The Analytical Bootstrap: a New Method for Fast Error Estimation in Approximate Query Processing

Kai Zeng[†] Shi Gao[†] Barzan Mozafari[‡] Carlo Zaniolo[†]
[†]University of California, Los Angeles [‡]University of Michigan, Ann Arbor
[†]{kzeng, gaoshi, zaniolo}@cs.ucla.edu [‡]mozafari@umich.edu

ABSTRACT

Sampling is one of the most commonly used techniques in Approximate Query Processing (AQP)—an area of research that is now made more critical by the need for timely and cost-effective analytics over "Big Data". Assessing the quality (i.e., estimating the error) of approximate answers is essential for meaningful AQP, and the two main approaches used in the past to address this problem are based on either (i) analytic error quantification or (ii) the bootstrap method. The first approach is extremely efficient but lacks generality, whereas the second is quite general but suffers from its high computational overhead. In this paper, we introduce a probabilistic relational model for the bootstrap process, along with rigorous semantics and a unified error model, which bridges the gap between these two traditional approaches. Based on our probabilistic framework, we develop efficient algorithms to predict the distribution of the approximation results. These enable the computation of any bootstrap-based quality measure for a large class of SQL queries via a single-round evaluation of a slightly modified query. Extensive experiments on both synthetic and real-world datasets show that our method has superior prediction accuracy for bootstrap-based quality measures, and is several orders of magnitude faster than bootstrap.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query processing

Keywords

Approximate Query Processing; Error Estimation; Bootstrap

1. INTRODUCTION

Data-driven activities in business, science, and engineering are rapidly growing in terms of both data size and significance. This situation has brought even more attention to the already-active area of Approximate Query Processing (AQP), and in particular to sampling approaches as a critical and general technique for coping with the ever-growing size of big data. Sampling techniques are widely used in databases [7, 11, 16, 22, 24, 32], stream processors [12, 31], and even Map-Reduce systems [9, 30].

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2588579.

This most commonly used technique consists in evaluating the queries on a small random *sample* of the original database. Of course, the approximate query answers obtained this way are of very limited utility unless they are accompanied by some *accuracy guarantees*. For instance, in estimating income from a small sample of the population, a statistician seeks assurance that the derived answer falls within a certain interval of the exact answer computed on the whole population with high confidence (e.g., within $\pm 1\%$ of the correct answer with probability $\geq 95\%$). This enables the users to decide whether the current approximation is "good enough" for their purpose. Thus, assessing the quality (i.e., error estimation) of approximate answers is a fundamental aspect of AQP.

The extensive work on error estimation in the past two decades can be categorized into two main approaches. The first approach [9, 15, 16, 22, 23, 24, 25, 33, 44] analytically derives closed-form error estimates for common aggregate functions in a database, such as `SUM`, `AVG`, etc. Although computationally appealing, analytic error quantification is restricted to a very limited set of queries. Thus, for every new type of queries, a new closed-form formula must be derived. This derivation is a manual process that is ad-hoc and often impractical for complex queries [34].[1]

To address this problem, a second approach, called *bootstrap*, has emerged as a more general method for routine estimation of errors [27, 30, 34]. Bootstrap [19] is essentially a Monte Carlo procedure, which for a given initial sample, (i) repeatedly forms simulated datasets by resampling tuples i.i.d. (identically and independently) from the given sample, (ii) recomputes the query on each of the simulated datasets, and (iii) assesses the answer quality on the basis of the empirical distribution of the produced query answers. The wide applicability and automaticity of bootstrap is confirmed both in theory [13, 41] and practice [27, 30, 34]. Unfortunately, bootstrap tends to suffer from its high computational overhead, since hundreds or even thousands of bootstrap trials are typically needed to obtain reliable estimates [27, 34].

In this paper, we introduce a new technique, called the *Analytical Bootstrap Method* (ABM), which is both computationally efficient and automatically applicable to a large class of SQL queries, and thus combines the benefits of these two approaches. Thus, this paper's main contribution is *a probabilistic relational model for the bootstrap process that allows for automatic error quantification of a large class of SQL queries (defined in Section 2.2) under sampling, but without performing the actual Monte Carlo simulation.* We show that our error estimates are provably equivalent to those produced by the simulation-based bootstrap (Theorems 1 and 2).

[1]This is evidenced by the difficulties faced by previous analytic approaches in supporting approximation for queries that are more complex than simple group-by aggregate queries.

The basic idea of ABM is to annotate each tuple of the sampled database with an integer-valued random variable that represents the possible multiplicities with which this tuple would appear in the simulated datasets generated by bootstrap. This small annotated database is called a *probabilistic multiset database* (PMDB). This PMDB succinctly models all possible simulated datasets that could be generated by bootstrap trials. Then, we extend relational operators to manipulate these random variables. Thus executing the query on the PMDB generates an annotated relation which encodes the distribution of all possible answers that would be produced *if* we actually performed bootstrap on the sampled database. In particular, using a single-round query evaluation, ABM accurately estimates the empirical distribution of the query answers that would be produced by hundreds or thousands of bootstrap trials.[2]

We have evaluated ABM through extensive experiments on the TPC-H benchmark and on the actual queries and datasets used by leading customers of Vertica Inc. [5]. Our results show that ABM is an accurate prediction of the simulation-based bootstrap. Additionally, it is 3–4 orders of magnitude faster than the state-of-the-art parallel implementations of bootstrap [28].

Therefore, ABM promises to be a technique of considerable practical significance: The immediate implication of this new technique is that, the quality assessment module of any AQP system that currently relies on bootstrap (e.g., [27, 30, 34]) can now be replaced by a process that uses 3–4 orders of magnitude fewer resources (resp. lower latency) if it currently uses a parallel (resp. sequential) implementation of bootstrap. We envision that by removing bootstrap's computational overhead, ABM would also significantly broaden the application of AQP to areas which require *interactive* and *complex* analytics expressible in SQL, such as root cause analysis and A/B testing [9], real-time data mining, and exploratory data analytics.

The paper is organized as follows. Section 2 exemplifies bootstrap and the query evaluation problem considered in this paper. Section 3 provides the necessary theoretical background. In Section 4, we present our probabilistic multiset relational model. We explain our efficient query evaluation technique in Sections 5 and 6. In Section 7, we discuss several extensions of our technique. We report the experimental study in Section 8, followed by the related work in Section 9. We conclude in Section 10.

2. PROBLEM STATEMENT

In this section, we formally state the problem addressed by this paper. We then provide a small example of bootstrap in Section 2.1, and a high-level overview of our approach in Section 2.2.

Given a database $\mathfrak{D}$ and a query $\mathfrak{q}$, let $\mathfrak{q}(\mathfrak{D})$ denote the exact answer of evaluating $\mathfrak{q}$ on $\mathfrak{D}$. An approximate answer can be obtained by (i) extracting from $\mathfrak{D}$ a random sample D, (ii) evaluating a potentially modified version of $\mathfrak{q}$ (say q) on D, and (iii) using $q(D)$ as an approximation of $\mathfrak{q}(\mathfrak{D})$. In some cases, such as AVG, q is the same as $\mathfrak{q}$, but in other cases, q can be a modified query that produces *better* results. For instance, when evaluating SUM on a sample of size $\frac{1}{f}|\mathfrak{D}|$, one will choose $q = f\mathfrak{q}$ to scale up the sample sum by a factor of f. The particular selection of q for a given $\mathfrak{q}$ is outside the scope of this paper.[3] Instead, we focus on estimating the quality of $q(D)$ for a given q, as described next.

[2]Note that we do not claim to provide low approximation error for arbitrary queries. For instance, random sampling is known to be futile for certain queries (e.g., joins, MIN, MAX). However, we guarantee the same empirical distribution that would be produced by thousands of bootstrap trials (which is the state-of-the-art error quantification technique for AQP of complex SQL analytics), whether or not sampling leads to low-error or unbiased answers.

[3]Similar to the original bootstrap [27, 30, 34], we assume that q is given. Deriving q for a given $\mathfrak{q}$ has been discussed in [31].

Let $D_1, \cdots, D_N$ be all possible sample instantiations of $\mathfrak{D}$. Then, $q(D)$ could be any of the $\{q(D_i), i = 1, \cdots, N\}$ values. Therefore, to assess the quality of $q(D)$, one needs to (conceptually) consider all possible query answers $\{q(D_i)\}$, and then compute some user-specified measure of quality, denoted by $\xi(\mathfrak{q}(\mathfrak{D}), \{q(D_i)\})$. For instance, when approximating an AVG query, ξ could be the variance, or the 99% confidence interval for the $\{q(D_i)\}$ values. However, since computing $\xi(\mathfrak{q}(\mathfrak{D}), \{q(D_i)\})$ directly is typically infeasible, a technique called *bootstrap* is often used to approximate this value.

Bootstrap. Bootstrap [19] is a powerful technique for approximating unknown distributions. Bootstrap consists in a simple Monte Carlo procedure: it repeatedly carries out a sub-routine, called a *trial*. Each trial generates a simulated database, say $\hat{D}_j$, which is the same size as D (by sampling $|D|$ tuples i.i.d. from D *with replacement*), and then computes query q on $\hat{D}_j$. The collection $\{q(\hat{D}_j)\}$ from all the bootstrap trials forms an empirical distribution, based on which $\xi(q(D), \{q(\hat{D}_j)\})$ is computed and returned as an approximation of $\xi(\mathfrak{q}(\mathfrak{D}), \{q(D_i)\})$. Bootstrap is effective and robust across a wide range of practical situations [27, 30, 34].

Problem Statement. Our goal is to devise an efficient algorithm for computing the empirical distribution $\{q(\hat{D}_j)\}$ produced by bootstrap, but *without* executing the actual bootstrap trials. In particular, we are interested in the marginal distribution of each individual result tuple, i.e., the probability of each tuple appearing in any $q(\hat{D}_j)$.[4] This marginal distribution enables us to compute the commonly used quality measures ξ (e.g., mean, variance, standard deviation, and quantiles as used in [27, 30, 34]). Next, we demonstrate this in an example.

2.1 An Example of Bootstrap

Bootstrapping a Database. A single bootstrap trial on a relation R produces a multiset relation, since a tuple may be drawn more than once. Thus, different trials could result in different multiset relations. This set of multiset relations can be modeled as a single *probabilistic* multiset relation, where each tuple has a random multiplicity. Let R^r denote the probabilistic relation that results from bootstrapping R. Likewise, let D^r denote the probabilistic database obtained by bootstrapping the relations of a database sample D. For instance, consider D in Figure 1(a), which has a single relation R (named *stock*) containing three tuples. Figure 1(b) shows a possible instance $\hat{R}$ of R^r, produced by a single bootstrap trial, where tuple t_2 and t_3 are drawn twice and once, respectively, while t_1 is not selected. For brevity, we denote this resample as $\{(t_2, 2), (t_3, 1)\}$.

$R =$ *stock*

	Part	Type	Qty
t_1	p01	a	4
t_2	p02	b	5
t_3	p03	a	3

(a)

$\hat{R} =$ *stock*

	Part	Type	Qty
t_2	p02	b	5
t_2	p02	b	5
t_3	p03	a	3

(b)

Figure 1: (a) An example of a database sample D with one relation R (named *stock*), and (b) a resample instance of R^r

Bootstrapping this particular database sample D generates 10 possible multiset relations. We refer to these as the *possible multiset worlds* of D^r, denoted as $pmw(D^r)$. Figure 2 shows all the ten possible instances with their probabilities of being generated.

Queries on the Resampled Database. Consider the "Important Stock Types" query q in Example 1, which finds the stock types having a quantity $> 30\%$ of the total quantity. To answer q on D^r,

[4]The set of all possible query answers, $\{q(\hat{D}_j)\}$, may have up to $O(2^{|D|})$ elements. Thus, due to its extremely large size, returning this set to the user is impractical.

multiset world	prob.
$D_1 = \{(t_1, 1), (t_2, 1), (t_3, 1)\}$	2/9
$D_2 = \{(t_1, 2), (t_2, 1)\}$	1/9
$D_3 = \{(t_1, 2), (t_3, 1)\}$	1/9
$D_4 = \{(t_2, 2), (t_1, 1)\}$	1/9
$D_5 = \{(t_2, 2), (t_3, 1)\}$	1/9
$D_6 = \{(t_3, 2), (t_1, 1)\}$	1/9
$D_7 = \{(t_3, 2), (t_2, 1)\}$	1/9
$D_8 = \{(t_1, 3)\}$	1/27
$D_9 = \{(t_2, 3)\}$	1/27
$D_{10} = \{(t_3, 3)\}$	1/27

Figure 2: The possible multiset worlds of D^r

one needs to evaluate q against each world in $pmw(D^r)$, and add up the probabilities of all the worlds returning the same answer. For instance, since $q(D_1) = q(D_2) = \{t_a, t_b\}$ (as in Figure 3(a)), then $\Pr(\{t_a, t_b\}) = \frac{2}{9} + \frac{1}{9} = \frac{1}{3}$. All possible answers from evaluating q on D^r, denoted by $q(D^r)$, are shown in Figure 3(b).

EXAMPLE 1 (IMPORTANT STOCK TYPES).
```
SELECT distinct Type FROM stock
WHERE Qty > 0.3 * (SELECT SUM(Qty) FROM stock)
```

In general, when R has n tuples, $q(D^r)$ can have up to $O(2^n)$ possible answers, which makes it impractical to deliver the distribution on all possible answers. Instead, for each possible tuple in the query result (t_a and t_b in this case) we compute its marginal probability of appearing in the query result, as shown in Figure 3(c). For example, t_a may appear in the results $\{t_a, t_b\}$ or $\{t_a\}$. Thus, the marginal probability of t_a appearing in any result is $\frac{1}{3} + \frac{8}{27} = \frac{17}{27}$. We denote this marginal distribution as $q^{mrg}(D^r)$.

In real life, instead of our three-tuple example, we have tables with millions or billions of tuples, where enumerating all possible worlds is impossible. Thus, bootstrap uses Monte Carlo simulation to approximate the above process, which requires hundreds or thousands of trials to achieve an accurate estimate for $q^{mrg}(D^r)$.

With $q^{mrg}(D^r)$, we can now measure the quality of $q(D)$. For instance, the average false negative rate of tuples in $q(D)$, i.e., the average probability of missing t_a or t_b, can be computed by $\frac{1}{2}\{(1 - \Pr(t_a)) + (1 - \Pr(t_b))\} = \frac{1}{3}$.

2.2 Scope of Our Approach

In this paper, we propose a new approach, called the Analytical Bootstrap Method (ABM), which avoids the computational overhead of bootstrap. We study the set of conjunctive queries with aggregates, i.e., queries expressed in the following relational algebra: σ (selection), Π (projection), $\bowtie$ (join), δ (deduplication), and γ (aggregate). $\gamma_{A,\alpha(B)}$ denotes applying aggregate α on B grouped by A. In this paper, we focus on queries without nested aggregates, i.e., B is not the output of another aggregate. (Nested aggregates are discussed in our technical report [45].) To simplify presentation, we first focus on aggregates SUM, COUNT, and AVG, and defer the extension of ABM to more general aggregates to Section 7. Since Bootstrap only works for "smooth" queries,[5] we also restrict ourselves to such queries. In particular, any of the following constructs can easily lead to non-smooth queries: (1) extrema aggregates (MIN/MAX) and (2) equality checks on aggregate results (e.g., projection/group-by/equi-join on aggregate results). Thus, we only discuss queries without these constructs.

Among the studied queries, we identify *eligible queries*, for which we provide an efficient extensional evaluation. Furthermore, our query evaluation techniques enjoy a DBPTIME complexity for a large subset of eligible queries (see Section 5.2).

To provide an intuitive sense of how general/restrictive these classes of queries are in practice, in Table 1 we summarize the syntactic constraints imposed by different error estimation techniques

[5]A discussion of smoothness can be found in [19, 34].

	Type
t_a	a
t_b	b

(a)

answer	prob.
$\{t_a, t_b\}$	1/3
$\{t_a\}$	8/27
$\{t_b\}$	10/27

(b)

answer	prob.
t_a	17/27
t_b	19/27

(c)

Figure 3: (a) The result of $q(D_1)$ and $q(D_2)$, (b) all possible answers of q, and (c) their marginal probabilities

along with some statistics. Table 1 compares the set of queries supported by our formal semantics, our ABM (eligible queries and those with DBPTIME complexity), previous analytical techniques (which are strictly subsumed by ABM), and bootstrap (which strictly subsumes ABM). We have analyzed 22 TPC-H benchmark queries, as well as a real-life query log from Conviva Inc.[1] consisting of 6660 queries.[6] Table 1 reports the fraction of queries from TPC-H and the Conviva log that is supported by different techniques, i.e., queries that satisfy the constraints imposed by each technique.

The set of queries supported by ABM strictly subsumes those of previous analytical approaches, and it also constitutes a majority subset of those supported by bootstrap. For instance, ABM supports $19/22$ TPC-H queries, and 98.6% of the Conviva ones, covering most of the queries supported by bootstrap (i.e., $19/22$ in TPC-H and 99.1% in the Conviva log). Previous analytical approaches only support $9/22$ of TPC-H and 36.9% of the Conviva queries. Also, note that 81.0% of the Conviva queries are supported by ABM while enjoying a guaranteed DBPTIME complexity.

Note that User Defined Aggregate Functions (UDAFs) can only be handled by (simulation-based) bootstrap. Fortunately, UDAFs are quite rare in day-to-day data analysis, as most users find it more convenient to write pure SQL queries. For instance, *none* of the 6660 queries in the Conviva log contained a UDAF. [7]

3. BACKGROUND

In this section, we provide background on semirings and their connection with relational operators, followed by a brief overview of semiring random variables. These concepts will be used in our probabilistic relational model and query evaluation technique.

3.1 Semirings and Relational Operators

Here, we provide an overview on semirings as well as how they can be used to compute queries on a database.

A **monoid** is a triplet $(S, +, 0)$, where:

- S is a set closed under $+$, i.e., $\forall s_1, s_2 \in S$, $s_1 + s_2 \in S$
- $+$ is an associative binary operator, i.e., $\forall s_1, s_2, s_3 \in S$, $(s_1 + s_2) + s_3 = s_1 + (s_2 + s_3)$
- 0 is the identity element of $+$, i.e., $\forall s \in S$,$s+0 = 0+s = s$

For example, $(\mathbb{N}, +, 0)$ is a monoid, where $\mathbb{N}$ is the set of natural numbers, and 0 and $+$ are the numerical zero and addition.

A **semiring** is a quintuplet $(S, +, \cdot, 0, 1)$ which follows the four axioms below:

- $(S, +, 0)$ is a monoid where $+$ is commutative, i.e., $\forall s_1, s_2 \in S$, $s_1 + s_2 = s_2 + s_1$ (a.k.a. a **commutative monoid**)
- $(S, \cdot, 1)$ is a monoid

[6]Due to proprietary reasons, we had restricted access to Conviva queries; we were only allowed to run a customized parser on the query log to compute the breakdown of different query types. Thus, for performance evaluations, we use Vertica and TPC-H queries.

[7]UDAFs should not be confused with User Defined Functions (UDFs) which are quite common in practice (e.g., 42% of Conviva queries contain UDFs); UDFs operate on a single tuple and return a single value, while UDAFs operate on multiple tuples and return a single value. UDFs are usually used to transform or extract data from each tuple. By treating UDFs as extra columns in the table, they can be easily supported by both analytic approach and ABM.

Technique	Constraints	in TPCH	in Conviva Log
Analytic Approach	Simple group-by aggregate (no MIN/MAX) queries [9, 15, 16, 22, 23, 24, 25, 33, 44]	9/22	36.9%
ABM Semantics	Conjunctive queries with (a) no nested aggregates, (b) no MIN/MAX and (c) no equality checks on aggregate results	19/22	99.1%
ABM Eligible	Queries that (a) satisfy all ABM semantics constraints, and (b) have an eligible query plan (see Definition 9)	19/22	98.6%
ABM Eligible with DBPTIME Query Evaluation	ABM eligible queries that (a) have DBPTIME-eligible plans or (b) can be optimized by the containment join optimization (see Section 5.2)	15/22	81.0%
Bootstrap	"Smooth" queries	19/22	99.1%

Table 1: Classes of SQL queries supported by different techniques, and their coverage of TPC-H and Conviva queries

- $\cdot$ is left and right distributive over $+$, i.e., $\forall s_1, s_2, s_3 \in S$, $s_1 \cdot (s_2 + s_3) = s_1 \cdot s_2 + s_1 \cdot s_3$ and $(s_2 + s_3) \cdot s_1 = s_2 \cdot s_1 + s_3 \cdot s_1$
- 0 annihilates S, i.e. $\forall s \in S, 0 \cdot s = s \cdot 0 = 0$

A **commutative semiring** is a semiring in which $(S, \cdot, 1)$ is also a commutative monoid. In this paper, all semirings are commutative semirings. E.g., $S_{\mathbb{N}} = (\mathbb{N}, +, \cdot, 0, 1)$ is a commutative semiring.

Green et al. [26] showed that many different extensions of relational algebra can be formulated by annotating database tuples with semiring elements and propagating these annotations during query processing. Consider a multiset database as an example. Tuples in a multiset relation are annotated with natural numbers $\mathbb{N}$, representing their multiplicities in the database. Formally, a multiset relation R with the tuple domain U is described by an annotation function $\pi_R : U \to \mathbb{N}$, where a tuple $t \in R \Leftrightarrow \pi_R(t) \neq 0$.

During query processing, the relational algebra is extended with the $+$ and $\cdot$ operators in semiring $S_{\mathbb{N}}$, which manipulate the annotated multiplicities: for projection, we add the multiplicities of all input tuples that are projected to the same result tuple, while for join, we multiply the multiplicities of the joined tuples. That is, inductively we have:

- **Selection** $\sigma_c(R)$. $\pi_{\sigma_c(R)}(t) = \pi_R(t) \cdot \mathbb{1}(c(t))$, where $\mathbb{1}(c(t))$ returns 1 if $c(t)$ is true and 0 otherwise.
- **Projection** $\Pi_A(R)$. $\pi_{\Pi_A(R)}(t) = \sum_{t'[A]=t} \pi_R(t')$ where $t'[A]$ is the projection of t' on A.
- **Join** $R_1 \bowtie R_2$. $\pi_{R_1 \bowtie R_2}(t) = \pi_{R_1}(t_1) \cdot \pi_{R_2}(t_2)$, where t_i is t on U_i.

Green et al. showed that by annotating tuples with elements from $S_{\mathbb{N}}$ and using the extended relational algebra defined above, we obtain the relational algebra with multiset semantics. However, this extension is not applicable to bootstrap, because the multiset relation generated by bootstrap is probabilistic (shown in Section 2.1). Thus, we introduce our *probabilistic multiset relational model* in Section 4 by using *semiring random variables* (described next).

3.2 Semiring Random Variables

A random variable that takes values from the elements of a semiring S is called a *semiring random variable*, denoted by S-rv. Analogous to operators $+$ and $\cdot$ on semiring elements, there are corresponding operators that operate on semiring random variables, called *convolutions*. Below we define the $+$ convolution (denoted as $\oplus$), and the $\cdot$ convolution (denoted as $\odot$).

DEFINITION 1 (CONVOLUTION). *Let r_1 and r_2 be two S-rvs. The convolution $\oplus$ (resp. $\odot$) is a binary operator defined on monoid $(S, +, 0)$ (resp. $(S, \cdot, 0)$), such that $r_1 \oplus r_2$ (resp. $r_1 \odot r_2$) is a S-rv, where $\forall s \in S$,*

$$\Pr(r_1 \oplus r_2 = s) = \sum \{\Pr(r_1 = x \wedge r_2 = y) | \forall x, y \in S, x + y = s\}$$

$$\Pr(r_1 \odot r_2 = s) = \sum \{\Pr(r_1 = x \wedge r_2 = y) | \forall x, y \in S, x \cdot y = s\}$$

Similar to conventional random variables, we define the *disjointness*, *independence* and *entailment* between S-rvs, but with some special treatment for $0 \in S$.

R/D	deterministic relation/database
R^r/D^r	probabilistic relation/database from bootstrap
$\hat{R}$/$\hat{D}$	a resample instance from bootstrap
$m_{\hat{R}}(t)$	multiplicity of t in $\hat{R}$
U	tuple domain of a relation
$head(\cdot)$	set of attributes in the output of a query
$rels(\cdot)$	set of relations occurring in a query
π,ϖ	annotation functions for PMRs
0/1	Constant random variables with value 0/1

Table 2: Summary of Notations

DEFINITION 2. *Let r_1 and r_2 be two S-rvs.*

- *r_1 and r_2 are **disjoint**, if $\Pr(r_1 \neq 0 \wedge r_2 \neq 0) = 0$.*
- *r_1 and r_2 are **independent**, if $\forall s_1, s_2 \in S$,*

$$\Pr(r_1 = s_1 \wedge r_2 = s_2) = \Pr(r_1 = s_1) \cdot \Pr(r_2 = s_2)$$

- *r_1 **entails** r_2, if $\Pr(r_1 \neq 0 | r_2 \neq 0) = 1$.*

The marginal distribution of a S-rv r can be represented by a vector $\mathbf{p}^r$ indexed by S, namely $\forall s \in S$, $\mathbf{p}^r[s] = \Pr(r = s)$.

In general, using Definition 1 to compute convolutions of S-rvs can be quite inefficient, especially when the semiring S is large. However, we make the observation that under certain conditions, the convolution of S-rvs can be quickly computed by simply manipulating their probability vectors, as stated next.[8]

PROPOSITION 1 (EFFICIENT CONVOLUTION). *Let r_1 and r_2 be two S-rvs on semiring $(S, +, \cdot, 0, 1)$:*

- *If r_1 and r_2 are disjoint, then $\mathbf{p}^{r_1 \oplus r_2} = \mathbf{p}^{r_1} \uplus \mathbf{p}^{r_2}$, where*

$$(\mathbf{p}^{r_1} \uplus \mathbf{p}^{r_2})[s] \stackrel{\text{def}}{=\!=} \begin{cases} \mathbf{p}^{r_1}[s] + \mathbf{p}^{r_2}[s] & \text{if } s \neq 0 \\ \mathbf{p}^{r_1}[0] + \mathbf{p}^{r_2}[0] - 1 & \text{if } s = 0 \end{cases}$$

- *If r_1 entails r_2 where r_2 can only take value 0 or 1, then $\mathbf{p}^{r_1 \odot r_2} = \mathbf{p}^{r_1}$.*

4. SEMANTICS & QUERY EVALUATION

Next, we introduce our probabilistic multiset relational model, which annotates each tuple with a $S_{\mathbb{N}}$-rv representing its nondeterministic multiplicity and extends the relational algebra to propagate these annotations during query processing. Table 2 lists the notations we use.

4.1 Formal Semantics

Probabilistic Multiset Database Semantics. A *probabilistic multiset relation* (PMR) is a multiset relation whose tuples have nondeterministic multiplicities. Formally, the functional representation of a PMR R^r is an annotation function $\pi_R : U \to \{S_{\mathbb{N}}\text{-rv}\}$, where $\pi_R(t) = \pi_t$ when tuple t occurs in R^r with a random multiplicity π_t; otherwise, $\pi_R(t) = \mathbf{0}$.

Specifically, the PMR R^r resulting from bootstrapping R (of size n) is modeled thus: for all tuples $\{t_i | i = 1, \cdots, n\}$ in R,

[8]All the proofs in this paper are omitted due to space constraints but can be found in our technical report [45].

$(\pi(t_1), \cdots, \pi(t_n))$ jointly follow a multinomial distribution $M(n, [\frac{1}{n}, \cdots, \frac{1}{n}])$. We can also model a deterministic relation R as a special PMR, where $\forall t \in R, \pi_R(t) = \mathbf{1}$.

DEFINITION 3 (PMR SEMANTICS). *The semantics of a PMR R^r annotated by $\pi_R : U \to \{S_{\mathbb{N}}\text{-}rv\}$ is a set of possible multiset worlds $pmw(R^r)$, where the probability of each world (i.e., resample instance) $\hat{R}$ is*

$$\Pr(\hat{R}) = \Pr\left(\bigwedge\{\pi_R(t) = m_{\hat{R}}(t) \mid t \in R\}\right)$$

A *probabilistic multiset database* (PMDB) D^r is a database with PMRs. We omit the semantics of D^r, which have similar definitions (see [45]).

Query Semantics. Evaluating a query q on a PMDB D^r is equivalent to evaluating q on every possible multiset world in $pmw(D^r)$.

DEFINITION 4 (QUERY SEMANTICS). *The semantics of evaluating a query q on a PMDB D^r is a set of possible answers $q^{pmw}(D^r)$, where each possible answer's probability is:*

$$\forall \hat{R}_q \subseteq U_q : \ \Pr(\hat{R}_q) = \sum\{\Pr(\hat{D}) \mid \hat{D} \in pmw(D^r) \wedge q(\hat{D}) = \hat{R}_q\}$$

Since computing $q^{pmw}(D^r)$ is infeasible (see Section 2.1), we return a marginal summary $q^{mirg}(D^r)$. i.e., for each tuple $t \in U_q$, we return a probability vector $\mathbf{p}$, where $\mathbf{p}[m]$ is the probability of t appearing in any possible answer m times, i.e.,

$$\mathbf{p}[m] = \sum\{\Pr(\hat{R}_q) | \hat{R}_q \subseteq U_q, \pi_{\hat{R}_q}(t) = m\}$$

4.2 Intensional Query Evaluation

A prohibitive number of worlds makes direct application of the possible worlds semantics impractical. Thus, this section introduces our *intensional evaluation* and its semantics, which lay the theoretical foundation of the evaluation technique we introduce in Section 5. Intensional evaluation extends relational algebra to propagate the annotations symbolically throughout query execution.

Extending Relational Algebra. Analogous to the multiset semantics introduced in Section 3, we extend relational algebra with $\oplus$ and $\odot$ operators from semiring $(S_{\mathbb{N}}\text{-rv}, \oplus, \odot, \mathbf{0}, \mathbf{1})$ to manipulate the random multiplicities of each tuple, so that query evaluation using these operators will produce the correct results (with respect to the possible multiset worlds semantics). We define this extended relational algebra as follows. (We first discuss how aggregates manipulate tuple multiplicities, postponing discussion of how aggregates compute values later.)

DEFINITION 5 (INTENSIONAL EVALUATION). *Intensional evaluation is defined inductively on a query Plan P:*

- *If $P = \sigma_c(P_1)$, then $\pi_P(t) = \pi_{P_1}(t) \odot \mathbb{1}(c(t))$.*
- *If $P = \Pi_A(P_1)$, then $\pi_P(t) = \bigoplus_{t'[A]=t} \pi_{P_1}(t')$.*
- *If $P = P_1 \bowtie P_2$, then $\pi_P(t) = \pi_{P_1}(t_1) \odot \pi_{P_2}(t_2)$, where $t_i = t[head(P_i)]$.*
- *If $P = \delta(P_1)$, then $\pi_P(t) = \mathbb{1}(\pi_{P_1}(t))$.*
- *If $P = \gamma_{A,\alpha(B)}(P_1)$, then $\pi_P(t) = \pi_{\delta(\Pi_A(P_1))}(t)$.*

where $\mathbb{1}(r)$ maps a random variable r to another random variable, i.e. $\mathbb{1}(r) = 1$ if $r \neq 0$, and $\mathbb{1}(r) = 0$ otherwise. When r is deterministic, $\mathbb{1}(r)$ degenerates to $\mathbf{0}$ or $\mathbf{1}$.

Aggregates. Besides manipulating π, an aggregate produces values absent from the input relations; an extra semiring annotation (denoted as ϖ) is thus required to define its computation.

Next, we define the intensional evaluation of SUM. Since manipulating π is defined in Definition 5, we focus on the manipulation of ϖ. (COUNT is a special case of SUM and AVG can be defined as an arithmetic function of SUM and COUNT.)

DEFINITION 6 (INTENSIONAL EVALUATION OF $\gamma_{A,\text{SUM}(B)}$). *Intensional evaluation of $\gamma_{A,SUM(B)}(P)$ is defined as follows:*[9]

1. *Annotate P with $\varpi : U \to \{S_{\mathbb{R}}\text{-}rv\}$ where $S_{\mathbb{R}} = (\mathbb{R}, +, \cdot, 0, 1)$, i.e., $\varpi(t) = t[B] \odot \pi_P(t)$, where $t[B]$ is treated as a degenerated $S_{\mathbb{R}}$-rv taking a constant value.*
2. *Compute $SUM(B) = \varpi_{\gamma_{A,SUM(B)}(P)}(t) = \bigoplus_{t'[A]=t} \varpi_P(t')$.*

EXAMPLE 2. *Consider the database D in Figure 1(a) and the query q in Example 1. Figure 4 shows the intensional evaluation of q on D^r using the query plan shown in Figure 4(a). Figure 4(g) shows the truth table for tuple t_a. From this, we can determine the probability of t_a appearing in the result as $\frac{17}{27}$.*

Intensional evaluation of q on D^r produces a new PMR, denoted by $q^i(D^r)$, whose semantics is defined as follows.

DEFINITION 7 (INTENSIONAL SEMANTICS). *The semantics of $q^i(D^r)$ is a set of possible multiset worlds, $pmw(q^i(D^r))$, where any assignment of the annotations of each tuple t, namely $\pi_q(t)$ and $\varpi_q(t)$, yields a possible world instance $\hat{R}_q$, such that:*

$$\Pr(\hat{R}_q) = \Pr\left(\bigwedge\{\pi_q(t) = m_{\hat{R}_q}(t) \wedge \varpi_q(t) \models t \mid t \in U_q\}\right)$$

Here, $\varpi_q(t) \models t$ means that $\varpi_q(t)$ takes the corresponding aggregate value in t.

Because of the commutative and distributive properties of semirings, $q^i(D^r)$ is independent of the plan chosen for q [26]. Furthermore, this semantics is equivalent to the possible multiset worlds semantics $q^{pmw}(D^r)$, as stated next.

THEOREM 1. *We have $pmw(q^i(D^r)) \equiv q^{pmw}(D^r)$ for every conjunctive query q with aggregates and every PMDB D^r, as long as the projected, group-by, and aggregated columns are not the output of another aggregate.*

Theorem 1 proves the correctness of intensional evaluation. However, intensional evaluation is quite inefficient, since in the worst case the size of each annotation can grow to the same order of magnitude as the database, and computing the distribution requires enumerating all possible annotation values. Thus, our next section develops an efficient evaluation technique.

5. EXTENSIONAL QUERY EVALUATION

This section presents an *extensional evaluation* technique that increases efficiency over its intensional counterpart by manipulating succinct multinomial representations of the annotations rather than the annotations themselves,

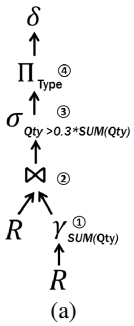
For simplicity's sake, we limit discussion to queries with a single (re)sampled relation and without any self-joins of the (re)sampled relation. (Section 7 discusses extensions to general queries.) Next, we introduce our multinomial representation, and then describe extensional evaluation for queries without and with aggregates.

5.1 The Multinomial Representation

This observation prompts our proposed multinomial representation of the annotations:

OBSERVATION 1. *A bootstrap trial on a relation R of n tuples comprises n i.i.d. experiments. The i-th experiment picks a single tuple at random, hence producing a probabilistic relation R_i^r. Formally, each R_i^r is annotated by $\rho_i : U \to \{S_{\mathbb{N}}\text{-}rv\}$, such that $\rho_i(t) = 1$ when tuple t is selected and $\rho_i(t) = 0$ otherwise. The relation R^r resulting from bootstrap is the union of $\{R_i^r\}$. One can easily verify that $\pi(t) = \bigoplus_{i=1}^n \rho_i(t)$.*

[9]W.l.o.g, in the following discussion, we assume $dom(B) = \mathbb{R}$.



(a)

tuple	ϖ_{SUM}	π
t_1	$4\pi_1$	π_1
t_2	$5\pi_2$	π_2
t_3	$3\pi_3$	π_3

(b)

ϖ_{SUM}	π
$4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3$	$\mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$

(c) Step ① in (a)

tuple	ϖ_{SUM}	π
t_1	$4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3$	$\pi_1 \odot \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$
t_2	$4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3$	$\pi_2 \odot \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$
t_3	$4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3$	$\pi_3 \odot \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$

(d) Step ② in (a)

tuple	π
t_1	$\pi_1' = \mathbb{1}(4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3 < 13.3) \odot \pi_1 \odot \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$
t_2	$\pi_2' = \mathbb{1}(4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3 < 16.7) \odot \pi_2 \odot \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$
t_3	$\pi_3' = \mathbb{1}(4\pi_1 \oplus 5\pi_2 \oplus 3\pi_3 < 10) \odot \pi_3 \odot \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3)$

(e) Step ③ in (a)

	Type	π
t_a	a	$\pi_1' \oplus \pi_3'$
t_b	b	π_2'

(f) Step ④ in (a)

π_1	π_2	π_3	prob.
1	1	1	2/9
2	1	0	1/9
2	0	1	1/9
1	0	2	1/9
3	0	0	1/27
0	0	3	1/27

(g)

Figure 4: (a) Query plan of Example 1, (b) initial annotation of R^r, (c-d) intensional evaluation steps, and (f) truth table of $\pi_q(t_a) = 1$

Based on Observation 1, we define *atoms* of an annotation $\pi(t)$ as the set of annotations comprising $\pi(t)$, which are generated from each experiment, i.e., $atom(\pi(t)) = \{\rho_i(t) \mid i = 1, \cdots, n\}$. The atoms hold these properties:

- Within an experiment i, $\rho_i(t)$ and $\rho_i(t')$ are disjoint for any two tuples t, t';
- Across different experiments i and j, $\rho_i(\cdot)$ and $\rho_j(\cdot)$ are independent;
- $\{\rho_i(t) \mid i = 1, \cdots, n\}$ are i.i.d. S-rvs with the same marginal probability vector $\mathbf{p}$, where $\mathbf{p}[0, 1] = [\frac{n-1}{n}, \frac{1}{n}]$.

These properties allow us to uniquely and succinctly represent $\pi(t)$ by a pair $[n, \mathbf{p}]$, namely a *multinomial representation*, reinterpretable as *the convolution sum of n i.i.d. semiring random variables with the probability vector* $\mathbf{p}$. The probability distribution of $\pi(t)$ can be easily reconstructed from its multinomial representation, using the probability mass function of the Multinomial distribution $M(n, \mathbf{p})$.

Moreover, when $atom(\pi(t))$ satisfies certain properties, $\oplus$ and $\odot$ can be directly computed on its multinomial representation, i.e.,

PROPOSITION 2. *Let $\pi_1 = [n, \mathbf{p}_1]$ and $\pi_2 = [n, \mathbf{p}_2]$.*

- *If $atom(\pi_1) \cap atom(\pi_2) = \emptyset$, then $\pi_1 \oplus \pi_2 = [n, \mathbf{p}_1 \uplus \mathbf{p}_2]$.*
- *If $atom(\pi_1) \subseteq atom(\pi_2)$, then $\pi_1 \odot \mathbb{1}(\pi_2) = [n, \mathbf{p}_1]$.*

Next, we show how our multinomial representation and its properties enable us to devise the extensional evaluation.

5.2 Queries without Aggregates

This section first formally defines the set of queries *eligible* for efficient evaluation, then presents our extensional evaluation.

Preliminaries. Let us denote the sampled relation as R^f. To simplify discussion, we base it on *canonical query plans*, obtainable by repeating the procedure below on an arbitrary query plan (using relational algebra's rewriting rules [20]):

1. Distinguish different occurrences of R^f by renaming them.
2. Push σ below Π and δ, e.g., $\sigma_c(\Pi_A(R)) \equiv \Pi_A(\sigma_c(R))$.
3. Eliminate duplicate δ, e.g., $\delta(\Pi_A(\delta(R))) \equiv \delta(\Pi_A(R))$.
4. Pull σ and Π above $\bowtie$, i.e., $R_1 \bowtie \sigma_c(R_2) \equiv \sigma_c(R_1 \bowtie R_2)$ and $R_1 \bowtie \Pi_A(R_2) \equiv \Pi_{head(R_1),A}(R_1 \bowtie R_2)$.
5. If both subqueries of $\bowtie$ are deduplicated, pull one δ above $\bowtie$, i.e. $\delta(R_1) \bowtie \delta(R_2) \equiv \delta(R_1 \bowtie \delta(R_2))$.

Note that in the canonical form δ is always the last operator of a join subtree. Since we do not consider self-joins of R^f, w.l.o.g. we use $\bowtie \delta$ instead of $\bowtie$, e.g., $q_1 \bowtie \delta(q_2)$ means q_1 joined with the deduplicated q_2.

We also define the *induced functional dependencies* of query q, denoted by $\Gamma(q)$ as in [18]:

- Any functional dependency of $rels(q)$ is in $\Gamma(q)$.
- $R^f.\pi \rightarrow head(R^f)$ and $head(R^f) \rightarrow R^f.\pi$ are in $\Gamma(q)$, i.e., each tuple in R^f has a unique annotation.
- For every join predicate $R_i.A = R_j.B$, both $R_i.A \rightarrow R_j.B$ and $R_j.B \rightarrow R_i.A$ are in $\Gamma(q)$;
- For every selection $R_i.A = constant$, $\emptyset \rightarrow R_i.A$ is in $\Gamma(q)$.

Following the convention from previous work [17], we recursively define the *lineage* of $\pi_P(t)$, denoted by $L(\pi_P(t))$, as

DEFINITION 8 (LINEAGE). *The lineage of $\pi_P(t)$ is defined inductively on a query plan P:*

- *If $P = \sigma_c(P_1)$, then $L(\pi_P(t)) = L(\pi_{P_1}(t))$.*
- *If $P = \Pi_A(P_1)$, then $L(\pi_P(t)) = \bigcup_{t'[A]=t} L(\pi_{P_1}(t'))$.*
- *If $P = P_1 \bowtie \delta(P_2)$, then $L(\pi_P(t)) = L(\pi_{P_1(t)})$.*
- *If $P = \delta(P_1)$, then $L(\pi_P(t)) = L(\pi_{P_1}(t))$.*
- *If P is a relation, $L(\pi_{R^f}(t)) = \{t\}$, $L(\pi_R(t)) = \emptyset$.*

It is easy to see that for any subplan P, the lineage $L(\pi_P(t))$ for any tuple t consists of tuples from the same base relation. We denote the relation as R_{L_P}.

Eligible Plan. Queries can be efficiently computed by the extensional evaluation with an eligible query plan. Given $\Gamma(q)$, we define an *eligible plan* thus.

DEFINITION 9 (ELIGIBLE PLAN). *A canonical plan P is eligible if all its operators are eligible:*

- *Operators σ, δ and $\bowtie$ are always eligible*
- *Operator $\Pi_A(P_1)$ is eligible, if $\Gamma(P_1)$ implies $\langle A, R_{L_{P_1}}.\pi \rangle \rightarrow head(P_1)$.*

A query's eligibility can be efficiently checked at compile time. Next, we introduce the extensional evaluation restricting ourselves to a strict subset of eligible queries (i.e., those with simple joins) before generalizing to all eligible queries.

Extensional Evaluation of Queries with Simple Joins. A join $P_1 \bowtie \delta(P_2)$ is a *simple join* if $R^f \notin rels(P_2)$. Given an eligible plan P where all joins are simple joins and standalone deduplication is the last operator of P, one can evaluate P via the following extensional evaluation procedure, which directly manipulates the multinomial representations of the annotations.

DEFINITION 10 (EXTENSIONAL EVALUATION (PART 1)). *Extensional evaluation is defined inductively on a query plan P. Let $\pi_{P_1}(t) = [n, \mathbf{p}_t]$, then:*

- *If $P = \sigma_c(P_1)$, then $\pi_P(t) = [n, \mathbf{p}_t]$ if $c(t)$ is true, and $[n, \mathbf{0}]$ otherwise.*
- *If $P = \Pi_A(P_1)$, then $\pi_P(t) = [n, \biguplus_{t'[A]=t} \mathbf{p}_{t'}]$.*
- *If $P = P_1 \bowtie \delta(P_2)$, then $\pi_P(t) = [n, \mathbf{p}_{t_1}]$ where $t_1 = t[head(P_1)]$.*
- *If $P = \delta(P_1)$, then $\pi_P(t) = [1, [\mathbf{p}_t[0]^n, 1 - \mathbf{p}_t[0]^n]]$.*

To reconstruct $q^{mrg}(D^r)$, we compute the probability distribution of $\pi_P(t) = [n, \mathbf{p}]$ by $\mathbf{p}^{\pi_P(t)}[k] = \binom{n}{k} \mathbf{p}[0]^{n-k} \mathbf{p}[1]^k$.

Intuitively, an eligible plan ensures that the tuple annotations are still disjoint after a projection. We formalize this intuition by tracking the lineage of tuples.

LEMMA 1. *For any subplans P_1 and P_2,*

1. $\forall t \in P_1$, $\pi_{P_1}(t) = \bigoplus_{t \in L(\pi_{P_1}(t))} \pi_{R^f}(t)$.
2. $\forall t_1, t_2 \in P_1$, $L(\pi_{P_1}(t_1)) \cap L(\pi_{P_1}(t_2)) = \emptyset$.

Lemma 1 ensures that the extensional evaluation correctly estimates $q^{mrg}(D^r)$ on any D^r by following Proposition 2.

Extensional Evaluation of General Queries. The extensional evaluation from Definition 10 does not apply to eligible queries with general joins. E.g., $P_1 \bowtie \delta(P_2)$ may produce different annotations: if $\pi_{P_2}(t_2) \neq 0$, $\pi_{P_1 \bowtie \delta(P_2)}(t) = \pi_{P_1}(t_1)$, and otherwise $\pi_{P_1 \bowtie \delta(P_2)}(t) = \mathbf{0}$. To resolve this, we enumerate all possible π_{P_2} values; since for each π_{P_2} value, $P_1 \bowtie \delta(P_2)$ is deterministic, we can apply the extensional evaluation from Definition 10.

To aid the enumeration, we represent $\pi(t)$ as a set of triplets $\{(\mathfrak{c}_i, \pi_i, \mathfrak{L}_i)\}$, where

- $\mathfrak{c}_i$ is a set of conjunctive conditions $\{c_j(\pi_{i,j}), \forall j\}$ on annotations $\{\pi_{i,j}, \forall j\}$,
- $\mathfrak{L}_i$ is a set of lineages $\{L(\pi_{i,j}), \forall j\} \cup \{L(\pi_i)\}$.

A triplet $(\mathfrak{c}_i, \pi_i, \mathfrak{L}_i)$ is interpreted as follows: *if all $c_j(\pi_{i,j}) \in \mathfrak{c}_i$ are true, $\pi(t)$ takes value π_i, and the lineages of $\{\pi_{i,j}, \forall j\}$ and π_i are stored in $\mathfrak{L}_i$*. The set $\{\mathfrak{c}_i, \forall i\}$ has two properties: (1) any $\mathfrak{c}_i$ and $\mathfrak{c}_j$ are disjoint, and (2) $\{\mathfrak{c}_i, \forall i\}$ enumerates all possible conditions on $\{\pi_{i,j}, \forall j\}$. Thus, the distribution of $\pi(t)$ can be reconstructed by Equation 1, which computes a weighted sum of π_i's distributions under all conditions. Since $\mathfrak{L}_i$ captures the correlation between $\mathfrak{c}_i$ and π_i, Equation 1 can easily be computed. (For details, see [45].)

$$f(\pi(t)) = \sum_{\forall i} \Pr(\mathfrak{c}_i) f(\pi_i \mid \mathfrak{c}_i) \tag{1}$$

At query time, we extend the extensional evaluation to manipulate $\mathfrak{c}_i$, π_i, and $\mathfrak{L}_i$. When combining two sets of conditions $\mathfrak{c}_i$ and $\mathfrak{c}_j$, we also modify π_i and π_j following Definition 10 and modify $\mathfrak{L}_i$ and $\mathfrak{L}_j$ following Definition 8. Below, we show the extended extensional evaluation. For the sake of simplicity, we only show the manipulation of $\mathfrak{c}_i$, omitting that of π_i and $\mathfrak{L}_i$. Furthermore, lineage maintenance has been well addressed in database provenance literature (e.g.,[21]). We denote the set $\{\mathfrak{c}_i, \forall i\}$ of $\pi_q(t)$ by $\mathbb{C}_q(t)$. Here, $\times$ denotes the set Cartesian product.

DEFINITION 11 (EXTENSIONAL EVALUATION (PART 2)). *Extensional evaluation is defined inductively on a query plan P:*

- *If $P = \sigma_c(P_1)$, then $\mathbb{C}_P(t) = \mathbb{C}_{P_1}(t)$.*
- *If $P = \Pi_A(P_1)$, then $\mathbb{C}_P(t) = \times_{t'[A]=t} \mathbb{C}_{P_1}(t')$.*
- *If $P = P_1 \bowtie \delta(P_2)$, then $\mathbb{C}_P(t) = \mathbb{C}_{P_1}(t_1) \times \mathbb{C}_{P_2}(t_2) \times \{\{\pi_{P_2}(t_2) \neq 0\}, \{\pi_{P_2}(t_2) = 0\}\}$ where $t_i = t[head(P_i)]$.*
- *If $P = \delta(P_1)$, then $\mathbb{C}_P(t) = \mathbb{C}_{P_1}(t) \times \{\{\pi_{P_1}(t) \neq 0\}, \{\pi_{P_1}(t) = 0\}\}$.*

This extensional evaluation works for any eligible query q. For certain queries, one can theoretically create a worst-case database that makes $|\mathbb{C}_q(t)|$ grow exponentially in the size of the database. Next, we identify situations where $|\mathbb{C}_q(t)| = O(1)$, i.e., when DBPTIME evaluation is guaranteed.[10]

DEFINITION 12 (DBPTIME-ELIGIBLE PROJECTION). *$\Pi_A(P_1)$ is DBPTIME-eligible, if for every join $P_2 \bowtie \delta(P_3)$ or $\delta(P_3)$ in P_1, $\Gamma(P_1)$ implies $A \rightarrow head(P_3)$.*

If $\Pi_A(P_1)$ is DBPTIME-eligible, then $\mathbb{C}_{\pi_A(P_1)}(t) = \mathbb{C}_{P_1}(t')$ for any $t'[A] = t$, giving us the following lemma:

LEMMA 2. *If every Π_A in an eligible plan P is DBPTIME-eligible, then $|\mathbb{C}_P(t)| = O(1)$.*

[10]However, some queries may still be evaluated efficiently on real-life databases despite being DBPTIME-ineligible, e.g., queries Q17, Q18, Q20, Q22, V3, and V4 in Section 8.

We can optimize the extensional evaluation for eligible queries (both DBPTIME-eligible and DBPTIME-ineligible queries) by detecting containment joins. A join $P_1 \bowtie \delta(P_2)$ is a *containment join* if (1) $\Gamma(P_1 \bowtie \delta(P_2))$ implies $head(P_1) \rightarrow head(P_2)$ and (2) $Q = \delta(\Pi_{head(P_1)}(P_1 \bowtie \delta(P_2)))$ contains $\delta(P_1)$, i.e., $\delta(P_1) \subseteq Q$ for any input database. Whether a join is a containment join can be decided in polynomial time for a large class of conjunctive queries [29]. A containment join ensures that joined tuples' annotations satisfy entailment, i.e.,

LEMMA 3. *For any tuple t generated by a containment join $P_1 \bowtie \delta(P_2)$, $L(\pi_{P_1}(t_1)) \subseteq L(\pi_{P_2}(t_2))$ where $t_i = t[head(P_i)]$.*

Therefore, we have the following optimization:

- If $P = P_1 \bowtie \delta(P_2)$ is a containment join, then $\mathbb{C}_P(t) = \mathbb{C}_{P_1}(t_1)$ where $t_i = t[head(P_i)]$.

Note that the size of $\mathbb{C}_P(t)$ is reduced by pruning the set $\mathbb{C}_{P_2}(t_2) \times \{\{\pi_{P_2}(t_2) \neq 0\}, \{\pi_{P_2}(t_2) = 0\}\}$. Furthermore, the corresponding lineage can be dropped from the annotation. Specifically, if every join in plan P either satisfies Definition 12 or is a containment join, P can be evaluated in DBPTIME. The following is an example of the extensional evaluation for general queries.

EXAMPLE 3. *Consider the query in Example 1 and its evaluation steps in Figures 4(c) and 4(d). Let $\pi_i = [3, \mathbf{p}_i]$ for $i = 1, 2, 3$, where $\mathbf{p}_i[0, 1] = [2/3, 1/3]$.*

- *$\pi_1 \oplus \pi_2 \oplus \pi_3 = [3, \biguplus_{i=1}^{3} \mathbf{p}_i] = [3, \mathbf{p}]$ where $\mathbf{p}[0, 1] = [0, 1]$.*
- *$\pi_i \odot \mathbb{1}(\pi_1 \oplus \pi_2 \oplus \pi_3) = [3, \mathbf{p}_i]$, for $i = 1, 2, 3$ as Step ② is a containment join.*

5.3 Queries with Aggregates

This section extends our extensional evaluation to queries with aggregates. We first consider aggregates in the return clause of a query, then aggregates in predicates.

Handling Aggregates in Return Clauses. Similar to Section 5.1, we can represent the annotation ϖ of aggregate $\gamma_{A,\alpha(B)}$ in a multinomial representation, such that Proposition 2 still applies to ϖ. Taking $\gamma_{A,\texttt{SUM}(B)}(P)$ as an example, let $\pi_p(t) = [n, \mathbf{p}]$. We can represent the annotation $\varpi(t)$ as $[n, \mathbf{p}']$ where $\mathbf{p}'[0] = \mathbf{p}[0]$ and $\mathbf{p}'[t[B]] = \mathbf{p}[1]$.

To check the eligibility of plan P of a query with aggregates, one can simply substitute every $\gamma_{A,\alpha(B)}$ in P with $\delta(\Pi_A)$, and check modified plan P^*'s eligibility. P is eligible if and only if P^* is eligible. Extensional evaluation of γ is very similar to Π (as in Definition 10 and 11), and is omitted. (When maintaining the multinomial representation $[n, \mathbf{p}]$ of ϖ, $\mathbf{p}$ could grow to the database size. Space-efficient approximation is introduced in Section 6.)

Handling Aggregates in Predicates. In operator σ_c, if the predicate c contains aggregate $\gamma(q)$ and $R^f \in rel(q)$ (e.g., see step ③ in Figure 4(a)), then $c(t)$ is uncertain since γ is random. Thus, we must enumerate all possible value ranges of γ. We modify Definition 11 thus:

DEFINITION 13 (EXTENSIONAL EVALUATION (PART 3)). *We extend Definition 11 by modifying the rule of σ as follows:*

- *If $P = \sigma_c(P_1)$, then $\mathbb{C}_P(t) = \mathbb{C}_{P_1}(t) \times enum_c(t)$ where $enum_c(t)$ enumerates all possible valuations of predicate c, that is, if c is deterministic, $enum_c(t) = \{\mathbf{T}\}$ where $\mathbf{T}$ is the event true; otherwise, $enum_c(t) = \{\{c(t)\}, \{\neg c(t)\}\}$.*

Similar to Section 5.2, the size of $\mathbb{C}_q(t)$ may grow exponentially with the size of the database due to projection and aggregation. Next, we propose an optimization technique to prune conditions with 0 probability, greatly reducing $\mathbb{C}_q(t)$'s size. Then, we again identify cases where DBPTIME evaluation is guaranteed.

OBSERVATION 2. *Consider two sets of conditions* $\{\gamma < 4, \gamma \geq 4\}$ *and* $\{\gamma < 3, \gamma \geq 3\}$. *A Cartesian product of these two sets produces* 4 *conditions. However, clearly* $\gamma \geq 4 \wedge \gamma < 3$ *is false. In fact,* 3 *and* 4 *partition the domain of* γ *into three parts:* $(-\infty, 3)$, $[3, 4)$ *and* $[4, \infty)$. *Now a Cartesian product of these two sets produces exactly* 3 *valid conditions corresponding to the three partitions, i.e.,* $\gamma \in (-\infty, 3)$, $\gamma \in [3, 4)$ *and* $\gamma \in [4, \infty)$.

We can generalize this observation into the following pruning rule, reducing the size of the Cartesian product of m condition-sets from $O(2^m)$ to $O(m)$.

PROPOSITION 3. *Let* $c_0 = -\infty < c_1 < \cdots < c_m < c_{m+1} = \infty$ *and* $\mathbb{C}_i = \{\{\gamma < c_i\}, \{\gamma \geq c_i\}\}$, $i = 1, \cdots, m$. *Then,*

$$\times_{i=1}^{m} \mathbb{C}_i = \{\{\gamma \in (c_j, c_{j+1}]\} \mid j = 0, \cdots, m\}$$

Next, we identify situations with efficient condition enumeration.

LEMMA 4. *After substituting every* $\gamma_{A,\alpha(B)}$ *with* $\delta(\Pi_A)$ *in an eligible plan* P, *if all the* Π *are DBPTIME-eligible, then* $|\mathbb{C}_P(t)| = O(n^{|P|})$, *where* n *is the size of the database.*

5.4 Correctness and Complexity

Let $q^e(D^r)$ be the result of extensional evaluation of query q on D^r. The next theorem ensures that extensional evaluation gives (1) the correct $q^{mrg}(D^r)$ for eligible queries on any D^r, and (2) an efficient evaluation time for DBPTIME-eligible queries.

THEOREM 2. *For any eligible query* q *and any PMDB* D^r, $q^e(D^r) \equiv q^{mrg}(D^r)$. *For any DBPTIME-eligible query* q, $q^e(D^r)$ *can be computed in DBPTIME.*

The extensional evaluation efficiently computes the annotations in multinomial representation. Nevertheless, reconstructing the exact distributions of aggregates from their annotations can be computationally expensive. For example, the multinomial representation of SUM could be of size $O(n)$ where n is the size of the database. This requires enumerating $O(2^n)$ possible cases to compute the distribution of SUM. However, many real-world users willingly trade accuracy for efficiency. Thus, Section 6 introduces a fast approximation technique for reconstructing the distribution from the annotations.

6. EFFICIENT APPROXIMATION

We now describe our solution to the problem introduced in Section 5.4. By approximating multinomial representations as asymptotically Gaussian distributions, we efficiently reconstruct the required distributions from extensional evaluation outputs. Further, we approximate the multinomial representation by maintaining a few moments (i.e., mean, variance, and covariance) instead of the probability vector itself, which reduces the space overhead.

Given an extended multinomial representation $(\mathfrak{c}, \varpi^*, \mathfrak{L})$, Theorem 3 states that we can model $\{\varpi^*\} \cup \{\varpi_j \in \mathfrak{c}, \forall j\}$ as an asymptotically multivariate normal distribution. Here, we slightly abuse the notation to refer to a random variable by its probability vector.

THEOREM 3. *Let* $\varpi_i, i = 1, \cdots, k$ *be* k *semiring random variables, whose multinomial representations are* $[n, \mathbf{p}_i]$, *respectively. When* $n \to \infty$, $\vec{\varpi} = [\varpi_1, \cdots, \varpi_k]$ *jointly follows asymptotically a Gaussian distribution:*

$$\vec{\varpi} \sim \mathcal{N}(n\vec{\mu}, n\Sigma)$$

where $\vec{\mu}_i = \mu_{\mathbf{p}_i}$ *is the mean of* $\mathbf{p}_i$, $\Sigma_{i,i} = \sigma^2_{\mathbf{p}_i}$ *is the variance of* $\mathbf{p}_i$, *and* $\Sigma_{i,j} = \sigma_{\mathbf{p}_i,\mathbf{p}_j}$ *is the covariance of* $\mathbf{p}_i, \mathbf{p}_j$. *Furthermore, applying any differentiable function* ψ *on* $\vec{\varpi}$ *still yields an asymptotic Gaussian distribution:*

$$\psi(\vec{\varpi}) \sim \mathcal{N}\left(\psi(n\vec{\mu}), n\nabla\psi(n\vec{\mu})^T \Sigma \nabla\psi(n\vec{\mu})\right)$$

where $\nabla\psi$ *is the gradient of* ψ.

This theorem provides accurate approximation whenever the size n of the fact relation is sufficiently large; this is typically the case in large-scale real-life applications. Also, various rules of thumb [14] can be used to check the quality of approximation. Next, we explain how to compute the distribution of AVG, and the conditional probability/distributions in Equation 1.

Computing AVG. AVG can be expressed as SUM/COUNT. Thus, to compute $\gamma_{A,\text{AVG}(B)}$, we rewrite it as two aggregates $\gamma_{A,\text{SUM}(B)}$ and $\gamma_{A,\text{COUNT}(B)}$, and evaluate them instead of the original AVG. Let ϖ be the AVG, and ϖ_1, ϖ_2 be the corresponding SUM and COUNT. We have $\varpi = \psi(\varpi_1, \varpi_2)$ where $\psi(x, y) = x/y$, for which we can directly apply Theorem 3 to compute the distribution.

Computing Conditions. As discussed in Section 5.3, there are two types of conditions: (1) comparing an aggregate with a constant, and (2) comparing two aggregates. As discussed in Section 2.2, we focus on comparison operators $\{<, >, \leq, \geq\}$.

W.l.o.g., consider extensional evaluation output $(\mathfrak{c}, \varpi^*, \mathfrak{L})$ where conditions $\mathfrak{c} = \{\varpi_i < 0 \mid i = 1, \cdots, m\} \cup \{\varpi'_i < \varpi''_i \mid i = 1, \cdots, n\}$. Let $\vec{\varpi}$, $\vec{\varpi}'$ and $\vec{\varpi}''$ denote the vectors $[\varpi_i]$, $[\varpi'_i]$ and $[\varpi''_i]$ respectively. We have the following corollary:

COROLLARY 1. *Let* $[\varpi^*, \vec{\varpi}, \vec{\varpi}', \vec{\varpi}''] \sim \mathcal{N}(n\mu, n\Sigma)$. *Then*

$$[\varpi^*, \vec{\varpi}, \vec{\varpi}' - \vec{\varpi}''] \sim \mathcal{N}(nA\mu, nA^T\Sigma A) = f$$

where $A = [1, \mathbf{0}_{1(m+2n)}; \mathbf{0}_{m1}, I_m, \mathbf{0}_{m(2n)}; \mathbf{0}_{n(m+1)}, I_n, -I_n]$. *Let* $\vec{l} = -\vec{\infty}$ *and* $\vec{u} = [\infty; \vec{0}]$. *Thus, (1)* $f(\pi \mid \mathfrak{c})$ *is the marginal distribution of* ϖ *when* f *is truncated by the range* $[\vec{l}, \vec{u}]$ *and (2)* $\Pr(\mathfrak{c})$ *is the cumulative probability in the range* $[\vec{l}, \vec{u}]$.

Computing the truncated probability/distribution in Corollary 1 is well-studied in statistics. Efficient solutions can be found in [43].

Moreover, Corollary 1 gives us a pruning method for further reducingthe number of conditions we need to enumerate, as in Corollary 2, which we can use to quickly prune conditions with extremely low probability.

COROLLARY 2. *Let* $\mathfrak{c}$ *be a set of conditions.* $\Pr(\mathfrak{c})$ *is computed by the cumulative probability of* $\mathcal{N}(n\vec{\mu}, n\Sigma)$ *in the range* $[\vec{l}, \vec{u}]$,

$$\Pr(\mathfrak{c}) \leq \Phi(\frac{\vec{u}_i - n\vec{\mu}_i}{\sqrt{n\Sigma_{i,i}}}) - \Phi(\frac{\vec{l}_i - n\vec{\mu}_i}{\sqrt{n\Sigma_{i,i}}}), \forall i$$

Sketching Multinomial Representations. As discussed above, our approximation technique only requires mean, variance, and covariance to reconstruct the distributions. We can use these moments to sketch the multinomial representations without maintaining the actual probability vector; this greatly reduces the storage overhead of our algorithms. During extensional evaluation, we directly maintain these moments using the following equations.

$$\mu_{\mathbf{p}_1 \uplus \mathbf{p}_2} = \mu_{\mathbf{p}_1} + \mu_{\mathbf{p}_2}, \sigma^2_{\mathbf{p}_1 \uplus \mathbf{p}_2} = \sigma^2_{\mathbf{p}_1} + \sigma^2_{\mathbf{p}_2} - 2\mu_{\mathbf{p}_1}\mu_{\mathbf{p}_2}$$
$$\sigma_{\mathbf{p}_1 \uplus \mathbf{p}'_1, \mathbf{p}_2} = \sigma_{\mathbf{p}_1, \mathbf{p}_2} + \sigma_{\mathbf{p}'_1, \mathbf{p}_2}$$

Note that, in the special case of simple group-by aggregates, our extensional evaluation becomes equivalent to previous analytical approaches. This is because in these cases extensional evaluation using the approximate multinomial representation simply consists of summing up the annotations of the corresponding tuples, thereby computing the basic statistics in the same way as previous analytical approaches. However, as we show in Table 1, our technique can handle a much larger class of queries.

7. EXTENSIONS OF ABM

In this section, we discuss several extensions of ABM.

General Aggregates. Many common aggregates can be written as functions of simple aggregates. E.g., VAR and STDEV can be written as functions of 1st and 2nd moments. Other aggregates such as MEDIAN, QUANTILE can also be computed using our annotation by a different CLT approximation [38].

Multiple Sampled Relations. Uniform sampling on multiple relations participating in a join yields non-uniform and undesirably sparse results [6]. Join synopses [6] are usually used to solve this problem, where the basic idea is to pre-compute the join (including self-joins), and uniformly sample from the join results. ABM can easily support join synopses by simply treating the join synopsis as the input sampled relation.

Using Stratified Samples. Although uniform samples are widely used in practice, in some cases stratified samples enable better approximations (e.g., for skewed data) where each stratum is itself a uniform sample, but with a different sampling rate than other strata [9, 16]. Bootstrap can also work on stratified samples, by bootstrapping each stratum and combining the resamples from all strata as the simulated dataset [19].

ABM can be easily extended to support stratified samples. Instead of using a single pair $[n, \mathbf{p}]$ to represent each annotation, we extend the multinomial representation to a set of pairs $\{[n_i, \mathbf{p}_i] \mid i = 1, \cdots, h\}$, one for each stratum. We can manipulate this generalized multinomial representation following a similar procedure, as described in Section 5. ABM can even handle the case where some strata are sampled while other small strata are not. In Section 8.4, we study ABM's accuracy on stratified samples.

8. EXPERIMENTS

To evaluate the effectiveness and efficiency of ABM, we conduct experiments on both synthetic and real-world workloads, and compare the results against both sequential and parallel/distributed implementations of bootstrap.

We use MonetDB (v11.15.19) [2] for implementing both bootstrap and ABM. We do not modify the internals of the relational engine, but rather implement a middle layer in Java to re-write the SQL queries to support our extensional evaluation. These modified queries are then executed by the relational engine. The returned results are fed into a post-processing module (implemented in R [3] to compute the probabilities/distributions. All pre- and post-processing times are included in ABM's execution times.

8.1 Experiment Setup

Parallel/distributed experiments are performed on a cluster of 15 machines, each with two 2.20 GHz Intel Xeon E5-2430 CPU cores and 96GB RAM. Sequential experiments use only one of these machines. We report experiments on three workloads: (1) TPC-H benchmark [4], (2) skewed TPC-H benchmark [35] and (3) a real-world dataset and query log from the biggest customers of Vertica Inc. [5] (referred to as Vertica). For details of the datasets and queries see [45].

TPC-H. We use a 100 GB benchmark (scale factor of 100). We use 17 queries out of the 22 TPC-H queries, namely: Q1, Q3, Q5-Q12, Q14, Q16-Q20, and Q22.[11] The other queries contain aggregates MIN/MAX, or otherwise do not satisfy our eligible query conditions. We (re)sample the largest relation *lineitem*. For queries without *lineitem*, we (re)sample the second largest relations, i.e., *customer* and *partsupp*.

[11]As some of queries produce undesirably sparse results under sampling, we keep the query structures but modify the very selective WHERE predicates and/or GROUP BY clauses.

Skewed TPC-H. We generate a 1 GB micro-benchmark (scale factor 1) using the SSB benchmark [35] (a star schema variation of TPC-H). All the numeric columns follow Zeta distribution with parameter $s \in [2.1, 2.3]$. We use 13 out of the above 17 TPC-H queries after modifying them according to the SSB schema; we leave out Q11, Q16, Q20 and Q22, which are inconsistent with the SSB schema. Again, we (re)sample the largest relation *lineorder*.

Vertica. The Vertica benchmark consists of 52 GB and 310 relations. We have chosen the 6 most complex queries (denoted as V1-V6) from the query logs, which have similar query structures as TPC-H queries Q1, Q11, Q18, Q22. Again, for each query, we (re)sample the largest relation. All (re)sampled relations have 9.2 million tuples each, and are 37.8 GB in total.

8.2 Error Quantification Accuracy

In this section, we evaluate the accuracy of our ABM. For each workload and each query q, we conduct three sets of experiments:

1. Ground Truth (GT). Similar to [27], we take an $x\%$ ($x = 1, 2, 5, 10$) random sample from a single relation, leave the other relations intact, and compute q on them. We repeat this procedure n times to collect the empirical distribution of all the n results.

2. Bootstrap (BS). We take an $x\%$ random sample from a single relation, we bootstrap this sample, and compute q on each resample and the other intact relations. We repeat the resampling process n times to collect the empirical distribution of all the n results.

3. ABM. We take an $x\%$ random sample from a single relation, and apply extensional evaluation to compute q on the sample and the other intact relations. For comparison purposes, we use the same random sample as bootstrap.

To compare the predicted distribution of query results given by ABM with the empirical distributions given by the ground truth and bootstrap, we measure various distribution statistics, including mean (MEAN), standard deviation (SD), quantiles (QN), 5%-95% confidence interval (CI), and existence probability (EP) (the probability of each tuple appearing in the query's output). For ground truth and bootstrap, we compute these statistics based on the empirical distribution of the collected results, whereas for ABM, we compute these statistics directly on the estimated distribution of the query results. We report the relative error of these statistics given by different methods. In the figures, we use the notation S-A-B to denote the relative error of the statistic S given by method A compared to the same statistic given by method B. For example, the relative error of our mean prediction μ_{ABM} to the empirical mean produced by Ground Truth μ_{GT} is defined as follows MEAN-ABM-GT $= |\frac{\mu_{\text{ABM}} - \mu_{\text{GT}}}{\mu_{\text{GT}}}|$. Note that some test queries (e.g., Q20 in TPC-H, V5 and V6 in Vertica) do not return aggregate values, for which we only report the existence probability.

When a query returns more than one column, we compute both the average and maximum relative error of all the columns in the query. Both errors are shown in our figures, where the histograms represent average error, and the T-shaped error bars represent the maximum error.

1. Predicting the Empirical Distribution of Bootstrap. In the first set of experiments, we study whether the approximate distribution produced by ABM is an accurate prediction of the empirical distribution given by bootstrap. For this purpose, we compare the two distributions in terms of: (1) EP-ABM-BS and QNk-ABM-BS for ($k = 5\%$ to 95%) as shown in Figure 5(a), and (2) the Kolmogorov-Smirnov (KS) distribution distance as shown in Figure 5(b).[12]

We perform the experiments on both TPC-H and Vertica benchmarks with different sample rates ($x = 1, 2, 5, 10$) and $n = 1000$ bootstrap trials. Due to space limitations, we only report the results

[12]http://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test

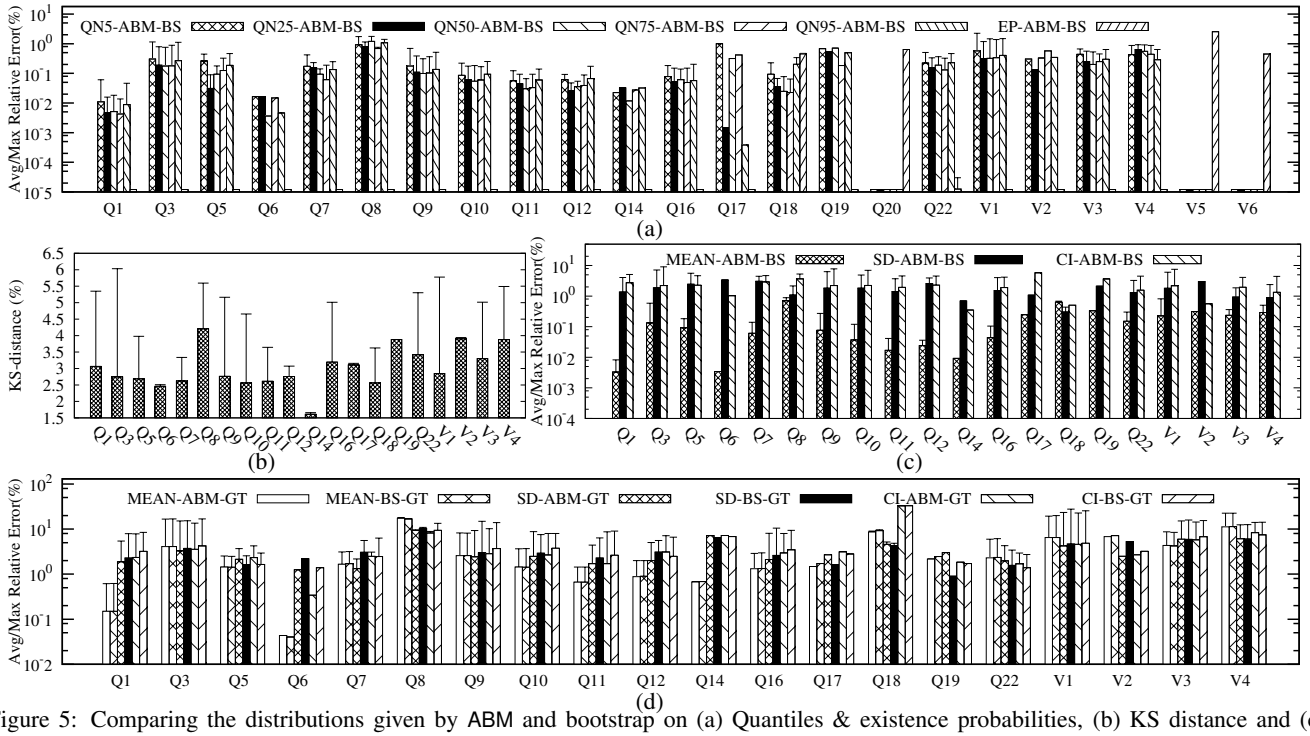


Figure 5: Comparing the distributions given by ABM and bootstrap on (a) Quantiles & existence probabilities, (b) KS distance and (c) User-defined quality measures; (d) Comparing user-defined quality measures given by ABMand bootstrap to ground truth

on the smallest sample size x=1%. The results on larger sample sizes have better accuracy, and thus are omitted. ABM provides highly accurate predictions of bootstrap across all different metrics and queries: On all quantiles, ABM's largest average relative error is less than 2%, while most of the average relative errors are even below 0.1%. Also, the maximum relative error is always below 5% across all quantiles. For most queries, both distributions predict the same existence probabilities, i.e., `EP-ABM-BS` is 0. The average KS distance is about 5%, which is relatively small for 1% sampling rate. In summary, these results show that ABM produces highly accurate predictions of the empirical distribution of bootstrap.

2. Predicting User-defined Quality Measures. In this set of experiments, we study the accuracy of various user-defined quality measures predicted by ABM. We take 1% samples and conduct 1000 sampling/bootstrap trials. The comparison results between ABM and bootstrap on TPC-H and Vertica workloads are reported in Figures 5(c). Again, ABM provides highly accurate predictions of bootstrap: On all statistics, ABM's maximum relative error is less than 10%, while most of the relative errors are even below 2%. We also report the results of comparing both ABM and bootstrap against the ground truth in Figure 5(d) on TPC-H and Vertica workloads. More interestingly, ABM can also serve as an accurate predication of the ground truth. As shown, comparing to the ground truth, most of ABM's average relative errors are below 5%. Noticeably, ABM is very consistent with bootstrap, i.e., when bootstrap approximates the ground truth well, ABM makes very accurate predictions; when bootstrap has a relatively large error, so does ABM. This evidences that ABM is equivalent to bootstrap.

3. Evaluating on skewed TPC-H benchmark. To study how ABM performs when encounters skewed data, we conduct a micro-benchmark study using the skewed TPC-H workload. We directly run 1000 bootstrap trials on the whole dataset and compare the user-defined quality measures predicted by ABM and bootstrap. The results are shown in Figure 6(a). Over all queries and all the compared statistics, ABM's maximum relative error is less than 10%, while most of the relative errors are even below 2%, which shows that the data skewness does not impact the accuracy of ABM much.

4. Varying Number of Bootstrap Trials & Sample Size. We also study the effect of the sample size and the number of bootstrap trials on the prediction accuracy of ABM. We compare ABM with bootstrap on both the distance measures used in Experiment 1 and the user-defined measures used in Experiment 2. Due to space limitations, we only report some representative measures on TPC-H and take the average and maximum of their relative errors across all queries. The other measures and the results on Vertica workload are similar, and are thus omitted.

To study the effect of the number of bootstrap trials, we fix the sampling rate to 1%, but vary the number of bootstrap trails (n = 100 to 1000). For each n, we compute the relative error of the statistics given by ABM against those given by bootstrap. As shown in Figure 6(b), the relative error between ABM and bootstrap decreases as the number of trials increases, which clearly shows that bootstrap suffers from accuracy loss with a finite number of trials, whereas ABM's analytical modeling of all possible worlds overcomes this limitation. Moreover, ABM saves the user from error-prone parameter tuning required by bootstrap.

To study the effect of sample size, we conduct the experiments using 1000 bootstrap trials, but varying the sampling rates (x = $1, 2, 5, 10$). As shown in Figure 6(c), ABM performs stably for `CI`, `SD`, and `KS`, while for more linear statistics (i.e., mean and quantiles) its error further decreases with higher sampling rates. Nonetheless, the average relative error of all statistics consistently stays below 3%, even for the smallest sample size (x = 1%).

8.3 Error Quantification Performance

This section demonstrates the superior speed of ABM by comparing it against both sequential and parallel/distributed state-of-the-art bootstrap implementations, as well as CLT-based analytical approach. We show that ABM is 5 orders of magnitude faster than the naïve bootstrap and 2-4 orders of magnitude faster than highly optimized variants of bootstrap.

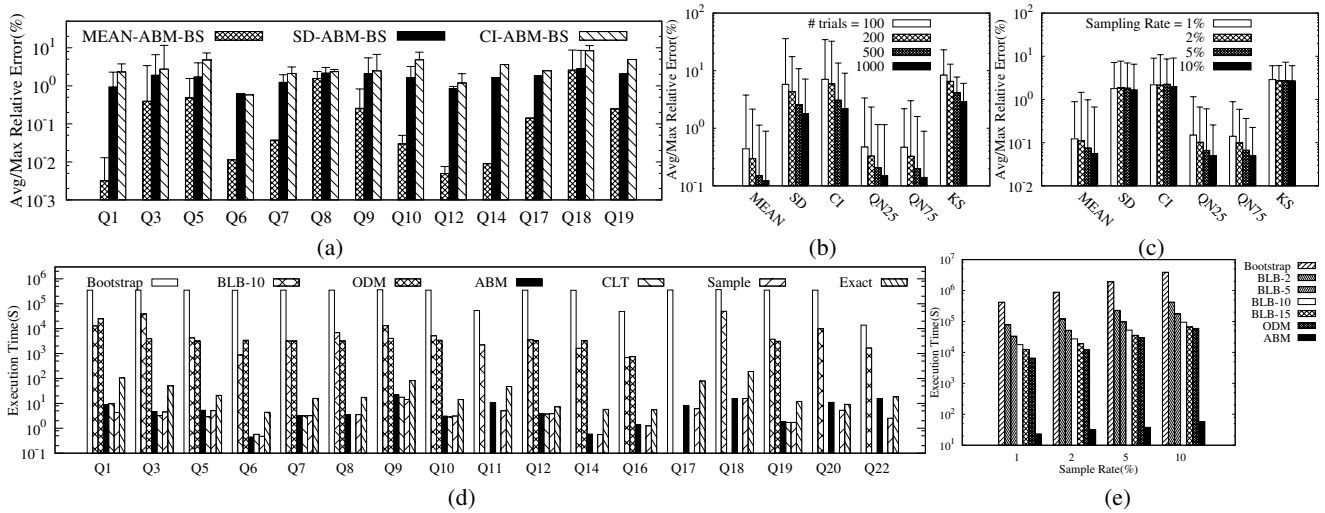


Figure 6: (a) ABM vs. bootstrap on user-defined quality measures for Skewed TPC-H; effect of varying (b) number of bootstrap trails, and (c) sampling rate; comparing time performance of ABM & various techniques (d) under 10% sampling rate, (e) under different sampling rates

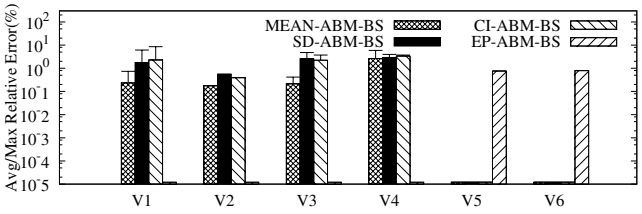


Figure 7: ABM vs. bootstrap under stratified sampling

5. Comparing Time Performance. In the first experiment, we compare ABM against three algorithms: (1) naïve bootstrap, (2) *On-Demand Materialization*(ODM) [34], and (3) Bag of Little Bootstrap (BLB) [28]. To the best of our knowledge, ODM is the best sequential bootstrap algorithm reported. However, it is limited to simple group-by aggregate queries. Bag of Little Bootstrap (BLB) is a bootstrap variant optimized for distributed and parallel platforms. We deploy the sequential algorithms (naïve bootstrap and ODM) and ABM on a single machine, while deploying the parallel/distributed algorithm (BLB) on 10 machines. We compare ABM with all the three counterparts on TPC-H ($x = 10\%$). All the counterpart bootstrap algorithms use 1000 trials. Figure 6(d) reports the running time. ABM is 5 orders of magnitude faster than the naïve bootstrap, and more than 2-4 orders of magnitude faster than ODM. Compared with BLB, ABM is 2-4 orders of magnitude faster although BLB is using 10 times more computation resources.

We also compare ABM with (1) CLT-based analytical approach (which is only applicable to simple group-by aggregates), (2) executing the approximate query on the sample (Sample), and (3) executing the exact query on the original DB (Exact). This comparison clearly demonstrates that ABM achieves almost identical running time as CLT and Sample, incurring little overhead. Since ABM is computed on 10% sample, ABM achieves 10X speed up on almost every query compared with Exact.

6. Varying the Sample Size. Furthermore, as shown in Figure 6(e), the execution time of naïve bootstrap and ODM increases with the sample size. On the contrary, our ABM does not vary much in terms of execution time as the sample size increases, since a large portion of ABM's time is spent on query evaluation, which can be highly optimized by modern database engines.

8.4 Using Stratified Samples

We study the accuracy of ABM when applied to stratified samples using the Vertica dataset. We apply different stratification on the 4 relations used in the experiments, consisting of 74 and up to 360 strata. The dataset is skewed such that the smallest stratum contains $1/100000$ of the corresponding relation, while the largest stratum contains 63%. We apply the same stratification configuration described in [9]. In particular, we take a stratified sample sized at 1% of the original relation equally from each stratum, while the overly small strata are not sampled. We report the relative error of the predictions given by ABM and bootstrap in Figure 7. As shown, all relative errors are below 5%, which evidences that ABM can be extended to support stratified sampling with high accuracy.

9. RELATED WORK

There has been a large body of research on using sampling to provide quick answers to database queries, on database systems [9, 15, 16, 22, 23, 24, 25, 33, 44], and data stream systems [12, 31]. Approximate aggregate processing has been the focus of many of these works, which study randomized joins [24], optimal sample construction [9, 16], sample reusing [44], and sampling plan in a stream setting [12, 31]. Most of them use statistical inequalities and the central limit theorem to model the confidence interval or variance of the approximate aggregate answers [9, 16, 22, 23, 24, 44]. Recently, Pansare et al. [33] develop a very sophisticated Bayesian framework to infer the confidence bounds of approximate aggregate answers. However, this approach is limited to simple group-by aggregate queries and does not provide a systematic way of quantifying approximation quality.

Many other works have focused on specific types of queries. For example, Charikar et al. [15] study distinct value estimation from a sample; Joshi and Jermaine [25] propose an EM algorithm to quantify aggregate queries with subset testing.

The bootstrap has become increasingly popular in statistics during the last two decades. Various theoretical [13, 19, 41] and experimental works [8, 27, 30, 34] have proven its effectiveness as a powerful quality assessment tool. Recent works [30, 34] have used bootstrap in a database setting, in order to quantify the quality of approximate query answers. Nevertheless, all these works focus on improving the Monte-Carlo process of the bootstrap. Thus, Pol et al. [34] focus on efficiently generating bootstrap samples in a relational database setting, while Laptev et al. [30] target MapReduce platforms and study how to overlap computation across different bootstrap trials or bootstrap samples. A diagnostic procedure is proposed in [8, 27] to determine when bootstrap's error estimation is reliable. This procedure applies bootstrap to multiple sample

sizes. Since ABM is equivalent to bootstrap, it can be seamlessly used in this diagnostic procedure, as long as the input query is supported by ABM (e.g., no UDAFs).

Another line of related work is approximate query processing in probabilistic databases. Much existing work in this area [10, 18, 36, 37, 42] uses possible world semantics to model uncertain data and its query evaluation. Tuples in a probabilistic database have binary uncertainty, i.e., they either exist or not with a certain probability. Specifically, [18, 36] use semirings for modeling and querying probabilistic databases, focusing on conjunctive queries with `HAVING` clauses. On the contrary, we focus on the bootstrap process and model resampled data, using a possible multiset world semantics where database tuples have uncertain multiplicities. Furthermore, bootstrap is fundamentally different from probabilistic databases, since tuples in a resampled relation are always correlated, whereas many probabilistic databases assume that tuples are independent [10, 18, 36, 37], or propose new query evaluation methods to handle particular correlations. For instance, [39, 40] propose Gaussian models to process continuous uncertainty data. Our work is instead based on the bootstrap, which is naturally characterized by discrete distributions, rather than the continuous distributions required by previous techniques.

10. CONCLUSION

In this paper, we developed a probabilistic model for the statistical bootstrap process and showed how it can be used for automatically deriving error estimates for complex database queries. First, we provided a rigorous semantics and a unified analytical model for bootstrap-based error quantification; then we developed an efficient query evaluation technique for a general class of analytical SQL queries. Evaluation using the new method is 2–4 orders of magnitude faster than the state-of-the-art bootstrap implementations. Extensive experiments on a variety of synthetic and real-world datasets and queries confirm the effectiveness and superior performance of our approach.

11. ACKNOWLEDGEMENTS

The authors would like to extend their gratitude to Andrew Lamb, James Finnerty and others at Vertica Inc. (for sharing an anonymized version of their customer queries and datasets), and to Ion Stoica and Sameer Agarwal (for sharing Conviva Inc. queries).

12. REFERENCES

[1] Conviva Inc. http://www.conviva.com/.
[2] MonetDB. http://www.monetdb.org/Home.
[3] The R Project. http://www.r-project.org/.
[4] TPC-H Benchmark. http://www.tpc.org/tpch/.
[5] Vertica Inc. http://www.vertica.com/.
[6] S. Acharya, P. B. Gibbons, et al. Join Synopses for Approximate Query Answering. In *SIGMOD*, pages 275–286, 1999.
[7] S. Acharya, P. B. Gibbons, et al. The Aqua Approximate Query Answering System. In *SIGMOD*, pages 574–576, 1999.
[8] S. Agarwal, H. Milner, et al. Knowing When You're Wrong: Building Fast and Reliable Approximate Query Processing Systems. In *SIGMOD*, 2014.
[9] S. Agarwal, B. Mozafari, et al. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *EuroSys*, pages 29–42, 2013.
[10] L. Antova, T. Jansen, et al. Fast and Simple Relational Processing of Uncertain Data. In *ICDE*, pages 983–992, 2008.
[11] B. Babcock, S. Chaudhuri, et al. Dynamic Sample Selection for Approximate Query Processing. In *SIGMOD*, pages 539–550, 2003.
[12] B. Babcock, M. Datar, et al. Load Shedding for Aggregation Queries over Data Streams. In *ICDE*, page 350, 2004.
[13] P. J. Bickel and D. A. Freedman. Some Asymptotic Theory for the Bootstrap. *The Annals of Statistics*, 9(6):1196–1217, 1981.
[14] G. Box, J. S. Hunter, et al. *Statistics for Experimenters: Design, Innovation, Discovery*. Wiley-Interscience, 2005.
[15] M. Charikar, S. Chaudhuri, et al. Towards Estimation Error Guarantees for Distinct Values. In *PODS*, pages 268–279, 2000.
[16] S. Chaudhuri, G. Das, et al. Optimized stratified sampling for approximate query processing. *TODS*, 32(2):9, 2007.
[17] Y. Cui, J. Widom, et al. Tracing the lineage of view data in a warehousing environment. *TODS*, 25(2):179–227, 2000.
[18] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16:523–544, 2007.
[19] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.
[20] H. Garcia-Molina, J. D. Ullman, et al. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.
[21] B. Glavic and G. Alonso. Perm: Processing Provenance and Data on the Same Data Model through Query Rewriting. In *ICDE*, pages 174–185, 2009.
[22] J. M. Hellerstein, P. J. Haas, et al. Online Aggregation. In *SIGMOD*, pages 171–182, 1997.
[23] Y. Hu, S. Sundara, et al. Estimating Aggregates in Time-Constrained Approximate Queries in Oracle. In *EDBT*, pages 1104–1107, 2009.
[24] C. Jermaine, S. Arumugam, et al. Scalable Approximate Query Processing with DBO Engine. In *SIGMOD*, pages 1–54, 2007.
[25] S. Joshi and C. Jermaine. Sampling-Based Estimators for Subset-Based Queries. *VLDB J.*, 18(1):181–202, 2009.
[26] G. Karvounarakis and T. J. Green. Semiring-Annotated Data: Queries and Provenance? *SIGMOD Record*, 41(3):5–14, 2012.
[27] A. Kleiner, A. Talwalkar, et al. A General Bootstrap Performance Diagnostic. In *KDD*, pages 419–427, 2013.
[28] A. Kleiner, A. Talwalkar, et al. The Big Data Bootstrap. In *ICML*, 2012.
[29] P. G. Kolaitis and M. Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. In *PODS*, pages 205–213, 1998.
[30] N. Laptev, K. Zeng, et al. Early Accurate Results for Advanced Analytics on MapReduce. *PVLDB*, 5(10):1028–1039, 2012.
[31] B. Mozafari and C. Zaniolo. Optimal Load Shedding with Aggregates and Mining Queries. In *ICDE*, pages 76–88, 2010.
[32] C. Olston, E. Bortnikov, et al. Interactive Analysis of Web-Scale Data. In *CIDR*, 2009.
[33] N. Pansare, V. R. Borkar, et al. Online Aggregation for Large MapReduce Jobs. *PVLDB*, 4(11):1135–1145, 2011.
[34] A. Pol and C. Jermaine. Relational Confidence Bounds Are Easy With The Bootstrap. In *SIGMOD*, pages 587–598, 2005.
[35] T. Rabl, M. Poess, et al. Variations of the Star Schema Benchmark to Test the Effects of Data Skew on Query Performance. In *SPEC*, pages 361–372, 2013.
[36] C. Ré and D. Suciu. The Trichotomy of HAVING Queries on a Probabilistic Database. *VLDBJ*, 18(5):1091–1116, 2009.
[37] P. Sen, A. Deshpande, et al. Read-Once Functions and Query Evaluation in Probabilistic Databases. *PVLDB*, 3(1):1068–1079, 2010.
[38] M. M. Siddiqui and C. Butler. Asymptotic Joint Distribution of Linear Systematic Statistics from Multivariate Distributions. *Journal of the American Statistical Association*, 64(325):300–305, 1969.
[39] T. T. L. Tran, Y. Diao, et al. Supporting User-Defined Functions on Uncertain Data. *PVLDB*, 6(6):469–480, 2013.
[40] T. T. L. Tran, L. Peng, et al. CLARO: Modeling and Processing Uncertain Data Streams. *VLDBJ*, (5):651–676, 2012.
[41] A. van der Vaart and J. Wellner. *Weak Convergence and Empirical Processes*. Springer, corrected edition, Nov. 2000.
[42] D. Z. Wang, E. Michelakis, et al. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. *PVLDB*, 1(1):340–351, 2008.
[43] S. Wilhelm. tmvtnorm: A Package for the Truncated Multivariate Normal Distribution. *sigma*, 2:2.
[44] S. Wu, B. C. Ooi, et al. Continuous Sampling for Online Aggregation over Multiple Queries. In *SIGMOD*, pages 651–662, 2010.
[45] K. Zeng, S. Gao, et al. The Analytical Bootstrap: A New Method for Fast Error Estimation in Approximate Query Processing. Technical Report CSD #130028, UCLA, 2013.