Weighted Distinct Sampling: Cardinality Estimation for SPJ Queries

Yuan Qiu[1] Yilei Wang[1] Ke Yi[1,3] Feifei Li[2] Bin Wu[2] Chaoqun Zhan[2]
[1]Hong Kong University of Science and Technology
[2]Alibaba Group
[3]SICS, Shenzhen University
(yqiuac,ywanggq,yike)@cse.ust.hk;(lifeifei,binwu.wb,lizhe.zcq)@alibaba-inc.com

ABSTRACT

SPJ (select-project-join) queries form the backbone of many SQL queries used in practice. Accurate cardinality estimation of these queries is thus an important problem, with applications in query optimization, approximate query processing, and data analytics. However, this problem has not been rigorously addressed in the literature, despite the fact that cardinality estimation techniques of the three relational operators, selection, projection, and join, have each been extensively studied (but not when used in combination) in the past 30+ years. The major technical difficulty is that (distinct) projection seems to be difficult to combine with the other two operators when it comes to cardinality estimation.

In this paper, we give the first formal study of cardinality estimation for SP queries. While it was studied in a prior work in 2001, there is no guarantee on its optimality. We define a class of algorithms, which we call *weighted distinct sampling*, for estimating SP query sizes, and show how to find a near-optimal sampling strategy that is away from the optimum only by a lower order term. We then extend it to handling SPJ queries, giving the first non-trivial solution for SPJ cardinality estimation. We have also performed an extensive experimental evaluation to complement our theoretical findings.

ACM Reference Format:
Yuan Qiu, Yilei Wang, Ke Yi, Feifei Li, Bin Wu and Chaoqun Zhan. 2021. Weighted Distinct Sampling: Cardinality Estimation for SPJ Queries. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3452821

1 INTRODUCTION

Query size estimation (*a.k.a.* cardinality estimation) is a fundamental problem in database systems that requires little justification for its importance. Size estimation techniques for the three basic relational operators: Selection (σ), Projection[1] (π), and (natural) Join ($\bowtie$), have each been extensively studied in the literature. However, most existing methods do not compose, which becomes a serious problem to their practical adoption as these relational operators rarely appear on their own in a query. While a number of techniques can handle SJ queries [8, 25, 32, 42], how to deal with P jointly with SJ largely remains an unsolved problem. There is one early work investigating size estimation for SP queries [19], which is reviewed in more detail in Section 1.3, but no (non-trivial) techniques can handle SPJ queries.

When P is the only operator applied on a table R, the size of $\pi_A(R)$ is the number of distinct values (NDV) in attribute A. The problem of estimating the NDV using small space has been studied for more than 30 years by both the database and the algorithms community [5, 6, 11, 13–15, 30, 41]. However, P by itself does not yield many interesting queries. Instead, it is often combined with S and J, as illustrated by the following examples on the TPC-H schema:

```
SELECT count(distinct o_custkey) FROM orders
 WHERE o_orderdate > 2020-01-01 AND o_orderpriority = "1-URGENT"
SELECT count(distinct o_custkey) FROM orders, lineitem
 WHERE o_orderdate > 2020-01-01 AND l_extendedprice > 100
  AND o_orderpriority = "1-URGENT" AND o_orderkey = l_orderkey
```

The first is an SP query that counts the number of distinct customers who have ever placed an urgent order this year, while the second is an SPJ query that adds one more constraint that the order must contain one item priced at least 100.

The current practice for dealing with such queries decouples P and SJ, *i.e.*, it just returns the smaller of the NDV of projection attribute and the estimated size of SJ query. This over-simplistic approach ignores any interaction between P and SJ, which often leads to a gross overestimation. As an extreme case, imagine that there are many customers, but only one has placed urgent orders this year, and it happens that s/he placed many urgent orders.

In this paper, we first design a new technique for estimating the size of SP queries, significantly improving over the results from [19] in both theory and practice. Then we consider SPJ queries, advancing the state of the art of cardinality estimation one step closer to general SQL queries.

1.1 Problem Definition

Any SPJ query can be expressed by the following normal form [1]:

$$Q := \pi_A(\sigma_\phi(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m)). \quad (1)$$

[1]In this paper, the projection operator π_A uses its relational algebra semantics, *i.e.*, duplicates in A are removed after the projection. If not, this operator need not be considered as it does not affect the query size. This corresponds to `SELECT DISTINCT A` in SQL. Furthermore, `GROUP BY A` yields a result size identical to that of π_A, so our techniques also apply to queries where the projection is replaced by a group-by.

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452821

When $m = 1$, this becomes an SP query. For simplicity, we only consider the case where the projection is on a single attribute A. For SP queries, this is not a restriction, since we can (conceptually) concatenate all the projection attributes into a single one. This does not work for SPJ queries, though, and we leave SPJ queries with multiple projection attributes to future studies.

We assume that all the relations $R_1, \ldots, R_m$ and the projection attribute A are known in advance. However, the selection condition ϕ is only given at query time and can be arbitrary. In particular, ϕ may involve attributes from different relations, so our formulation also incorporates *θ-joins with arbitrary join conditions given at query time*. The goal is to construct a synopsis on $R_1, \ldots, R_m$, such that given any ϕ, we can estimate $|Q|$ from the synopsis with small error.

Our definition reflects how cardinality estimation is needed in practical database systems: The join structure of the queries has only limited forms, often confined by the database schema; the projection attribute also has limited possibilities. So it is affordable to build a synopsis for each possible query in the form of (1) that the database expects to receive. On the other hand, the selection condition ϕ may use arbitrary expressions (even UDFs) over any attributes, which is not possible to be given in advance.

We introduce some more notations. For SP queries, we use R to denote the only relation in (1), whose size is N. For SPJ queries, we use $N_\bowtie = |R_1 \bowtie \cdots \bowtie R_m|$ to denote the join size. When the projection attribute A is clear from the context, we use D to denote the NDV of A, and use D^ϕ to represent the NDV of A subject to the selection condition ϕ, which is also the query size $|Q|$ we wish to estimate. We use $\hat{D}^\phi$ to denote an estimator of D^ϕ. Table 1 summarizes the notations used in this paper.

Notation	Meaning
$R, R_1, \ldots, R_m$	Input relation(s)
A	Projection attribute
$\mathbf{dom}(A)$	Domain of attribute A
ϕ	Selection condition
N	Number of tuples in R (for SP queries)
N_i	Number of tuples with $A = i$ where $i \in \mathbf{dom}(A)$
N_i^ϕ	Number of tuples with $A = i$ that pass ϕ
$N_\bowtie$	Join size $\lvert R_1 \bowtie \cdots \bowtie R_m \rvert$
D	NDV of A
D^ϕ	NDV of A with selection condition ϕ
$\hat{D}^\phi$	Estimator for D^ϕ
n	Sample budget
A_s	Set of sampled values from $\mathbf{dom}(A)$
p_i	The probability that value i is sampled
τ_i	Sample budget allocated for value i
n_i^ϕ	Number of sampled tuples with $A = i$ that pass ϕ where $i \in \mathbf{dom}(A)$
$\mu_i(\phi)$	Conditional probability of sampling a passing tuple for i, $\mathbf{Pr}[n_i^\phi \geq 1 \mid i \in A_s]$
ϕ°	Special filter where $N_i^{\phi^\circ} = 1$ for all $i \in \mathbf{dom}(A)$
μ_i	Abbreviation of $\mu_i(\phi^\circ) = \tau_i / N_i$
$\boldsymbol{N}, \boldsymbol{p}, \boldsymbol{\tau}, \boldsymbol{\mu}$	Vectors of above symbols

Table 1: Notation used in the paper.

1.2 Error Metric

Most algorithms for the standard NDV problem (*i.e.*, no selection condition) provide multiplicative error guarantees. However, there is a simple argument that such a guarantee is not possible for SP queries (hence also for SPJ queries), when the selection condition is only given at query time and can be arbitrary: Consider two selection conditions ϕ_0 and ϕ_1. ϕ_0 blocks all tuples in R, while ϕ_1 blocks all but one that is arbitrarily chosen. So $D^{\phi_0} = 0$ and $D^{\phi_1} = 1$. Thus, unless the synopsis has kept more than half of the tuples, it cannot distinguish the two cases with confidence more than $1/2$. Note that not being able to distinguish between 1 and 0 means that the multiplicative error is unbounded.

Therefore, we aim at designing synopses with additive errors. This helps circumvent the 0 vs 1 problem above, as an additive error of Err means that the synopsis can be insensitive to any query size smaller than Err. In fact, additive errors are more appropriate in many applications of query size estimation, in particular query optimization. In query optimization, the goal is actually not to find *the* optimal query plan, but to avoid bad plans [18]. A multiplicative error would require us to estimate small query sizes accurately, which is unnecessary, while incurring larger errors for large query sizes, which may correspond to plans we wish to avoid.

Specifically, we measure the accuracy of an estimator $\hat{D}^\phi$ for a particular selection condition ϕ by the standard Mean Squared Error (MSE), and:

$$\mathbf{MSE}[\hat{D}^\phi] = \mathbf{E}[(\hat{D}^\phi - D^\phi)^2] = \mathbf{Bias}^2[\hat{D}^\phi] + \mathbf{Var}[\hat{D}^\phi]\,.$$

Note that by Chebyshev inequality, an MSE can be translated into an additive error of $O\left(\sqrt{\mathbf{MSE}[\hat{D}^\phi]}\right)$ with arbitrary constant probability.

We would like the synopsis to handle arbitrary selection conditions given at query time with a guaranteed MSE. Thus, we measure the quality of a synopsis on the worst possible ϕ, *i.e.*, $\max_\phi \mathbf{MSE}[\hat{D}^\phi]$.

1.3 Distinct Sampling

Let $\mathbf{dom}(A)$ be the domain of the projection attribute A. All algorithms for the standard NDV problem are based on the idea of *distinct sampling*: For a parameter $0 < p < 1$, we take each $i \in \mathbf{dom}(A)$ into the sample with probability p, which is often implemented by checking if $h(i) \leq p$ for a random hash function $h : \mathbf{dom}(A) \to [0, 1]$. Let A_s be the set of distinct values of A sampled. Then, $|A_s|/p$ is a good estimator of D.

Now consider SP queries. As the selection condition ϕ is only given at query time and can take an arbitrary form, one natural idea is to augment A_s with additional tuples sampled from R. More precisely, for a parameter τ, we take τ tuples into the sample, randomly chosen from all tuples with $A = i$, for each $i \in A_s$; if there are less than τ tuples with $A = i$ for some i, all of them are taken. The sample can be collected easily in one pass over the data by running the reservoir sampling algorithm for each i with $h(i) \leq p$.

The estimator for SP queries is then changed to

$$\hat{D}^\phi = \frac{1}{p}\left|\{i \in A_s \mid n_i^\phi \geq 1\}\right| = \frac{1}{p}\sum_{i \in \mathbf{dom}(A)} \mathbf{I}[n_i^\phi \geq 1]\,, \qquad (2)$$

where $\mathbf{I}[\cdot]$ is the indicator function and n_i^ϕ is the number of tuples with $A = i$ in the sample that pass the condition ϕ. Indeed, this was the idea of the first and only work addressing size estimation for SP queries, due to Gibbons [19].

The key drawback of the simple sampling strategy above is that the same p and τ are applied for all distinct values in $\mathbf{dom}(A)$. While this makes sense for the standard NDV problem, since each distinct value in A contributes only once to the count no matter how many times it appears in R, it may lead to a sub-optimal solution for SP queries. The following example shows a scenario where it can be quadratically worse than the optimal sampling strategy in terms of MSE.

Example 1. Consider a table R with D distinct values in A. $\sqrt{D}$ of these distinct values are called *heavy* values, each of which has $3\sqrt{D}$ tuples, and the rest of the distinct values are called *light* values, each of which has 1 tuple. Note that the entire table R has $\sqrt{D} \cdot 3\sqrt{D} + (D - \sqrt{D}) \approx 4D$ tuples in total. Suppose the sample budget is $2D$. We apply the above strategy with parameter τ and p .

If $\tau > 2\sqrt{D}$, p cannot be larger than $3/4$, otherwise the expected sample size $\frac{9}{4}D - o(D)$ would be larger than the sample budget $2D$. Consider a selection condition ϕ that passes all the tuples of the light values and blocks all the rest (*i.e.*, blocks all heavy values). We have $\mathbf{Var}[\hat{D}^\phi] = \frac{1}{p^2} \cdot p(1-p) \cdot (D - \sqrt{D}) = \Omega(D)$ (as $p \le 3/4$ in this construction, $\frac{1}{p} - 1$ is larger than $1/3$).

If $\tau \le 2\sqrt{D}$, then consider a series of selection conditions $\phi(x)$ that passes x tuples for each of the heavy values, for some $1 \le x \le 3\sqrt{D}$, and blocks the rest. We have $D^{\phi(x)} = \sqrt{D}$, and $\mathbf{E}[\hat{D}^{\phi(x)}] = \mu(x)\sqrt{D}$, where $\mu(x) = 1 - \binom{3\sqrt{D}-x}{\tau} \Big/ \binom{3\sqrt{D}}{\tau}$ is the probability that at least one of the x tuples is sampled. Observing that x is unknown to the estimator and $\mu(x)$ can range from $\frac{\tau}{3\sqrt{D}}$ to 1, an uncertainty gap of at least $1/3$, so $\mathbf{Bias}[\hat{D}^{\phi(x)}]$ must be $\Omega(\sqrt{D})$ for some values of x, *i.e.*, $\max_x \mathbf{MSE}[\hat{D}^{\phi(x)}] = \Omega(D)$. Note that this bias cannot be removed by scaling up or down the estimator (2).

On the other hand, a better sampling strategy is the following: We sample all the light values , and sample each heavy value with probability $1/3$. If a value is sampled (heavy or light), we take all of its tuples. We correspondingly modify the estimator to

$$\hat{D}^\phi = \sum_{i \in \mathbf{dom}(A)} \frac{1}{p_i} \mathbf{I}[n_i^\phi \ge 1], \tag{3}$$

where p_i is the probability that i is sampled, *i.e.*, $p_i = 1$ for a light i and $p_i = 1/3$ for a heavy i. It can be verified that the (expected) sample size is $2D$, and $\mathbf{MSE}[\hat{D}^\phi] = \mathbf{Bias}^2[\hat{D}^\phi] + \mathbf{Var}[\hat{D}^\phi] = 0 + O(\sqrt{D}) = O(\sqrt{D})$.

1.4 Weighted Distinct Sampling

The example above suggests that, unlike the standard NDV problem where all distinct values are treated equally, SP queries call for a *weighted* sampling strategy according to the frequencies of the values, where the "weight" corresponds to the sample budget, *i.e.*, $p_i\tau_i$, allocated to each $i \in \mathbf{dom}(A)$.

More precisely, in this paper we study the *weighted distinct sampling* problem: For each $i \in \mathbf{dom}(A)$, we sample it with probability p_i. This can be implemented by checking if $h(i) \le p_i$ for a random hash function[2]. If it is sampled, then we take τ_i tuples into the sample, randomly chosen from all tuples with $A = i$. The goal is to minimize the worst-case MSE of the estimator (3). Denoting by N_i the number of tuples in R with $A = i$, then finding the optimal sampling strategy can be formulated as the following optimization problem (we use bold symbols to represent the vector forms of p_i, τ_i, N_i):

$$\begin{aligned} \underset{\boldsymbol{p},\boldsymbol{\tau}}{\text{minimize}} \quad & \max_\phi f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi) \\ \text{subject to} \quad & \mathbf{0} \prec \boldsymbol{p} \preceq \mathbf{1}, \quad \mathbf{0} \preceq \boldsymbol{\tau} \preceq \boldsymbol{N}, \\ & \boldsymbol{p} \cdot \boldsymbol{\tau} \le n. \end{aligned} \tag{4}$$

Here $f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi) = \mathbf{MSE}[\hat{D}^\phi]$, $\prec$ and $\preceq$ denote product (element-wise) order, and n is the given sample budget. So the last constraint ensures that the (expected) sample size does not exceed the budget. Note that we require $p_i > 0$ for all i, so that (3) is well defined. If a sampling strategy does not want to sample a particular i, it can set $\tau_i = 0$ instead.

Intuitively, higher sample budgets should be allocated to values i's with higher N_i's; indeed, determining this relationship quantitatively is a key technical problem we solve in this paper. To contrast, we call the sampling strategy with the same p and τ for all i *uniform distinct sampling*, which simply allocates the same sample budget to all i.

1.5 Our Contributions

In addition to proposing the idea of weighted distinct sampling, we make the following technical contributions in this paper:

- We design an $O(D)$-time algorithm that, given the frequency vector $\boldsymbol{N}$ (in sorted order), finds a near-optimal solution to the optimization problem (4). The solution found by the algorithm is larger than the optimum by only a lower order term. Note that after we have solved (4) for $\boldsymbol{p}$ and $\boldsymbol{\tau}$, the sample can be collected in one pass over the data using a hash function h and reservoir sampling.
- As illustrated by Example 1, weighted distinct sampling works better on more skewed data. We substantiate this intuition by analyzing its MSE against the Zipfian distribution, which quantitatively shows that the MSE reduces as data skewness increases.
- We also show that our sampling strategy has a worst-case MSE of $O(\min\{DN/n, D^2\})$ for SP queries on *any* database instance, while Gibbons' uniform distinct sampling strategy as described in [19] may have an MSE of $\Omega(D^2)$ on certain instances and queries, even with a sample size of $n = \Theta(N)$. In this case, our MSE is quadratically better than that of uniform distinct sampling.
- The straightforward way to dealing with SPJ queries is to first compute the join $R_1 \bowtie \cdots \bowtie R_m$, and then build the SP-query synopsis on the join results. This can be expensive, especially for multi-way cyclic joins. We extend our sampling strategy to a random walk based algorithm that can collect the sample much more efficiently, while incurring a small loss to the accuracy.

[2] Our analysis only requires h to be taken from a pairwise-independent hash family.

- Finally, we conducted an extensive experimental study to evaluate these algorithms on synthetic, benchmark, and real data, giving recommendations to their practical use.

2 RELATED WORK

Cardinality estimation is a fundamental problem that has been studied extensively. However, most previous work only studied each relational operator in isolation.

As mentioned, the projection operator acting on its own is the standard NDV problem. This problem has a long history and is well understood by now [5, 6, 11, 13–15, 30, 41]. None of them works in conjunction with selection and/or joins, though, except the distinct sampling algorithm [19], which was reviewed in Section 1.3.

Estimating the result size of a selection operator is the classical selectivity estimation problem, and has also been widely studied. When the selection condition ϕ can be arbitrary, however, the only way to estimate its selectivity is by random sampling, which is also the approach we take in this paper. On the other hand, if ϕ is restricted to a particular form, most commonly a range condition, a host of techniques have been proposed, including sampling [7, 26, 39], histograms [20, 21, 24, 28, 34, 35, 40], and quantiles [16, 22, 23, 33].

Join size estimation is another well studied problem, and there are two main categories: sketch-based [4, 9, 10, 37] and sample-based [2, 8, 12, 17, 42]. The former usually offers better accuracy, but cannot support selection conditions. The latter may incur larger errors compared with sketch-based synopses, but can often incorporate selection conditions easily. However, none of these techniques can handle SPJ queries.

Query size estimation falls under the general umbrella of *approximate query processing (AQP)* [3, 27, 29, 32, 38, 43]. AQP techniques generally fall into two categories: query-time sampling and pre-computed samples. The former takes samples according to the query, so can often return more accurate estimates, but the downside is that it needs to access the entire database at query time, which may not be desirable for use in query optimizers for large-scale distributed databases. Our sampling method falls into the category of pre-computed samples, and aims at producing query size estimates by only examining the pre-computed sample at query time.

3 OPTIMAL SAMPLING STRATEGY

In this section, we provide a near-optimal solution to problem (4) in Section 1.4 . Without loss of generality, we assume $\mathbf{dom}(A) = \{1, \ldots, D\}$ and their frequencies N_i are given in ascending order. Denote the optimal solution to problem (4) by $(\boldsymbol{p}^*, \boldsymbol{\tau}^*)$ and the optimal worst-case MSE by OPT, namely,

$$\mathrm{OPT} = \max_{\phi} f_0(\boldsymbol{p}^*, \boldsymbol{\tau}^*, \phi) = \min_{\boldsymbol{p}, \boldsymbol{\tau}} \max_{\phi} f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi)\,.$$

The main result of this section is the following characterization of a near-optimal solution to problem (4):

Theorem 3.1. *There exists* $0 \le M \le D$, $\kappa > 0$ *and*

$$p_i = \min\left\{1, \frac{\kappa}{\sqrt{N_i}}\right\}, \quad \tau_i = \begin{cases} N_i, & \text{if } i \le M, \\ 0, & \text{if } i > M, \end{cases}$$

such that $\boldsymbol{p} \cdot \boldsymbol{\tau} \le n$ *and*

$$\max_{\phi} f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi) \le \mathrm{OPT} + \sqrt{\mathrm{OPT}} + 1\,.$$

The key message in this result is that the sample budget $p_i\tau_i$ should be proportional to $\sqrt{N_i}$, except for a few very large N_i's, which we simply ignore (by setting $\tau_i = 0$). Another way of looking at this result is that each tuple with $A = i$ has a per-tuple sample budget of $p_i\tau_i/N_i \sim 1/\sqrt{N_i}$, *i.e.*, inversely proportional to $\sqrt{N_i}$. Fundamentally, this is the balance point between two forces in a tug of war: On the one hand, a value $i \in \mathbf{dom}(A)$ will contribute to the SP query size as long as at least one tuple with $A = i$ passes the filter ϕ, so values with higher N_i are more likely to contribute. On the other hand, all N_i tuples with the same $A = i$ can only contribute one to the query size, no matter how many of them pass ϕ, so the per-tuple importance is actually less than those tuples with a smaller N_i. Below, we illustrate this idea with a more concrete example:

Example 2. In Table 2 we give a concrete example with $D = 10$ distinct values and show the near-optimal solutions under different sample budgets n. The values of κ and M are found by our algorithm to be discussed in Section 4. This example illustrates that for most values i in the middle, the sample budget $p_i\tau_i$ is proportional to $\sqrt{N_i}$. For very small number of corresponding tuples N_i's, their sampling probability p_i hits the ceiling of 1 and we take all their tuples; for very large N_i's, we decide not to sample them at all, the intuition being that the sample budget they would consume is so high that it actually makes more sense to ignore them and accept the error thus caused.

Now consider applying a filter ϕ° that passes one tuple for each value i (so $D^{\phi^\circ} = 10$). Suppose sample budget $n = 20$ and $A_s = \{1, 2, 3, 4, 5, 6, 7, 9\}$, our estimator will be

$$\hat{D}^{\phi^\circ} = \sum_{i \in A_s} p_i^{-1} = 6 + 0.93^{-1} + 0.57^{-1} = 8.82\,.$$

In the rest of this section we prove Theorem 3.1. The proof involves a series of transformations starting from problem (4), each of which either preserves optimality or introduces a quantifiable small error.

3.1 Rewriting the Optimization Problem

Problem (4) is a minimax problem. To convert it to a regular optimization problem, we need to write $\max_\phi f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi)$ into an explicit form of $\boldsymbol{p}$ and $\boldsymbol{\tau}$, *i.e.*, to identify the worst ϕ that maximizes f_0 for *any* given $\boldsymbol{p}$ and $\boldsymbol{\tau}$. However, this is another optimization problem and it has no closed-form solutions. To get around this difficulty, we first minimize $f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi^\circ)$ for a particular ϕ°, *i.e.*, find $(\boldsymbol{p}^\circ, \boldsymbol{\tau}^\circ) = \arg\min_{\boldsymbol{p}, \boldsymbol{\tau}} f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi^\circ)$. This is a regular optimization problem. Then we show that $(\boldsymbol{p}^\circ, \boldsymbol{\tau}^\circ)$ must also be an optimal solution to problem (4).

We can write $\hat{D}^\phi$ in Equation (3) as

$$\hat{D}^\phi = \sum_{i=1}^{D} X_i\,, \quad \text{where } X_i = \frac{1}{p_i} \cdot \mathbf{I}[n_i^\phi \ge 1]\,.$$

Denote the expectation of X_i as a function of ϕ by

$$\mu_i(\phi) = \mathbf{E}[X_i] = \frac{1}{p_i} \cdot p_i \cdot \mathbf{Pr}[n_i^\phi \ge 1 \mid i \in A_s]$$

Value	$i=1,2,3$			$i=4,5,6$			$i=7$			$i=8$			$i=9$			$i=10$			
Frequency	$N_1=N_2=N_3=1$			$N_4=N_5=N_6=2$			$N_7=3$			$N_8=5$			$N_9=8$			$N_{10}=20$			
	p_i	τ_i	$p_i\tau_i$	p_i	τ_i	$p_i\tau_i$	p_7	τ_7	$p_7\tau_7$	p_8	τ_8	$p_8\tau_8$	p_9	τ_9	$p_9\tau_9$	p_{10}	τ_{10}	$p_{10}\tau_{10}$	M
$n=10$	0.89	1	0.89	0.63	2	1.26	0.51	3	1.54	0.40	5	1.99	0.32	0	0	0.20	0	0	8
$n=15$	1	1	1	1	2	2	0.87	3	2.62	0.68	5	3.38	0.53	0	0	0.34	0	0	8
$n=20$	1	1	1	1	2	2	0.93	3	2.80	0.72	5	3.62	0.57	8	4.58	0.36	0	0	9

Table 2: Example of near-optimal solutions

$$= 1 - \binom{N_i - N_i^\phi}{\tau_i} \Big/ \binom{N_i}{\tau_i},$$

where N_i^ϕ is the number of tuples of value i that pass ϕ. This is equivalent to the conditional probability of sampling at least one passing tuple for i given i is sampled ($i \in A_s$). Note that $\mu_i(\phi)$ is an increasing function of N_i^ϕ (for any fixed τ_i), and $0 \le \mu_i(\phi) \le 1$.

Since the values in $\mathbf{dom}(A)$ are sampled pairwise independently, we have

$$\mathbf{E}[\hat{D}^\phi] = \sum_{i=1}^{D} \mu_i(\phi), \quad \mathbf{Var}[\hat{D}^\phi] = \sum_{i=1}^{D} \left[\frac{\mu_i(\phi)}{p_i} - \mu_i^2(\phi) \right],$$

$$\mathbf{MSE}[\hat{D}^\phi] = \left[D^\phi - \sum_{i=1}^{D} \mu_i(\phi) \right]^2 + \sum_{i=1}^{D} \left[\frac{\mu_i(\phi)}{p_i} - \mu_i^2(\phi) \right].$$

Note that for a value i with $N_i^\phi = 0$, we have $\mu_i(\phi) = 0$, so it has no contribution to the $\mathbf{Var}[\hat{D}^\phi]$. The bias can be written as $\mathbf{Bias}[\hat{D}^\phi] = \sum_{i=1}^{D} (\mathbf{I}[N_i^\phi \ge 1] - \mu_i(\phi))$, so the i with $N_i^\phi = 0$ has no contribution to the bias either. Therefore, the filter ϕ that maximizes $f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi)$ must have $N_i^\phi \ge 1$ for all i.

Consider the special filter ϕ° where $N_i^{\phi^\circ} = 1$ for all value i, then the expectation $\mu_i(\phi^\circ) = \tau_i / N_i$, which we abbreviate to μ_i. μ_i is also the fraction of tuples we take from i given $i \in A_s$. With ϕ fixed to ϕ°, we rewrite problem (4) to the following optimization problem, where τ_i is replaced by $\mu_i N_i$:

$$\begin{aligned} \underset{\boldsymbol{p},\boldsymbol{\mu}}{\text{minimize}} \quad & f_1(\boldsymbol{p}, \boldsymbol{\mu}) \\ \text{subject to} \quad & \mathbf{0} < \boldsymbol{p} \le \mathbf{1}, \quad \mathbf{0} \le \boldsymbol{\mu} \le \mathbf{1}, \\ & \sum_{i=1}^{D} p_i \mu_i N_i \le n, \end{aligned} \tag{5}$$

where

$$f_1(\boldsymbol{p}, \boldsymbol{\mu}) = f_0(\boldsymbol{p}, \boldsymbol{\tau}, \phi^\circ) = \left(D - \sum_{i=1}^{D} \mu_i \right)^2 + \sum_{i=1}^{D} \left(\frac{\mu_i}{p_i} - \mu_i^2 \right).$$

3.2 Relating the Two Optimization Problems

Now we prove that an optimal solution to problem (5) is also an optimal solution to problem (4).

Lemma 3.2 shows the existence of an optimal solution to problem (5) where p_i is not set infinitely close to 0. Then, we present Lemma 3.3 and 3.4, each of which reveals some properties of such an optimal solution.

In this paper, we omit the proofs of all lemmas and theorems, which can be found in the full version of the paper [36].

Lemma 3.2. *The objective function of problem* (5) *can take an optimal value in the feasible domain.*

Lemma 3.3. *Suppose* $(\boldsymbol{p}, \boldsymbol{\mu})$ *is an optimal solution to problem* (5)*, then for any* i *with* $\mu_i > 0$*,* $p_i \ge (D - \sum_{j \ne i} \mu_j)^{-1}$*. Besides, if* $0 < \mu_i < 1$*, then* $p_i = (D - \sum_{j \ne i} \mu_j)^{-1}$*.*

Lemma 3.4. *Suppose* $(\boldsymbol{p}, \boldsymbol{\mu})$ *is an optimal solution to problem* (5)*, then* $\mu_i = 0$ *or* $\mu_i = 1$ *for all but one* i*.*

With the above lemmas, we can now argue that solving problem (5) on the particular ϕ° is enough for solving the original problem (4).

Theorem 3.5. *Let* $(\boldsymbol{p}^*, \boldsymbol{\mu}^*)$ *be an optimal solution to problem* (5)*, then* $(\boldsymbol{p}^*, \boldsymbol{\tau}^*)$ *is an optimal solution to problem* (4) *in Section 1.4 where* $\tau_i^* = \mu_i^* N_i$ *for all* i*.*

3.3 A Near-Optimal Solution

Problem (5) has linear constraints and a closed-form objective function, but the objective function is non-convex. General solvers for such optimization problems only return a local-optimal solution without any guarantee on its global optimality. Furthermore, they usually use iterative algorithms that can be slow on large instances. In this section, we exploit the special structure of problem (5) and transform it to an easier one that admits a near-optimal solution.

By Lemma 3.4, for any optimal solution, there is at most one value i with fraction of tuples $0 < \mu_i < 1$. For simplicity, we also sets $\mu_i = 0$ for this i, *i.e.*, we only consider solutions such that $\mu_i \in \{0, 1\}$ for all i. The problem now becomes the following:

$$\begin{aligned} \underset{\boldsymbol{p},\boldsymbol{\mu}}{\text{minimize}} \quad & f_1(\boldsymbol{p}, \boldsymbol{\mu}) \\ \text{subject to} \quad & \mathbf{0} < \boldsymbol{p} \le \mathbf{1}, \quad \boldsymbol{\mu} \in \{0, 1\}^D, \\ & \sum_{i=1}^{D} p_i \mu_i N_i \le n. \end{aligned} \tag{6}$$

We quantify its near-optimality by the following lemma.

Lemma 3.6. *Let* $(\boldsymbol{p}, \boldsymbol{\mu})$ *be the optimal solution to problem* (6)*, and* $\mathrm{OPT} = f_1(\boldsymbol{p}^*, \boldsymbol{\mu}^*)$ *be the optimal objective value of problem* (5)*, then*

$$f_1(\boldsymbol{p}, \boldsymbol{\mu}) \le \mathrm{OPT} + \sqrt{\mathrm{OPT}} + 1.$$

Now we focus on solving problem (6). We first give a characterization of its optimal solutions, similar to Lemma 3.4.

Lemma 3.7. *There exists an optimal solution* $(\boldsymbol{p}, \boldsymbol{\mu})$ *to problem* (6) *such that* $\mu_i = 1$ *for* $i \le M$ *and* $\mu_i = 0$ *for* $i > M$*, for some* $M \in \{0, 1, \ldots, D\}$*.*

Thus the problem of finding $\boldsymbol{\mu}$ is reduced to finding $M = \|\boldsymbol{\mu}\|_1$, whose possible values are $0, \ldots, D$. Suppose we have obtained an

optimal M. We still need to find $\boldsymbol{p}$. For any given M, this is another optimization problem:

$$\begin{aligned} \underset{\boldsymbol{p}}{\text{minimize}} \quad & f_2(\boldsymbol{p}; M) = \sum_{i=1}^{M} \frac{1}{p_i} \\ \text{subject to} \quad & \mathbf{0} \prec \boldsymbol{p} \preceq \mathbf{1}\,, \quad \sum_{i=1}^{M} p_i N_i \le n\,. \end{aligned} \tag{7}$$

Note that f_2 and f_1 differ only in terms independent of $\boldsymbol{p}$, so it is sufficient to minimize f_2. Without the constraint $\boldsymbol{p} \preceq \mathbf{1}$, problem (7) could be solved easily by the Cauchy-Schwartz inequality, and the optimal solution would be $p_i \sim 1/\sqrt{N_i}$. The constraint $\boldsymbol{p} \preceq \mathbf{1}$ has introduced some technical complications, but we still manage to get the following closed-form solution.

Lemma 3.8. *An optimal solution to problem* (7) *is*

$$p_i = \min\left\{1, \frac{\kappa}{\sqrt{N_i}}\right\},$$

where $\kappa = \dfrac{n - \sum_{j=1}^{K} N_j}{\sum_{j=K+1}^{M} \sqrt{N_j}}$, *where* K *is the largest integer such that*

$$n - \sum_{i=1}^{K} N_i > \sqrt{N_K} \sum_{i=K+1}^{M} \sqrt{N_i}\,. \tag{8}$$

If $n \le \sqrt{N_1} \sum_{i=1}^{M} \sqrt{N_i}$, *we set* $K = 0$.

Combining Theorem 3.5, Lemma 3.6, 3.7, and 3.8, we obtain Theorem 3.1. In addition, these lemmas yield a method to find M and κ, although not very efficient: For each $M = 0, 1, \ldots, D$, we use Lemma 3.8 to find the optimal $\boldsymbol{p}$. Then we choose the best M.

4 EFFICIENT ALGORITHMS

Recall that we assume $N_1 \le N_2 \le \cdots \le N_D$ are given in sorted order. The straightforward algorithm described above has a running time of $O(D^2)$. For each M, we spend $O(D)$ time to find the corresponding K (which we denote as K_M to stress that it depends on M) and $\boldsymbol{p}$, as specified in Lemma 3.8. Then we calculate the objective function

$$f_3(M) = (D - M)^2 + \frac{(\sum_{i=K_M+1}^{M} \sqrt{N_i})^2}{n - \sum_{i=1}^{K_M} N_i} + K_M - M\,,$$

and return the M with the minimum $f_3(M)$.

In this section, we first give an improved algorithm with running time $O(D)$. Then we also show how the algorithm can be made to work if only a histogram is given in lieu of all value frequencies.

4.1 An $O(D)$-time Algorithm

We see that the bottleneck of the above algorithm is in finding all the K_M's. To improve it, we make the key observation that K_M is non-increasing with respect to M [36] . This means that we can find K_M incrementally for $M = M_0, \ldots, D$, starting from $K = M_0$. This suggests the $O(D)$-time algorithm as described in Algorithm 1. After M^* and K_{M^*} are found, we can calculate κ, $\boldsymbol{p}$ and $\boldsymbol{\tau}$ in $O(D)$ time following Lemma 3.8.

Algorithm 1: Find M^* and $K^* = K_{M^*}$

```
Input  : D, {N_i}_{i=1}^D and n
Output : M* and K* = K_{M*}
M_0 ← 0, sum ← 0;
while M_0 ≤ D and n ≥ sum do
    M_0 ← M_0 + 1, sum ← sum + N_{M_0};
sum ← sum − N_{M_0}, M_0 ← M_0 − 1, M* ← 0, K* ← 0;
min_mse ← +∞, K ← M_0, sum_sqrt ← 0;
for M ← M_0 to D do
    while K > 0 and n − sum ≤ sqrt(N_K) · sum_sqrt do
        sum ← sum − N_K, sum_sqrt ← sum_sqrt + sqrt(N_K);
        K ← K − 1;
    mse ← (D − M)^2 + sum_sqrt^2/(n − sum) + K − M;
    if mse < min_mse then
        min_mse ← mse, M* ← M, K* ← K;
    sum_sqrt ← sum_sqrt + sqrt(N_M);
return M* and K*;
```

Example 3. We illustrate how Algorithm 1 works using the example in Table 2 with $n = 20$. The algorithm first finds $M_0 = 8$, as the first 8 frequencies add up to 17, while adding N_9 would exceed n.

When $M = 8$, we find $K_M = 8$, and $f_3(M) = (10 - 8)^2 + 0 = 4$.

When $M = 9$, we reduce K_M to 6, and update $f_3(M)$ as

$$f_3(9) = (10 - 9)^2 + \frac{(\sqrt{3} + \sqrt{5} + \sqrt{8})^2}{20 - 9} + 6 - 9 = 2.20\,.$$

When $M = 10$, we reduce K_M to 3, while $f_3(M)$ is updated to

$$f_3(10) = 0^2 + \frac{(3\sqrt{2} + \sqrt{3} + \sqrt{5} + \sqrt{20})^2}{20 - 3} + 3 - 10 = 2.46\,.$$

The algorithm maintains the prefix sum $\sum N_i$ and the suffix sum $\sum \sqrt{N_i}$ so that both K_M and $f_3(M)$ can be updated incrementally. The optimal value is therefore taken at $M = 9$.

Theorem 4.1. *There is an* $O(D)$*-time algorithm that, given* $\{N_i\}$ *in ascending order, finds a near-optimal solution to problem* (4).

4.2 Histogram Approximation

If not all frequencies N_i are available, our algorithm can also work with a heavy hitters histogram, which is commonly maintained by database systems. A heavy hitters histogram only maintains the H values in $\mathbf{dom}(A)$ with the largest frequencies, namely, $N_{D-H+1}, \ldots, N_D$. Then, for the low-frequency values, we simply assume that they have equal frequencies, *i.e.*, we use $\hat{N_L} = (N - \sum_{j=D-H+1}^{D} N_j)/(D - H)$ as an approximation of N_i for $i = 1, \ldots, D - H$. The running time improves to $O(H)$ [36] .

5 MSE ANALYSIS

Our algorithm finds a near-optimal weighted distinct sampling strategy. However, we still need to answer the question: how good is its MSE, in particular, in comparison with uniform distinct sampling [19]? In this section, we give a theoretical analysis; results from empirical studies will be presented in Section 7.

Worst-case data distribution. We first show that the MSE of our optimal weighted distinct sampling strategy is $O(\min\{DN/n, D^2\})$ in the worst case.

Theorem 5.1. *For any input data and SP query, the MSE of the near-optimal weighted distinct sampling strategy found by our algorithm is bounded by* $O(\min\{DN/n, D^2\})$.

This bound is tight in the worst case. The hardest input distribution is simply the uniform distribution. In fact, it is not surprising that uniform distribution is the worst-case input for weighted distinct sampling; on uniform data, all distinct values are equally important, losing the advantage of weighted sampling.

Analysis of Gibbons' algorithm. On the other hand, the MSE of the uniform distinct sampling strategy of Gibbons [19] is $\Omega(D^2)$ on certain datasets and queries, even with a sample budget of $n = \Omega(N)$, as we show in [36]. Recall that the worst-case MSE of our weighted distinct sampling strategy is $O(DN/n)$ when $n \geq N/D$. Therefore, for any sample size $n = \omega(N/D)$, we obtain an asymptotic improvement over Gibbons' uniform distinct sampling algorithm, and the improvement is quadratic when $n = \Theta(N)$.

Analysis on Zipfian distribution. We have also conducted analysis on the Zipfian distribution, which shows that the MSE reduces as data skewness increases [36].

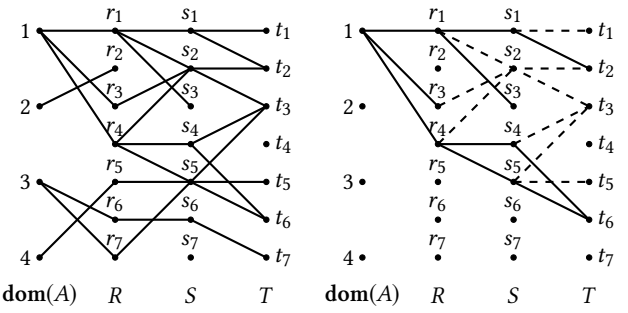
6 SPJ QUERIES

A straightforward way of handling SPJ queries is to first compute the join $R_1 \bowtie \cdots \bowtie R_m$, and then run our optimal weighted distinct sampling algorithm on the join results. This way, an SPJ query becomes an SP query, and all our MSE analysis from the previous section will hold. In particular, we can guarantee a maximum MSE of $O(DN_\bowtie/n)$ for SPJ queries over any database instance and any ϕ, where $N_\bowtie = |R_1 \bowtie \cdots \bowtie R_m|$ is the join size. However, computing the full join and running the algorithm on all join results can be very expensive, especially for multi-way joins. In this section, we describe a random walk based algorithm that collects the sample in just one pass over the data, without performing the join. The resulting sample is not optimal, though, but we demonstrate through our experiments that the accuracy loss is still small enough in comparison to the significant savings in the sample construction cost.

6.1 Random Walk Algorithm

Our random walk algorithm is inspired by the Wander Join algorithm of Li *et al.* [32]. However, Wander Join does not support projections. Thus, we make the following modifications. To make things more concrete, consider a 3-way line join $R \bowtie S \bowtie T$, where the projection attribute A belongs to R. We conceptually add one more table that consists of a single attribute and D tuples, one for each distinct value in $\mathbf{dom}(A)$ (see Figure 1a). For each tuple $r \in R$ with $\pi_A(r) = i$, we add an edge between value i and tuple r. We also add an edge between any two tuples that can join. Then, the idea is to start the random walks from the conceptual table $\mathbf{dom}(A)$ and walk towards T, where each random walk yields a sample from the join results.

Another major difference is that Wander Join [32] takes samples at query time, which requires indexes to be available, and results in a lot of random accesses in the database at query time. This does not fit the scenario of query optimization, which is the main application of query size estimation. Here, we would like to have a pre-computed small synopsis from which the query size can be estimated quickly *without having to rely on indexes.*



(a) Adding conceptual column. **(b) Distinct sampling from join.**

Figure 1: Sampling by random walks.

Below, we describe our random walk algorithm for obtaining a weighted distinct sample from a multi-way join. It does not need indexes, either, and the sample can be collected in one pass over each relation.

We first compute the optimal sampling strategies $\boldsymbol{p}, \boldsymbol{\tau}$ using the frequencies of values of $\mathbf{dom}(A)$ in R as described in Section 3,[3] and collect a distinct sample on R. Suppose $i = 1$ has been sampled, and $\tau_1 = 3$ tuples r from R have been sampled with $\pi_A(r) = i$. Next, we extend each of the τ_1 tuples into a join result, sampled from all join results whose projection on A is i. This corresponds to the subgraph rooted at $i = 1$ (see Figure 1b).

As we may encounter a "deadend" when extending a tuple into a join result, such as (r_1, s_3) in Figure 1b, we take the following two-phase approach. We use a space budget of $\tau'_i = c\tau_i$ for each distinct $i \in \mathbf{dom}(A)$ sampled from R, where $c > 1$ is an empirical constant scaling parameter. We are in phase one as long as the space budget is not reached. In phase one, we try to obtain a random sample of size up to τ'_1 from the partial join results $\sigma_{A=1}(R) \bowtie S$, using reservoir sampling. Consider $i = 1$ and suppose $\tau'_1 = 4$ in Figure 1b. Our initial random walks consist of 3 tuples in R, which is smaller than $\tau'_1 = 4$, so we are in phase one. Then when scanning S, for each $s_i \in S$, we join it with the current sample from R, and feed the join results to reservoir sampling. If $|\sigma_{A=1}(R) \bowtie S| \leq \tau'_1$, we take all the join results into the sample and remain in phase one; otherwise we go to phase two. In Figure 1b, $|\sigma_{A=1}(R) \bowtie S| = 7 > 4$, so only 4 partial join results have been sampled (solid lines), and we move onto phase two when scanning T.

We will be in phase two when scanning T. In phase two, for each partial random walk (r, s), we extend it to one tuple in T, randomly chosen from all those that join with s (if such tuples exist). An example is shown in Figure 1b, where 3 join results, (r_1, s_1, t_2), (r_4, s_4, t_6) and (r_4, s_5, t_6), have been eventually sampled. Note that (r_1, s_3) fails to extend to T, which is why we use a larger space budget τ'_1. In the end, if more than τ_1 join results have been obtained, we sub-sample them down to τ_1.

[3] Essentially, we use the frequencies of values in R to approximate their frequencies in the full join results, which is why we cannot guarantee optimality of the sampling strategy.

For general multi-way joins, we reorder relations so that each relation (except the first one) shares at least one common attribute with a previous relation. We then scan the relations one by one in this order. When scanning, we only take tuples that join with the corresponding tuples sampled from all previous relations so far. For example, if there is also a join condition between R and T, then when extending a partial random walk (r, s) to a $t \in T$, we only sample from all tuples $t \in T$ such that t joins with both r and s.

Since we scan the relations one by one, at the end of processing each relation, we get a sample for the prefix of the join. For example in Figure 1b, the four paths $(r_1, s_1), (r_1, s_3), (r_4, s_4), (r_4, s_5)$ is a sample for queries on $R \bowtie S$. By storing all the random walks (including those not generating to a join result), the same sample collected for $R_1 \bowtie \cdots \bowtie R_m$ can be used in estimating any prefix of the join $R_1 \bowtie \cdots \bowtie R_j$ where $j \le m$.

Finally, we run the algorithm above for all $i \in \mathbf{dom}(A)$ sampled in the first relation in parallel. So the algorithm makes one pass over each relation (*i.e.*, its time complexity is *only linear to the sum of all relation sizes from the join*), using space $cn = O(n)$, where n is the total sample budget .

6.2 Bias Reduction

Another technical difference between our SPJ algorithm and SP algorithm as described in Section 3 is that, in the SP algorithm, τ_i is either N_i or 0, so individual estimators have no bias. In the random walk algorithm, however, for a sampled $i \in A_S$, only up to τ_i of its join results are sampled. As such, there is a downward bias in the estimator. To see this problem more clearly, for a selection condition ϕ, let $\mu_i(\phi)$ be the probability that any sampled tuples for i pass ϕ, given i is sampled. Then $\hat{D}^\phi = \sum_{i=1}^{D} Y_i$ would be an unbiased estimator of D^ϕ, where $Y_i = \frac{\mathbf{I}[n_i^\phi \ge 1]}{p_i \mu_i(\phi)}$. However, as illustrated in Example 1, the problem is that $\mu_i(\phi)$ is unknown[4]. Due to the uncertainty in $\mu_i(\phi)$, bias cannot be eliminated. Nevertheless, we introduce the following heuristic to reduce it to some extent.

Our idea is to reduce the bias based on the value of n_i^ϕ, which is known. If $n_i^\phi \ge 2$, then N_i^ϕ is also likely large, so the chance that some tuple passes ϕ is also large, so we take $\mu_i(\phi) = 1$.

If $n_i^\phi = 1$, then $\mu_i(\phi)$ is likely small. We observe that $p_t \le \mu_i(\phi) \le 1$, where p_t is the probability that the only tuple t in the sample that passes ϕ was sampled when the sample was initially collected, conditioned upon i being sampled. Then we strike a balance between the two extremes by taking geometric mean and setting $\mu_i(\phi) \approx \sqrt{p_t}$.

To summarize, our bias-reduced estimator is $\hat{D}^\phi = \sum_{i=1}^{D} \hat{Y}_i$, where

$$\hat{Y}_i = \begin{cases} 0, & \text{if } n_i^\phi = 0\,, \\ \frac{1}{p_i \sqrt{p_t}}, & \text{if } n_i^\phi = 1\,, \\ \frac{1}{p_i}, & \text{if } n_i^\phi > 1\,. \end{cases}$$

The remaining issue is how to compute p_t. Recall our two-phase sample collection algorithm. If the algorithm terminates during phase one, then p_t is the same for all tuples with $A = i$:

$$p_t = \frac{\tau_i'}{|\sigma_{A=i}(R_1) \bowtie \cdots \bowtie R_m|}\,.$$

Note that the partial join size $|\sigma_{A=i}(R_1) \bowtie \cdots \bowtie R_m|$ can be computed when we scan R_m.

If the algorithm goes on to phase two, then p_t might be different for different t. Suppose phase one terminates after scanning R_s, for some $s < m$. Consider a tuple $t = r_1 \bowtie r_2 \bowtie \cdots \bowtie r_m$ sampled in the end, where $r_j \in R_j$. First, $(r_1, \ldots, r_s)$ are sampled during phase one, so the probability that it is sampled is $\frac{\tau_i'}{|\sigma_{A=i}(R_1) \bowtie \cdots \bowtie R_s|}$. During phase two, we perform a series of random walks, where in each step, we pick a tuple uniformly at random from all tuples that join with previous tuples, so

$$p_t = \frac{\tau_i'}{|\sigma_{A=i}(R_1) \bowtie \cdots \bowtie R_s|} \cdot \prod_{j=s+1}^{m} \frac{1}{|r_1 \bowtie \cdots \bowtie r_{j-1} \bowtie R_j|}\,. \quad (9)$$

For example, in Figure 1b, for $t = (r_4, s_5, t_6)$, $p_t = \frac{4}{7} \cdot \frac{1}{3} = \frac{4}{21}$.

Finally, note that p_t's for all the sampled tuples t can be computed easily during the sample collection process.

7 EXPERIMENTAL EVALUATION

In this section, we report the results from an extensive experimental evaluation on various cardinality estimation methods for both SP queries and SPJ queries.

7.1 Methods Evaluated

Exact. The exact method has been implemented to provide a reference. Note that computing the exact query size for SPJ queries requires the entire data set and is very slow.

Upper Bound (UB). As mentioned in Section 1, a simple and commonly used method for estimating the size of SPJ queries is to estimate the NDV of A and the size of the remaining query (without the projection) separately, and then return the smaller of the two. In our evaluation, we implement this method in its best-case scenario, *i.e.*, we use the exact NDV and the remaining query size, and return the smaller of the two.

Uniform Distinct Sampling (UDS). Uniform Distinct Sampling (UDS) refers to the algorithm described in [19] for estimating SP query sizes. For SPJ queries, we compute the full join first, and then run this algorithm on the join results. MurmurHash3, implemented in SMHasher[5], is used as the hash function for deciding if a value in $\mathbf{dom}(A)$ is sampled.

Weighted Distinct Sampling (WDS). This refers to our weighted distinct sampling algorithm using the optimal sampling strategy as described in Section 3. For SPJ queries, similar to UDS, we compute the full join and run the algorithm on the join results.

Histogram Approximation (HISTO). We have also implemented the algorithm in Section 4.2, where the algorithm is only given a heavy hitters histogram as an approximation of the value frequencies. The histogram contains the frequencies of the top 10% most frequent values.

[4] More precisely, it is always 1 if all tuples with $A = i$ are sampled; otherwise, it depends on N_i^ϕ, which is unknown.

[5] https://github.com/aappleby/smhasher

Random Walk Algorithm (RW). The random walk algorithm in Section 6 can be considered an approximate version of WDS for SPJ queries. The sample can be collected in one pass over the data, at the expense of some loss in accuracy.

We implemented all the algorithms above in C++. Each relation is stored as a `std::vector` of `std::tuple`'s. The experiments were carried out on a machine with an Intel(R) 2.10GHz Xeon(R) Silver 4116 CPU running CentOS Linux 7.7. The memory is large enough to hold all input relations and join results. For each dataset, we run the algorithm 30 times using independent random seeds, and report the root of the average MSE (RMSE) from these 30 runs. The sample budget is 1% of the data size by default.

We used a wide range of datasets in our evaluation, including synthetic datasets with different skewness, the TPC-DS benchmark, and two real-world datasets. Below, we describe these datasets and queries in turn, followed by our experimental findings.

7.2 Results on Synthetic Datasets

Datasets. Our theoretical analysis shows that WDS works better on more skewed data, so we used two synthetic datasets, one uniform and the other following a Zipfian distribution, to validate this claim. The table has roughly one million tuples. The uniform dataset contains $D = 1000$ distinct A-values, each appearing in 1000 tuples. The Zipfian data have a skewness factor $\alpha = 1.1$. The distinct values are integers ranging from 1 to 50000, and the frequency of value i is proportional to $1/i^{\alpha}$.

We introduced a second attribute B, which is used in the selection condition ϕ. We tested three conditions. The first one represents an average case that is considered easy, while the other two are designed to model worst-case scenarios.

(1) For both uniform and Zipfian data, ϕ_1 passes each tuple independently with the same probability p. Essentially, p corresponds to the selectivity of the query.
(2) ϕ_2 applies on the uniform dataset. First, we pick a random value from the D distinct values of A, and pass all its tuples. Then, we randomly take a set of $D/2$ values randomly from the remaining $D-1$ distinct values, and passes t tuples for each. Note that $D^{\phi_2} = D/2 + 1$. The smaller t is, the more difficult the query is considered, as it is more difficult to catch those passing tuples by sampling.
(3) ϕ_3 applies on the Zipfian dataset. It passes all tuples of the most frequent value. It also passes a single tuple for each value whose frequency is at least t. As with ϕ_2, the smaller t is, the more difficult the query will be.

Results on uniform data. Figure 2a shows the estimates of query ϕ_1 returned by various algorithms on the uniform dataset, where we vary the selectivity p. Thus, UB simply reports $\min\{D, pN\}$, which is shown as straight line turning at $p = D/N = 10^{-3}$. UDS and WDS are both sampling algorithms, so in addition to the mean, we also plot the 10% and 90% percentiles of the 30 runs. Figure 2b shows the R MSE of these algorithms. From these results, we see that on this easy query, UB actually performs quite well, while UDS is even worse than UB due to large underestimation. On the other hand, WDS has generally outperformed UB in this case. The performance of WDS is more stable across different values of p, while UB is a good estimator for either small p (when pN is a good estimate since very likely, no A-value has more than one tuple passing ϕ_1) or large p (when D is a good estimate, since almost every distinct A-value has at least one tuple passing ϕ_1).

On ϕ_2, which is a hard query, the situation becomes quite different (see Figure 2c and 2d). Since more than D tuples in total pass ϕ_2, UB always returns D, which is a bad estimate. On the other hand, the two sampling algorithms give more accurate estimates. Between the two, we see that WDS has stable performance across different values of t, and is always better than UDS. For larger values of t, the two have similar MSE (UDS has larger bias and smaller variance, while WDS has smaller bias but larger variance), but the accuracy of UDS drops as t gets smaller (the query gets more difficult).

Results on Zipfian data. Figure 2e to 2h show the results on the Zipfian dataset. Since Zipfian distribution produces a skewed dataset with a long tail, UB fails miserably; we cannot even plot its results on the hard query ϕ_3. The performance of HISTO is similar to WDS, showing that our algorithm still works well even if only a histogram is given. It even outperforms WDS on ϕ_3. This is because by computing the optimal solution based on the histogram, we set higher probabilities for more frequent values, which happens to suit the scenario of ϕ_3. Note that this does not contradict the optimality of the solution based on full frequency information, which is assuming the *worst* filter condition ϕ.

The comparison between UDS and WDS is interesting on the Zipfian dataset. We see from Figure 2e and 2f that on ϕ_1, which passes each tuple with equal probability, the two algorithms have very similar performance, while WDS significantly outperforms UDS on ϕ_3 across all values of t. These results are consistent with the design and analysis of WDS, which is aimed at minimizing the MSE over worst-case queries on skewed data. Meanwhile, it is also reassuring to see that, on easier, non-worst-case data and queries, WDS is no worse than its competitors.

7.3 Results on Benchmark Data

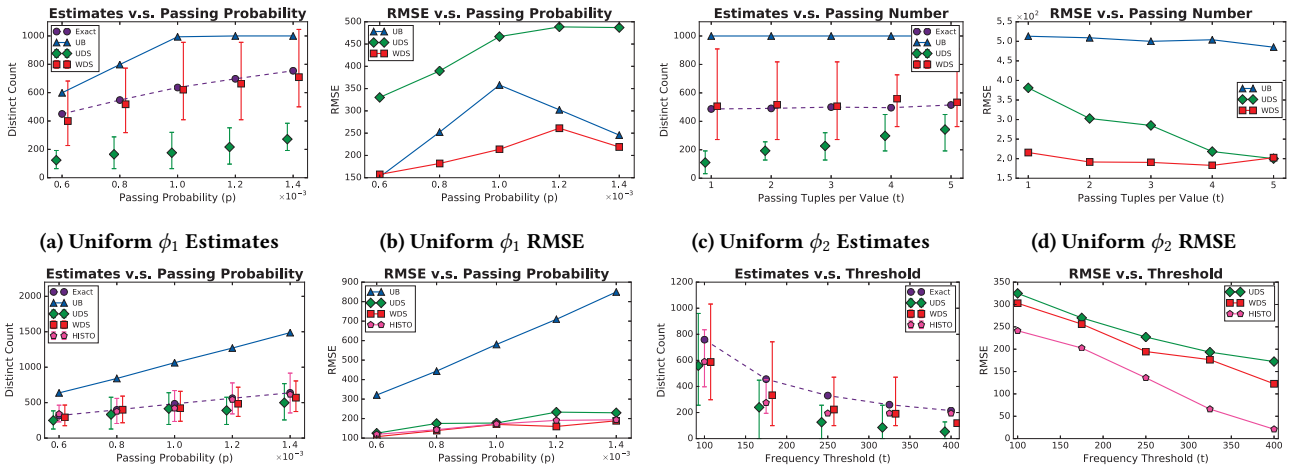
We used the TPC-DS benchmark data generator to generate a dataset of size 2 GB, and tested the following 2 queries[6] from the benchmark.

SP queries. Query 28 finds the number of distinct list prices. It consists of 6 distinct count subqueries on a single table over different sales buckets. Thus, each subquery is an SP query with a different selection condition. The results on these 6 SP queries are shown in Figure 3a and 3b. We see that WDS achieves a smaller MSE than UDS in all cases. HISTO performs similar to WDS, and is always better than UDS as well.

SPJ queries. Query 54 finds the number of customers that purchased a given item during a given period. With the union between catalog and web sales carried out first, the query is an SPJ query involving a star join between one fact table and two dimension tables. We tested 4 queries with selection conditions on different attributes of the dimension tables. For SPJ queries, we also included the random walk algorithm (RW) in the experimental comparison.

The results are shown in Figure 3c and 3d. We see that both WDS and RW have smaller MSE than UDS. The performances of WDS

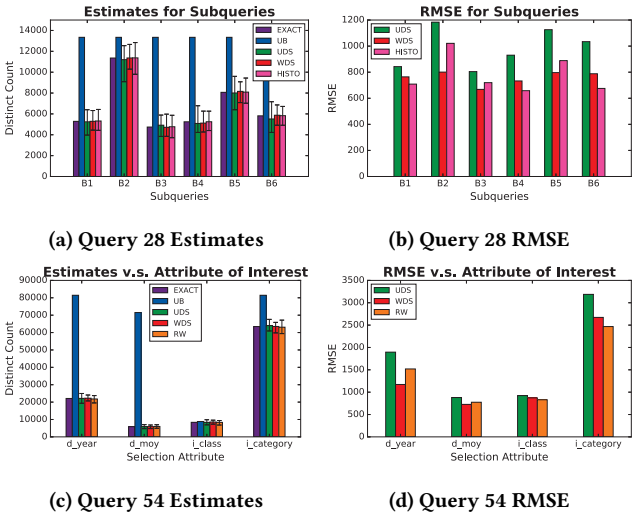
[6] The SQL statements of all queries can be found in [36].



(a) Uniform ϕ_1 Estimates (b) Uniform ϕ_1 RMSE (c) Uniform ϕ_2 Estimates (d) Uniform ϕ_2 RMSE

(e) Zipfian ϕ_1 Estimates (f) Zipfian ϕ_1 RMSE (g) Zipfian ϕ_3 Estimates (h) Zipfian ϕ_3 RMSE

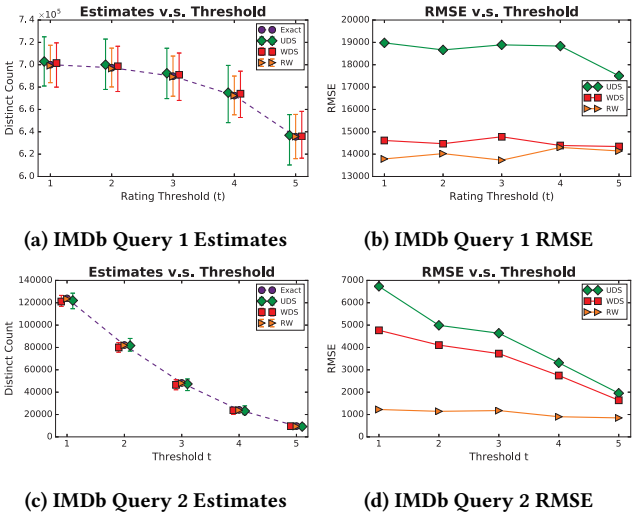
Figure 2: Performance Evaluation for Synthetic Datasets



(a) Query 28 Estimates (b) Query 28 RMSE

(c) Query 54 Estimates (d) Query 54 RMSE

Figure 3: Performance Evaluation for TPC-DS Benchmark



(a) IMDb Query 1 Estimates (b) IMDb Query 1 RMSE

(c) IMDb Query 2 Estimates (d) IMDb Query 2 RMSE

Figure 4: Performances Evaluation of IMDb Data

and RW are very similar. This is because the join is a foreign-key star join, *i.e.*, each tuple in the fact table joins with exactly one tuple in each of the dimension tables, and the projection attribute is in the fact table. Thus, the random walks will start from the fact table and walk towards the dimension tables. This has the same effect as sampling from all the join results, since there is a one-to-one correspondence between the tuples in the fact table and the full join results. However, we can run RW much more efficiently to collect the sample as there is no need to compute and materialize the join results. Results on the sample construction time will be given in Section 7.6.

7.4 Results on Real Data

We used two real datasets and tested the following SPJ queries:

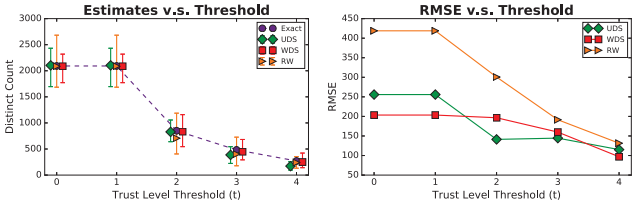
IMDb data. We tested IMDb Query 1 that finds all actors and actresses that ever acted in a movie whose rating is above a given threshold t on IMDb dataset[7]. This involves data stored in two tables: `principals`, the acting list, and `ratings`, rating information of movies. We cleaned up the data by removing movies without ratings. This results in 7 million tuples in the `principals` table and 1 million tuples in the `ratings` table. Then, the query can be written as an SPJ query with a selection condition on multiple attributes.

As this is a large dataset, we used a sample budget that is 0.1% of the dataset. The results are shown in Figure 4a and 4b as we vary the parameter t. Here, we omit UB as it is way off the chart. We see that WDS performs much better than UDS, while RW and WDS are close, because this join is also a foreign-key join.

We also evaluated another more complicated query (IMDb Query 2). It finds all actors and actresses that acted in two movies released more than 50 years apart, where the rating of the latter movie is

[7] https://www.imdb.com/interfaces/

higher by t compared to the former one. This query involves 5 joins where one of them is a complicated many-to-many join that generates more than 200 million tuples. The selection condition is also more complicated as it involves arithmetic operations between different attributes. The results are shown in Figure 4c and 4d as we vary the parameter t. While WDS still dominates UDS, RW also behaves accurately in this case. This is because values in the result are likely to have many passing tuples. So RW does not experience a loss of accuracy due to bias. Contrarily, it sampled more distinct values so that the variance is reduced.



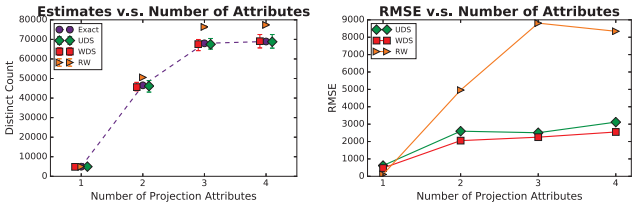
(a) Trust Network Estimates **(b) Trust Network RMSE**

Figure 5: Performance Evaluation of Network Data

Network data. The second dataset is the network dataset [31], which stores information about a who-trusts-whom network on a Bitcoin platform. Vertices represent users and directed edges represent trust relationships. The edges are weighted -10 to 10, denoting the degree of trust from the source user to the destination user. There are around 6k vertices and 36k edges. Our query of interest finds the users involved in at least one *trust triangle*. Specifically, three users X, Y, Z form a trust triangle if there exist edges from X to Y, Y to Z, and Z to X, and the weight of each edge is at least a given threshold t. We store the graph as a relation, so the above query can be written as an SPJ query involving a triangle join.

The results are shown in Figure 5a and 5b where we vary the parameter t. We see that WDS and UDS have similar performance. Recall that these two algorithms are run on the join results, which consist of all triangles in the network. This is a skewed dataset, as some vertices are involved in many triangles while the rest have few triangles. Each triangle has the same probability to pass the selection condition, so this SP query (on the join results) is an easy case, similar to what we have observed in Figure 2f. The triangle join presents a challenge for RW, since many random walks may not be able to close as a triangle. This results in a smaller effective sample size, which in turn leads to larger MSE.

7.5 Results Varying Other Parameters



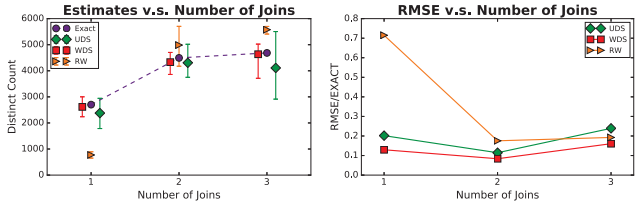
(a) Query 38 Estimates **(b) Query 38 RMSE**

Figure 6: Performance Evaluation of TPC-DS Q38

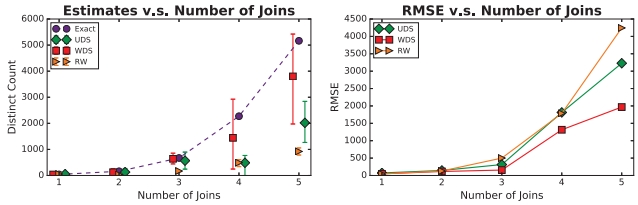
Number of Projection Attributes. To evaluate the effect of concatenating projection attributes, we tested a query modified from TPC-DS Q38. In SQL, the query is written as

```
SELECT count(distinct c_last_name),
  count(distinct c_last_name, c_birth_year),
  count(distinct c_last_name, c_first_name, c_birth_year),
  count(distinct c_last_name, c_first_name, c_birth_year,
      c_birth_country)
FROM store_sales, date_dim, customer
WHERE store_sales.ss_sold_date_sk = date_dim.d_date_sk
  AND store_sales.ss_customer_sk = customer.c_customer_sk
  AND d_month_seq between 1188 and 1188+11
```

Figure 6a and 6b show the accuracy comparison when the number of projection attributes vary from 1 to 4. For UDS and WDS, the error magnitude is insensitive to the number of projection attributes. WDS still dominates UDS in all cases. For RW, increasing the number of projection attributes also increases the number of distinct values at the source of the random walk. Therefore less tuples will be stored for each distinct value under the constraint of space. This leads to a larger bias.



(a) Network Self-Join Estimates **(b) Network Self-Join RMSE**



(c) Graph Self-Join Estimates **(d) Graph Self-Join RMSE**

Figure 7: Performance Evaluation for Self-Joins

Number of Joins. We tested the algorithms under more complicated queries when there are many joins, and the joins are many-to-many. For graph data, the queries find all vertices that has a length-d path to vertex 1. To compose this into a query, we perform $d-1$ self-joins to include all length-d paths in the join result. The selection condition passes only those paths ending in vertex 1, where all vertices in the path are distinct. For the network dataset above, we could only compute at most 3 joins, which already generates over 4 billion join results. So we include a less dense graph[8] to evaluate queries with more joins. Figure 7 shows the results on the network dataset and the graph dataset. When $d = 5, 6$, we set the sample size to 0.1%.

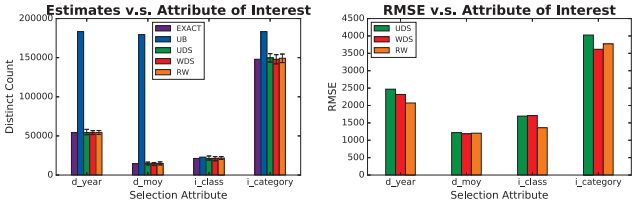
Since the selectivity of this filter is low, the bias of UDS makes it inaccurate. WDS sacrifices its variance for the bias reduction, which results in a smaller MSE when summed up. For RW, with the

[8] https://snap.stanford.edu/data/p2p-Gnutella04.html

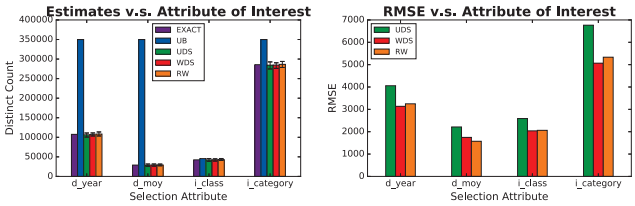
Dataset	Query	Sample construction cost				Query cost			
		UDS	WDS	HISTO	RW	EXACT	UDS	WDS	RW
Uniform	ϕ_1	125 ms	163 ms	156 ms	-	100 ms	< 1 ms	< 1 ms	-
	ϕ_2					95 ms	< 1 ms	< 1 ms	-
Zipfian	ϕ_1	300 ms	406 ms	356 ms	-	115 ms	< 1 ms	< 1 ms	-
	ϕ_3					182 ms	< 1 ms	< 1 ms	-
TPC-DS	Q28	561 ms	782 ms	753 ms	-	385 ms	1 ms	1 ms	-
	Q38	13.6 s	14.3 s	-	1.34 s	14.9 s	15 ms	7 ms	5 ms
	Q54 (2GB)	5.51 s	5.75 s	-	281 ms	6.41 s	< 1 ms	< 1 ms	< 1 ms
	Q54 (5GB)	8.85 s	9.02 s	-	558 ms	9.87 s	11 ms	10 ms	3 ms
	Q54 (10GB)	18.8 s	19.2 s	-	1.35 s	21.2 s	27 ms	21 ms	8 ms
IMDb	Query 1	16.3 s	17.3 s	-	2.43 s	21.3 s	45 ms	36 ms	21 ms
	Query 2	224 s	251 s	-	3.83 s	259 s	32 ms	52 ms	112 ms
Network	Triangle	4.33 s	4.35 s	-	61 ms	4.46 s	< 1 ms	< 1 ms	< 1 ms
	3 Joins	2004 s	2309 s	-	31.0 s	1946 s	185 ms	435 ms	120 ms
Graph	1 Join	81.9 ms	91.0 ms	-	31.7 ms	72.6 ms	< 1 ms	1 ms	1 ms
	5 Joins	56.7 s	60.3 s	-	1.78 s	55.3 s	2 ms	6 ms	2 ms

Table 3: Comparison of Time Costs

increase of join relations, each distinct value has more tuples in the join results, forcing us to move to phase 2 of the algorithm. It thus leads to an accuracy penalty. Nevertheless, at the same time, the benefit in efficiency becomes more significant.



(a) Estimates on 5GB dataset **(b) RMSE on 5GB dataset**



(c) Estimates on 10GB dataset **(d) RMSE on 10GB dataset**

Figure 8: Performance Evaluation under Different Data Sizes

Data Sizes. In addition tests on the 2GB TPC-DS dataset, we tested Query 54 on a 5GB and a 10GB dataset to examine the effect of data sizes on the algorithms. These are shown in Figure 8. As can be observed, the performance of algorithms are stable across different sizes of data. The growth of running times are also near linear.

7.6 Running Times

We have also measured the wall-clock running times of the algorithms, including both the time for building the sample and the time for producing an estimate for a given query, which are summarized in Table 3. First, we see that the query costs of all estimation algorithms are order-of-magnitude smaller than of computing the query size exactly. This justifies their use in a query optimizer.

The sample construction cost is larger for WDS compared to UDS, as we need to find the optimal sampling strategy first. But the costs are small anyway for SP queries. For SPJ queries, the sample construction cost is dominated by that of computing the join, which dwarfs the additional cost of finding the optimal sampling strategy. On the other hand, RW is able to build the sample without computing the join, resulting in significant savings.

7.7 Summary of Experimental Evaluation

From our experimental evaluation, we can draw the following conclusions: For SP queries, WDS performs at least as well as UDS, and outperforms it on hard queries, especially on skewed data. Thus, as long as one is willing to pay the small extra cost to find the optimal sampling strategy, WDS is always the recommended method. HISTO performs similar to WDS and is desirable when the frequency vector is not known. For SPJ queries, if one can tolerate the cost to pre-compute the join, then running WDS on the join results is the best choice. Otherwise, RW can be used, especially for foreign-key joins, where its performance is as good as that of running WDS on the full join results. On cyclic joins, however, RW may lose some accuracy compared with WDS.

8 CONCLUSIONS

In this paper, we have introduced weighted distinct sampling to tackle the problem of size estimation for SP queries, and extended it to handling SPJ queries. We have presented algorithms that can collect samples in time linear to the size of the database. It is also not difficult to adapt them to run in a constant number of MapReduce rounds, enabling it to run over large distributed datasets.

ACKNOWLEDGMENTS

This work has been supported by HKRGC under grants 16202317, 16201318, 16201819, and 16205420, and by an Alibaba Innovative Research (AIR) grant.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Addison-Wesley Longman Publishing Co., Inc.

[2] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. Join synopses for approximate query answering. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[3] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. 2014. Knowing when youâĂŹre wrong: Building fast and reliable approximate query processing systems. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The Space Complexity of Approximating the Frequency Moments. *J. Comput. System Sci.* 58, 1 (1999), 137–147. https://doi.org/DOI:10.1006/jcss.1997.1545

[5] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. 2002. Counting Distinct Elements in a Data Stream. In *Proc. International Workshop on Randomization and Approximation Techniques in Computer Science*. 1–10.

[6] Kevin S. Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. 2007. On synopses for distinct-value estimation under multiset operations. In *Proc. ACM SIGMOD International Conference on Management of Data*. 199–210.

[7] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1998. Random sampling for histogram construction: how much is enough?. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[8] Yu Chen and Ke Yi. 2017. Two-Level Sampling for Join Size Estimation. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[9] Graham Cormode and Minos Garofalakis. 2005. Sketching streams through the net: Distributed approximate query tracking. In *Proc. International Conference on Very Large Data Bases*.

[10] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2002. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 61–72.

[11] Marianne Durand and Philippe Flajolet. 2003. Loglog Counting of Large Cardinalities (Extended Abstract). In *Proc. European Symposium on Algorithms*. 605–617.

[12] Cristian Estan and Jeffrey F Naughton. 2006. End-biased samples for join cardinality estimation. In *Proc. IEEE International Conference on Data Engineering*. IEEE, 20–20.

[13] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms (AOFA)*.

[14] Philippe Flajolet and G. Nigel Martin. 1985. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.* 31, 2 (1985), 182–209.

[15] Michael J. Freitag and Thomas Neumann. 2019. Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates. In *Proc. Biennial Conference on Innovative Data Systems Research*.

[16] Edward Gan, Jialin Ding, Kaisheng Tai, Vatsal Sharan, and Peter Bailis. 2018. Moment-based quantile sketches for efficient high cardinality aggregation queries. *Proceedings of the VLDB Endowment* 11 (2018).

[17] Sumit Ganguly, Phillip B Gibbons, Yossi Matias, and Avi Silberschatz. 1996. Bifocal sampling for skew-resistant join size estimation. *ACM SIGMOD Record* 25, 2 (1996), 271–281.

[18] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2008. *Database Systems: The Complete Book*. Prentice Hall.

[19] Phillip B. Gibbons. 2001. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *Proc. International Conference on Very Large Data Bases*. 541–550.

[20] P. B. Gibbons, Y. Matias, and V. Poosala. 2002. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems* 27, 3 (2002), 261–298.

[21] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. 2002. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. ACM Symposium on Theory of Computing*.

[22] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. 2002. How to Summarize the Universe: Dynamic Maintenance of Quantiles. In *Proc. International Conference on Very Large Data Bases*.

[23] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[24] S. Guha, N. Koudas, and K. Shim. 2006. Approximation and streaming algorithms for histogram construction problems. *ACM Transactions on Database Systems* 31, 1 (2006), 396–438.

[25] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. 1996. Selectivity and Cost Estimation for Joins Based on Random Sampling. *J. Comput. System Sci.* 52 (1996), 550–569.

[26] Sariel Har-Peled and Micha Sharir. 2011. Relative (p, ε)-approximations in geometry. *Discrete & Computational Geometry* 45, 3 (2011), 462–496.

[27] J. M. Hellerstein, P. J. Haas, and H. J. Wang. 1997. Online Aggregation. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[28] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. 1998. Optimal Histograms with Quality Guarantees. In *Proc. International Conference on Very Large Data Bases*.

[29] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex Ad-Hoc Queries in Big Data Clusters. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[30] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. 2010. An optimal algorithm for the distinct elements problem. In *Proc. ACM Symposium on Principles of Database Systems*. 41–52.

[31] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 221–230.

[32] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[33] Charles Masson, Jee E. Rim, and Homin K. Lee. 2019. DDSketch: A Fast and Fully Mergeable Quantile Sketch with Relative Error Guarantees. *Proceedings of the VLDB Endowment* 12 (2019).

[34] Y. Matias, J. S. Vitter, and M. Wang. 1998. Wavelet-Based Histograms for Selectivity Estimation. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[35] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. 1996. Improved histograms for selectivity estimation of range predicates. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[36] Yuan Qiu, Yilei Wang, Ke Yi, Feifei Li, Bin Wu, and Chaoqun Zhan. 2020. *Weighted Distinct Sampling: Cardinality Estimation for SPJ Queries (full version)*. http://www.cse.ust.hk/~yike/spj-full.pdf

[37] Johanna Sommer, Matthias Boehm, Alexandre V. Evfimievski, Berthold Reinwald, and Peter J. Haas. 2019. MNC: Structure-Exploiting Sparsity Estimation for Matrix Expressions. In *Proc. ACM SIGMOD International Conference on Management of Data*. 1607–1623.

[38] Surajit Surajit, Bolin Ding, and Srikanth Kandula. 2017. Approximate Query Processing: No Silver Bullet. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[39] S. Suri, C. Toth, and Y. Zhou. 2006. Range counting over multidimensional data streams. *Discrete and Computational Geometry* (2006).

[40] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. 2002. Dynamic multidimensional histograms. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[41] Daniel Ting. 2019. Approximate Distinct Counts for Billions of Datasets. In *Proc. ACM SIGMOD International Conference on Management of Data*. 69–86.

[42] David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P Chakkappen. 2015. Join size estimation subject to filter conditions. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1530–1541.

[43] C. Zhan, M. Su, C. Wei, X. Peng, L. Lin, S. Wang, Z. Chen, F. Li, Y. Pan, F. Zheng, and C. Chai. 2019. AnalyticDB: real-time OLAP database system at Alibaba cloud. In *Proceedings of the VLDB Endowment*, Vol. 12.