Sampling-Based Cardinality Estimation Algorithms: A Survey and An Empirical Evaluation

Wentao Wu
Department of Computer Sciences
University of Wisconsin-Madison
wentaowu@cs.wisc.edu

Abstract

Cardinality estimation is a fundamental problem that has been studied for several decades in database community. It has wide applications in many database management issues such as query optimization, query monitoring, query progress indicator, query execution time prediction, and approximate query answering. Existing cardinality estimation techniques can generally fall into two categories: i) methods based on histograms, and ii) methods based on sampling. Methods based on one-dimensional histograms currently dominate in the implementation of modern database systems (specifically, query optimizers), due to their simplicity and light overhead. However, they have inherent difficulty in estimating queries over more than one columns/tables, especially when data correlation exists. Sampling-based techniques, on the other hand, do not have such limitations. While they are not used in implementing query optimizers because of the high efficiency required by query optimization, they do find their ways in many other applications. In this paper, we give a survey of existing sampling-based algorithms proposed in the literature. We focus our discussion on important milestones along the timeline of history. In addition to the description and analysis of the algorithms, we also present our experimental evaluation results for several state-of-the-art methods on the TPC-H benchmark.

1 Introduction

Relational database management systems have achieved great success in the past forty years. One important reason for its success is the revolutionary notion of *data independence*, which completely separates the *logical* and *physical* view of the underlying data. The system is then allowed to manage the physical data in whatever ways it wishes, as long as the logical view of the data remains unchanged. This abstraction significantly reduces the burden of data management for application developers. They now only need to understand the logical view of the data without knowledge of how the data is actually stored and manipulated inside the database. Currently, the logical view of the data is defined and manipulated by the SQL query language. Database users can only access the data by issuing SQL queries.

SQL is a *declarative* language that only describes *what* the query is supposed to do without specifying *how* the query should be executed. This gives the system great flexibility to choose different *plans* for a given query. A plan is a tree-like structure that encodes the execution details of the query. Each node of the tree is some *operator*, such as *sequential scan*, *sort*, *hash join*, and so on. As indicated by its name, an operator executes certain functionality built inside the system. Since the execution cost of different plans can be dramatically different, a critical task of the database system is then to pick a good (i.e., low-cost) plan before the query is run. For this purpose, every existing database system contains such a module called *query optimizer*, and the procedure of seeking a good query plan is thus called *query optimization*.

As a result, the basic effort involved in query optimization is to estimate the cost of a plan. Query optimizers usually define effective cost models considering the CPU and I/O overhead of the query. To estimate these quantities, the key knowledge is the input/output cardinality of each operator in the plan. For example, to estimate the CPU cost of a *sort* operator, we need the number of tuples it is going to sort. This problem is termed *cardinality estimation* in database literature, and database community has spent several decades in studying it. Besides its usage in query optimization, cardinality estimation also has wide applications in many other areas, including *approximate query answering* (e.g., [1, 2]), *query monitoring* (e.g., [3]), *progress indicator* (e.g. [4, 5]), and *query execution time prediction* (e.g., [6, 7]).

Existing methods on cardinality estimation can generally fall into two categories: i) methods based on *histograms*, and ii) methods based on *sampling*. Histograms [8] maintain certain *statistics* of the underlying database *tables*, which try to capture the important statistical characteristics of the data distribution. One-dimensional histogram, which summarizes the data distribution over a single *column* of a table, currently dominates in the implementation of query optimizers, for its simplicity and light overhead. However, one-dimensional histogram cannot work well for many queries. Query optimizers have to rely on unrealistic assumptions such as independence and uniformity for queries involving data over more than one columns. A natural extension for one-dimensional histogram is *multi-dimensional* histogram, which captures the *joint* data distribution over multiple columns. However, multi-dimensional histograms are rarely used in practice due to their huge storage cost as the number of dimensions (i.e., columns) grows. Meanwhile, they still can only work well for queries over a single table. Sampling-based algorithms, on the other hand, generally do not have such limitations. A number of interesting algorithms have been proposed throughout the years (e.g. [9, 10, 11]). Sampling-based algorithms usually perform much better than histogram-based approaches by giving much more accurate estimations with theoretical guarantees. These algorithms, however, have not been implemented in existing query optimizers yet, due to their additional runtime overhead compared with histogram-based approaches. Cardinality estimation in query optimization needs to be very efficient since it is invoked for *every* plan considered by the optimizer, while the search space for possible query plans is usually quite big. Nonetheless, sampling-based algorithms have found their way in various other applications where cardinality estimation is required (e.g. [1, 2, 3]).

In this paper, we give a survey of existing sampling-based algorithms proposed in the database literature for cardinality estimation. While good surveys for histogram-based approaches already exist (e.g., [8]), we are not aware of any counterpart for sampling-based algorithms. This is partially due to the ignorance in current database systems for these algorithms, and sampling-based algorithms were scattered in the literature without being stitched together. We organize this survey along the timeline of history, and focus our discussion on important milestones. In addition to the description and theoretical analysis that may have been presented in original papers, we also evaluate the performance of several state-of-the-art approaches on a modern database system for typical SQL query workloads nowadays, in terms of both *efficiency* (i.e., runtime overhead) and *effectiveness* (i.e. estimation accuracy), with histogram-based approaches as baselines. To the best of our knowledge, this kind of benchmark study seems never to be done before.

The rest of the paper is organized as follows. Section 2 covers basic preliminary knowledge that is necessary for in-depth discussion of sampling-based cardinality estimation. Section 3 and 4 then introduce two basic estimation problems, namely, estimating the size of selection/join queries, and estimating the size of projection queries, respectively. As we will see, these two problems are inherently different and should be treated separately. We will describe important algorithms proposed in the literature as well as their analysis. In Section 5, we present our results from an experimental evaluation of several state-of-the-art algorithms on the TPC-H benchmark queries. Section 6 concludes the paper.

2 Preliminaries

In this section, we go over basic concepts in relational database theory and probability theory that will be frequently referred to in the subsequent sections of this survey. Meanwhile, we also introduce necessary notations that will be used throughout this paper. Readers that are already familiar with these concepts can safely skip this section.

2.1 Basic Relational Database Theory

We focus our discussion on the *relational model* and *relational algebra*. A comprehensive introduction to relational database theory is given in [12].

2.1.1 The Relational Model

A *relation schema* is represented as $R(a_1, a_2, ..., a_k)$, where R is the name and $(a_1, a_2, ..., a_k)$ is an *ordered* tuple. Each a_i is called an *attribute*, associated with a name and a type (e.g., integers, strings, ...). The number k is called the *arity* of R. A *relation instance* of R is a set of tuples $\langle u_1, u_2, ..., u_k \rangle$ conforming to R, i.e., u_i is of the type associated with a_i. In the rest of this survey, we simply use "*relation* R" to stand for a relation instance with the relation schema R. A relational database $\mathcal{D}$ is then defined as a collection of relations.

2.1.2 The Relational Algebra

Relational algebra is a formal *query language* that can be used to construct *queries* over relational databases[1]. It defines a set of *operators* that can manipulate relations. The result returned by any operator is still a relation. Hence it is easy to compose these operators to form a complex query, which is basically a *relational algebra expression*. The set of relational algebra expressions $\mathcal{Q}$ is recursively defined as follows:

1. Let R be a relation. Then $R \in \mathcal{Q}$.
2. Let R be a relation, and o^u be a unary operator. Suppose that the result relation by applying o^u to R is $o^u(R)$. Then $o^u(R) \in \mathcal{Q}$.
3. Let R and S be two relations, and o^b be a binary operator. Suppose that the result relation by applying o^b to R and S is $o^b(R, S)$. Then $o^b(R, S) \in \mathcal{Q}$.

Let $t \in R$ be any tuple of the relation $R(a_1, ..., a_k)$. We use $t(i)$ to denote the value of t on attribute a_i. The relational algebra specifies five basic operators:

- *Selection* $\sigma_F(R)$: This is a unary operator which returns a relation consisting of tuples from the relation R that pass the selection condition F. Here F is a Boolean combination (i.e., an expression using the logical connectives $\wedge$ and $\vee$) of *terms* with the form a_i `op` c or a_i `op` a_j, where a_i and a_j are attributes, c is some constant, and `op` is one of the comparison operators $<, \leq, >, \geq, =$, or $\neq$. Formally, we have
$$\sigma_F(R) = \{t | t \in R, \texttt{ and } F(t) \texttt{ is true}\}.$$
- *Projection* $\pi_{a_{j_1}, a_{j_2}, ..., a_{j_m}}(R)$: This is a unary operator which returns a relation consisting of tuples from relation R by restricting the tuples on the attributes a_{j_1}, a_{j_2}, ..., a_{j_m}. Formally, we have
$$\pi_{a_{j_1}, a_{j_2}, ..., a_{j_m}}(R) = \{\langle t(j_1), t(j_2), ..., t(j_m) \rangle | t \in R\}.$$
- *Cross-Product* $R \times S$: This is a binary operator that provides the capability for combining relations. It takes as input two relations R and S with arities m and n, and returns a relation with arity $m + n$. Formally, we have
$$R \times S = \{\langle t(1), ..., t(m), s(1), ..., s(n) \rangle | t \in R \texttt{ and } s \in S\}.$$
- *Union* $R \cup S$: This is a binary operator that returns the union of the tuples in R and S. R and S must be *union-compatible*, which means they should have the same arity and the corresponding attributes must be of the same type. Formally, we have
$$R \cup S = \{t | t \in R \texttt{ or } t \in S, \ R \texttt{ and } S \texttt{ are union-compatible}\}.$$

[1] SQL is a more powerful query language that contains relational algebra as a subset. In this survey, we focus on queries that can be expressed by relational algebra. Most of the real-world SQL queries fall into this subset.

- *Difference* $R - S$: This is a binary operator that returns the difference of the tuples in R and S. R and S must be *union-compatible*. Formally, we have

$$R - S = \{t | t \in R \texttt{ and } t \notin S, \texttt{ } R \texttt{ and } S \texttt{ are union-compatible}\}.$$

We introduce one more operator, i.e., the *join* operator, which can be expressed with the *selection* and *cross-product* operators:

- *Join* $R \bowtie_F S$: This is a binary operator defined as $R \bowtie_F S = \sigma_F(R \times S)$.

It is easy to see that the expressive power of the algebra with the operators $\{\sigma, \pi, \times, \cup, -\}$ and the algebra with the operators $\{\sigma, \pi, \bowtie, \cup, -\}$ are equivalent, which means they can compose exactly the same set of queries [12]. However, since the join operator is used very frequently, all database systems have several implementations of it. In fact, no database system implements the cross-product operator.

While relational algebra contains five operators, most of the existing work is about cardinality estimation of the three operators σ, π and $\bowtie$ (or equivalently, $\times$). There are several reasons. First, the operators $\cup$ and $-$ appear much less frequently in real-world queries than the other three queries. For example, none of the 22 queries in the TPC-H benchmark involves the operators $\cup$ and $-$. Second, even if the operators $\cup$ and $-$ present in the query, they are usually the operators executed at the end of the query plan. Therefore, the accuracy of their cardinality estimation is not important in query optimization, since they will always be performed in the same way no matter which query plan is considered. This is quite different from the join operator, where different join orders may lead to several magnitudes of overhead difference in executing the query. Third, the major errors in cardinality estimation usually come from the operators σ, π and $\bowtie$ (or equivalently, $\times$). As a result, for the purpose of this survey, we will focus our discussions on selection, projection, and join queries in the following sections.

2.2 Basic Probability Theory

We assume that the readers are familiar with basic concepts from probability theory, such as *sample space*, *event*, *probability measure*, *conditional probability*, *random variable*, *expectation* (or *mean*), *variance*, and so on. In the following, we list several important facts that will be used in the rest of this survey. They can be found in any standard textbook on probability theory (e.g., [13]). Throughout this survey, we use $Pr\big[\mathcal{E}\big]$ to denote the probability of an event $\mathcal{E}$, and use $E\big[X\big]$ and $Var\big[X\big]$ to denote the expectation and variance of a random variable X, respectively.

Proposition 1 (Linearity of Expectation)**.** *Let X_1, ..., X_n be arbitrary random variables. If $X = \sum_{i=1}^{n} a_i X_i$, namely, X is a linear combination of X_1, ..., X_n, then*

$$E\big[X\big] = E\big[\sum_{i=1}^{n} a_i X_i\big] = \sum_{i=1}^{n} a_i E\big[X_i\big].$$

Proposition 2. *Let X_1, ..., X_n be* independent *variables. If $X = \sum_{i=1}^{n} a_i X_i$, then*

$$Var\big[X\big] = Var\big[\sum_{i=1}^{n} a_i X_i\big] = \sum_{i=1}^{n} a_i Var\big[X_i\big].$$

Proposition 3 (Chebyshev's Inequality)**.** *Let X be a random variable with finite mean μ and variance σ^2. Then, for any $k > 0$,*

$$Pr\big[\big|X - \mu\big|\big] \leq \frac{\sigma^2}{k^2}.$$

Proposition 4 (The Central Limit Theorem)**.** *Let X_1, ..., X_n be a sequence of* independent *and* identically distributed *random variables, each having mean μ and variance σ^2. Then, for $-\infty < a < \infty$,*

$$\lim_{n\to\infty} Pr\big[\frac{\sum_{i=1}^{n} X_i - n\mu}{\sigma\sqrt{n}} \leq a\big] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-\frac{x^2}{2}} dx = \Phi(a),$$

where Φ is the cumulative distribution function of the standard normal distribution.

3 Selection and Join Queries

Selection and join are the two "work-horse" operators of relational systems. In this section, we describe algorithms dedicated to estimating the cardinality for selection and join queries. We focus our discussion on two sampling-based techniques. One is the so-called *adaptive sampling*, as proposed in [14, 9]. The other is the so-called *sequential sampling*, as proposed in [15, 10].

3.1 Adaptive Sampling

Adaptive sampling, as indicated by its name, has the nice *adaptive* nature in practice. Briefly speaking, the number of samples we need to estimate the query size within the desired error rate depends on the *size* of the samples. The bigger the sizes of the samples are, the smaller number of samples we need. Therefore, if the cost of a sample is some function of its size (as is commonly the case), then adaptive sampling can keep the running time constant rather than growing with the number of samples. We next describe the details of this idea.

3.1.1 The Urn Model

In [14], Lipton et al. presented the following urn model as the basis for adaptive sampling:

Definition 1 (Urn Model)**.** *Let U be an urn that contains n balls, where each ball i is associated with some number a_i such that $1 \leq a_i \leq b$, for $1 \leq i \leq n$. Here b is the upper bound of a_i's. Suppose that the quantity to be estimated is $A = \sum_{i=1}^{n} a_i$. The urn model then works as follows: Repeatedly sample with replacement until S, the sum of the costs on the balls sampled, satisfies $S \geq \alpha b$. Let the number of samples taken to be m. Estimate A as $\tilde{A} = nS/m$.*

One nice property of this urn model is that it can provide certain probabilistic guarantee to the estimation error, as stated in Theorem 1.

Theorem 1. *For $0 \leq p < 1$ and $d > 0$, if $\alpha = d(d+1)/(1-\sqrt{p})$, then*

$$Pr\big[|\tilde{A} - A| \leq \frac{A}{d}\big] \geq p.$$

The proof of Theorem 1 relies on several lemmas stated below. Let X_i be the value of the i-th sample ($1 \leq i \leq m$) (so X_i is a random variable). Since we sample with replacement, X_i's are independent of each other. Further more, X_i's conform to the same probabilistic distribution $F(x)$ with (unknown) mean μ and variance σ^2.

Lemma 1. *$b \geq \sigma^2/\mu$, where b is the upper bound for $X \sim F(x)$.*

Proof. [2] By definition of σ^2, we have

$$\sigma^2 = E\big[(X-\mu)^2\big] = E\big[X^2\big] - \mu^2 \leq E\big[X^2\big].$$

Since

$$E\big[X^2\big] \leq E\big[Xb\big] = bE\big[X\big] = b\mu,$$

we have $\sigma^2 \leq E\big[X^2\big] = b\mu$, which gives $b \geq \sigma^2/\mu$. □

Lemma 2. *Let $m = \beta\sigma^2/\mu^2$. Then*

$$Pr\big[\sum_{i=1}^{m} X_i \geq \frac{\alpha\sigma^2}{\mu}\big] \leq \frac{\beta}{(\alpha-\beta)^2}.$$

Proof. It is clear that $E\big[\sum_{i=1}^{m} X_i\big] = m\mu$, and $Var\big[\sum_{i=1}^{m} X_i\big] = m\sigma^2$ (ref. Section 2.2). We hence have

$$\begin{aligned} Pr\big[\sum_{i=1}^{m} X_i \geq \frac{\alpha\sigma^2}{\mu}\big] &= Pr\big[\sum_{i=1}^{m} X_i - \frac{\beta\sigma^2}{\mu} \geq (\alpha-\beta)\frac{\sigma^2}{\mu}\big] = Pr\big[\sum_{i=1}^{m} X_i - m\mu \geq (\alpha-\beta)\frac{\sigma^2}{\mu}\big] \\ &\leq Pr\big[\big|\sum_{i=1}^{m} X_i - m\mu\big| \geq (\alpha-\beta)\frac{\sigma^2}{\mu}\big] \leq \frac{m\sigma^2}{\frac{\sigma^4}{\mu^2}(\alpha-\beta)^2} = \frac{\beta}{(\alpha-\beta)^2}. \end{aligned}$$

[2] Proofs without citations are done by the author of this survey. Otherwise a citation to the original source will be pointed out.

The last second step follows by applying Chebyshev's inequality (ref. Section 2.2). □

Lemma 3. *Let $m \geq \beta\sigma^2/\mu^2$ and $d > 0$. Then*

$$Pr\big[\big|\frac{n}{m}\big(\sum_{i=1}^{m} X_i\big) - A\big| \geq A/d\big] \leq \frac{d^2}{\beta}.$$

Proof. Since $\mu = \frac{A}{n}$, we have

$$\begin{aligned} Pr\big[\big|\frac{n}{m}\big(\sum_{i=1}^{m} X_i\big) - A\big| \geq \frac{A}{d}\big] &= Pr\big[\big|\sum_{i=1}^{m} X_i - \frac{m}{n}A\big| \geq \frac{mA}{nd}\big] \\ &= Pr\big[\big|\sum_{i=1}^{m} X_i - m\mu\big| \geq \frac{m\mu}{d}\big] \leq \frac{m\sigma^2}{\frac{m^2\mu^2}{d^2}} \leq \frac{d^2}{\beta}. \end{aligned}$$

Similarly as in the proof of Lemma 2, the last second step follows by applying Chebyshev's inequality. □

We are now ready to prove Theorem 1.

Proof. (of Theorem 1) Note that

$$\tilde{A} = \frac{nS}{m} = \frac{n}{m}\sum_{i=1}^{m} X_i.$$

By Lemma 1 and 2, the probability that sampling will stop with less than $\beta\sigma^2/\mu^2$ samples is

$$Pr\big[\sum_{i=1}^{m} X_i \geq \alpha b\big] \leq Pr\big[\sum_{i=1}^{m} X_i \geq \frac{\alpha\sigma^2}{\mu}\big] \leq \frac{\beta}{(\alpha-\beta)^2}.$$

So the probability that sampling will stop with more than $\beta\sigma^2/\mu^2$ samples is at least $1 - \frac{\beta}{(\alpha-\beta)^2}$. By Lemma 3, when this happens, the probability that $Pr\big[\big|\tilde{A} - A\big| \geq A/d\big] \leq \frac{d^2}{\beta}$. Hence, the probability that $Pr\big[\big|\tilde{A} - A\big| \geq A/d\big] \geq 1 - \frac{d^2}{\beta}$. Therefore the probability that the urn model will successfully estimate A within A/d is at least $(1 - \frac{\beta}{(\alpha-\beta)^2})(1 - \frac{d^2}{\beta})$. We can bound this product by setting $1 - \frac{\beta}{(\alpha-\beta)^2} = 1 - \frac{d^2}{\beta} = \sqrt{p}$, where p is the desired probability. This gives $\beta = d^2/(1-\sqrt{p})$ and $\alpha = d(d+1)/(1-\sqrt{p})$, which completes the proof of the theorem. □

3.1.2 Query Partitioning

Before discussing the adaptive sampling algorithm, we need one more notion of *query partitioning* [14].

Definition 2 (Query Partitioning)**.** *A query q over a database D is n-partitionable if*

1. *The answer to q can be partitioned into n* disjoint *subsets q_i, for $1 \leq i \leq n$.*
2. *The size of q_i, written as $|q_i|$, is bounded by some constant b (i.e., $|q_i| \leq b$), for $1 \leq i \leq n$.*
3. *It is possible to randomly select a subset q_i and compute its size $|q_i|$.*

We next present two examples to illustrate this notion.

Example 1. *(Selection Query) Consider a selection query $q = \sigma_F(R)$. The answer set can be partitioned based on the tuples in R. Each tuple of R can be considered as a representative of a subset of the answer to the query. If the tuple satisfies the selection, then the size of the subset of 1. If not, the size is then 0. Here, the upper bound $b = 1$.*

Example 2. *(Two-way Join Query) Consider a two-way join query $q = R \bowtie_F S$. The answer set can be partitioned as follows. For each tuple $r \in R$, the partition is all tuples t such that t is generated by joining r with some tuple of S. The size of the subset is the number of S-tuples that join with r. Here, the upper bound $b = \max_{r \in R} |\{r\} \bowtie_F S|$.*

3.1.3 The Algorithm

The basic form of the algorithm is then quite straightforward, as illustrated in Algorithm 1.

Algorithm 1: Adaptive Sampling to Estimate Query Size

Input: q, a query; b, bound of partitions of q; d, error ratio; $0 \leq p < 1$, confidence on the estimation

Output: $\tilde{A}$, estimated size of q

```
S ← 0;
m ← 0;
while S < bd(d + 1)/(1 − √p) do
    S ← S + |RandomSample(q)|;
    m ← m + 1;
end
return Ã = nS/m;
```

The procedure *RandomSample* randomly picks some q_i (for $1 \leq i \leq n$) and returns the size $|q_i|$ of q_i after executing it. Based on Theorem 1, with probability p, $\tilde{A}$ will be within A/d of A, where $A = |q|$.

The bound $bd(d+1)/(1-\sqrt{p})$ in the algorithm can be further lowered if we assume the *central limit approximation* (CLA) applies, namely, we assume that $\frac{\sum_i^m X_i - m\mu}{\sqrt{m\sigma^2}}$ has the standard normal distribution (ref. Section 2.2). This is usually true in practice given that m is bigger than several tens.

Theorem 2. *Suppose that the central limit approximation applies. For* $0 \leq p < 1$ *and* $d > 0$*, if* $\alpha = d(d+1)\left[\Phi^{-1}\left(\frac{1+\sqrt{p}}{2}\right)\right]^2$*, then* $Pr\left[|\tilde{A} - A| \leq A/d\right] \geq p$*.*

The proof of Theorem 2 is very similar to the proof of Theorem 1 and hence omitted here. Readers can refer to [9] for the details. Table 1 compares the values of θ when CLA applies.

p	$\theta = 1/(1-\sqrt{p})$	$\theta = \left[\Phi^{-1}\left(\frac{1+\sqrt{p}}{2}\right)\right]^2$
0.80	9.5	2.6
0.90	19.5	3.8
0.95	39.5	5.0
0.99	199.5	12.2

Table 1: Comparison of θ when CLA applies or not

The remaining problem for this algorithm (and, in fact, for any sampling-based algorithms) is *data skewness*, namely, when b is much bigger than μ. Consider the following example:

Example 3. *Suppose that we are estimating a selection on a relation with 1,000,000 tuples, and there is only one tuple satisfies the selection condition. Then as discussed in Example 1, we will have 999,999 partitions of size 0, and one partition of size 1. This means that the expected size of a random sample is 1/1,000,000, and hence sampling until* $s > \theta bd(d+1)$ *(where* θ *is either* $\theta = 1 - \sqrt{p}$ *or* $\theta = \left[\Phi^{-1}\left(\frac{1+\sqrt{p}}{2}\right)\right]^2$*) requires expected to* $1,000,000 \cdot \theta bd(d+1)$ *samples.*

To address the issue of data skewness, the authors in [9] further proposed a technique called *sanity check*. The idea is to bound the number of samples by considering the error within the *worst-case* size (i.e., $A_{max} = bn$) instead of the actual size. To implement this, we only need to add a *sanity bound* $m < \eta e^2$ in the *while* condition of Algorithm 1. The following theorem summarizes the theoretical guarantee by adding this sanity check.

Theorem 3. *Suppose that the while loop terminates because of* $m \geq \eta e^2$*. For* $0 \leq p < 1$ *and* $e > 0$*, if* $\eta \geq 1/(1-p)$ *(or* $\eta \geq \left[\Phi^{-1}\left(\frac{1+p}{2}\right)\right]^2$ *if central limit approximation applies), then* $Pr\left[|\tilde{A} - A| \leq A_{max}/e\right] \geq p$*.*

Theorem 3 intuitively makes sense. Note that the sanity bound does not depend on the specific samples taken in each run of Algorithm 1. Instead, it only depends on the two constants p and e.

Therefore, it puts a uniform upper bound on the samples we shall take no matter which query whose size we want to estimate. Since it is so general, it is reasonable to see that its theoretical guarantee on the estimation accuracy is weaker than that was given in Theorem 1. The two *while* conditions given by Theorem 1 and 3 can hence be viewed as some trade-off between the estimation accuracy (i.e., effectiveness) and sampling overhead (i.e., efficiency). The proof of Theorem 3 can be found in [16].

3.1.4 Remarks

While the theoretical part of adaptive sampling is impressive, there are several issues that prevent Algorithm 1 to be applied in a real query optimizer. First, the algorithm has significant random I/O overhead on taking samples and evaluating sample queries. In [9], the authors proposed to use an *index-assistant* scheme to alleviate this problem, by assuming that each relation in the database has at least one available index. The index is usually small and can be cached in memory, although fetching the samples from the underlying relation via index look up still requires one random disk I/O per sample. The experimental results reported in [9] show that, for very simple queries (i.e., selection queries over a single relation, and two-way join queries that join two relations), it usually takes about 10% to 20% of the execution time of the original query for Algorithm 1 to finish. However, this is still inefficient compared with histogram-based methods. Second, the algorithm can only estimate the size of the query as a whole, while in query optimization, we need the cardinality of all subqueries of the query. Consider, for example, a three-way join query $q = R_1 \bowtie R_2 \bowtie R_3$. Algorithm 1 can estimate the size of q. But to estimate the subquery $q' = R_1 \bowtie R_2$, we have to run the algorithm on q' again. It is difficult to extend Algorithm 1 to estimate the size of q' and q simultaneously, due to the negative result that the join operator is not commutable with the sampling operator (see Theorem 10 in [17]). Third, the algorithm relies on some *a priori* upper bound b that is difficult to determine exactly. This issue is addressed by the so-called *sequential sampling*, as will be described next.

3.2 Sequential Sampling

Sequential sampling [15, 10] is a generalization of adaptive sampling. It overcomes the problem of the dependency on the a priori upper bound as required by adaptive sampling. The key idea is to use the samples observed so far to decide either to continue taking observations or to stop and return a final estimate. The most general form of sequential sampling was given in [10]. However, the authors only discussed its application on join queries. In fact, this framework can be applied to the more general class of queries involving arbitrary number of selections and joins. In this section, we describe sequential sampling in this setting.

3.2.1 The Estimator

Let $\mathcal{D}$ be a database consisting of K relations R_1, ..., R_K. Suppose that R_i is partitioned into m_i blocks each with size N_i, namely, $|R_i| = m_i N_i$. Consider the two basic relational operators: *selection* σ_F, and *cross-product* $\times$. For σ_F, we define the output of an input block B to be $\sigma_F(B)$. For $\times$, we define the output of the two input blocks B and B' to be $B \times B'$. Instead of estimating the cardinality of the output relation directly, the estimator will estimate the *selectivity* of the operator, which is defined as the output cardinality divided by the input cardinality. Specifically, the selectivity of the selection operator σ_F is $\rho_R = |\sigma_F(R)|/|R|$ where R is the input relation. Moreover, the selectivity of σ_F on a particular block B of R is $\rho_B = |\sigma_F(B)|/|B|$. On the other hand, the selectivity of the cross-product operator $\times$ is always 1. It is then straightforward to obtain the output cardinality once we know the selectivity of the operator (we always assume that the input cardinality is already known before the estimation procedure runs).

Lemma 4. *Consider $\sigma_F(R)$, where $R \in \mathcal{D}$ is a relation partitioned as $R = \{B_1, ..., B_m\}$. Let B_1^s, ..., B_n^s be a sequence of n blocks randomly picked from R (with replacement). Define $\tilde{\rho}_R = \frac{1}{n}\sum_{j=1}^{n} \rho_{B_j^s}$, where $\rho_{B_j^s} = |\sigma_F(B_j^s)|/|B_j^s|$, and let $b = |B_i|$. Then $E\left[\tilde{\rho}_R\right] = \rho_R$.*

Proof. We have

$$\rho_R = \frac{|\sigma_F(R)|}{|R|} = \frac{\sum_{j=1}^{m} |\sigma_F(B_j)|}{mb}.$$

Note that

$$E\big[\big|\sigma_F(B_j^s)\big|\big] = \frac{\sum_{j=1}^{m} |\sigma_F(B_j)|}{m}.$$

Thus,

$$E\big[\tilde{\rho}_R\big] = \frac{1}{nb}\sum_{j=1}^{n} E\big[\big|\sigma_F(B_j^s)\big|\big] = \frac{1}{nb}\sum_{j=1}^{n} b\rho_R = \frac{1}{nb}\cdot nb\rho_R = \rho_R.$$

□

Lemma 5. *Consider $\sigma_F(R_{i_1} \times \cdots \times R_{i_l})$, where $R_{i_j} \in \mathcal{D}$ and $l \geq 2$. Let $\mathbf{B}_1^s$, ..., $\mathbf{B}_n^s$ be a sequence of n blocks randomly picked from $R_{i_1} \times \cdots \times R_{i_l}$, where $\mathbf{B}_j^s = B_{i_1}^s \times \cdots \times B_{i_l}^s$ and each $B_{i_k}^s$ is a block randomly picked from the relation R_{i_k} (with replacement). Let $\mathbf{R} = R_{i_1} \times \cdots \times R_{i_l}$. Define $\tilde{\rho}_{\mathbf{R}} = \frac{1}{n}\sum_{j=1}^{n} \rho_{\mathbf{B}_j^s}$, where $\rho_{\mathbf{B}_j^s} = |\sigma_F(\mathbf{B}_j^s)|/|\mathbf{B}_j^s|$. Then $E\big[\tilde{\rho}_{\mathbf{R}}\big] = \rho_{\mathbf{R}}$.*

Proof. Let $B(i,j)$ be the j-th block of relation i. Then we have

$$\rho_{\mathbf{R}} = \frac{|\sigma_F(\mathbf{R})|}{|\mathbf{R}|} = \frac{\sum_{k=1}^{l}\sum_{j=1}^{m_{i_k}} |\sigma_F(B(i_k,j))|}{\prod_{k=1}^{l} m_{i_k} N_{i_k}}.$$

On the other hand,

$$E\big[\tilde{\rho}_{\mathbf{R}}\big] = \frac{1}{n}\cdot\frac{1}{\prod_{k=1}^{l} N_{i_k}}\sum_{j=1}^{n} E\big[|\sigma_F(\mathbf{B}_j^s)|\big].$$

Since

$$E\big[\big|\sigma_F(\mathbf{B}_j^s)\big|\big] = \rho_{\mathbf{R}}\prod_{k=1}^{l} N_{i_k},$$

we thus have

$$E\big[\tilde{\rho}_{\mathbf{R}}\big] = \frac{1}{n}\cdot\frac{1}{\prod_{k=1}^{l} N_{i_k}}\sum_{j=1}^{n}\rho_{\mathbf{R}}\prod_{k=1}^{l} N_{i_k} = \frac{1}{n}\cdot\frac{1}{\prod_{k=1}^{l} N_{i_k}}\cdot n\rho_{\mathbf{R}}\prod_{k=1}^{l} N_{i_k} = \rho_{\mathbf{R}}.$$

□

Lemma 4 and 5 tell us that the estimator $\tilde{\rho}_{\mathbf{R}}$ so defined is an *unbiased* estimator of the true selectivity $\rho_{\mathbf{R}}$. This result can be easily extended to the case of queries involving only selections and joins, as stated in the following theorem:

Theorem 4. *Let q be any query involving only selections and joins over $\mathbf{R}$. Then $E\big[\tilde{\rho}_q\big] = \rho_q$.*

Proof. The proof is easy by noticing that q can be written as its normal form [12]: $\sigma_F(\mathbf{R})$. We then apply Lemma 4 and 5 to complete the proof. □

3.2.2 Properties

We want to estimate the selectivity ρ to within $\epsilon\rho$ with probability p, for some $\epsilon > 0$ and $0 < p < 1$.

Definition 3 (Asymptotical Consistency)**.** *An estimator is* asymptotically consistent *if it satisfies*

$$\lim_{\epsilon\to 0} Pr\big[\big|\tilde{\rho} - \rho\big| \leq \epsilon\rho\big] = p.$$

Our next goal is to develop an approximate formula for the number of samples $N(\epsilon)$ we need to take to guarantee $Pr\big[\big|\tilde{\rho} - \rho\big| \leq \epsilon\rho\big] = p$, for some $\epsilon > 0$. To do this, we need some conditions. First, we assume that the distribution of the selectivity estimator is approximately normal after a large number of sampling steps. Second, we assume that the sample mean $\tilde{\rho}_n$ and sample variance S_n^2 are *strongly consistent* for the actual mean ρ and actual variance σ^2, namely $\lim_{n\to\infty}\tilde{\rho}_n = \rho$ a.s. and $\lim_{n\to\infty} S_n^2 = \sigma^2$ a.s. Based on these two assumptions, we can develop an explicit formula for $N(\epsilon)$ (i.e., an explicit stopping rule for the sampling procedure) as follows:

Let Φ be the cumulative distribution function for the standard normal distribution. Assume that central limit theorem applies, namely, $\frac{\tilde{\rho}_n - \rho}{\sigma/\sqrt{n}} \to N(0,1)$ as $n \to \infty$. Then we have

$$Pr\big[\big|\tilde{\rho}_n - \rho\big| \le \epsilon\rho\big] = Pr\big[\big|\frac{\sqrt{n}}{\sigma}(\tilde{\rho}_n - \rho)\big| \le \frac{\epsilon\rho\sqrt{n}}{\sigma}\big] \approx 2\Phi\big(\frac{\epsilon\rho\sqrt{n}}{\sigma}\big) - 1,$$

when n is large and ϵ is small enough so that $\epsilon\sqrt{n}$ is not too large. Let z_p be the unique constant such that $\Phi(z_p) = \frac{1+p}{2}$. Set $2\Phi\big(\frac{\epsilon\rho\sqrt{n}}{\sigma}\big) - 1 = p$, we get

$$n^* = \frac{z_p^2\sigma^2}{\epsilon^2\rho^2}, \tag{1}$$

which means that by taking n^* samples, the estimator ρ_{n^*} can estimate ρ within $\epsilon\rho$ with probability p. However, Equation (1) cannot be directly used in practice to determine the required number of samples in ahead, since both ρ and σ^2 are unknown. Nonetheless, since we assume that $\tilde{\rho}_n$ and S_n^2 are *strongly consistent* for ρ and σ^2, it is reasonable to use $\tilde{\rho}_n$ and S_n^2 to approximate ρ and σ^2 in Equation 1. This leads to the stopping rule

$$N(\epsilon) = \inf\{n \ge 1 : S_n^2 > 0 \ \texttt{and} \ \epsilon\tilde{\rho}_n \ge z_p S_n/\sqrt{n}\}.$$

Definition 4 (Asymptotical Efficiency)**.** *An estimator is* asymptotically efficient *if its expected number of sampling steps* $E\big[N(\epsilon)\big]$ *satisfies*

$$\lim_{\epsilon\to 0} \epsilon^2 E\big[N(\epsilon)\big] = z_p^2\sigma^2/\rho^2.$$

In other words, an estimator is asymptotically efficient if the expected number of sampling steps when ϵ is small is approximately equal to the optimal number of sampling steps n^* given by Equation (1).

A nice property of the estimator $\tilde{\rho}_q$ is that, under certain assumptions (such as the strong consistency of $\tilde{\rho}_n$ and S_n^2), it is both asymptotically consistent and efficient (see Theorem 3 in [10]). The estimator is asymptotically consistent, so the confidence is close to p when the estimation error ϵ is small. The estimator is asymptotically efficient, so on average the estimator will stop with the theoretically minimum number of required samples.

3.2.3 The Algorithm

Based on the previous discussion, the sequential sampling algorithm is then straightforward, as described in Algorithm 2.

Algorithm 2: Sequential Sampling to Estimate Query Size

Input: q, a query; $0 \le p < 1$, confidence on the estimation
Output: ρ_n, estimated size of q

$M \leftarrow 0$;
$S^2 \leftarrow 0$;
$n \leftarrow 1$;
repeat
  $S^2 \leftarrow S^2 + (M - (n-1)\rho_n)^2/(n(n-1))$;
  $M \leftarrow M + \rho_n$;
  $\rho_n \leftarrow M/n$;
  $S_n^2 \leftarrow S^2/(n-1)$;
until $S_n^2 > 0$ `and` $\epsilon\rho_n \ge z_p S_n/\sqrt{n}$;
return ρ_n;

Here, we use the numerically stable recurrence formula to compute S_n^2:

$$(n-1)S_n^2 = (n-2)S_{n-1}^2 + \frac{\big(\sum_{i=1}^{n-1}\rho_i - (n-1)\rho_n\big)^2}{n(n-1)}.$$

3.2.4 Remarks

The sequential sampling procedure shown in Algorithm 2 takes samples from each relation uniformly and independently (called *independent sampling*). Therefore, after n steps, we have n observations in total, with each observation containing one block from each underlying relation. In [10], the authors further discussed another alternative called *cross-product* sampling. The idea is that, at the i-the step, assuming the K blocks taken from the K relations are B_{i_1}, ..., B_{i_K}, we can actually join each B_{i_j} with each $B_{l_{j'}}$ such that $1 \leq l \leq i$ and $j \neq j'$ (note that in the case of independent sampling, we only join among the blocks with $l = i$). In this way, we can obtain n^K observations after n steps, and the estimator by averaging the selectivity over these n^K samples is still unbiased. However, since these samples are no longer independent, we need more sophisticated techniques to estimate the sample variance S_n^2. It is also shown that cross-product sampling is always better than independent sampling due to its lower sample variance. Interested readers can see [10] for more details. For a query plan with a fixed join order, the size of each subquery can be estimated simultaneously based on the samples taken so far. This is better than the case of adaptive sampling, where the size of each subquery has to be estimated by calling the sampling procedure again.

4 Projection Queries

Estimating the cardinality of projection queries is very different from that for selection and join queries discussed in the previous section. Projection will remove duplicates from the results, and hence the problem is equivalent to estimating the number of *distinct values* in the answer. This is a well-studied problem called *estimating the number of species* in statistical literature (see [18] for a survey).

Suppose that a population is partitioned into C classes. Unlike most of the estimation problems whose focus is the relative sizes of the classes (including the problem of cardinality estimation for selection and join queries, as mentioned in the previous section), the estimation here is on the C itself. This problem is quite difficult, in the sense that it is quite resistant to statistical solution, essentially because no matter how many classes have been observed, there may still be a large number of very small unobserved classes.

There is only limited work on sampling-based distinct-values estimation in database literature [19, 11]. In this section, we focus our discussion on the state-of-the-art Guaranteed-Error Estimator (GEE) as introduced in [11].

4.1 The Estimator

Let n be the number of rows in the relation R. Consider a specific column of R. Suppose there are D distinct values in this column, w.l.o.g. say, $\{1, 2, ..., D\}$. The estimator is based on examining a random sample of r tuples chosen uniformly at random from the relation. Let d be the number of distinct values in the sample, and let f_i be the number of values that occur exactly i times in the sample. Then $d = \sum_{i=1}^{r} f_i$. The Guaranteed-Error Estimator is defined as:

$$\widehat{D} = \sqrt{\frac{n}{r}} f_1 + \sum_{i=2}^{r} f_i. \tag{2}$$

The intuition behind GEE is as follows. The set from which the samples come consists of high frequency and low frequency values. When the sample size is reasonable, high frequency values will be picked up almost surely. Although *high* and *low* here are relative terms, we can think of the values that appear more than once in the random sample as being the values of high frequency, and think of the singleton values in the sample as the low frequency values. For high frequency values, we need to count them only once. This interprets the $\sum_{i=2}^{r} f_i$ part of Equation (2). For low frequency values, since the random sample contains only some of them, we need to scale up the count we observed. It is easy to see that the actual number of low frequency values should be expected between f_1 and $\frac{n}{r} f_1$. GEE then chooses to use the geometric mean of these two bounds as the scaling factor, which interprets the $\sqrt{\frac{n}{r}} f_1$ part of Equation (2).

4.2 Properties

We use the *ratio error* defined as

$$err(\widehat{D}) = \max(\frac{D}{\widehat{D}}, \frac{\widehat{D}}{D}) \tag{3}$$

as our error metric. We first show an upper bound of the estimation error of GEE. We then show that it is actually the best we can do in the worst case. Therefore, GEE has the optimality in terms of worst-case estimation errors.

Theorem 5. *The expected ratio error of GEE is $O\big(\sqrt{\frac{n}{r}}\big)$ when it samples r values from any possible input of size n.*

Proof. [3] Let n_i be the number of times value i occurs in the column where we sample from, and let $p_i = \frac{n_i}{n}$. We then have $\sum_{i=1}^{D} n_i = n$ and $\sum_{i=1}^{D} p_i = 1$. The probability x_i that a particular value i does not appear in the r samples is then $x_i = 1 - (1-p_i)^r$. In particular, the probability y_i that i appears exactly once in the r samples is $y_i = rp_i(1-p_i)^{r-1}$. The number of distinct values d in a random sample of size r is then a random variable with expectation $E\big[d\big] = \sum_{i=1}^{D} x_i \cdot 1 = \sum_{i=1}^{D} x_i$. In particular, the number of singleton values f_1 is a random variable with expectation $E\big[f_1\big] = \sum_{i=1}^{D} y_i \cdot 1 = \sum_{i=1}^{D} y_i$. Therefore, the expected value returned by GEE is

$$E\big[GEE\big] = E\big[d + \big(\sqrt{\frac{n}{r}} - 1\big)f_1\big] = E\big[d\big] + \big(\sqrt{\frac{n}{r}} - 1\big)E\big[f_1\big] = \sum_{i=1}^{D} \big(x_i + \big(\sqrt{\frac{n}{r}} - 1\big)y_i\big).$$

Our next goal is to show that $x_i + \big(\sqrt{\frac{n}{r}} - 1\big)y_i$ is between $c_1\sqrt{\frac{r}{n}}$ and $c_2\sqrt{\frac{n}{r}}$ for some constants c_1 and c_2. If this holds, then $E\big[GEE\big]$ is between $D \cdot c_1\sqrt{\frac{r}{n}}$ and $D \cdot c_2\sqrt{\frac{n}{r}}$, which results in

$$err(GEE) = \max(\frac{E[GEE]}{D}, \frac{D}{E[GEE]}) = O(\sqrt{\frac{n}{r}}),$$

and the theorem is proved.

To show that $x_i + \big(\sqrt{\frac{n}{r}} - 1\big)y_i$ is between $c_1\sqrt{\frac{r}{n}}$ and $c_2\sqrt{\frac{n}{r}}$, consider two cases:

1. $p_i \geq \frac{1}{r}$. In this case,

   $$x_i = 1 - (1-p_i)^r \geq 1 - (1 - \frac{1}{r})^r.$$

   Since $(1 - \frac{1}{r})^r \leq e^{-1}$ (note that $1 + x \leq e^x$ for $x \in \mathcal{R}$), we have $x_i \geq 1 - e^{-1}$. Clearly $x_i \geq 1$, so $1 - e^{-1} \leq x_i \leq 1$. On the other hand, we have $0 \leq y_i \leq 1$. Hence, we have

   $$1 - e^{-1} \leq x_i + \big(\sqrt{\frac{n}{r}} - 1\big)y_i \leq \sqrt{\frac{n}{r}}.$$

   We can further pick $c_1^{(1)} = 1 - e^{-1}$ (we have $c_1^{(1)}\sqrt{\frac{r}{n}} \leq c_1^{(1)} = 1 - e^{-1}$ since $r \leq n$) and $c_2^{(1)} = 1$ so that

   $$c_1^{(1)}\sqrt{\frac{r}{n}} \leq x_i + \big(\sqrt{\frac{n}{r}} - 1\big)y_i \leq c_2^{(1)}\sqrt{\frac{n}{r}}.$$

2. $p_i < \frac{1}{r}$. In this case, we have $0 \leq x_i \leq 1$. On the other hand, note that $p_i \geq \frac{1}{n}$. Hence

   $$y_i = rp_i(1-p_i)^{r-1} \geq \frac{r}{n}(1 - \frac{1}{r})^{r-1}.$$

[3] This is adapted from the proof of Theorem 2 in [11] with some revisions. The original proof has some flaws in its **Case 1** for $p_i \geq \frac{1}{r}$ and **Case 2** for $p_i < \frac{1}{r}$.

Let $f(r) = (1 - \frac{1}{r})^{r-1}$. We have

$$f'(r) = \frac{r-1}{r^2}\big(1 - \frac{1}{r}\big)^{r-2} \geq 0$$

given that $r \geq 1$. Hence $f(r)$ is nondecreasing, and

$$y_i \geq \frac{r}{n} f(r) \geq \frac{r}{n} f(2)$$

when $r \geq 2$. Thus $y_i \geq \frac{r}{2n}$ when $r \geq 2$. Since $y_i \leq 1$, we obtain

$$\frac{r}{2n} \leq y_i \leq 1$$

when $r \geq 2$. Therefore

$$\frac{r}{2n}\big(\sqrt{\frac{n}{r}} - 1\big) \leq x_i + \big(\sqrt{\frac{n}{r}} - 1\big) y_i \leq \sqrt{\frac{n}{r}}$$

when $r \geq 2$, which gives

$$\frac{1}{2}\sqrt{\frac{r}{n}}\big(1 - \sqrt{\frac{r}{n}}\big) \leq x_i + \big(\sqrt{\frac{n}{r}} - 1\big) y_i \leq \sqrt{\frac{n}{r}}.$$

We can then pick $c_1^{(2)} = \frac{1}{2}\big(1 - \sqrt{\frac{r}{n}}\big)$ and $c_2^{(2)} = 1$ so that

$$c_1^{(2)} \sqrt{\frac{r}{n}} \leq x_i + \big(\sqrt{\frac{n}{r}} - 1\big) y_i \leq c_2^{(2)} \sqrt{\frac{n}{r}}.$$

When $r = 1$, we have $x_i = y_i = p_i$, and hence

$$x_i + \big(\sqrt{\frac{n}{r}} - 1\big) y_i = p_i \sqrt{\frac{n}{r}}.$$

Since $\frac{1}{n} \leq p_i \leq 1$, we obtain

$$\frac{1}{r}\sqrt{\frac{r}{n}} \leq x_i + \big(\sqrt{\frac{n}{r}} - 1\big) y_i \leq \sqrt{\frac{n}{r}}.$$

Therefore, we can pick $c_1^{(3)} = \frac{1}{r}$ and $c_2^{(3)} = 1$ so that

$$c_1^{(3)} \sqrt{\frac{r}{n}} \leq x_i + \big(\sqrt{\frac{n}{r}} - 1\big) y_i \leq c_2^{(3)} \sqrt{\frac{n}{r}}$$

when $r = 1$.

Picking $c_1 = \min(c_1^{(1)}, c_1^{(2)}, c_1^{(3)})$ and $c_2 = \max(c_2^{(1)}, c_2^{(2)}, c_2^{(3)})$ then completes the proof. □

Theorem 6. *Consider any estimator that examines at most r values from any possible input of size n. For any $\gamma > e^{-r}$, there is a choice of input data such that with probability at least γ,*

$$err(\widehat{D}) \geq \sqrt{\frac{n-r}{2r} \ln \frac{1}{\gamma}}.$$

Proof. (Sketch) The proof is constructive. We consider two scenarios:

1. The column contains only one distinct value x.
2. The column contains $k+1$ distinct values $\{x, y_1, ..., y_k\}$ of which x appears $n-k$ times and each y_i appears only once. What's more, the y_i's are placed in k rows that are uniformly chosen at random.

Let X be the random variable that denotes the value sampled from the column. Now suppose that in the second scenario, we take r samples X_1, X_2, ..., X_r. If $X_1 = X_2 = \cdots X_r = x$, then the estimator has no idea whether it is in the first or second scenario. Let this event be $\mathcal{E}$. Our goal is to bound the probability $Pr[\mathcal{E}]$:

$$\begin{aligned} Pr[\mathcal{E}] &= Pr[X_1 = x, X_2 = x, \cdots, X_r = x] = \prod_{i=1}^{r} Pr[X_i = x | X_1 = x, \ldots \cdots X_{i-1} = x] \\ &= \prod_{i=1}^{r} \frac{[n-(i-1)]-k}{n-(i-1)} \geq \big(\frac{n-r-k}{n-r}\big)^r = \big(1 - \frac{k}{n-r}\big)^r \geq e^{-\frac{2kr}{n-r}}. \end{aligned} \quad (4)$$

The last step follows from the fact that $1 - z \geq e^{-2z}$ for $0 \leq z \leq \frac{1}{2}$, which is easy to verify. Now for $\gamma \geq e^{-r}$, we pick

$$k = \frac{n-r}{2r} \ln \frac{1}{\gamma}.$$

Since

$$k \leq \frac{n-r}{2r} \cdot \ln e^r = \frac{n-r}{2},$$

this choice of k satisfies both $\frac{k}{n-r} \leq \frac{1}{2}$ (as required by the last inequality in Equation (4)) and $k + 1 \leq n$ (as required by the second scenario).

This means, with the particular choice of k, in the second scenario, the probability that E occurs can be at least γ, for which the estimator cannot distinguish the two scenarios. Now let the value returned by the estimator be v ($1 \leq v \leq k$). Then the error w.r.t. the first scenario is v, while the error w.r.t. the second scenario is $\frac{k}{v}$. If $v \geq \sqrt{k}$, then the error w.r.t. the first scenario is at least $\sqrt{k}$, while if $v \leq \sqrt{k}$, then the error w.r.t. the second scenario is at least $\sqrt{k}$. Therefore, no matter which case the estimator is in, it incurs an estimation error at least $\sqrt{k}$. This completes the proof of the theorem (see Theorem 1 of [11] if more details are desired). □

4.3 Remarks

Different from the estimators for selection and join queries as discussed in Section 3, where those estimators have nice properties such as asymptotical consistency and efficiency, the GEE estimator introduced in this section for projection queries is rather frustrating. The fact that GEE is worst-case optimal provides no optimistic reason for practitioners, since the worst-case error given by Theorem 6 is quite big unless the number of samples r is proportional to the number of rows n in the table. This is prohibitively expensive for a sampling-based algorithm to be useful in practice. Modern query optimizers hence use simple heuristics instead of sampling when they need to estimate the number of distinct values as in projection queries. The GEE estimator, however, has been used in recent work of query progress monitoring [3]. The overhead in this scenario is not important since the samples are taken online when the query is running.

5 Experimental Evaluation

In this section, we evaluate the estimators discussed in previous sections on the standard TPC-H benchmark[4]. The TPC-H benchmark is dedicated to testing the performance of a DBMS by simulating OLAP (short for Online Analytical Processing) workloads. OLAP queries are usually quite complicated in syntax and long-running, so it is important to find good query plans, for which cardinality estimation is the key. We report our results from this empirical study, with a focus on two aspects: 1) the *effectiveness* of the estimator, in terms of the estimation accuracy; and 2) the *efficiency* of the estimator, in terms of the time spent on estimation compared with the time if the query is actually run. To the best of our knowledge, such evaluation on a modern benchmark for sampling-based cardinality estimation algorithms is never done before.

[4] http://www.tpc.org/tpch/

5.1 Experimental Settings

We use the 1GB TPC-H database in our experiments. The experiments are conducted on a PC with 2.27GHz Intel CPU and 2GB memory, and we run PostgreSQL 9.0.4 on Linux Kernel 2.6.18. The samples are taken offline and materialized as additional relations. Since the original TPC-H database generator uses uniform distributions, to test the robustness of the estimators, we also use a skewed TPC-H database generator[5]. This database generator populates a TPC-H database using a Zipf distribution. This distribution has a parameter z that controls the degree of skewness. $z = 0$ generates a uniform distribution, and as z increases, the data becomes more and more skewed. We create a skewed 1GB database generated using $z = 1$. We also test the efficiency and effectiveness of the algorithms with different number of samples. In the results reported following, we use the ratio error defined in Equation (3) as the accuracy metric. Currently, all the algorithms are implemented outside the database system with pure SQL queries. Hence the efficiency results reported can be treated as the worst-case performance, since an inside-DBMS implementation will be more efficient.

5.2 Results for Selection/Join Estimators

We test the selection/join estimators with 7 queries containing only selection and join operators (see Appendix A for the list of queries written in SQL). These queries are subqueries from the standard TPC-H queries.

5.2.1 Results of Adaptive Sampling

Table 2 and 3 show the results of the adaptive sampling algorithm (Algorithm 1). Here, err_o is the ratio error of the estimation given by the PostgreSQL optimizer, based on one-dimensional histograms; err_s is the ratio error of the estimation given by the adaptive sampling algorithm; T_s is the time spent on sampling; and T is the time of running the query itself. To test the the effect of the number of samples taken, we vary p in 0.8, 0.9, 0.95, and 0.99, for which the sampling algorithm will take 170, 270, 380, and 670 samples, respectively.

		$p = 0.8$		$p = 0.9$		$p = 0.95$		$p = 0.99$	
Q	err_o	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T
SJ1	1.02	1.08	0.14	1.16	0.05	1.10	0.06	1.05	0.08
SJ2	1.74	1.04	1.10	1.02	0.83	1.11	0.68	1.21	0.65
SJ3	1.01	1.00	8.96	1.26	4.13	1.07	4.21	1.25	8.43
SJ4	2067.41	1.51	0.06	1.22	0.14	1.16	0.18	1.01	0.20
SJ5	1.70	1.14	0.13	1.13	0.26	1.02	0.21	1.03	0.21
SJ6	1.04	1.03	2.40	1.01	1.76	1.06	1.14	1.05	1.19
SJ7	2.08	1.12	0.61	1.10	0.61	1.30	0.66	1.11	0.61

Table 2: Adaptive sampling on uniform data

		$p = 0.8$		$p = 0.9$		$p = 0.95$		$p = 0.99$	
Q	err_o	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T
SJ1	1.17	1.17	0.14	1.02	0.09	1.09	0.10	1.03	0.17
SJ2	1.71	1.25	6.58	1.12	0.96	1.08	0.71	1.02	0.63
SJ3	1.28	1.05	12.75	1.43	4.18	1.42	4.92	1.20	8.50
SJ4	237.55	1.59	0.07	1.64	0.25	1.10	0.29	1.32	0.37
SJ5	1.71	1.02	0.13	1.18	0.24	1.04	0.19	1.07	0.19
SJ6	1.06	1.04	3.74	2.36	3.83	1.56	2.59	2.31	1.66
SJ7	2.15	1.17	0.61	1.14	0.54	1.01	0.54	1.07	0.52

Table 3: Adaptive sampling on skewed data

We have several observations. First, in the case of uniform data, the estimations based on sampling are consistently good for all the queries tested, while the estimations based on histograms are good for some queries but bad for the others (this is also true in the skewed case). Second, in the case

[5] ftp://ftp.research.microsoft.com/users/viveknar/TPCDSkew/

of skewed data, the estimation based on sampling is worse than that on the uniform data, which is consistent with our discussion in Section 3.1 that data skewness is a general problem for any sampling-based estimators (ref. Example 3). Third, the efficiency based on the current implementation is not so good. For some queries, it takes even longer for sampling than running the original query.

5.2.2 Results of Sequential Sampling

Table 4 and 5 summarize the results of the sequential sampling algorithm (Algorithm 2). However, instead of sampling enough until the specified estimation confidence is reached, in this experiment, we simply vary the number of the samples taken to see how accurate the estimation can be based on a certain amount of samples. The rational is that, in practice we have limited budget on the time that can be spent on sampling. Due to the nice property of asymptotical consistency of sequential sampling, we are able to make a tradeoff between the number of samples we can afford and the estimation accuracy we can achieve. In the results shown below, f stands for the percentage of samples we take from each relation in the database.

		$f = 0.05$		$f = 0.1$		$f = 0.2$		$f = 0.3$		$f = 0.4$	
Q	err_o	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T
SJ1	1.02	0.99	0.01	1.01	0.01	1.04	0.04	1.06	0.07	1.02	0.12
SJ2	1.76	0.97	0.06	0.97	0.11	0.99	0.20	1.01	0.31	0.99	0.39
SJ3	1.01	1.01	0.05	1.00	0.11	1.00	0.21	0.99	0.29	0.99	0.37
SJ4	2677.46	9.80	0.00	1.84	0.00	0.85	0.01	1.06	0.03	1.02	0.04
SJ5	1.92	1.39	0.01	0.92	0.03	0.98	0.06	1.05	0.12	1.02	0.17
SJ6	1.03	0.98	0.16	1.05	0.19	0.97	0.24	1.03	0.29	1.00	0.39
SJ7	2.23	0.98	0.05	0.98	0.09	1.00	0.20	1.02	0.26	0.99	0.40

Table 4: Sequential sampling on uniform data

		$f = 0.05$		$f = 0.1$		$f = 0.2$		$f = 0.3$		$f = 0.4$	
Q	err_o	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T
SJ1	1.17	1.00	0.01	1.07	0.03	1.03	0.07	1.00	0.11	1.03	0.17
SJ2	1.70	1.02	0.07	1.05	0.12	1.04	0.21	1.02	0.31	1.01	0.39
SJ3	1.28	1.01	0.06	1.01	0.12	1.02	0.22	1.00	0.33	1.00	0.43
SJ4	180.68	N/A	N/A	N/A	N/A	1.49	0.02	1.44	0.03	1.48	0.05
SJ5	1.68	1.04	0.02	1.09	0.04	1.01	0.08	1.01	0.13	1.06	0.19
SJ6	1.09	1.02	0.06	1.30	0.13	1.09	0.23	1.09	0.33	1.14	0.42
SJ7	2.20	1.04	0.05	1.02	0.11	1.04	0.21	1.02	0.30	1.03	0.40

Table 5: Sequential sampling on skewed data

Not surprisingly, usually the more samples we take, the better the estimation is, and the more time we need to spend on sampling. This is true for both the uniform and the skewed case. However, an advantage of sequential sampling over adaptive sampling is that we can control the number of samples to take. For the particular setting here, taking 20% samples from each relation is sufficient to give very accurate estimation to all the queries. This usually incurs less than 20% additional time attributing to sampling, which is much better than that in adaptive sampling and may be affordable in practice. We also notice that skewness remains an issue. As denoted in Table 3, the estimator fails to give reasonable estimation[6] on the query SJ4 in the skewed case when less than 20% samples are available.

5.3 Results for Projection Estimators

We test the GEE estimator with 7 queries that merely count the number of distinct values in a specific column (see Appendix B for the list of queries written in SQL). Table 6 and 7 present our results of the GEE estimator in the case of uniform and skewed data, respectively. Here, we directly compare the estimation from GEE with the actual size of the query, without comparing it with other approaches.

[6]It estimates the size to be 0 so that the ratio error goes to infinity.

	$f = 0.05$		$f = 0.1$		$f = 0.2$		$f = 0.3$		$f = 0.4$	
Q	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T
P1	1.92	0.09	1.28	0.21	1.02	0.44	1.00	0.69	1.00	0.83
P2	1.00	0.02	1.00	0.04	1.00	0.06	1.00	0.09	1.00	0.12
P3	1.62	0.18	1.40	0.31	1.13	0.41	1.03	0.80	1.00	1.15
P4	1.00	0.04	1.00	0.07	1.00	0.14	1.00	0.20	1.00	0.27
P5	4.40	0.17	3.25	0.28	2.49	0.64	2.27	0.93	2.17	1.07
P6	4.62	0.72	3.58	2.74	2.58	0.83	2.35	1.13	2.24	1.49
P7	1.00	0.03	1.00	0.05	1.00	0.09	1.00	0.13	1.00	0.16

Table 6: GEE on uniform data

	$f = 0.05$		$f = 0.1$		$f = 0.2$		$f = 0.3$		$f = 0.4$	
Q	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T	err_s	T_s/T
P1	1.39	0.14	1.29	0.12	1.12	0.27	1.03	0.36	1.01	0.46
P2	1.00	0.02	1.00	0.04	1.00	0.06	1.00	0.09	1.00	0.12
P3	1.67	0.18	1.46	0.31	1.14	0.39	1.03	0.77	1.01	1.12
P4	1.00	0.04	1.00	0.07	1.00	0.14	1.00	0.20	1.00	0.28
P5	3.27	0.18	2.68	0.14	2.18	0.27	2.07	0.33	1.99	0.40
P6	3.42	0.63	2.69	0.80	2.15	2.80	2.12	1.56	2.10	1.02
P7	1.00	0.03	1.00	0.05	1.00	0.09	1.00	0.13	1.00	0.17

Table 7: GEE on skewed data

While the estimation accuracy improves as more samples are taken, it seems that there are some queries whose sizes are inherently difficult to estimate. For instance, the ratio error is still around 2 for the queries P5 and P6 even if the estimator has already examined 40% of the data. On the other hand, skewness does not affect the performance much. In fact, for the particular case here, the estimation is even better on the skewed data. This may be explained as follows. Suppose we have two distributions with the same support. Then skewness only changes the probability mass associated with each sample point, which may not significantly change the proportion of *frequent* and *infrequent* (or, singleton) values observed in a random sample. Since the accuracy of the GEE estimator heavily relies on the frequency of singleton values, the accuracy may not vary much as long as this proportion remains similar when data becomes skewed. Nonetheless, a formal analysis on the impact of data skewness seems to be an interesting future work.

6 Conclusion

In this survey, we studied several important sampling-based cardinality estimation algorithms in database literature. Due to different nature of the estimation problems, these estimation algorithms can fall into two categories. One is for estimating the size of selection/join queries, and sampling is an effective technique for this purpose. The other is for estimating the size of projection queries, or more basically, estimating the number of distinct values in a set, and sampling has inherent difficulty to make accurate estimations in this case, unless a significant number of samples are taken. We described the details of each algorithm covered in this survey, with an focus on their theoretical properties. We also provided an empirical study of their performance on a modern database benchmark, in terms of both effectiveness and efficiency. To the best of our knowledge, this kind of evaluation is never done before. We hence hope that the results reported in this paper can provide further insight motivating future research in this old but still active field.

References

[1] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online aggregation," in *SIGMOD Conference*, 1997, pp. 171–182.

[2] C. M. Jermaine, S. Arumugam, A. Pol, and A. Dobra, "Scalable approximate query processing with the dbo engine," in *SIGMOD Conference*, 2007, pp. 725–736.

[3] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou, "The researcher's guide to the data deluge: Querying a scientific database in just a few seconds," *PVLDB*, vol. 4, no. 12, pp. 1474–1477, 2011.

[4] S. Chaudhuri, V. R. Narasayya, and R. Ramamurthy, "Estimating progress of execution for SQL queries," in *SIGMOD*, 2004.

[5] G. Luo, J. F. Naughton, C. J. Ellmann, and M. Watzke, "Toward a progress indicator for database queries," in *SIGMOD*, 2004.

[6] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *ICDE*, 2009.

[7] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. Zdonik, "Learning-based query performance modeling and prediction," in *ICDE*, 2012.

[8] Y. E. Ioannidis, "The history of histograms (abridged)," in *VLDB*, 2003, pp. 19–30.

[9] R. J. Lipton, J. F. Naughton, and D. A. Schneider, "Practical selectivity estimation through adaptive sampling," in *SIGMOD*, 1990.

[10] 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.*, vol. 52, no. 3, pp. 550–569, 1996.

[11] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya, "Towards estimation error guarantees for distinct values," in *PODS*, 2000, pp. 268–279.

[12] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[13] S. Ross, *A First Course in Probability*, 8th ed. Prentice Hall, 2009.

[14] R. J. Lipton and J. F. Naughton, "Query size estimation by adaptive sampling," in *PODS*, 1990.

[15] P. J. Haas and A. N. Swami, "Sequential sampling procedures for query size estimation," in *SIGMOD Conference*, 1992, pp. 341–350.

[16] R. J. Lipton, J. F. Naughton, and D. A. Schneider, "Practical selectivity estimation through adaptive sampling," University of Wisconsin-Madison Computer Sciences Department, Tech. Rep., 1990.

[17] S. Chaudhuri, R. Motwani, and V. R. Narasayya, "On random sampling over joins," in *SIGMOD Conference*, 1999, pp. 263–274.

[18] J. Bunge and M. Fitzpatrick, "Estimating the number of species," *Journal of the American Statistical Association*, vol. 88, no. 421, pp. 364–373, 1993.

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

A SQL Queries for Testing Selection/Join Estimators

(SJ1)
```
SELECT *
FROM supplier, partsupp, nation, region
WHERE s_suppkey = ps_suppkey
    AND s_nationkey = n_nationkey
    AND n_regionkey = r_regionkey
    AND r_name = 'AFRICA';
```

(SJ2)
```
SELECT *
FROM customer, orders
WHERE c_custkey = o_custkey
    AND c_mktsegment = 'BUILDING'
    AND o_orderdate < date '1995-03-01';
```

(SJ3)
```
SELECT *
FROM customer, nation, region
WHERE c_nationkey = n_nationkey
    AND n_regionkey = r_regionkey
    AND r_name = 'EUROPE';
```

(SJ4)
```
SELECT *
FROM part, supplier, lineitem, partsupp, orders, nation
WHERE s_suppkey = l_suppkey
    AND ps_suppkey = l_suppkey
    AND ps_partkey = l_partkey
    AND p_partkey = l_partkey
    AND o_orderkey = l_orderkey
    AND s_nationkey = n_nationkey
    AND p_name like '%lime%';
```

(SJ5)
```
SELECT *
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND o_orderdate >= date '1993-07-01'
    AND o_orderdate < date '1993-07-01' + interval '3 month'
    AND l_returnflag = 'R'
    AND c_nationkey = n_nationkey;
```

(SJ6)
```
SELECT *
FROM lineitem, part
WHERE l_partkey = p_partkey
    AND l_shipdate >= date '1993-10-01'
    AND l_shipdate < date '1993-10-01' + interval '1 month';
```

(SJ7)
```
SELECT *
FROM partsupp, part
WHERE p_partkey = ps_partkey
    AND p_brand <> 'Brand#51'
    AND p_type NOT LIKE 'SMALL BURNISHED%'
    AND p_size IN (9, 37, 1, 31, 6, 19, 45, 29)
    AND ps_suppkey NOT IN (
        SELECT s_suppkey
        FROM supplier
        WHERE s_comment LIKE '%Customer%Complaints%');
```

B SQL Queries for Testing Projection Estimators

(P1) `SELECT COUNT(DISTINCT l_partkey) FROM lineitem;`

(P2) `SELECT COUNT(DISTINCT l_shipdate) FROM lineitem;`

(P3) `SELECT COUNT(DISTINCT o_custkey) FROM orders;`

(P4) `SELECT COUNT(DISTINCT o_shippriority) FROM orders;`

(P5) `SELECT COUNT(DISTINCT c_acctbal) FROM customer;`

(P6) `SELECT COUNT(DISTINCT s_acctbal) FROM supplier;`

(P7) `SELECT COUNT(DISTINCT p_size) FROM part;`