Improved Correlated Sampling for Join Size Estimation

TaiNing Wang
National University of Singapore
taining_wang@u.nus.edu

Chee-Yong Chan
National University of Singapore
chancy@comp.nus.edu.sg

***Abstract*—Recent research on sampling-based join size estimation has focused on a promising new technique known as correlated sampling. While several variants of this technique have been proposed, there is a lack of a systematic study of this family of techniques. In this paper, we first introduce a framework to characterize its design space in terms of five parameters. Based on this framework, we propose a new correlated sampling based technique to address the limitations of existing techniques. Our new technique is based on using a discrete learning method for estimating the join size from samples. We experimentally compare the performance of multiple variants of our new technique and identify a hybrid variant that provides the best estimation quality. This hybrid variant not only outperforms the state-of-the-art correlated sampling technique, but it is also more robust to small samples and skewed data.**

***Index Terms*—query processing, database systems, sampling methods**

I. Introduction

Estimating query result size plays an important role in cost-based query optimization. In particular, estimating join size is considerably harder than estimating result size of single-table queries. There have been two main approaches for join estimation, namely, histogram-based approaches [2], [3], [5]–[7], [13], [18], [22]–[24] (including sketches and wavelets which can be viewed as compressed histograms [14], [18], [30]) and sampling-based approaches [1], [4], [11], [28], [30]. Histogram-based approaches generally do not scale well with the number of attributes in selection and join predicates. The storage space grows dramatically and computation over multi-dimensional histograms becomes very complex [19], [30]. Moreover, it is difficult to estimate the selectivity of general selection predicates (e.g., LIKE predicates) using histograms.

Sampling-based approaches do not have scalability issues with the number of attributes involved and they could support various selection predicates. However, using the straightforward approach based on independent table samples often does not have good estimation quality because it ignores the join relationship between tables [4]. In recent years, a new sampling-based approach for join estimation, named *correlated sampling*, has been proposed [4], [28], [30]. Given a join graph representing $A \bowtie B$, the correlated sampling proposed in [30] samples both A and B in an offline phase: a random sample S_A is drawn with some sampling probability p from table A, and a subset of the tuples in the semijoin of B with S_A (i.e., $B \ltimes S_A$) forms the sample S_B from table B. Correlated sampling thus ensures that each tuple from S_B joins with some tuple in S_A. At runtime, given a query $\sigma_c(A \bowtie B)$ on the join graph, the query's join size is estimated by scaling up the sampled join size $|S''_A \bowtie S''_B|$ by $1/p$, where $S''_A \subseteq S_A$ and $S''_B \subseteq S_B$ denote the subsets of sampled records that satisfy the selection condition c. We refer to S''_A and S''_B as *filtered samples*.

Several variants of correlated sampling have been developed by both industry and academia, and they have been shown to have good estimation quality compared to independent sampling of base tables [4], [28]. The start-of-the-art variant is *two-level correlated sampling* [4] (denoted as `CS2L` in this paper) which is an unbiased estimator with the sampling probabilities optimized to minimize estimation variance. However, our experimental results demonstrate that the estimation variance of `CS2L` could still be high leading to poor estimation quality especially when the filtered samples are small or when the *join value density* (to be defined in Section III) is small. In addition, the optimization of the sampling probabilities to minimize the estimation variance is a complex problem that is not amenable to a closed form solution, and `CS2L` provides approximate solutions based on heavy hitters [4].

In this paper, we present a systematic study of correlated sampling techniques and make the following contributions. First, while there are currently three proposed variants of correlated sampling, the design options and tradeoffs for this new class of sampling-based techniques are not well understood. To fill this gap, we present a framework to characterize the design space of correlated sampling techniques (Section II). Our framework, which is based on five parameters, not only captures all the existing variants but more importantly uncovers many other possible variants in the design space that have yet to be explored.

Second, based on our proposed framework, we introduce a new class of correlated sampling variants, denoted as `CSDL` (Section IV). `CSDL` is the first approach that combines a biased estimation method (termed *discrete learning algorithm*, which is an advanced technique to learn a discrete distribution) [27] with correlated sampling for accurate join size estimation. This integration involves non-trivial techniques to enable the estimation method (which requires a simple random sample as input) to be applied to the samples from correlated sampling (which may not necessarily be simple random samples). Our work is the first to demonstrate that a correlation sampling

Notation	Meaning
data_size	size (byte size) of all base tables in a join graph
θ	the ratio of the size of the sample synopsis for a join graph to data_size
a_v, b_v	table frequencies of a value v, i.e. number of times v appear in table A, B
S_A, S_B	samples drawn for table A, B
S''_A, S''_B	filtered samples, obtained after applying query's selection predicates on S_A, S_B
$S_A(v), S_B(v)$	sample frequencies of v, i.e. number of times v appear in sample S_A, S_B
$s_A(v), s_B(v)$	sentry drawn for value v in table A and B
V_A, V_B	set of distinct join column values in tables A and B
$V_{A,B}$	set of distinct join column values that appear in both tables A and B; i.e., $V_{A,B} = V_A \cap V_B$
$V'_{A,B}$	set of sampled join column values that appear in both S_A and S_B
$V''_{A,B}$	set of sampled join column values that appear in both S''_A and S''_B
jvd	join value density for a join query defined as $\min\{\frac{\lvert V_A\rvert}{\lvert A\rvert}, \frac{\lvert V_B\rvert}{\lvert B\rvert}\}$

TABLE I: Notation

approach in combination with a biased estimation method can outperform traditional unbiased methods on estimation quality due to having lower estimation variance.

Third, we extend the estimation techniques for `CSDL` to handle not only two-table join queries but also two types of multi-table join query structures, namely, chain joins and star joins (Section V). Thus, our proposed `CSDL` can handle the same class of join queries as the state-of-the-art approach.

Fourth, we perform an experimental study to compare the estimation quality of our 10 proposed `CSDL` variants against the state-of-the-art approach `CS2L` using both real (JOB) and synthetic (TPC-H) benchmark datasets (Section VI). Based on our empirical results, we have identified an important data parameter, termed *join value density* (Section III), that is crucial for distinguishing the performance of the `CSDL` variants. Specifically, among our 10 `CSDL` variants, our experimental results have revealed two winning `CSDL` variants depending on whether the join value density is low or high. This pair of winning variants can be seen as a hybrid `CSDL` approach termed `CSDL-Opt`. Our experiments show that `CSDL-Opt` outperforms the state-of-the-art `CS2L` in both benchmark datasets. Specifically, for all queries with low join value density on JOB datasets, `CS2L` estimates the join size to be zero even though the true join size ranges from 10 to 1 million. In contrast, `CSDL-Opt` provides good estimates independent of the values of join value density.

II. Framework for Correlated Sampling

In this section, we propose a framework to characterize the design space of correlated sampling-based techniques for join size estimation for two-table equijoins represented by $A \bowtie B$. Table I summarizes the notations used in this paper.

All sampling-based techniques for join size estimation operate in two phases. In the offline sampling phase, samples of tables A and B, denoted by S_A and S_B, respectively, are created to form the sample synopsis for $A \bowtie B$. Correlated sampling requires that $S_B \subseteq B \ltimes S_A$. In the online estimation phase, given a query $\sigma_c(A \bowtie B)$, the join size of $A \bowtie B$ is estimated from the filtered samples, denoted by S''_A and S''_B, where $S''_A \subseteq S_A$ and $S''_B \subseteq S_B$ denote the subsets of sampled records that satisfy the query's selection condition c. For ease of presentation, we shall assume in this section that the run-time query has no selection predicate; i.e., $S''_A = S_A$ and $S''_B = S_B$.

A. Design Space Parameters

In this section, we shall explain the five parameters that are used in our framework with the help of a simple example of correlated sampling.

The first three parameters in our framework (denoted by p, q, and u) are used to specify the sampling probabilities used in the sampling phase to create S_A and S_B.

Parameters p and q. The first two parameters p and q are sampling probabilities used in the construction of S_A. First, from the set of distinct join column values in table A (denoted by V_A), we draw a sample $V \subseteq V_A$ where each value in V_A is sampled with a probability p. Next, S_A is created by sampling each tuple from table A that has a join column value in V with a probability of q.

Parameter u. The third parameter u is a sampling probability used in the construction of S_B. Similar to the construction of S_A, S_B is also created in two steps. First, we compute the set of tuples S given by the semijoin of B and S_A; i.e., $S = B \ltimes S_A$[1]. Tuples in S are also called *joinable tuples*. Next, S_B is created by sampling each tuple from S with a probability u. Since we are looking at equijoin queries, V contains all the join column values that appear in S_B.

Note that each of S_A and S_B is created using a two-step procedure that is referred to as *two-level sampling* [4].

Estimation Method Parameter. The fourth parameter is the *estimation method* which refers to the method for deriving the estimated join size $\hat{J}$ from the samples S_A and S_B. A well-known estimation method (which is used by all the existing correlated sampling techniques) is the *scaling up* technique which, as its name implies, simply scales up the join size of the samples by a factor defined in terms of the sampling probabilities as follows:

$$\hat{J} = \frac{1}{p} \sum_{v \in V_{A,B}} \frac{S_A(v)}{q} \frac{S_B(v)}{u} = \frac{1}{pqu} \sum_{v \in V_{A,B}} S_A(v) S_B(v) \quad (1)$$

Here, $V_{A,B}$ denote the set of join column values that appear in both A and B (i.e., $V_{A,B} = V_A \cap V_B$), and $S_A(v)$ and $S_B(v)$ denote the number of times that a join value v appears in S_A and S_B, respectively, (i.e., the sample frequencies of v). Thus, $\hat{J}$ is computed by scaling up the join size of the two samples (given by $\sum_{v \in V_{A,B}} S_A(v) S_B(v)$) by the factor $\frac{1}{pqu}$.

In general, it is possible to customize the sampling probabilities p, q, and u to be dependent on the specific join column

[1]Two approaches for computing $B \ltimes S_A$ in one table scan are discussed in our technical report [29].

value v being considered; i.e., we use sampling probabilities p_v, q_v, and u_v for the join value v. Then the estimated join size is computed as follows:

$$\hat{J} = \sum_{v \in V_{A,B}} \frac{1}{p_v} \frac{S_A(v)}{q_v} \frac{S_B(v)}{u_v} \tag{2}$$

Sentry parameter. This parameter indicates whether a technique called *sentry* is applied in the second-level sampling of S_A and S_B. It is first introduced in [4]. If the *sentry* technique is applied, it will ensure that at least one tuple is drawn for each sampled join value. Specifically, if the sentry technique is applied in the second-level sampling step to construct S_A, then for each join value $v \in V$, a tuple with join value v (denoted as $s_A(v)$) will be first drawn uniformly at random from table A. The remaining tuples in S_A will be sampled from A with sampling probability q_v. Similarly, if the sentry technique is applied in the second-level sampling step to construct S_B, then for each distinct join value v that exists in V, a tuple with join value v (denoted as $s_B(v)$) will be first drawn uniformly at random from the joinable tuples in B. The remaining tuples in S_B will be sampled from the joinable tuples with sampling probability u_v. We refer to tuples drawn using the sentry technique (i.e., $s_A(v)$ and $s_B(v)$) as sentries.

Having sentry avoids sampling 0 tuples for join values that appear infrequently in one of the tables. For example, let a_v and b_v denote the number of times the join value v appears in tables A and B, respectively. Assuming that for some v, a_v is large and b_v is small. Without sentry, we could get $S_B(v) = 0$ and $S_A(v) >> 0$, which causes the estimate of $a_v b_v$ to be $\frac{S_A(v)}{q_v}\frac{S_B(v)}{u_v} = 0$. In contrast, with sentry, we have at least one tuple $s_B(v)$ to join with S_A. The estimate of $a_v b_v$ will be $(\frac{S_A(v)}{q_v} + 1)(\frac{S_B(v)}{u_v} + 1)$ which is larger than 0.

With the sentry technique applied, the estimated join size is computed as follows:

$$\hat{J} = \sum_{v \in V_{A,B}} \frac{1}{p_v} (\frac{S_A(v)}{q_v} + 1)(\frac{S_B(v)}{u_v} + 1) \tag{3}$$

B. Capturing Existing Approaches

We now describe each of the three existing correlated sampling techniques using our proposed five-parameter framework for correlated sampling. A summary of their characterizations are shown in the first three rows in Table II. The space budget allocated for a join query's samples is given by $\theta \cdot$ *data_size*, where θ is some value less than 1 and *data_size* denotes the total size of the tables involved in the join query.

CS2: In the first proposed correlated sampling approach (denoted by `CS2`) [30], S_A is a simple random sample from table A with sampling probability θ, and $S_B = B \ltimes S_A$. For the construction of S_A, since all the join column values are considered and each tuple is drawn with a probability of θ, $p_v = 1$ and $q_v = \theta$, for all v. For the construction of S_B, since S_B is simply $B \ltimes S_A$, $u_v = 1$ for all v. `CS2` does not apply the sentry technique and the estimation method used is the simple scaling-up method. Although `CS2` generally has good estimation quality, its synopsis size can be unpredictably large if $B \ltimes S_A$ is large. Hence, `CS2` is not practical due to its large space requirement.

CSO: To address the large space overhead of `CS2`, a second variant of correlated sampling, denoted as `CSO`, was proposed [28]. In `CSO`, each join column value from V_A is sampled with probability θ; thus, $p_v = \theta$. All the tuples with these sampled join values in both A and B are included in the synopsis; thus, $q_v = u_v = 1$ for all v. Similar to `CS2`, `CSO` does not apply the sentry technique and the estimation method used is the simple scaling-up method. Since the join values are randomly sampled, the expected synopsis size for `CSO` is $\theta \cdot$ *data_size*. Since `CSO` has the property that $q_v = u_v = 1$ for all v, it means that for each join column value v, either all or none of the tuples with value v are included in the synopsis. This phenomenon is referred to by [4] as the "all or nothing" problem and it has been experimentally shown that this phenomenon leads to poor estimation quality due to high estimation variance.

CS2L: The state-of-the-art approach is the two-level correlated sampling approach [4], which we refer to as `CS2L`. `CS2L` introduces three key ideas. First, it incorporates the idea of end-biased sampling by using different p_v for each v, such that join values with higher frequencies are more likely to be sampled as they are likely to contribute more output tuples to the join result. Second, unlike `CSO` which sets both q_v and u_v to be 1 for all values of v, `CS2L` sets $q_v = u_v < 1$ to avoid the "all or nothing" problem in `CSO`. The values of p_v and q_v are optimized by minimizing the estimation variance. Third, `CS2L` introduces the sentry technique in the second-level sampling step to ensure that there is at least one sampled tuple for each sampled join value. `CS2L` has been demonstrated to have better estimation quality compared to `CSO` [4]. Similar to both `CS2` and `CSO`, `CS2L` uses the simple scaling-up technique for its estimation method.

Unlike `CS2`, where the size of its sample synopsis could be unpredictably large, both `CSO` and `CS2L` (as well as our new approach `CSDL` to be discussed in Section IV) could bound the sample synopsis size to a given allocated space budget (i.e., $\theta \cdot$ *data_size*).

III. Limitations of Existing Techniques

In this section, we discuss three limitations of existing correlated sampling techniques. These limitations will be addressed by our new approach to be described in Section IV.

Bias-variance tradeoff. First, all existing techniques are based on the unbiased scaling up technique for estimating join size. However, using an unbiased estimator does not necessarily guarantee low estimation errors as the estimation error comprises of both the bias and the variance [25]. More specifically, the bias of an estimator $\hat{f}$ for the ground truth f is given by $Bias[\hat{f}(x)] = E[\hat{f}(x)] - f(x)$, which has a value of 0 if $\hat{f}$ is an unbiased estimator of f. However, the squared error of $\hat{f}$, $E[((\hat{f}(x) - f(x))^2]$, is proportional to $(Bias[\hat{f}(x)])^2 + (Var[\hat{f}(x)])^2$ [25]; thus, an unbiased estimator $\hat{f}$ could still have a large estimation error if the estimation variance is high.

Approaches		Framework Parameters				
		p_v	q_v	u_v	sentry	estimation method
CS2 [30]		$p_v = 1$ for all v	$q_v = \theta$ for all v	$u_v = 1$ for all v	no	simple scaling
CSO [28]		$p_v = \theta$ for all v	$q_v = 1$ for all v	$u_v = 1$ for all v	no	simple scaling
CS2L [4]		p_v is proportional to $\sqrt{a_v b_v}$	same q_v for all v	$u_v = q_v$	yes	simple scaling
CSDL-Opt	jvd is small	$p_v = 1$ for all v	q_v is proportional to $\sqrt{a_v b_v}$	$u_v = q_v$	yes	discrete learning
	jvd is large	$p_v = \theta$ for all v	q_v is proportional to $\sqrt{a_v b_v}$	$u_v = q_v$	yes	discrete learning

TABLE II: Summary of existing and new correlated sampling techniques (space budget = $\theta \cdot data\_size$)

Our experimental evaluation actually shows using the unbiased scaling-up estimation method for join estimation produced high estimation variance resulting in high estimation error. Indeed, there are many estimation problems where unbiased estimators often have high estimation variance, and biased estimators with lower variance are therefore preferred. This is known as the *bias-variance tradeoff* [25]. For example, in many machine learning problems, regularizers (e.g., L1 and L2 regularizers) are usually added to reduce estimation variance, especially when the amount of training data is small compared to the population. This is because an unbiased estimator without regularizer (e.g., the least squares estimator) often have high estimation variance on real data. With the use of regularizers, the estimator becomes biased but has much better estimation quality on real data due to a much lower estimation variance [9], [26].

Accuracy for small samples. Second, existing approaches need relatively large samples to obtain satisfactory estimation quality, and they do not perform well with very small samples. While providing good estimations with very small samples is a desirable property for sampling techniques in general, this property is even more important for correlated sampling techniques because correlated sampling generally requires table samples to be created separately for each join graph to be estimated. Therefore, the space allocated for a join graph's samples could be limited if there are many join graphs in a query workload of interest. As an example, suppose that we have a query workload consisting of 10 frequently queried join graphs in a 100GB database. If we were to use a sampling rate of 1% for the samples (i.e., $\theta = 0.01$), then each join graph would require 1GB samples with a total space requirement of 10GB for all the sample synopsis. To reduce this storage overhead would require using a lower sampling rate; hence, it is important for a correlated sampling technique to provide good estimation quality even with very small samples.

One-size-fits-all approach. Third, all the existing correlated sampling techniques are essentially a one-size-fits-all approach that applies a single technique to all join size estimation problems independent of the characteristics of the tables being joined. However, our experimental study reveals that the accuracy of correlated sampling is affected by a data characteristic that we termed *join value density* defined as follows.

Consider the join between two tables A and B, where V_A (resp., V_B) denote the set of distinct join column values in table A (resp., B). The **join value density** (denoted by jvd) between A and B (or simply join value density when the context is clear) is defined as

$$jvd \ = \min\{\frac{|V_A|}{|A|}, \frac{|V_B|}{|B|}\}$$

where $|X|$ denote the cardinality of a set X.

Our experimental study shows that the join value density is an important data characteristic that affects the accuracy of correlated sampling: sampling techniques that perform well for joins with large jvd often do not perform well for joins with small jvd and vice versa. Indeed, for the state-of-the-art correlated sampling, CS2L, we have shown analytically that the estimation variance of CS2L actually increases as jvd becomes smaller.[2] [29]

Since there is no correlated sampling technique that performs well for both large and small jvd, it is therefore important to choose an appropriate sampling technique depending on the jvd of the join.

IV. Our Approach

In this section, we present a new class of correlated sampling techniques (denoted by CSDL). Our new approach is based on applying a recently proposed *discrete learning algorithm* [27] as the estimation method, which is a biased estimator that has a low empirical estimation variance. The algorithm can learn discrete distributions accurately for very small samples. This addresses the first two limitations of existing approaches discussed in Section III. To address the third limitation of existing approaches, we design different variants of CSDL with different sampling strategies, and identify the best variants for different data characteristics (i.e., with different jvd) through experiments and analysis.

The rest of this section is organized as follows. We first briefly introduce the estimation method, the discrete learning algorithm, that is used in CSDL in Section IV-A, and then present several variants of CSDL in Sections IV-B. Section IV-C and Section IV-D include some analysis on the variants. For ease of presentation, the preceding discussions focus on two-table (many-to-many) equijoin queries $A \bowtie B$ without any selection predicate. We discuss how to estimate for join

[2]Small jvd is quite common; for example, the join between sellers table and products table has small jvd (less than 0.001) in an e-commerce database where each seller can sell thousands of products.

	Same q_v for all v	**Different q_v for each v**
Same p_v for all v	CSDL(1,θ): $p_v = 1, q_v \approx \theta$	CSDL(1,diff): $p_v = 1$, q_v is proportional to $\sqrt{a_v b_v}$
	CSDL(θ,1): $p_v = \theta, q_v = 1$	CSDL(θ,diff): $p_v = \theta$, q_v is proportional to $\sqrt{a_v b_v}$
	CSDL($\sqrt{\theta}$,$\sqrt{\theta}$): $p_v = \sqrt{\theta}, q_v \approx \sqrt{\theta}$	CSDL($\sqrt{\theta}$,diff): $p_v = \sqrt{\theta}$, q_v is proportional to $\sqrt{a_v b_v}$
Different p_v for each v	CSDL(diff,1): p_v is proportional to $\sqrt{a_v b_v}$, $q_v = 1$	CSDL(diff,diff): both p_v and q_v are proportional to $\sqrt{a_v b_v}$
	CSDL(diff,θ): p_v is proportional to $\sqrt{a_v b_v}$, $q_v = \theta$	
	CSDL(diff,$\sqrt{\theta}$): p_v is proportional to $\sqrt{a_v b_v}$, $q_v = \sqrt{\theta}$	

TABLE III: CSDL variants classified into four categories. Detailed formulae for p_v and q_v is in our technical report [29].

queries with selection predicates in Section IV-E, and how to estimate for PK-FK (i.e., primary-key-foreign-key) join queries in Section IV-F.

A. Discrete learning algorithm

This section briefly introduces the estimation method that is used in our proposed CSDL. The method is based on a recently proposed discrete learning method (which we refer to as DL) [27] that is designed to accurately learn discrete distributions for very small samples.

Recall that in the offline phase of correlated sampling for the join $A \bowtie B$, we first draw a sample S_A from table A and then draw a sample S_B from $B \ltimes S_A$. These two samples are used during the online phase to estimate the join size for queries involving $A \bowtie B$. Our CSDL approach applies DL to estimate the distribution of join column values in table A from S_A during the online phase[3].

The intuition for DL is that, when a distribution is known, we can easily estimate how a sample would be like given the distribution; however, when a sample is known but the distribution is unknown, we should estimate the distribution such that the expected sample statistics from the distribution is as close as possible to our known sample.

The detailed algorithm of DL is discussed in Appendix A. A key step in DL is the construction of a linear program to minimize the difference between F_i and the expected F_i (i.e., $E[F_i]$), where the sample fingerprint F_i is the number of domain values that appear i times in the sample. Here, $E[F_i]$ is the sample statistics calculated from the estimated distribution, and F_i is the statistics of the given sample.

In DL, the input sample is assumed to be a simple random sample, which enables $E[F_i]$ to be computed using Equation (9) shown in Appendix A. This brings a challenge to applying discrete learning to correlated samples, especially when the sampling probabilities for each join value is different. This is because when we do correlated sampling with different sampling probabilities, the sample is not a simple random sample which means that Equation (9) is not applicable to compute $E[F_i]$.

Our approach addresses this challenge by constructing a virtual sample based on the real sample, where the virtual sample has the same $E[F_i]$ as that in a simple random sample, and $E[F_i]$ can be calculated using Equation (9).

[3]In the general case where there is a selection predicate on A in the runtime query (i.e., $\sigma_c(A) \bowtie B$), the algorithm is applied to estimate the distribution of the join values in $\sigma_c(A)$; we defer the discussion of the general case to Section IV-E.

We will discuss how the virtual sample is constructed to satisfy Equation (9) in Section IV-B3, and explain the advantage of using the virtual sample compared to directly using a simple random sample (given that they have the same expected fingerprints) in Section IV-C.

B. CSDL Variants

In this section, we present 10 variants of our proposed CSDL technique which are summarized in Table III. They all use the discrete learning algorithm as the estimation method and apply the sentry technique for the second-level sampling step. To avoid having unpredictably large sample synopsis, all the variants have $u_v = q_v$; the details are discussed in our technical report [29]. Note that all the existing approaches in Table II also have $u_v = q_v$ except for CS2 which has the unpredictable space problem because of this.

The CSDL variants therefore differ in the options for the remaining two parameters, namely, p_v and q_v. The options for these two sampling probability parameters can be classified into four cases: same p_v and q_v for all v (Section IV-B1); different p_v for each v, and same q_v for all v (Section IV-B2); same p_v for all v, and different q_v for each v (Section IV-B3); and different p_v and q_v for each v (Section IV-B4). Based on this classification in terms of p_v and q_v, we have chosen to examine 10 variants of CSDL in this paper that represent diverse instantiations of the parameter space.

Given that our CSDL variants differ only in p_v and q_v, we use CSDL(p_v,q_v) to refer to a specific CSDL variant. In the following, we explain how our approach works for each of the four cases; in particular, we explain how the discrete learning algorithm [27] is adapted as the estimation method for correlated sampling.

1) Case 1: Same p_v & Same q_v: For this case, we consider three options for p_v: 1 (the maximum value), θ (the minimum value to fully utilize the space budget), and $\sqrt{\theta}$ (a value between the minimum and maximum values). Given a space budget and p_v, q_v can be uniquely derived such that the sample size is within the space budget; the detailed derivation is given in our technical report [29].

We now explain how we adapt the discrete learning algorithm (Section IV-A) to estimate the join size for CSDL. Note that the following discussion applies for any values chosen for p_v and q_v (and not only for the specific values being considered for the setting in this section). We first consider the simpler case where $p_v = 1$, and then discuss the more general case where $p_v \leq 1$.

First, consider the case where $p_v = 1$ for all v. Given a space budget $\theta \cdot data\_size$, if we set $p_v = 1$, then $q_v \approx \theta$. Note that q_v is not exactly θ due to the use of sentry technique in CSDL; details are shown in our technical report [29]. Here we are essentially drawing a simple random sample S_A from A with sampling probability $q_v = \theta$. Then we sample the joinable tuples in the second table B with sampling probability $u_v = q_v = \theta$ to obtain S_B.

In the estimation phase, since the sample S_A is a simple random sample, the sample will satisfy the discrete learning algorithm's input requirement (i.e., Equation (9) discussed in Section IV-A). We first apply the discrete learning algorithm to estimate the join column distribution of A. Then we obtain a mapping between the join column values and their estimated distribution probabilities in table A (excluding sentries). If the probability of a value v is x_v and $|A| = N$, then the table frequency of v is estimated as $a_v = x_v \cdot N + 1$. Since S_B is also sampled with sentries, for each value v, its frequency in B is estimated as $b_v = \frac{S_B(v)}{u_v} + 1$. Putting these together, when $p_v = 1$, the join size is estimated as

$$\hat{J} = \sum_{v \in V_{A,B}} a_v b_v = \sum_{v \in V_{A,B}} (x_v N + 1)(\frac{S_B(v)}{u_v} + 1) \quad (4)$$

We now discuss the general case where $p_v \le 1$. In this case, only join values sampled in the first-level sampling are considered in the second-level sampling. We denote the set of sampled join values as $V'_{A,B}$.

Let N' denote the total number of tuples in A with sampled join values; i.e., $N' = \sum_{v \in V'_{A,B}} a_v$. Thus, if $p_v \le 1$, the join size is estimated as

$$\hat{J} = \sum_{v \in V'_{A,B}} \frac{1}{p_v}(x_v N' + 1)(\frac{S_B(v)}{u_v} + 1) \quad (5)$$

Note that N' is computed during the sampling phase and is stored together with samples S_A and S_B.

2) Case 2: Different p_v & Same q_v: For this case, we also consider three variants with $q_v \in \{1, \theta,$ and $\sqrt{\theta}\}$ as summarized in Table III. Note that for the CSDL variants discussed in this paper, the heuristic that guides our selection of different p_v (or q_v) values is that p_v should be proportional to $\sqrt{a_v b_v}$. The rationale for this is so that the contribution of v to the join cardinality is proportional to $a_v b_v$. If our approach was using only one-level sampling, then we would have set the sampling probability to be proportional to $a_v b_v$. However, as our approach is using two-level sampling with two sampling probabilities p_v and q_v, we set each of p_v and q_v to be proportional to $\sqrt{a_v b_v}$. The details of the derivation are in our technical report [29]. Note that although CS2L chooses values for p_v to minimize the variance of their unbiased estimation, their optimal p_v is also roughly proportional to $\sqrt{a_v b_v}$ [4].

The sampling and estimation steps for this case are similar to those for Case 1 discussed in Section IV-B1 except that p_v can be different for each v. The join size is also estimated using Equation (5).

3) Case 3: Same p_v & Different q_v: For this case, we consider three variants with $p_v \in \{1, \theta, \sqrt{\theta}\}$ as summarized in Table III. The sampling and estimation steps are similar to those discussed for Cases 1 and 2 but with one important change in the estimation phase. Instead of using S_A as input sample of the discrete learning algorithm, we construct a virtual sample, denoted by S_{A2}, that is related to S_A as follows: $S_{A2}(v) = S_A(v) \cdot t$, where t is a function of q_v, to be explained next. Then we use S_{A2} instead of S_A in the discrete learning algorithm. Next, we will explain why constructing S_{A2} is necessary.

Let us denote the sample we draw using same $p_v = p$ and different q_v as S_A, and denote the sample we draw in Section IV-B1 using same $p_v = p$ and same $q_v = q$ as S_{A1}. Note that S_A and S_{A1} are using the same $p_v = p$, and the respective q_v and q are calculated based on the same space budget. As we discussed in Section IV-B1, we could apply the discrete learning algorithm to S_{A1}, based on Equation (9). However, Equation (9) no longer holds for S_A as Lemma 1 shows. The high level intuition is that Equation (9) is essentially using the Poisson distribution to approximate the binomial distribution of the sample fingerprint F_i. F_i follows the binomial distribution if the sample is a simple random sample. However, if each domain value is drawn using different probabilities, F_i no longer follows the binomial distribution and thus cannot be approximated by the Poisson distribution as in Equation (9).

Lemma 1. *Let F_i and G_i be the number of values that appear i times in S_A and S_{A1}, respectively.* $E[F_i] \neq E[G_i] = \sum_{x \in X} poi(nx, i) \cdot r_x$.

Proof. By definition of F_i and G_i, we have $F_i = |\{v | S_A(v) = i\}|$, $G_i = |\{v | S_{A1}(v) = i\}|$. Because when we draw S_A, each tuple with join value v is drawn with probability q_v, and when we draw S_{A1}, each tuple with value v is drawn with probability q (same for all v), we have $E[S_A(v)] = a_v q_v$, $E[S_{A1}(v)] = a_v q$. Since $q_v \neq q$ in general, $E[S_A(v)] \neq E[S_{A1}(v)]$ for each v. Thus, in general, $E[F_i] \neq E[G_i]$. □

Therefore, we cannot directly use S_A as input sample to the discrete learning algorithm. Instead, we construct S_{A2} from S_A at estimation time by letting

$$S_{A2}(v) = S_A(v) \frac{q}{q_v} \quad (6)$$

Then it can be easily shown that $E[S_{A2}(v)] = E[S_{A1}(v)]$. Thus, we have $E[H_i] = E[G_i] = \sum_{x \in X} poi(nx, i) \cdot r_x$, where H_i is the number of values that appear i times in S_{A2}. We could then use S_{A2} in the discrete learning algorithm.

We will discuss the advantage of Case 3 using S_{A2} compared to Case 1 using the simple random sample S_{A1}, given that S_{A2} and S_{A1} have the same expected fingerprint, in Section IV-C.

4) Case 4: Different p_v & Different q_v: The sampling and estimation phases for this case are similar to the discussion in Section IV-B3. We denote the sample we draw using different p_v and q_v for each v as S_A, and denote the sample we draw in Section IV-B2 using different p_v and same q as S_{A1}. We construct S_{A2} with the property that $S_{A2}(v) = S_A(v)\frac{q}{q_v}$, and use S_{A2} in the discrete learning algorithm. Finally, we use Equation (5) to estimate the join size.

C. Discussion on Case 3 and Case 4

Note that in Case 3, the input sample to the discrete learning algorithm is the virtual sample S_{A2}, not the originally drawn sample S_A. S_{A2} is constructed such that it has the same expected sample fingerprint as the simple random sample S_{A1} in Case 1. As a result, the Case 3 input S_{A2} and Case 1 input S_{A1} have the same expected fingerprint. Then a natural question to ask is, what is the difference between Case 3 and Case 1?

The difference is that the Case 3 virtual sample $S_{A2}(v)$ has lower variance for frequent join values than the Case 1 sample $S_{A1}(v)$ does, and thus frequencies of these values can be estimated more accurately. Same logic applies to Case 4 and Case 2 where the Case 4 virtual sample has the same expected fingerprint as in Case 2.

Indeed, the point of Case 3 using different q_v for different v is to allocate more synopsis space to values with high frequencies in the table, because they are likely to contribute more to the join size. We show in Lemma 2 that if we use higher sampling probability for a value v when drawing S_A, the variance of constructed $S_{A2}(v)$ will be smaller than that of $S_{A1}(v)$.

Lemma 2. *$Var[S_{A2}(v)] < Var[S_{A1}(v)]$ when $q_v > q$.*

Proof. By variance of binomial distribution, $Var[S_{A1}(v)] = a_v q(1-q)$, $Var[S_A(v)] = a_v q_v(1-q_v)$. Then $Var[S_{A2}(v)] = Var[S_A(v)\frac{q}{q_v}] = \frac{q^2}{q_v^2}Var[S_A(v)] = \frac{q^2}{q_v^2}a_v q_v(1-q_v) = a_v q\frac{q}{q_v}(1-q_v) = a_v q(\frac{q}{q_v} - q)$. Since $\frac{q}{q_v} < 1$, $Var[S_{A2}(v)] < Var[S_{A1}(v)]$. ☐

D. Best Performing Variants

The variants we introduced in Section IV-B are using different sampling strategies. As we discussed in Section III, data with different characteristics (especially join value density jvd) need different sampling strategies. To find out which variants are the best, we conduct experiments on the Join Order Benchmark (JOB), where queries have large or small jvd. As we will show in Section VI, for queries with small jvd, the best variants are CSDL(1,θ) from Case 1 and CSDL(1,diff) from Case 3. CSDL(1,diff) from Case 3 is slightly better in terms of empirical variance. For queries with large jvd, each case has some variant performing quite well (CSDL(θ,1), CSDL(diff,1), CSDL(θ,diff), CSDL(diff,diff)). The best is CSDL(θ,diff) from Case 3 for large jvd.

To summarize, we observe from our experiments that Case 3 usually performs the best. The best variants are shown in the last two lines of Table II. More specifically, we observe that a) Using different p_v (Case 2 and Case 4) is often not the best strategy, especially when the join value density is small. b) Case 3 often performs better than Case 1, which corresponds to our variance analysis in Section IV-C.

E. Join Queries with Selection Predicates

So far, we have explained our estimation approach for queries without any selection predicate. In this section, we extend the ideas for queries with selection predicates of the form $\sigma_{c_A}(A) \bowtie \sigma_{c_B}(B)$.

The offline sampling phase remains unchanged since the selection in the queries are only known during the online estimation phase. At online time, we first derive the filtered sample $S''_A = \sigma_{c_A}(S_A)$ by applying the query's selection predicate c_A on table A to our sample S_A. We then apply the discrete learning algorithm on S''_A to learn the distribution of the join column values for tuples that satisfy c_A. The selectivity of c_A on table A is estimated as $f^{c_A} = \frac{|S''_A|}{|S_A|}$. Let $S''_B = \sigma_{c_B}(S_B)$, $V''_{A,B}$ be the set of join column values that are present in both S''_A and S''_B, and N'' denote the estimated number of tuples in A with sampled join values that satisfy c_A. Thus, we have $N'' = N' \cdot f^{c_A}$. Then the estimated join size is

$$\hat{J} = \sum_{v \in V''_{A,B}} \frac{1}{p_v}(x_v N'' + I''_A(v))(\frac{S''_B(v)}{u_v} + I''_B(v)) \quad (7)$$

where $S''_B(v)$ is the set of tuples in S_B that satisfy the selection predicate c_B on table B; $I''_A(v)$ is 1 when the sentry $s_A(v)$ satisfies c_A, and 0 otherwise; and $I''_B(v)$ is 1 when the sentry $s_B(v)$ satisfies c_B, and 0 otherwise.

F. PK-FK Join Queries

The algorithm for handling PK-FK joins is not much different from that for many-to-many equijoins. However, there are two issues to take note. First, for PK-FK joins, we should always sample the FK table first and get joinable tuples from the PK table, and apply the discrete learning algorithm on the sample of the FK table. We do not want to apply the discrete learning algorithm to the PK table because we know exactly how the join column values are distributed (each join value appears exactly once in the table). The difficulty of estimating join size is on estimating distribution of the FK table.[4] Also, estimating the join size for PK-FK joins with no selection predicate is trivial since the join size is just the size of the FK table. Moreover, if the selection predicates are only on the FK table, the join size estimation is no different from single-table selectivity estimation, which is not the focus of this paper. The problem is only interesting when the query has selection predicates on the PK table.

Second, since we always draw a sentry first in second-level sampling, we are not actually doing second-level sampling on the PK table as each join value only has one tuple. The algorithm does not change; it is just that we will be drawing tuples from an empty set.

[4]For many-to-many joins, we randomly choose one of the tables to sample first, since the join column distributions in both tables are unknown before sampling.

V. Multi-table Joins

We have extended our approach to handle two common types of multi-table join queries, namely, chain joins and star joins, with arbitrary number of joined tables. Extending to other multi-table query structures is part of our future work. In this section, we present our approach for chain join queries; due to space constraint, star join queries are discussed in our technical report [29].

Let us consider a 3-table chain join query as an example: $A \overset{A.PK=B.FK}{\bowtie} B \overset{B.PK=C.FK}{\bowtie} C$, where A joins B on A's primary key, and B joins C on B's primary key. Extending to chain join queries with more than three tables is straightforward. Right now, we can only support this join structure where each PK-FK join has the FK table on the right side; this is the same chain join structure supported by `CS2L`.

Let $V_{A,B}$ and $V_{B,C}$ be the set of join column values for $A \bowtie B$ and $B \bowtie C$, respectively. We use p_v and q_v as the first-level and second-level sampling probabilities for sampling C, and obtain all the joinable tuples from A and B. Note that we will not be doing second-level sampling on A and B because for each join column value v in S_C ($v \in V_{B,C}$), there is at most one joinable tuple in B, and for each (u, v) pair in S_B ($u \in V_{A,B}$, $v \in V_{B,C}$), there is at most one joinable tuple in A, given that the join attributes are the primary keys of B and A. In other words, we will only be getting sentries from B and A. For chain join queries with any number of joined tables, we always choose the rightmost table (which joins on its foreign key) to be the first sampled table.

We apply the discrete learning algorithm on S_C to obtain the mapping between each join value $v \in V_{B,C}$ and its estimated probability x_v. The chain join size is estimated as

$$\hat{J} = \sum_{u \in V''_{A,B}, v \in V''_{B,C}} \frac{1}{p_v} I''_A(u) I''_B(u,v)(x_v N''_0 + I''_C(v)) \quad (8)$$

where $V''_{A,B}$ is the set of join values in S''_B for $A \bowtie B$, $V''_{B,C}$ is the set of join values in S''_C for $B \bowtie C$, $I''_A(u)$ is 1 if there exists a tuple in S_A with join values u that passes c_A and 0 otherwise, $I''_B(u,v)$ is 1 if there exists a tuple in S_B with join values u, v that passes c_B and 0 otherwise, $I''_C(u)$ is 1 if sentry $s_C(u)$ passes c_C and 0 otherwise, $N''_0 = N_0 \cdot f^{c_C} = N_0 \cdot \frac{|S''_C|}{|S_C|}$.

For setting p_v and q_v, we follow the design choices of `CSDL-Opt` in Table II: When the join value density $jvd = min\{\frac{|V_B|}{|B|}, \frac{|V_C|}{|C|}\}$ is small, we use `CSDL`(1,diff); when jvd is large, we use `CSDL`(θ,diff). The design choices of `CSDL-Opt` will be discussed in Section VI with experimental results.

VI. Experimental Results

In this section, we present experimental results comparing the 10 variants of `CSDL` against `CS2L`, the state-of-the-art correlated sampling technique[5]. Our comparison metric is the *q-error* [20] which is defined as q-error $= max(J, \hat{J})/min(J, \hat{J})$, where J denotes the true join size and $\hat{J}$ denotes the estimated join size. The q-error metric has been used by recent papers on cardinality estimation [11], [16], [21] as it has the nice property that a theoretical upper bound can be established for a query plan quality if its q-error is bounded, unlike other previous error metrics (e.g., absolute error, relative error, L2 error) [20].

All the correlated sampling techniques were implemented using Python[6]. Our experiments were conducted on an Intel Xeon Processor E5-2603 v2 server with 32GB memory, running Ubuntu Linux 16.04. Our experiments used both real dataset (Join Order Benchmark (JOB) [15]) as well as synthetic dataset (TPC-H Benchmark). Each reported estimation error for a query (w.r.t. a space budget) is the median value from performing 20 join size estimations.

We report the results for four experiments. Experiments 1 to 3 focus on two-table join queries. Experiment 1 (Section VI-A) compares our proposed 10 variants of `CSDL` against `CS2L` in terms of both estimation quality as well as estimation runtime. Our results reveal two winning `CSDL` variants: `CSDL`(1,diff) performs the best for queries with low join value density (jvd), while `CSDL`(θ,diff) performs the best for queries with high jvd. We combine these two variants into a hybrid variant that we refer to as `CSDL-Opt`. Our results also show that `CSDL-Opt` outperforms `CS2L` in terms of better estimation quality. Experiment 2 (Section VI-B) examines the effect of small filtered samples by varying the query's selection predicate selectivity, while Experiment 3 (Section VI-C) examines the effect of data skewness. Our experimental results show that `CSDL-Opt` is more robust than `CS2L` for both small filtered samples as well as skewed data. Experiment 4 (Section VI-D) focus on multi-table join queries, and our results show that `CSDL-Opt` outperforms `CS2L` as well for such queries.

A. CSDL variants vs. CS2L

In this section, we compare the 10 `CSDL` variants (shown in Table III) against `CS2L`, the state-of-the-art correlated sampling technique. The comparison focuses on two-table join queries that are derived from the multi-table join queries in the Join Order Benchmark (JOB). JOB is based on the IMDB movie data set, and it consists of 113 multi-table join queries that are derived from 33 different join graphs. Queries derived from the same join graph differ only on their selection predicates. For a multi-table join query in JOB, we can derive multiple two-table join queries from it by choosing different pairs of tables being joined (and their associated selection predicates). Our technical report [29] has more details on the queries used.

For this experiment, we used all the 6 multi-table join queries (Q1a, Q1b, Q2a, Q2b, Q2c, and Q2d) that are based on the first two join graphs (Q1 and Q2) in JOB to derive 14 two-table join queries. The identifier of each two-table join query

[5]We did not compare against `CSO` and `CS2` because `CS2L` has been shown to outperform `CSO` [4] and `CS2` is not practical due to its unpredictably large samples (Section II-B). Some early work on end-biased sampling [8] has also been shown to be outperformed by `CSO` [28].

[6]The code for `CS2L` was obtained from the authors. We use $D = 0.08, E = 0.05$ in Algorithm 1 of our technique. We have tried different values of D, E that satisfy the requirement $0 < \frac{D}{2} < E < D < 0.1$ in Algorithm 1, which does not have much impact on experimental results.

Query	θ	**1,θ**	θ,1	$\sqrt{\theta}$,$\sqrt{\theta}$	diff,1	diff,θ	diff,$\sqrt{\theta}$	**1,diff**	θ,diff	$\sqrt{\theta}$,diff	diff,diff	`CS2L`
Q1a1	0.001	**1.13**	∞	∞	∞	1.19	∞	**1.12**	∞	∞	∞	1.11
($J = 28657$)	0.0001	**1.44**	∞	∞	∞	1.36	∞	**1.41**	∞	∞	∞	∞
Q1a4	0.001	**140.93**	∞	∞	∞	∞	∞	**146.05**	∞	∞	∞	∞
($J = 250$)	0.0001	**3.11**	∞	∞	∞	∞	∞	**3.55**	∞	∞	∞	∞
Q1b1	0.001	**1.00**	∞	∞	∞	1.01	∞	**1.00**	∞	∞	∞	1.01
($J = 1334832$)	0.0001	**1.04**	∞	∞	∞	1.03	∞	**1.02**	∞	∞	∞	∞
Q1b4	0.001	**5.88**	∞	∞	∞	∞	∞	**5.79**	∞	∞	∞	∞
($J = 10$)	0.0001	**7.72**	∞	∞	∞	∞	∞	**6.94**	∞	∞	∞	∞

TABLE IV: Comparison of q-error for queries with small join value density ($jvd < 0.001$). J denotes the true result size.

Query	θ	1,θ	**θ,1**	$\sqrt{\theta}$,$\sqrt{\theta}$	**diff,1**	diff,θ	diff,$\sqrt{\theta}$	1,diff	**θ,diff**	$\sqrt{\theta}$,diff	diff,diff	`CS2L`
Q1a2	0.001	∞	**1.29**	∞	**1.11**	∞	9.69	∞	**1.27**	∞	1.44	1.69
($J = 28889$)	0.0001	∞	**3.14**	∞	**1.76**	∞	∞	∞	**2.57**	∞	5.27	1.75
Q1a3	0.001	∞	**1.24**	∞	**1.81**	∞	8.46	∞	**1.31**	∞	1.25	1.85
($J = 62658$)	0.0001	∞	**4.43**	∞	**11.82**	∞	∞	∞	∞	∞	8.39	2.46
Q1b2	0.001	268317.42	**1.50**	∞	**1.38**	101.99	1.67	1.66	**1.42**	∞	1.81	1.04
($J = 622478$)	0.0001	670791.36	**1.35**	∞	**1.49**	∞	13.93	107.71	**1.35**	∞	1.89	1.21
Q1b3	0.001	378374.30	**1.24**	∞	**1.16**	39.22	1.40	2.20	**1.07**	∞	1.74	1.05
($J = 4072924$)	0.0001	397834.69	**1.20**	∞	**2.44**	∞	5.98	45.37	**1.18**	∞	1.94	1.16
Q1b5	0.001	287478.31	**1.33**	∞	**1.35**	158.85	1.35	2.30	**1.31**	∞	1.34	1.08
($J = 357091$)	0.0001	459964.33	**1.57**	∞	**1.31**	∞	20.10	47.35	**1.23**	∞	1.32	1.28
Q2a1	0.001	43043.15	**3.13**	∞	**3.46**	1.31	1.32	9.17	**4.62**	∞	2.02	1.34
($J = 148132$)	0.0001	43043.15	**14.73**	∞	∞	114.90	2.50	16.94	**14.53**	∞	8.53	11.48
Q2a2	0.001	5.54	∞	∞	∞	1.16	∞	30247.24	∞	∞	∞	∞
($J = 41840$)	0.0001	9.38	∞	∞	∞	∞	∞	832.21	∞	∞	∞	∞
Q2b1	0.001	67650.76	**6.02**	∞	∞	1.55	1.47	16.64	**4.40**	∞	2.95	1.48
($J = 56410$)	0.0001	12025.80	∞	∞	∞	175.43	12.40	10.65	∞	∞	6.16	∞
Q2c1	0.001	326141.62	∞	∞	∞	∞	∞	1.80	∞	∞	∞	∞
($J = 2$)	0.0001	235546.86	∞	∞	∞	∞	∞	32.47	∞	∞	∞	∞
Q2d1	0.001	47965.34	**2.51**	∞	**1.60**	1.1038	1.11	3.14	**2.13**	∞	4.55	1.13
($J = 1153798$)	0.0001	47965.06	**4.72**	∞	**3.34**	315.01	1.35	3.22	**2.64**	∞	20.33	2.01

TABLE V: Comparison of q-error for queries with large join value density ($jvd \geq 0.001$). J denotes the true result size.

Query	θ	1,θ	1,diff	`CS2L`
Q1a1	0.001	0.0164	0.0067	0.0384
	0.0001	0.4338	1.1707	∞
Q1a4	0.001	4847.6262	3.5977	∞
	0.0001	1.5283	0.1479	∞
Q1b1	0.001	0.00008	0.00005	0.00013
	0.0001	0.0015	0.0003	∞
Q1b4	0.001	31.8502	0.0072	∞
	0.0001	1731.5847	0.6813	∞

TABLE VI: Comparison of estimation variance for `CSDL`(1,θ), `CSDL`(1,diff) and `CS2L` on queries with small jvd

has a prefix from the identifier of the query from which it is derived. For example, we have derived four two-table queries (Q1a1, Q1a2, Q1a3, and Q1a4) from JOB's Q1a.

Tables IV and V compare the q-errors for queries with low and high jvd, respectively. For queries with low jvd, Table IV shows that the two variants with $p_v = 1$, `CSDL`(1,θ) and `CSDL`(1,diff) (shown in bold), have the best estimation quality. The reason for this is that for low $jvd = min\{\frac{|V_A|}{|A|}, \frac{|V_B|}{|B|}\}$, the number of join values $|V_A|$ (resp. $|V_B|$) is small compared to $|A|$ (resp. $|B|$); therefore, sampling with a small p_v would yield very few sampled join values (possibly none) in the first-level sampling resulting in poor estimation. Thus, the two CSDL variants with $p_v = 1$ perform the best. Between `CSDL`(1,θ) and `CSDL`(1,diff), we observe that `CSDL`(1,diff) has a slightly lower estimation variance than `CSDL`(1,θ) as shown by the comparison of their estimation variances in Table VI. Among the remaining `CSDL` variants, `CSDL`(diff,θ) performs relatively well because its q_v is small which implies that its p_v values are relatively large. The other `CSDL` variants perform poorly as indicated by their infinity q-error values which means that the estimated join size is 0 (the filtered sample is empty); their poor performance is due to sampling with small p_v. Therefore, we pick `CSDL`(1,diff) to be the best performing variant for queries with low jvd.

For queries with high jvd, we observe an opposite trend to that for queries with low jvd. Specifically, Table V shows that the best performers are the `CSDL` variants with small p_v values (`CSDL`(θ,1), `CSDL`(diff,1), and `CSDL`(θ,diff)), and the poor performers are the `CSDL` variants with large p_v values (`CSDL`(1,θ), `CSDL`($\sqrt{\theta}$,$\sqrt{\theta}$), `CSDL`(diff,θ), and `CSDL`($\sqrt{\theta}$,diff)). The reason is that for high $jvd = min\{\frac{|V_A|}{|A|}, \frac{|V_B|}{|B|}\}$, the average number of tuples for each join value is small, and sampling with a large p_v value implies that (1) many join values would be included in the first-level sample, and (2) q_v would be small. Therefore, the resultant sample obtained from using large p_v is likely to have very few tuples (possibly none) giving poor estimation. Among the three best performing variants, our estimation variance comparison (not shown due to space constraint) indicates that CSDL(θ,diff) has slightly lower estimation variance than the other two variants. Therefore, we pick CSDL(θ,diff) to be the best performing variant for queries with high jvd.

Hybrid Variant: CSDL-Opt. Based on the two winning CSDL variants, namely CSDL(1,diff) and CSDL(θ,diff), that we have identified, in the rest of this paper, we use CSDL-Opt to refer to a hybrid CSDL variant that works as follows: CSDL-Opt estimates using CSDL(1,diff) if the jvd for a query is low (i.e., lower than 0.001); otherwise, CSDL-Opt estimates using CSDL(θ,diff). Comparing CSDL-Opt and CS2L in Table IV and Table V, CSDL-Opt outperforms CS2L when jvd is small, and gives comparable results when jvd is large. Specifically, CS2L has infinity median q-error for many queries with small jvd, which is due to CS2L's high estimation variance when jvd is small [29].

Estimation time. We conclude this section by comparing the estimation time taken by CSDL-Opt and CS2L. Note that as the focus of our experimental study is on the comparison of estimation quality, both the implementations of CSDL-Opt and CS2L have not been optimized (e.g., by exploiting parallel executions). Since the simple scaling-up estimation technique used in CS2L is much faster than the discrete learning estimation method used in CSDL-Opt (which needs to solve a linear program), the estimation time of CSDL-Opt is roughly 10 times longer than that of CS2L. Specifically, excluding the cases where the estimated join size is 0 (which means the runtime sample size is 0 and there is nothing to do in both approaches), given a space budget of $0.0001 \cdot data\_size$, CS2L has an average running time of 0.19 seconds over all queries, where the running time for 40% of the queries are under 0.15 seconds. CSDL-Opt have an average running time of 2.46 seconds, where the running time for 40% of the queries are under 0.5 seconds. Note that the JOB dataset is 3.7GB in size and the two largest tables $cast\_info$ and $movie\_info$ have 36 million and 15 million tuples, respectively.

B. *Effect of Selection Predicate Selectivity*

In this experiment, we study the effect of the selection predicate selectivity on the performance of CSDL-Opt and CS2L using two join queries over the JOB dataset. The first query is a PK-FK query that joins table *aka_title* and table *title* on *movie_id*, and the second query is a many-to-many join that self-joins *aka_title* on attribute *title* to find pairs of movies with the same alias. Both the join queries have large jvd. To vary the selection predicate selectivity for both queries, we added the selection predicate of the form *title LIKE "prefix %"* to both join queries, where prefix specifies the first word in a movie title. We used the top-100 most frequent title prefixes to vary the selection predicate selectivity. The space budget used for the samples is $0.001 \cdot data\_size$. Due to space constraint, we present the comparison results for 20 of the 100 prefix values used in Table VII. As the experimental results show, CSDL-Opt performs much better than CS2L in terms of number of failures (cases where the q-error is infinity) for both PK-FK and many-to-many join queries.

Top n most frequent prefix	Error of CSDL-Opt	Error of CS2L
1	1.24	1.40
6	1.16	1.17
11	1.42	∞
16	1.82	2.70
21	1.90	∞
26	∞	∞
31	1.19	∞
36	2.39	∞
41	2.14	∞
46	2.69	∞
51	1.61	∞
56	1.91	∞
61	3.32	∞
66	∞	∞
71	2.10	∞
76	2.14	∞
81	1.24	∞
86	1.53	∞
91	∞	∞
96	1.64	∞
Number of ∞	3	17
Number of ∞ for all 100 prefixes	20	83

(a) PK-FK join queries

Top n most frequent prefix	Error of CSDL-Opt	Error of CS2L
1	10.95	1.10
6	6.87	1.53
11	6.34	∞
16	8.11	∞
21	9.29	∞
26	18.19	∞
31	21.04	∞
36	28.24	∞
41	∞	∞
46	10.73	∞
51	∞	∞
56	∞	∞
61	∞	∞
66	85.09	∞
71	∞	∞
76	∞	∞
81	23.17	∞
86	∞	∞
91	∞	∞
96	31.25	∞
Number of ∞	8	18
Number of ∞ for all 100 prefixes	35	85

(b) Many-to-many join queries

TABLE VII: Effect of selection predicate selectivity

Dataset	θ	CSDL-Opt		CS2L	
		Q-error	Variance	Q-error	Variance
s1-z4	0.0001	37.99	83.34	∞	∞
	0.001	25.53	65.18	37.84	87.22
s0.1-z4	0.0001	1.82	∞	∞	∞
	0.001	3.23	2.61	∞	∞
s1-z2	0.0001	272.36	34873.09	342.20	∞
	0.001	2.32	13684.37	2.23	39154.99
s0.1-z2	0.0001	30.60	∞	∞	∞
	0.001	33.17	214.74	48.03	∞

TABLE VIII: Comparisons for skewed data

C. *Effect of Data Skew*

In this experiment, we compare the effectiveness of CSDL-Opt and CS2L on skewed data using skewed TPC-H datasets[7]. We used a many-to-many join query between the customer and supplier tables on c_nationkey = s_nationkey without any selection predicate. Note that this query's join value density jvd is small. The comparison results for different space budget ($\theta \in \{0.001, 0.0001\}$), scale factor ($s \in \{0.1, 1\}$) and skewness value ($z \in \{2, 4\}$) are shown in Table VIII. Observe that CS2L performs poorly in many cases, particularly for highly skewed data (s1-z4 and s0.1-z4). CSDL-Opt performs much better than CS2L (in terms of lower estimation errors) for highly skewed data, and is no worse than CS2L for not so skewed data.

[7]https://www.microsoft.com/en-us/download/details.aspx?id=52430

Dataset	Q-error of CSDL−Opt	Q-error of CS2L
s1-z4	1.51	∞
s0.1-z4	3.77	∞
s1-z2	1.45	1.11
s0.1-z2	3.49	1.31

TABLE IX: Comparisons for chain join queries

D. Multi-table Join Queries

In this experiment, we compare the performance of CSDL−Opt and CS2L for the following three-table chain query on skewed TPC-H data: customer ⋈ orders ⋈ lineitem, where all the joins are PK-FK joins with a selection predicate $c\_acctbal > 8000$ on customer table. Note that this query's join value density jvd is large.

Table IX shows the results for four skewed TPC-H datasets with different scale factor ($s \in \{0.1, 1\}$) and skewness value ($z \in \{2, 4\}$), where the space budget for the samples is $0.001 \cdot data\_size$. The results show that for highly skewed data, CSDL−Opt provides much better estimations than CS2L as CS2L tends to have infinity for its median q-error. For not very skewed data, both CSDL−Opt and CS2L have comparable performance.

VII. Related Work

There are two main approaches for join size estimation, histogram-based approaches and sampling-based approaches. Histogram-based approaches include multi-dimensional histograms [6], [22], [23], sketches [2], [5], [7], [13], [24] and wavelets [3], [18] (can be viewed as compressed histograms). The size of the synopsis for these approaches grows dramatically as number of attributes grows, including selection and join predicates [4], [28], [30]. Also, they usually have restrictions on types of predicates, supporting only equality or range predicates and not allowing predicates such as "title LIKE %hero%" [4].

Sampling-based approaches have the advantages that they do not have restrictions on selection predicates, and do not have problems with high dimensionality. However, estimation using independent random samples from base tables often has large errors for join queries. [1] proposed join synopses to keep a simple random sample of the join result, but it works only for PK-FK joins. [11] applies kernel density estimation (KDE) technique to join samples for estimating join sizes, but the online computation is very expensive and requires hardware accelerators like GPU. Another recent work [12] uses neural networks to learn the cardinalities of join queries, where joins can be correlated. The approach is enhanced using base table samples, but not directly related to correlated sampling discussed in this paper. Similar to [11], training and prediction of neural networks is done using GPU.

As we have discussed in Section II-B, correlated sampling based approaches have been shown to be simple and effective. This paper studies this family of approaches and proposes new variants that improve existing correlated sampling based solutions.

Besides, there are some works on sampling over join [10], [31], which is getting a sample of join result without actually computing the join. The techniques can be used in data analytics when computing the join is too expensive and a representative sample is desirable for getting insights. These works are not directly related to the problem of join size estimation. Also, the computation cost is usually prohibitively high in the context of query optimization.

Another interesting work is the wander join [17] designed for online aggregation over joins. Its sampling technique could potentially be used for join size estimation. We compare wander join with correlated sampling in our technical report [29].

VIII. Conclusions

In this paper, we presented a systematic study of correlated sampling techniques by introducing a framework to characterize their design space in terms of five parameters. Our proposed framework not only captures all the existing correlated sampling techniques (CS2, CSO, and CS2L), but also reveals many other unexplored possible variants. We also introduced a new class of correlated sampling techniques termed CSDL that is based on using the discrete learning algorithm for the estimation method parameter. We presented 10 variants of CSDL and experimentally compared them against CS2L, the state-of-the-art correlated sampling technique. Our empirical study reveals a pair of winning CSDL variants that together form a hybrid variant CSDL−Opt, which was shown to outperform CS2L in terms of providing better estimation quality particularly when the samples are small or when the join value density is small. As part of our future work, we plan to extend CSDL to other multi-table join graphs beyond chain and star joins, and also to handle non-equijoin queries.

Acknowledgements We would like to thank the reviewers for their constructive comments and the authors Yu Chen and Ke Yi of [4] for sharing their code for CS2L. This research is supported in part by MOE Grant R-252-000-A53-114.

Appendix A
Discrete Learning Algorithm

Algorithm 1 presents the details of the discrete learning algorithm [27] that our approach adapts for the estimation method of CSDL. The algorithm consists of two key steps: the first step (lines 1-7) estimates the "shape" of the distribution, that is, a statistical histogram with the x-axis representing probability values and the y-axis representing the number of discrete domain values with each probability value. The second step (lines 8-12) determines the probability for each value; that is, assigning each value to the correct bin in the histogram.

In the first step, F_i is the number of values that appear i times in the sample (line 1 of Algorithm 1). When drawing a sample of size n, the probability that a value with true probability x will be drawn exactly i times is

Algorithm 1: The discrete learning algorithm

```
Input: Sample $S$ of size $n$ from a discrete distribution $\mathcal{P}$
Output: A mapping between domain elements in $\mathcal{P}$ and the
        estimated probabilities
1  Let $F_i$ be the number of values that appear $i$ times in $S$
     (fingerprint of the sample)
2  Define $X = \{\frac{1}{n^2}, \frac{2}{n^2}, \frac{3}{n^2}, ..., \frac{n^D+n^E}{n}\}$, where $D, E$ are
     arbitrary constants with $0 < \frac{D}{2} < E < D < 0.1$.
3  Define variable $r_x$ for each $x \in X$, and solve the following
     linear program (LP):
4  Minimize $\sum_{i=1}^{n^D} \mid F_i - \sum_{x\in X} poi(nx,i) \cdot r_x \mid$ subject to
     $\sum_{x\in X} x \cdot r_x + \sum_{i>n^D+2n^E}^{n} \frac{i}{n} F_i = 1, \forall x \in X, r_x \geq 0$
5  Let $h_{LP}$ be the statistical histogram obtained from the LP,
     with $h_{LP}(x) = r_x$, where $r_x$ is the solution to the LP.
6  Increment $h_{LP}(\frac{i}{n})$ by $F_i$ for each integer $i > n^D + 2n^E$.
7  Construct a multiset of probabilities $M$ from the statistical
     histogram $h_{LP}$, by adding $r_x$ copies of $x$ to $M$ for each
     histogram bin $h_{LP}(x) = r_x$.
8  foreach fingerprint entry $j < \log^2 n$ do
9      Construct a weighted multiset $M_j$ by assigning weight
         $poi(nx', j)$ to each $x'$ in $M$
10     The probability of domain elements that appears $j$ times
         in the sample is estimated to be the median of the
         weighted multiset $M_j$
11 foreach fingerprint entry $j \geq \log^2 n$ do
12     The probability of domain elements that appears $j$ times
         in the sample is estimated to be their empirical
         probability $\frac{j}{n}$
```

$Prob[Binomial(n, x) = i] \approx poi(nx, i)$.[8] Therefore, the expected F_i will be

$$E[F_i] = \sum_{x\in X} poi(nx, i) \cdot r_x \tag{9}$$

where r_x denotes the number of domain values with probability x. We compute r_x by minimizing the difference of expected F_i and actual F_i using the linear program formulation (line 4). The two constraints in the linear program ensures that the total probability mass equals 1 and r_x is non-negative.

In the second step, we construct a multiset of probabilities M from the histogram h_{LP} (line 7). For values that appear j times in the sample S where $j < \log^2 n$, we weight each x' in M by $poi(nx', j)$, which is the probability that a domain value with probability x' appears j times in a sample of size n, to obtain a weighted multiset M_j (lines 8-9). The estimated probabilities for these values that appear j times is the median of M_j (lines 10-11). For values that appear $j \geq \log^2 n$ times in the sample S, the estimated probabilities is $\frac{j}{n}$ (lines 12-13).

References

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, volume 28, pages 275–286. ACM, 1999.

[2] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. *Journal of Computer and System Sciences*, 64(3):719–747, 2002.

[8]We use $Poi(\lambda)$ to denote the Poisson distribution of expectation λ, and $poi(\lambda, j)$ to denote the probability that a random variable with distribution $Poi(\lambda)$ takes value j

[3] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *VLDB Journal*, 10(2-3):199–223, 2001.

[4] Y. Chen and K. Yi. Two-level sampling for join size estimation. In *SIGMOD*, pages 759–774. ACM, 2017.

[5] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24. VLDB Endowment, 2005.

[6] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. *SIGMOD*, 30(2):199–210, 2001.

[7] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, pages 61–72. ACM, 2002.

[8] C. Estan and J. F. Naughton. End-biased samples for join cardinality estimation. In *IEEE ICDE*, pages 20–20. IEEE, 2006.

[9] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[10] N. Kamat and A. Nandi. A unified correlation-based approach to sampling over joins. In *SSDBM*, pages 20:1–20:12. ACM, 2017.

[11] M. Kiefer, M. Heimel, S. Breß, and V. Markl. Estimating join selectivities using bandwidth-optimized kernel density models. *PVLDB*, 10(13):2085–2096, 2017.

[12] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*. www.cidrdb.org, 2019.

[13] H. Lee, R. T. Ng, and K. Shim. Similarity join size estimation using locality sensitive hashing. *PVLDB*, 4(6):338–349, 2011.

[14] J.-H. Lee, D.-H. Kim, and C.-W. Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *SIGMOD*, volume 28, pages 205–214. ACM, 1999.

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

[16] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *VLDB Journal*, 27(5):643–668, 2018.

[17] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *SIGMOD*, pages 615–629. ACM, 2016.

[18] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, volume 27, pages 448–459. ACM, 1998.

[19] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, pages 101–110. Morgan Kaufmann, 2000.

[20] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *PVLDB*, 2(1):982–993, 2009.

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

[22] M. Muralikrishna and D. J. DeWitt. Equi-depth multidimensional histograms. In *SIGMOD*, volume 17, pages 28–36. ACM, 1988.

[23] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, volume 97, pages 486–495, 1997.

[24] F. Rusu and A. Dobra. Sketches for size of join estimation. *ACM TODS*, 33(3):15, 2008.

[25] C. Sammut and G. I. Webb. Bias-variance decomposition. In *Encyclopedia of Machine Learning*, pages 100–101. Springer, 2011.

[26] G. Valentini and T. G. Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *Journal of Machine Learning Research*, 5(Jul):725–775, 2004.

[27] G. Valiant and P. Valiant. Instance optimal learning of discrete distributions. In *STOC*, pages 142–155. ACM, 2016.

[28] D. Vengerov, A. C. Menck, M. Zait, and S. P. Chakkappen. Join size estimation subject to filter conditions. *VLDB Journal*, 8(12):1530–1541, 2015.

[29] T. Wang and C.-Y. Chan. Improved Correlated Sampling for Join Size Estimation. Technical report, National University of Singapore, June 2019. http://www.comp.nus.edu.sg/%7Etaining/estimation/report.pdf.

[30] F. Yu, W.-C. Hou, C. Luo, D. Che, and M. Zhu. Cs2: a new database synopsis for query estimation. In *SIGMOD*, pages 469–480. ACM, 2013.

[31] Z. Zhao, R. Christensen, F. Li, X. Hu, and K. Yi. Random sampling over joins revisited. In *SIGMOD*, pages 1525–1539. ACM, 2018.