *In this example we provide a complete walk-through for a larger query plan. The input is the query plan in Figure 4.a that contains 3 sampling operators, 3 joins and refers to relations* `lineitem, orders, customers` *and* `part`. *To analyze such a query, the first step is to re-write the sampling operators as GUS quasi-operators $\mathcal{G}_{(a_1,\bar{b}_1)}$, $\mathcal{G}_{(a_2,\bar{b}_2)}$, $\mathcal{G}_{(a_3,\bar{b}_3)}$ as in Figure 4.b. The second step, shown in Figure 4.c is to apply Proposition 6 to commute $\mathcal{G}_{(a_1,\bar{b}_1)}$ and $\mathcal{G}_{(a_2,\bar{b}_2)}$ with the join resulting in $\mathcal{G}_{(a_{12},\bar{b}_{12})}$. This step also shows the application of Proposition 4 above* `customers`. *The next step in Figure 4.d again uses Proposition 6 to commute $\mathcal{G}_{(a_{12},\bar{b}_{12})}$ and $\mathcal{G}_{(1,\bar{1})}$resulting in $\mathcal{G}_{(a_{121},\bar{b}_{121})}$. Figure 4.e shows the final transformation that uses the same proposition to get an overall GUS method $\mathcal{G}_{(a_{123},\bar{b}_{123})}$ just below the aggregate and on the top of the rest of the plan. Theorem 1 can now be used to obtain expected value and variance of the estimate. Using this and either the normal approximation or the Chebychev bounds, we obtain confidence intervals for the estimate. The computed coefficients for the GUS methods involved are depicted in Figure 4.*

5. PROPERTIES OF GUS OPERATORS

In the previous section we explored the interaction between GUS operators and relational algebra operators. In this section, we investigate interactions between GUS operators when applied to the same data. Intuitively, this will open up avenues for design of sampling operators, since it will indicate how to compute GUS quasi-operators that correspond to complex sampling schemes.

PROPOSITION 7 (GUS UNION). *For any expression R and GUS methods $\mathcal{G}_{(a_1,\bar{b}_1)}$, $\mathcal{G}_{(a_2,\bar{b}_2)}$,*

$$\mathcal{G}_{(a_1,\bar{b}_1)}(R) \cup \mathcal{G}_{(a_2,\bar{b}_2)}(R) \overset{SOA}{\Longleftrightarrow} \mathcal{G}_{(a,\bar{b})}(R)$$
$$\textit{where}, \qquad a = a_1 + a_2 - a_1a_2$$
$$b_T = 2a - 1 + (1 - 2a_1 + b_{1T})(1 - 2a_2 + b_{2T})$$

Union of GUS methods can be very useful when samples are expensive to acquire, thus there is value in reusing them. If two separate samples from relation R are available, Proposition 7 provides a way to combine them.

PROPOSITION 8 (GUS COMPACTION). *For any expression R, and GUS methods $\mathcal{G}_{(a_1,\bar{b}_1)}, \mathcal{G}_{(a_2,\bar{b}_2)}$,*

$$\mathcal{G}_{(a_1,\bar{b}_1)}\left(\mathcal{G}_{(a_2,\bar{b}_2)}(R)\right) \overset{SOA}{\Longleftrightarrow} \mathcal{G}_{(a,\bar{b})}(R),$$
$$\textit{where}, \qquad a = a_1a_2, \qquad b_T = b_{1,T_1}b_{2,T_2}$$

Compaction can be also viewed as intersection. It allows sampling methods to be *stacked* on top of each other to obtain smaller samples. We will make use of this in the next section.

Interestingly, union behaves like + with the null element $\mathcal{G}_{(0,\bar{0})}$ (the sampling method that blocks everything), the compaction/intersection behaves like $*$ with the null element $\mathcal{G}_{(1,\bar{1})}$(the sampling method that allows everything). Overall, the algebraic structure formed is that of a*semi-ring*, as stated in the following.

THEOREM 2. *The GUS operators over any expression R, form a* semiring *structure with respect to the union and compaction operations with $\mathcal{G}_{(0,\bar{0})}$ and $\mathcal{G}_{(1,\bar{1})}$ as the null elements, respectively.*

The semi-ring structure of GUS methods can be exploited to design sampling operators from ingredients.

PROPOSITION 9 (GUS COMPOSITION). *For any expressions R_1, R_2 and $\mathcal{G}_{(a_1,\bar{b}_1)}$, $\mathcal{G}_{(a_2,\bar{b}_2)}$,*

$$\mathcal{G}_{(a_1,\bar{b}_1)}(R_1) \circ \mathcal{G}_{(a_2,\bar{b}_2)}(R_2) \overset{SOA}{\Longleftrightarrow} \mathcal{G}_{(a,\bar{b})}(R)$$
$$a{=}a_1a_2, \qquad b_T = b_{1,T}b_{2,T}.$$

GUS composition is very useful for design of multi-dimensional sampling operators. We use it here to design a bi-dimensional Bernoulli.

EXAMPLE 5. *Suppose that we designed a bi dimensional sampling operator $B_{0.2,0.3}$(`l,o`) that combines Bernoulli sampling operators $B_{0.2}$(`l`) and $B_{0.3}$(`o`). Using the above result, the GUS operator $\mathcal{G}_{(a,\bar{b})}$ corresponding to the bi dimensional Bernoulli is $\mathcal{G}_{(a_1,\bar{b}_1)}$(`l`) $\circ\, \mathcal{G}_{(a_2,\bar{b}_2)}$, where $\mathcal{G}_{(a_1,\bar{b}_1)}$ is the GUS of $B_{0.2}$(`l`) and $\mathcal{G}_{(a_2,\bar{b}_2)}$ is the GUS of $B_{0.3}$(`o`). Working out the coefficients using Proposition 9 - the process is similar to the process in Example 3 we get: $a_3 = 0.06$, $b_{3,\varnothing} = 0.0036$, $b_{3,o} = 0.012$, $b_{3,l} = 0.018$, $b_{3,lo} = 0.06$.*

6. EFFICIENT VARIANCE ESTIMATION

In this section we present the major challenges in implementing an efficient variance estimator and demonstrate how the theoretical ideas in the previous section can be used to tackle those challenges. Due to space constraints we are unable to include details on the implementation. These can found in the extended version of the paper [2].

6.1 Estimating y_S Terms

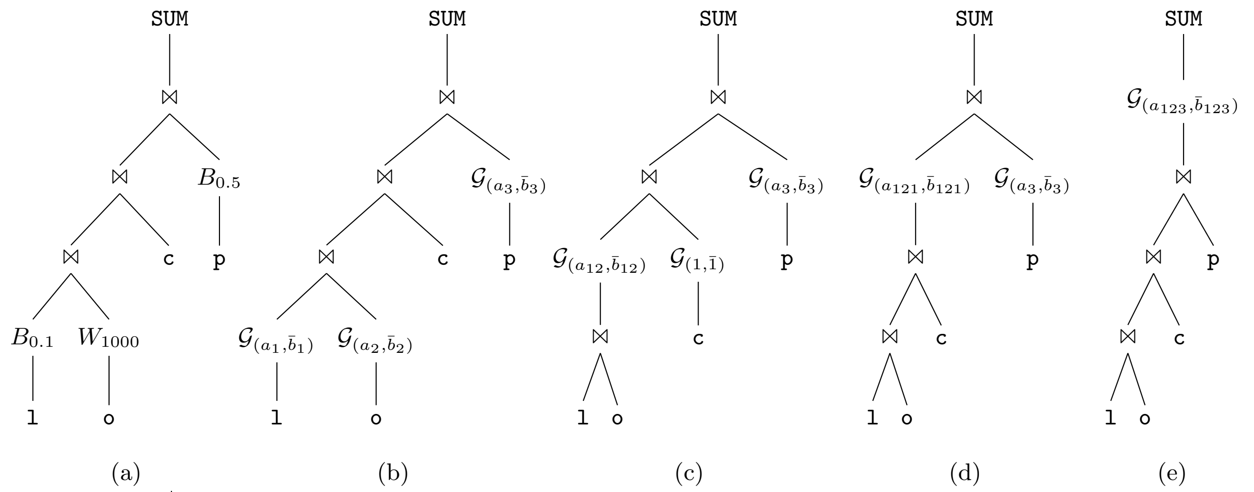
The computation of the variance of the sampling estimator in Theorem 1 uses the coefficients y_S defined as:

$$y_S = \sum_{t_i \in R_i | i \in S} \left(\sum_{t_j \in R_j | j \in S^C} f(t_i, t_j) \right)^2.$$

The terms y_S essentially require a *group by lineage* followed by a specific computation. This is better understood through an example – Query 1 – and equivalent expressions in SQL:

```
CREATE TABLE unagg AS
SELECT  l_orderkey*10+l_linenumber as f_l,
   o_orderkey as f_o, l_discount*(1.0-l_tax) as f
FROM lineitem TABLESAMPLE (10 PERCENT),
   orders TABLESAMPLE(1000 ROWS)
WHERE l_orderkey  = o_orderkey AND
   l_extendedprice > 100.0;

SELECT sum(f)^2 as y_empty FROM unagg;

SELECT sum(F*F) as y_l FROM ( SELECT sum(f) as F
       FROM unagg GROUP BY f_l);

SELECT sum(F*F) as y_o FROM ( SELECT sum(f) as F
       FROM unagg GROUP BY f_o);

SELECT sum(f*f) as y_lo FROM unagg;
```

The computation of the y_S terms using the above code is harder than the evaluation of the exact query, thus resulting in an impractical solution. We can use the sample



SUM
$B_{0.5}$
$B_{0.1}$
W_{1000}
$\mathcal{G}_{(a_1,\bar{b}_1)}$
$\mathcal{G}_{(a_2,\bar{b}_2)}$
$\mathcal{G}_{(a_3,\bar{b}_3)}$
$\mathcal{G}_{(a_{12},\bar{b}_{12})}$
$\mathcal{G}_{(1,\bar{1})}$
$\mathcal{G}_{(a_{121},\bar{b}_{121})}$
$\mathcal{G}_{(a_{123},\bar{b}_{123})}$
l
o
c
p
(a)
(b)
(c)
(d)
(e)

GUS method	Parameters
$\mathcal{G}_{(a_1,\bar{b}_1)}$	$a_1 = 0.1, b_{1,\varnothing} = 0.01, b_{1,\texttt{l}} = 0.1$
$\mathcal{G}_{(a_2,\bar{b}_2)}$	$a_2 = 6.667 \times 10^{-3}, b_{2,\varnothing} = 4.44 \times 10^{-5}, b_{2,\texttt{o}} = 6.667 \times 10^{-3}$
$\mathcal{G}_{(a_3,\bar{b}_3)}$	$a_3 = 0.5, b_{3,\varnothing} = 0.25, b_{3,\texttt{p}} = 0.5$
$\mathcal{G}_{(a_{12},\bar{b}_{12})}$	$a_{12} = 6.667 \times 10^{-4}, b_{12,\varnothing} = 4.44 \times 10^{-7}, b_{12,\texttt{o}} = 6.667 \times 10^{-5}, b_{12,\texttt{l}} = 4.44 \times 10^{-6}, b_{12,\texttt{lo}} = 6.667 \times 10^{-4}$
$\mathcal{G}_{(a_{121},\bar{b}_{121})}$	$a_{121} = 6.667 \times 10^{-4}, b_{121,\varnothing} = 4.44 \times 10^{-7}, b_{121,\texttt{c}} = 4.44 \times 10^{-7}, b_{121,\texttt{o}} = 6.667 \times 10^{-5}, b_{121,\texttt{oc}} = 6.667 \times 10^{-5}, b_{121,\texttt{l}} = 4.44 \times 10^{-6}, b_{121,\texttt{lc}} = 4.44 \times 10^{-6}, b_{121,\texttt{lo}} = 6.667 \times 10^{-4}, b_{121,\texttt{loc}} = 6.667 \times 10^{-4}$
$\mathcal{G}_{(a_{123},\bar{b}_{123})}$	$a_{123} = 3.334 \times 10^{-4}, b_{123,\varnothing} = 1.11 \times 10^{-7}, b_{123,\texttt{p}} = 2.22 \times 10^{-7}, b_{123,\texttt{c}} = 1.11 \times 10^{-7}, b_{123,\texttt{cp}} = 2.22 \times 10^{-7}, b_{123,\texttt{o}} = 1.667 \times 10^{-5}, b_{123,\texttt{op}} = 3.335 \times 10^{-5}, b_{123,\texttt{oc}} = 1.667 \times 10^{-5}, b_{123,\texttt{ocp}} = 3.335 \times 10^{-5}, b_{123,\texttt{l}} = 1.11 \times 10^{-6}, b_{123,\texttt{lp}} = 2.22 \times 10^{-6}, b_{123,\texttt{lc}} = 1.11 \times 10^{-6}, b_{123,\texttt{lcp}} = 2.22 \times 10^{-6}, b_{123,\texttt{lo}} = 1.667 \times 10^{-4}, b_{123,\texttt{lop}} = 3.334 \times 10^{-4}, b_{123,\texttt{loc}} = 1.667 \times 10^{-4}, b_{123,\texttt{locp}} = 3.334 \times 10^{-4}$

Figure 4: Transformation of the query plan to allow analysis

to *estimate these terms* essentially replacing the entire base relations in the above queries by their samples. These estimates, Y_S can be used to obtain unbiased estimates $\hat{Y}_S$ of terms y_S using the formula [22]

$$\hat{Y}_S = \frac{1}{c_{S,\emptyset}} \left(Y_S - \sum_{T \subset S^C, T \neq \emptyset} c_{S,T} \hat{Y}_{S \cup T} \right)$$

where

$$c_{S,T} = \sum_{U \subset T} (-1)^{|U|+|S|} b_{S \cup U}.$$

Yet again, the major effort is in evaluating Y_S terms over the sample. The number of terms to be evaluated is 2^n, where n is the number of base relations and each term consists of a `GROUP BY` query that is possibly expensive. To this end, we observe that the computation of the variance of the sampling estimator depends, in orthogonal ways, on properties of the data through terms y_S and on properties of the sampling through c_S. The base theory does not require any particular way to compute/estimate terms y_S. Using the available sample for estimating y_S terms is only one of the possibilities. While many ways to estimate terms y_S can be explored, a particularly interesting one in this context is to use *another sampling method* for the purpose. More specifically, we could use a sub-sample of the available sample for estimation of the terms y_S and the full sample for the estimation of the true value.

To understand what benefits we can get from this idea, we observe that we do not need very precise estimates of the terms y_S. Should we make a mistake, it will only affect the confidence interval by a small constant factor but will still allow the shrinking of the confidence interval with the increase of the sample. Based on the experience in DBO and TurboDBO, using 100K result tuples for the estimation of y_S terms suffices. Moreover, only for these 100K samples the system needs to provide lineage information[2]; samples used for evaluation of the expected value need no lineage.

6.2 Sub-Sampling

There are two alternatives when it comes to reducing the number of samples used for estimation of terms y_S: select a more restrictive sampling method, or further sample from the provided sample. The latter approach can be applied when needed, in case the size of the sample is overwhelming for the computation of terms y_S. Specifically, we can use a multi-dimensional Bernoulli GUS on top of the existing query plan for result tuples. This can be obtained by applying Proposition 9 until the desired size is reached. The extra results in Section 5 together with the core results in Section 4 provide the means to analyze this modified sampling process. Example 6 and the accompanying Figure 5 provide such analysis for Query 1 and exemplifies how the extra Bernoulli sampling can be dealt with.

EXAMPLE 6. *This example shows how the query plan from the introduction can be sampled further to efficiently obtain y_S terms. Figure 5.a shows the original query plan. Figure 5.b shows the sampling in terms of a GUS quasi-operator. Figure 5.c shows the placement of a bi-dimensional*

[2]By lineage, we mean the list of base relations that have participated in forming a given tuple.

Bernoulli sampling method. Figures 5.d, 5.e, 5.f make use of propositions in Section 4 to obtain a SOA-equivalent plan, suitable for analysis.

7. EXPERIMENTS

We have two main goals for this empirical study. Our first goal is to provide some experimental confirmation of our theory. Our second goal is to evaluate the efficiency of the estimation process and study how sub-sampling affects the running times and the variance estimates. More specifically:

- How does the running time depend on the selectivity of the query?
- How useful is the sub-sampling process? How does the running time depend on the sub-sampling?
- Does sub-sampling significantly reduce the precision of the confidence interval estimates?

As we will see in this section, the experiments validate the theory and the sub-sampling is invaluable for efficient estimation. In particular, sub-samples of size 100K to 400K tuples provide correct and meaningful confidence intervals and require less than 2% to the overall running time.

7.1 Experimental Setup

Data Sets. For our experiments, we use two versions of TPC-H [8] data—a small sized data set (scale factor 0.1, 100MB) for verification of confidence intervals, and a large one (scale factor 1000, 1TB) to benchmark the efficiency of the for the estimation process. We use following parameterized query:

```
SELECT SUM(l_discount*(1.0-l_tax))
FROM lineitem TABLESAMPLE (x PERCENT),
  orders TABLESAMPLE(y ROWS),
  part TABLESAMPLE(z PERCENT)
WHERE l_orderkey  = o_orderkey AND
   l_partkey = p_partkey AND o_totalprice < q AND
   p_retailprice < r;
```

where the parameters x, y and z govern the inclusiveness of the sampling methods and parameters q and r govern the selectivity.

Code. The analysis of the sampling query plan, as described in Section 4, is coded using SWI-Prolog [27]. When presented with the specific query plan that contains both the relational and sampling operators, the code derives the a, b_T coefficients. The sub-sampling operator and the confidence interval prediction based on the coefficients a, b_T from the symbolic analysis is implemented in C++. For the sub-sampling operator, we can specify a lower (L) and an upper (U) bound on the number of tuples to maintain. The implementation uses an adaptive algorithm that changes the probability of the Bernoulli sampling to meet these goals. In all the experiments we specify the range used to configure the sub-sampling – the actual number of tuples in the sub-sample is specified in some of the experiments.

DBMSs. Two DBMSs were used for experiments in this section. For the correctness experiments, we used Oracle 11g and made use of the `SAMPLE` operator. For the large scale experiments we used the DataPath[6] parallel database system and coded the sub-sampling and analysis as user-defined aggregates.

Hardware. We run our experiments on a 48 core AMD Magny-cours 1.9GHz with 256GB of RAM and 76 disks. The machine is capable of sustained I/O rates of 3.0GB/s through 3 Averatec RAID controllers. Oracle 11g used only one disk and did not make use of all the cores (Oracle was used only for the correctness experiments). DataPath usually makes use of all the resources and for most queries sustained data processing speeds of 2-3GB/s.

We now describe the experiments undertaken to answer the questions listed above. The detailed setup of each experiment is specified in the respective subsections.

7.2 Correctness

In this experiment, we provide a sanity check for our theory and measure how accurate the derived confidence intervals are.

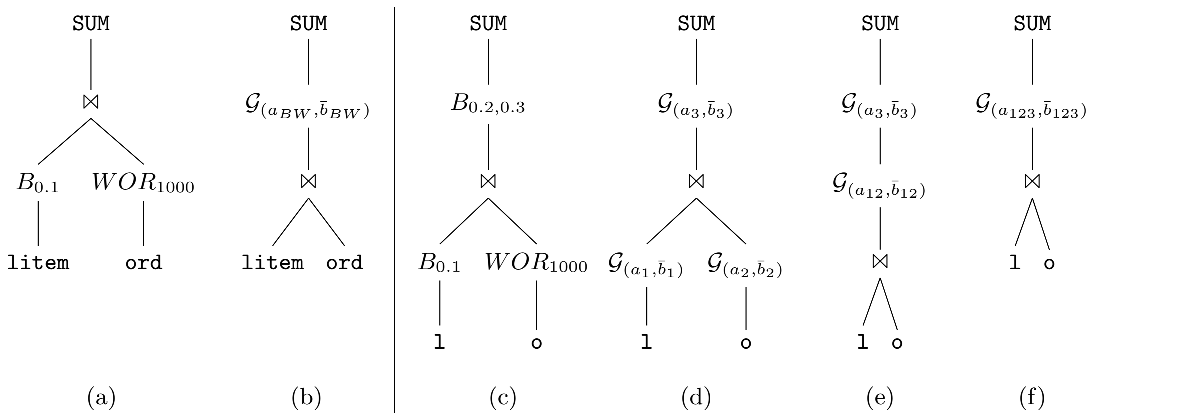
Setup. For this study, we use the 100MB TPC-H data (`lineitem` has 600,000 tuples and `orders` has 150,000 tuples). We selected the small size to allow us to perform many experiments to empirically evaluate the variance through Monte-Carlo simulation. We used the Oracle 11g database since it supports the `SAMPLE` operator from SQL and thus would support our claim that the theory in this paper can readily be used to predict sampling queries from existing commercial DBMSs.

Monte Carlo Validation. We validate the theoretically computed expected value and variance of the sampling estimate using a Monte Carlo simulation. We use the query as shown above with parameters `q` and `r` fixed at 10^6 and 4×10^5 respectively. This query is run for different values of (`x`,`y`,`z`) as shown in Figure 6. For each value of (`x`,`y`,`z`), we run the query 5000 times and obtain 5000 confidence intervals each for ten different confidence levels, ranging from 5% to 95%. We then compute the achieved confidence levels, i.e. for each of the ten desired confidence levels, we compute the percentage of times the true value falls within the corresponding confidence intervals. In Figure 6, we show a comparison between the desired and achieved confidence levels for 4 different sampling strategies. The achieved confidence levels are very close to the desired values, across the different sampling strategies and confidence levels. This provides strong empirical evidence that the confidence intervals obtained by using the theory in Section 4 are accurate and tight.

7.3 Running Time

The next goal is to evaluate the efficiency of the estimation process. We are especially interested in evaluating the variance of the estimators. This study performed with our research prototype should give the practitioner some idea on the overhead expected.

Setup. Intuitively, the analysis overhead will depend on the sample size. To ensure that we stress the analysis with large samples, we use the 1TB TPC-H instance and *treat the database as a sample of a 1PB database*. More specifically we assume that the 6 billion tuples in `lineitem` are a Bernoulli sample from the 6 trillion tuples in the same relation at 1PB scale (0.001 sampling fraction). Similarly, the


SUM
⋈
$B_{0.1}$
WOR_{1000}
litem
ord
(a)
SUM
$\mathcal{G}_{(a_{BW},\bar{b}_{BW})}$
⋈
litem
ord
(b)
SUM
$B_{0.2,0.3}$
⋈
$B_{0.1}$
WOR_{1000}
l
o
(c)
SUM
$\mathcal{G}_{(a_3,\bar{b}_3)}$
⋈
$\mathcal{G}_{(a_1,\bar{b}_1)}$
$\mathcal{G}_{(a_2,\bar{b}_2)}$
l
o
(d)
SUM
$\mathcal{G}_{(a_3,\bar{b}_3)}$
$\mathcal{G}_{(a_{12},\bar{b}_{12})}$
⋈
l
o
(e)
SUM
$\mathcal{G}_{(a_{123},\bar{b}_{123})}$
⋈
l
o
(f)

GUS method	Parameters
$\mathcal{G}_{(a_1,\bar{b}_1)}$	$a_1 = 0.1, b_{1,\varnothing} = 0.01, b_{1,\texttt{l}} = 0.1$
$\mathcal{G}_{(a_2,\bar{b}_2)}$	$a_2 = 6.667 \times 10^{-3}, b_{2,\varnothing} = 4.44 \times 10^{-5}, b_{2,\texttt{o}} = 6.667 \times 10^{-3}$
$\mathcal{G}_{(a_3,\bar{b}_3)}$	$a_3 = 0.06, b_{3,\varnothing} = 0.0036, b_{3,\texttt{o}} = 0.012, b_{3,\texttt{l}} = 0.018, b_{3,\texttt{lo}} = 0.06$
$\mathcal{G}_{(a_{12},\bar{b}_{12})}$	$a_{12} = 6.667 \times 10^{-4}, b_{12,\varnothing} = 4.44 \times 10^{-7}, b_{12,\texttt{o}} = 6.667 \times 10^{-5}, b_{12,\texttt{l}} = 4.44 \times 10^{-6}, b_{12,\texttt{lo}} = 6.667 \times 10^{-4}$
$\mathcal{G}_{(a_{123},\bar{b}_{123})}$	$a_{123} = 4 \times 10^{-5}, b_{123,\varnothing} = 1.598 \times 10^{-9}, b_{123,\texttt{o}} = 8 \times 10^{-7}, b_{123,\texttt{l}} = 7.992 \times 10^{-8}, b_{123,\texttt{lo}} = 4 \times 10^{-5}$

Figure 5: Transformation of the query plan to allow analysis

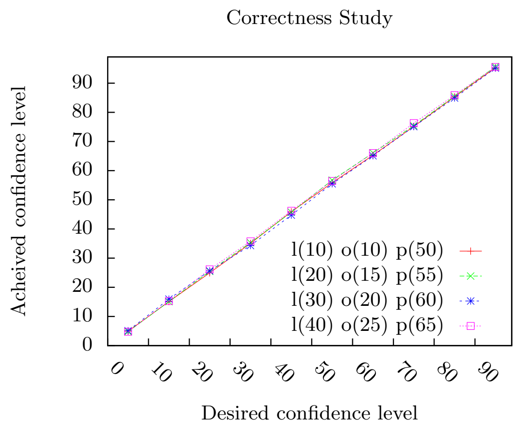


Figure 6: Plot of percentage of times the true value lies in the estimated confidence intervals vs desired confidence level.

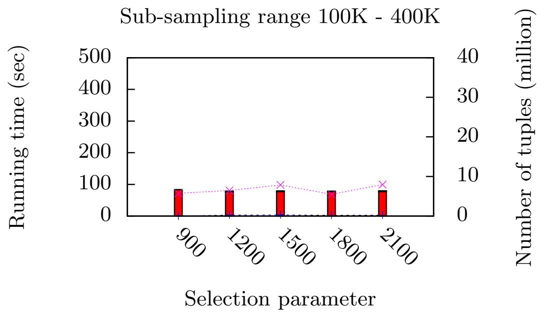
1.5 billion tuples in `orders` are a sample without replacement from the 1.5 trillion tuples of the 1PB database and the 200 million tuples in `part` are a Bernoulli sample (0.001 sampling fraction) from 200 billion tuples at 1PB scale. This ensures that the sample sizes the analysis has to deal with can be in the billions – a very harsh scenario for analysis indeed.

Since the database is the sample, there is no sampling needed in the execution of the query – the tuples that the analysis has to make use of are the tuples that are aggregated by the non-sampling version of the query. As described in Section 6, maintaining the estimator is as easy as performing the aggregation in the non-sampling query but computing the variance is much more involved. The technique we proposed is to sub-sample from the sample to limit the number of tuples used to estimate the variance. In our experiments we studied the impact of the query characteristics (various selection predicates) and sub-sampling size on the running time. For each experiment, we measured three running times. First, the running time of the non-sampling query (no statistical estimation, just the aggregate). Second, the running time of the query processing and sub-sampling process. Sub-sampling is interleaved with the rest of the processing and the two running times cannot be separated. Third, the time to perform the analysis. The analysis is single-threaded and starts only once the sub-sample is completely formed.

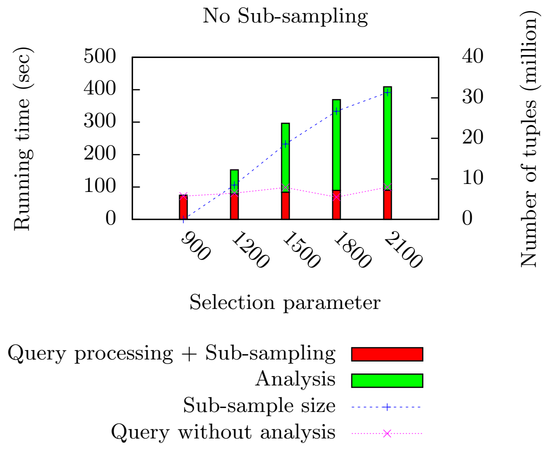
Impact of Selectivity with Fixed Sub-Sampling. In our first experiment, we will vary the selection predicate (thus indirectly the selectivity of the query) and set the range for the sub-sampling at 100K-400K tuples (i.e. sub-sampling obtains 100K-400K tuples that are a Bernoulli sample from the data provided for analysis). Results are depicted in Figure 7(a). We make three key observations; First, for this sub-sampling target, the analysis adds an insignificant amount of extra effort (about 2% of the overall running time). Second, selectivity of the query has no significant effect on the running time for either the non-sampling or for the sampling query. Last, the running time of the sampling version of the query seems to be more stable than the running time of the non-sampling version.[3] It seems that, when the size of sub-sample is below 500,000, the extra effort to perform sampling analysis is insignificant. We show later that such sub-samples are good enough to produce stable variance estimates.

Impact of Selectivity with No Sub-Sampling. An unresolved question from the previous experiment is what happens when no sub-sampling is performed, i.e. all the data is used for analysis. The selectivity of the query will now control the number of tuples used for analysis and give an

[3]A similar behavior was noticed in [6]: the execution is more stable and somewhat faster at higher CPU loads.



(a)



(b)

Figure 7: Plots of running time vs selection parameter with and without sub-sampling.

indication of the effort as a function of the size. Figure 7(b) depicts result of such an experiment in which the selection predicate was varied. Results reveal that, once the size of the sample exceeds 1M the analysis cost becomes unmanageable and starts to dominate. At the end of the spectrum (31M tuples) the analysis was 5 times slower than the the rest of the execution – this is clearly not acceptable in practice. As we hinted above, targets of 100K-400K produce good enough estimates of the variance; there is no need to base the variance analysis on millions of tuples, thus running time of analysis can be kept under control. Sub-sampling is thus a crucial technique for applicability of sampling estimation to large data.

7.4 Sub-Sample Size

This experiment sheds light on influence of sub-sampling sizes on the estimate for the variance and thus the quality of the confidence intervals.

Setup. Since we would like to get samples from all over the data source, we use the 1TB TPC-H instance as the *data source* and repeatedly derive samples from it. Remember that, according to Section 6, any estimates of the terms y_S can be used to analyze any of the sampling methods. Sub-sampling is used to estimate the terms y_S, but the entire sample is used to compute the estimate. The user expects a confidence interval of the estimate that is computed over the entire sample. An overly accurate estimate of variance does not help for any purpose. If the y_S estimates are based off a 10,000 times smaller sub-sample, it will produce only a 10% error in the confidence interval. Thus a much smaller sub-sample has only a secondary influence on the confidence interval. The plot in Figure 8 shows this fact. In this experiment we run around 250 instances of Query 1 for sub-sampling ranges of 10K-40K, 100K-400K and 1M-4M tuples each. In all cases, we calculate the fluctuation of the resultant confidence interval widths with respect to the confidence interval width obtained from an analysis without sub-sampling. In particular, we define error by ratio of the difference between the 5th and the 95th percentile values to the width of the confidence interval obtained without sub-sampling. The plot in Figure 8 shows that this error is only 1% when 100K-400K tuples are used.

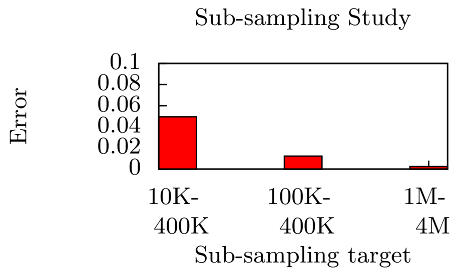


Figure 8: Plot of fluctuation of confidence interval widths obtained by analysis with sub-sampling, wrt the true confidence interval

Note on number of relations As we have seen in this section, for sampling over 3 relations, good confidence intervals can be obtained with a mere 2% extra effort since sub-samples of 100K tuples suffice. Since the analysis requires the computation of 2^n terms if n relations are sampled, the influence of the number of relations on the running time of the analysis is of concern. In practice, these concerns can be easily addressed as follows: (a) the computation of the y_S terms from the sub-samples can be parallelized – on our system this would result in a speedup of at least 32 (on 48 cores), (b) we noticed that foreign key joins result in repeated values for certain terms – about half the values are repeated, (c) we see no need to sample from more than 8 relations since there is no need to sample from small or medium size relations. Notice that the parallelization alone would allow us to scale from 3 to $3 + 5 = 8$ relations since $2^5 = 32$, the expected speedup.

8. CONCLUSIONS AND FUTURE WORK

While technically challenging to create, the theory in this paper is in essence easy to use. Sampling is treated as a quasi-operator. In order to incorporate sampling based approximations, such operators are introduced in the query plans and the mechanisms described in Section 4 are used to analyze the estimators. We have already seen an example of use of the theory: the sub-sampling technique in Section 6. With very little effort (introducing a final Bernoulli sampling quasi operator), we dealt with a seemingly hard problem: how to use a subsample to predict the behavior of the main sample. The straightforwardness of this process encourages us to suggest that the theory presented here will allow significant progress in a number of hard to solve problems explored in the approximate query processing literature. We briefly mention such potential in the remaining of this section.

Database as a sample. By viewing the database itself as a sample, *robustness analysis* is possible. In particular, if we assume that 1% of the tuples are mistakenly lost and we wish to predict the impact on the query results we can view the database as a 99% Bernoulli sample. A large variance will indicate that the query results are sensitive to such perturbations and thus not robust.

Choosing sampling parameters. By using the unbiased y_S estimates from a single sampling instance, the theory allows for plugging in coefficients for different sampling strategies to predict the respective variances. This can give the user insight on comparing different sampling strategies and parameters to suit his/her needs.

Estimating the size of intermediate relations. Query execution engines maintain a sample of the data and evaluate aggregates on it to predict the size of the intermediate relations. Our theory allows for the evaluation of the precision of these, thereby preventing the selection of *inferior* plans.

Data Streaming and Load Shedding. An interesting problem in load shedding is determining a sampling rate so that the system can keep up with fast-rate incoming data while minimizing the error[7]. While such analysis was done for single relations, our theory provides for similar analysis with multiple relations.

Acknowledgements. Material in this paper was supported by the National Science Foundation under Grant DIIS-1007062.

9. REFERENCES

[1] Sql-2003 standard, 2003.

[2] S. N. A Dobra, C Jermaine. A sampling algebra for aggregate estimation. Technical Report 749260, Dept of CISE, UF, 2013.

[3] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *In Proc. of 25th Intl. Conf. on Very Large Data Bases*, pages 754–755, 1999.

[4] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD '99, pages 574–576, New York, NY, USA, 1999. ACM.

[5] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *In SIGMOD*, pages 275–286, 1999.

[6] S. Arumugam, A. Dobra, C. M. Jermaine, N. Pansare, and L. L. Perez. The datapath system: a data-centric analytic processing engine for large data warehouses. In *SIGMOD Conference*, pages 519–530, 2010.

[7] B. Babcock, M. Datar, and R. Motwani. Load shedding in data stream systems. In *Data Streams - Models and Algorithms*, pages 127–147. 2007.

[8] T.-H. Benchmark. http://www.tpc.org/tpch/.

[9] S. Chaudhuri and R. Motwani. On sampling and relational operators. *IEEE Data Eng. Bull.*, 22(4):41–46, 1999.

[10] S. Chaudhuri, R. Motwani, and V. R. Narasayya. On random sampling over joins. In *SIGMOD Conference*, pages 263–274, 1999.

[11] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2007.

[12] A. Dobra, C. Jermaine, F. Rusu, and F. Xu. Turbo-charging estimate convergence in dbo. *PVLDB*, 2(1):419–430, 2009.

[13] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, S. Ramaswamy, and T. Suel. Aqua: System and techniques for approximate query answering. Technical report, 1998.

[14] J. Gryz, J. Guo, L. Liu, and C. Zuzarte. Query sampling in db2 universal database. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 839–843, New York, NY, USA, 2004. ACM.

[15] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *SSDBM*, pages 51–63. IEEE Computer Society Press, 1996.

[16] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. *SIGMOD Rec.*, 28:287–298, June 1999.

[17] P. J. Haas and J. M. Hellerstein. Online query processing. In *SIGMOD Conference*, page 623, 2001.

[18] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Selectivity and cost estimation for joins based on random sampling. *Journal of Computer and System Sciences*, 52(3):550 – 569, 1996.

[19] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.*, 52:550–569, June 1996.

[20] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *SIGMOD Rec.*, 26:171–182, June 1997.

[21] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. *ACM Trans. Database Syst.*, 33(4), 2008.

[22] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. *ACM Trans. Database Syst.*, 33(4), 2008.

[23] C. Jermaine, A. Dobra, S. Arumugam, S. Joshi, and A. Pol. The sort-merge-shrink join. *ACM Trans. Database Syst.*, 31(4):1382–1416, 2006.

[24] R. J. Lipton, J. F. Naughton, D. A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116(1):195 – 226, 1993.

[25] F. Olken. Random sampling from databases, 1993.

[26] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 256–276, New York, NY, USA.

[27] SWI-Prolog. http://www.swi-prolog.org.