Sampling over Union of Joins

Yurong Liu
University of Rochester
yliu217@u.rochester.edu

Yunlong Xu
University of Rochester
yxu103@u.rochester.edu

Fatemeh Nargesian
University of Rochester
fnargesian@rochester.edu

ABSTRACT

Data scientists often draw on multiple relational data sources for analysis. A standard assumption in learning and approximate query answering is that the data is a uniform and independent sample of the underlying distribution. To avoid the cost of join and union, given a set of joins, we study the problem of obtaining a random sample from the union of joins without performing the full join and union. We present a general framework for random sampling over the set union of chain, acyclic, and cyclic joins, with sample uniformity and independence guarantees. We study the novel problem of union of joins size evaluation and propose two approximation methods based on histograms of columns and random walks on data. We propose an online union sampling framework that initializes with cheap-to-calculate parameter approximations and refines them on the fly during sampling. We evaluate our framework on workloads from the TPC-H benchmark and explore the trade-off of the accuracy of union approximation and sampling efficiency.

1 INTRODUCTION

Data scientists often draw on multiple sources to collect training data. Since most relational sources are not stored as single tables due to normalization, users often need to perform joins before learning on the join output [11]. Moreover, data may be collected from distributed sources, each described as a join over internal databases, data lakes, or web data [3, 12]. Therefore, the target data is the union of the results of joins.

Joins are expensive and learning after joins leads to poor training performance due to the introduced redundancy avoided by normalization. There have been efforts to enable learning over joins, but they are limited to certain models, including linear regression, and cannot be applied to more general models [11, 22, 23, 35]. Fortunately, an important result [36] from the learning theory suggests that learning and approximate query answering [26] do not require the full results and an i.i.d sample can achieve a bounded error. This result holds for any model. Therefore, in data collection from

J_W : **Customer_W**$(\cdots)$ ⋈ **Part1_W**$(\cdots)$ ⋈ **Supplier1_W**$(\cdots)$ ⋈
    PartSupp1_W$(\cdots)$ ⋈ **PartSupp2_W**$(\cdots)$ ⋈
    LineItem1_W$(PartKey1, \cdots)$ ⋈ **Orders1_W**$(OrderKey1, \cdots)$ ⋈
    Orders2_W$(OrderKey1, \cdots)$ ⋈ **LineItem2_W**$(PartKey1, \cdots)$

⋃ J_E : **Customer_E**$(\cdots)$ ⋈ **PartSupplier_E1**$(\cdots)$ ⋈
    PartSupplier_E2$(\cdots)$ ⋈
    DoubleOrders_E$(OrderKey1, OrderKey2, \cdots)$ ⋈
    LineItem_E1$(OrderKey1, \cdots)$ ⋈ **LineItem_E2**$(OrderKey2, \cdots)$

⋃ J_MW : **Customer_MW**$(\cdots)$ ⋈ **DoublePartSupplier_MW**$(\cdots)$ ⋈
    DoubleOrdersLineItem_MW$(\cdots)$

Figure 1: Example of Union of Joins on Denormalized TPC-H.

multiple sources, the question to ask is how to obtain a sample from union of sources without executing join and union. Given a collection of joins, the goal is to return a sample of N tuples from the union of the results of joins, independently and at random.

Example 1. *Suppose a data scientist in an online retail company wants to train a model for applying a promotion to the future bundle orders of customers. To do so, the data scientist needs a random and independent sample of size k of customer data and their bundle purchases from the underlying distribution. Suppose the customer order data is stored in various databases, each having its own schema. For example, the company may have one database for suppliers of each east, west, and midwest region. Obtaining customer-bundle data requires constructing a query for each region database, as shown in Fig. 1, then unioning the results of queries. Since there is no single relation that contains all the required features, these queries need to join data from various relations. Note that J_W is a cyclic join and J_E and J_{MW} are acyclic. In J_W, relation* Orders *is self-joined to obtain the information of items in the same order (bundle purchases). All three joins have the same output schema. To construct the target dataset, the first challenge is although some of these queries are performed on heavily denormalized relations (or views), for example,* PartSupplier *relation in J_E, since some base relations, for example, the* LineItem *and* Orders, *are very large, performing a full join becomes very expensive.*

The problem of random sampling over a single join has been actively studied since the 1990s [1]. The goal is to obtain a random and independent sample from join J, without performing the full join, such that the probability of each tuple in the sample is $1/|J|$. One solution is to join samples of base relations to obtain sample join tuples [1]. However, the join of samples produces a much smaller number of join tuples than samples. Moreover, it is shown that the obtained join samples do not guarantee independence [18]. For approximate query answering, some techniques such as RippleJoin [16] and WanderJoin [25] manage to use non-random/independent and random/dependent samples, respectively.

Other techniques for sampling over join apply the accept/reject sampling paradigm to guarantee i.i.d [10, 33]. The most recent work by Zhao et al. proposes a framework for sampling over one join that handles general multi-way joins [38]. The motivation of random sampling over join is tightly connected to join size estimation which has also been a point of interest in the database community due to its application to query optimization [4, 10, 34].

Example 2. *Continuing with Ex. 1, the second challenge is to union join samples such that* uniformity *is guaranteed, i.e., each tuple has the probability $\frac{1}{|J_W \cup J_E \cup J_{MW}|}$ of being in the final sample. A naive solution is to union samples of joins, obtained in an offline manner. Suppose we apply an off-the-shelf sampling over join algorithm and obtain samples S_W, S_E, and S_{MW} from J_W, J_E, and J_{MW}, respectively. We have $P(t \in S_W) = 1/|S_W|, P(t' \in S_E) = 1/|S_E|$, and $P(t'' \in S_{MW}) = 1/|S_{MW}|$. It is easy to show that $U = J_W \cup J_E \cup J_{MW}$ does not guarantee uniformity and tuples have unequal probability of appearing in U. Consider the contradicting example of $r \in S_E, r \notin S_W, r \notin S_{MW}$, we have $P(r \in U) = 1/|S_E|$, however, if $r \in S_E \cap S_W \cap S_{MW}$, we get $P(r \in U) = (\frac{1}{|J_E|} + \frac{1}{|J_W|} + \frac{1}{|J_{MW}|}) \cdot \frac{|S_E \cap S_W \cap S_{MW}|}{|U|}$, because we do set union and keep one instance of overlapping tuples. An accept/reject sampling algorithm can help to adjust this probability to obtain $1/|U|$, however, as we show in § 2, the algorithm needs to know* apriori, *the size of each join and their union, which requires the overlap size of all combinations of J_E, J_W, and J_{MW}. One idea may be to estimate the overlaps and unions from the samples. However, that would not be a viable option, since just like joining samples or relations, the probability of obtaining samples from the overlapping regions of joins is low.*

In this paper, we present a generic framework for random sampling over the union of joins. In particular, we consider sampling set union with replacement. Sampling from the disjoint union is a straightforward extension of the set union. The classic join sampling [10, 34] and the recently revisited framework [38] consider random sampling *with replacement* over join. Another relevant problem is the random enumeration of the result of the union of acyclic conjunctive queries [8]. The intermediate results of a random query result enumeration algorithm can be considered as a random sample from the union *without replacement* which is different than our problem. Moreover, in this paper, we study union sampling over a larger class of joins (chain, cyclic, and acyclic). In § 3.2, we provide an elaborate discussion and analytical comparison of this line of work with our framework.

There are several challenges to addressing the sampling over the union of joins problem. First, unioning random samples from joins does not guarantee uniformity. Our solution is an accept/reject sampling algorithm that defines Bernoulli and non-Bernoulli probability distributions for selecting joins. The latter mimics the behavior of union calculation. Second, it turned out that to guarantee uniformity, the sampling framework needs to know the size of each join and the size of the union of joins apriori. Although the problem of set union size approximation [2, 7, 21] and its online extension to streams [9, 14, 21, 28] have been extensively studied in the approximate counting literature, to the best of our knowledge, there is no study that addresses the problem of approximating the union size of joins without performing the full join and overlap.

Third, HISTOGRAM-BASED estimation requires knowing the overlap of an exponential number of sets of joins, each set in the powerset of joins. We reduce the space of calculation by reformulating the problem to use smaller-unit statistics, called k-overlaps, of each join, which is the size of the subset of a join result that is shared with *exactly* $(k-1)$ other joins. Next, we propose two instantiations of the framework for estimating the overlap of joins with an arbitrary number of relations and all join types (chain, cyclic, and acyclic): a HISTOGRAM-BASED method and a RANDOM-WALK method. The HISTOGRAM-BASED technique is cheap and requires knowing limited statistics of joins. It may incur a loose bound, thus, a high rejection rate, under circumstances. The HISTOGRAM-BASED method is highly suitable for data in the wild or scenarios, such as data markets, where limited metadata is available but access to the whole data is infeasible. The RANDOM-WALK method is accurate in estimating parameters and results in low delay. It needs sampling for parameters warm-up and provides theoretical guarantees. To balance the trade-off of parameter estimation cost and sampling efficiency, we propose an ONLINE-UNION sampling algorithm that initializes and updates parameters with the HISTOGRAM-BASED and RANDOM-WALK methods, respectively, and reuses the samples obtained during RANDOM-WALK while ensuring uniformity.

In this paper, we make the following contributions:

- We present the problem of random sampling over the union of joins.
- We design a framework for sampling over the set union of joins of types chain, cyclic, and acyclic (§ 2). Any instantiation of the framework always returns uniform and independent samples from the full result (Theorem 1) but with different sampling efficiency (§ 6.2).
- We design HISTOGRAM-BASED (§ 4) and RANDOM-WALK (§ 6) methods to bound the size of overlap of any collection of chain, acyclic, or cyclic joins.
- We present an ONLINE-UNION sampling technique that balances the latency and warm-up cost trade-off (§ 6.2).
- We perform extensive experimental evaluations using the TPC-H benchmark to investigate the error and runtime of parameter estimation and sampling methods(§ 9). We also evaluate the scalability of our framework with respect to relation size, number of samples, and overlap size.

2 PROBLEM DEFINITION

Let $\mathcal{A}$ be the universe of attributes and $\mathcal{A}_i$ be the attributes in relation J_i. We are given a set of joins $S = \{J_1, \ldots, J_n\}$. A join J_j is defined as $J_j = R_{j,1} \bowtie_{A_{j,1}} R_{j,2} \bowtie_{A_{j,2}} \cdots \bowtie_{A_{j,n_1-1}} R_{j,n_1}$, where $R_{j,1}, \cdots, R_{j,n_1-1}$ are base relations. Similar to relational algebra, we assume all joins have the same output schema after performing the join in terms of the number and name of attributes. Note that joins can still have different lengths and different relations. We also assume that join attributes are standardized to have the same names. We only mention attribute names when needed. In relational algebra, there are two types of unions: set union and disjoint union. The former eliminates duplicate tuples from the result of a union and the latter keeps the duplicates. The notion of unionability [31] can be applied on base relations to align attributes such that joins incur the same schema.

The problem of sampling over a union of joins is to return each tuple with probability $1/|union(J_1, \cdots, J_n)|$, where union may be set or disjoint union. Returning just one sampled tuple is usually not enough, therefore, we would like to generate totally independent sampled tuples continuously until a certain desired sample size N is reached. We formulate the sampling set union and disjoint problems as follows.

Definition 1. (Sampling Disjoint Union of Joins) *Given a set of joins $S = \{J_1, \ldots, J_n\}$, return N independent samples from $V = J_1 \uplus \ldots \uplus J_n$ such that each sampled tuple is returned with probability $\frac{1}{|V|} = \frac{1}{J_1 + \ldots + J_n}$.*

Sampling from the disjoint union is straightforward. Given the disjoint union $V = J_1 \uplus \ldots \uplus J_n$, we first select a join J_j with probability $P(J_j) = \frac{|J_j|}{|J_1 + \ldots + J_n|}$, then, we select a random tuple from J_j. This means the probability of each sampled tuple t is $P(t) = \frac{|J_j|}{|V|} \cdot \frac{1}{|J_j|} = \frac{1}{|V|}$. We repeat the process until N sampled tuples are obtained. This algorithm always returns independent samples because a returned sample is always uniform regardless of the previous sampling iterations. Methods of sampling a tuple from a single join have long been a popular problem [10, 10, 33, 37, 38]. We revisit random sampling over join in § 3.2.

The set union operation eliminates duplicate tuples from the result of the union. As such, an i.i.d sampling algorithm over the set union should return each tuple in the universe of the set union with the probability of the size of a set union.

Definition 2. (Set Union of Joins Sampling) *Given a set of joins $S = \{J_1, \ldots, J_n\}$, let $\mathcal{U}$ be the discrete space of unique tuples in $U = J_1 \cup \ldots \cup J_n$. Return N independent samples from $\mathcal{U}$, such that each sampled tuple is returned with probability $\frac{1}{|J_1 \cup \cdots \cup J_n|}$.*

3 A UNION SAMPLING FRAMEWORK

Let U be the universe of tuples in the set union of joins. We assume there are no duplicates in each join. Given the set union $U = \bigcup_{j=1}^{n} J_j$, we want for each value $u \in U$, $P(t = u) = \frac{1}{|U|}$.

Example 3. *Consider joins J_1 and J_2 that have the same output schema. Suppose $t_1 = (3, 6, 4) \in J_1$ and $t_2 = (3, 6, 4) \in J_2$. The value of each tuple t, namely $t.val$, can be obtained by concatenating its attribute values using a standard convention. Then, by the definition of a set, t_1 and t_2 refer to the same tuple, say u, in the universe $U = J_1 \cup J_2$. We want $P(u)$, the probability of selecting a tuple with value u from U, to be $\frac{1}{|U|}$. Tuples t_1 and t_2 are distributed in different joins. Hence, u is obtained if t_1 or t_2 are sampled from their corresponding joins. That is, we want $P(t = u) = P(t_1) + P(t_2) = \frac{1}{|U|}$. Note that we may have a sampling with replacement or we may get both t_1 and t_2 in the sample. Our framework guarantees that $P(t = u) = P(t_1.val) = P(t_2.val) = \frac{1}{|U|}$, whether we choose to remove duplicates or not.*

At each sampling iteration, the framework performs two steps: join selection and join random sampling. The framework continuously samples tuples, with replacement, with $1/|U|$ probability, until the desired sample size N is reached. A straightforward way is based on the union trick [15]. At each iteration, we iterate through all joins

and select a join with the Bernoulli probability $P(J_j) = |J_j|/|U|$. This means multiple joins may be selected in each iteration. Upon selecting J_j, we randomly sample a tuple t from J_j with replacement. Recall $u = t.val$ denotes the value of tuple t. We accept tuples with duplicate values u, only if they are sampled from the same joins, otherwise, we accept the tuples. This means a duplicate tuple t is retained only if it is sampled from the first join where $u = t.val$ was observed. With this description, a tuple value $u \in U$ is returned upon first selecting a join J_j that contains u with probability $|J_j|/|U|$, then sampling J_j with probability $1/|J_j|$. This guarantees that every value $u \in U$ is returned with probability $\frac{|J_j|}{|U|} \cdot \frac{1}{|J_j|} = \frac{1}{|U|}$. Despite its simplicity, this algorithm has a high rejection ratio for highly overlapping joins and may result in high latency. This is attributed to the utilization of a two-phase framework, which is essential for ensuring uniformity in sampling. Next, we describe a join selection algorithm with a more careful selection of joins. In § 7, we propose a novel approach that leverages computation performed in the first phase to reduce latency in the second stage.

3.1 Non-Bernoulli Join Selection

The above technique keeps samples from an overlap area of joins only if they are sampled from exactly one predetermined join. Consider two joins J_1 and J_2 with overlapping data region B in Fig. 2a. We select and keep any sample $t_1 \in J_1$. Later, upon selecting J_2, if we sample a $t_2 \in B$, we reject t_2. The trick to avoiding rejection is to keep the B from J_1 as the only space we sample from J_1. Therefore, we have $P(J_1) = \frac{|A+B|}{|A+B+C|} = \frac{|J_1|}{|U|}$, and $P(J_2) = \frac{|C|}{|A+B+C|} = \frac{|J_2| - |B|}{|U|}$.

Our join selection is outlined in Algorithm 1. Prior to sampling, the algorithm needs to decide which overlapping region is restrictively sampled from which join. We call this division of joins a *cover* (line 2 of Algorithm 1). A cover over joins $S = \{J_1, \cdots, J_n\}$, namely $C = \{J_1', \cdots, J_n'\}$, is an ordering over S such that $J_i' = \{t \in J_i | t \notin \bigcup_{j<i} J_j'\}$. In fact, a cover J_i' of join J_i is a selection query over join J_i. A cover of S can be created by starting from the first join and keeping or removing overlapping parts. Fig. 2b illustrates an example of a cover for three overlapping joins. Given a cover C, to calculate the size of J_i', we simply follow the inclusion–exclusion principle. Let $O_\Delta = \bigcap_{J_j \in \Delta} J_j$ and S_i represent the set of joins that appear before J_i in the ordering offered by C, then we have the following.

$$|J_i'| = |J_i| + \sum_{m=1}^{i-1} \sum_{\Delta \subset S_i, |\Delta|=m} (-1)^m |O_\Delta \cup \{J_i\}|$$

Based on this cover, each J_i is selected with $P(J_i) = \frac{|J_i'|}{|U|}$. When sampling, we should always follow the cover we pre-defined, i.e., for any sample $t \in J_i$, we should discard it if $t \notin J_i'$. However, if we do not have overlap information apriori, upon selecting J_i and sampling t, it is not possible to verify whether t is in J_i' or not. Thus, we face a non-trivial case when we sample $t \in J_i \setminus J_i'$. If we later sample t from J_j with $J_j' \cap J_i \neq 0$, i.e., J_j' covers the overlapping part with J_i, we should do a critical operation, called *revision*. This means we remove $t \in J_j$ from the sample and re-sample J_j, while keeping the t from J_i.

Example 4. *Consider joins J_1, J_2, and J_3 of Fig. 2b. A cover for these joins are highlighted with blue, red, and green colors. The algorithm selects J_1, J_2, and J_3 with probability $|J_1'|/|U|$, $|J_2'|/|U|$, and $|J_3'|/|U|$, respectively. Suppose at some iteration we have selected J_2 and sampled $t \in J_2 \setminus J_2'$. Suppose now we select J_1' and sample the same t. Because the cover tells us to sample J_2 only from J_2' area, we remove t's from the target set, accept t that's sampled from J_1 and assign it to J_1 in the record.*
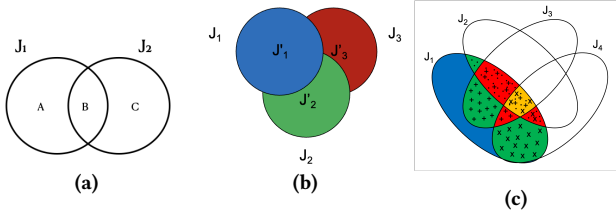
Figure 2: (a) union operation, (b) cover for three joins, and (c) $\mathcal{A}_j^k$ of four joins.

Theorem 1. *Given joins $S = \{J_1, \ldots, J_n\}$, Algorithm 1 returns each result tuple t with value u with probability $\frac{1}{|J_1 \cup \ldots \cup J_n|}$.*

PROOF. Intuitively, a cover defined by Algorithm 1 decides from which join exclusively a value in the overlap of a collection of joins is sampled. Recall U is the universe of the set union of tuples of joins, i.e., $\{u | u \in \cup_i J_i\}$. Algorithm 1 uses a mapping strategy function $f : U \to S$ that tells us to which J_i a specific u is assigned. Note that u could belong to multiple J_i's, however, f refers to the unique J_i from which u can be sampled. Let a cover C of S be the quotient space of U over f and $g : S \to C$ be a mapping function such that $g(J_i) = J_i'$. Then, $g \circ f$ will map each $u \in U$ to a join in cover C. For all u, we denote $|g(f(u))|$ to be $|\{u' | g(f(u')) = g(f(u))\}|$. In other words, the probability of sampling a $u \in U$ depends on the probability of selecting $g(f(u))$ followed by sampling u from $g(f(u))$. Therefore, we obtain the probability of $P(t = u)$ as follows.

$$P(t = u) = P(f(u)) \cdot \frac{1}{|g(f(u))|} = \frac{|g(f(u))|}{|U|} \cdot \frac{1}{|g(f(u))|} = \frac{1}{|U|}$$

□

Computing the probability distribution of line 2 of Algorithm 1 requires the knowledge of $|J_i'|$ as well as $|U|$. In § 4, we describe ways of estimating the overlap of k joins and $|J_i'|$.

3.2 Join Sampling Revisited

To sample a single join (line 7 of Algorithm 1), we consider the work by Zhao et al. [38], which is a generic framework for sampling from any type of join. The framework defines a join data graph where each tuple in a relation is a node. Each tuple t is labeled with a weight defined as the upper bound for the number of tuples in the join result that t yields. The framework performs accept/reject sampling. Each tuple from a relation is sampled with some probability based on its weight and is rejected with some rate in terms of the weights to guarantee uniformity. We make some design choices to adopt the join sample framework of Zhao et al. as a subroutine in our union sampling framework.

Algorithm 1 Union Sampling

Input: Joins $S = \{J_j, 1 \leq j \leq n\}$, tuple count N
Output: Tuples $\{t_i, 1 \leq i \leq N\}$
1: $\{|J_j|, 1 \leq j \leq n\}, |U| \leftarrow warmup(S)$ ▷ HISTOGRAM-BASED (§ 4) or RANDOM-WALK (§ 6)
2: $\{|J_j'|\} \leftarrow cover(S)$
3: $T \leftarrow \{\}$ ▷ target sample
4: $orig\_join_i \leftarrow \{\}$ ▷ record of original join of seen tuples
5: **while** $n < N$ **do**
6:     select J_j with probability $\frac{|J_j'|}{|U|}$
7:     $t \leftarrow$ a random sample from J_j
8:     **if** $t \in orig\_join_i$ for any $i < j$ **then** reject t
9:     **else**
10:         **if** $t \in orig\_join_i$ for any $i > j$ **then** ▷ revision
11:             remove t from $orig\_join_i$ and add t to $orig\_join_j$
12:             remove all t's from T
13:         **if** $t \notin orig\_join_i$ **then** add t to $orig\_join_j$
14:     $T \leftarrow T \cup \{t\}$
15: **return** T

First, for weight instantiation, we use three techniques: extended Olken's, exact, and Wander Join [25], proposed by Zhao et al. [38]. Second, this framework requires index structures over base relations to know which tuples can be joined together. Instead, we use hash tables for relations to maintain tuples' joinability information. Third, one limitation of Zhao et al.'s framework is the assumption of having only key-foreign key joins between relations. Since in a generic join, some tuples may not have a joinable tuple in other relations, we release this assumption by modifying the Extended Olken's to set the weights (and hence probabilities) of those tuples to zero with an extra linear search in the hash tables.

Finally, to obtain the accept/reject ratio, this framework allows us to plug in any of the join size upper-bound estimations. We also need to compute the size upper bound of joins in Algorithm 1. To do so, in what follows, we adopt parts of the algorithm proposed in Ngo et al. [32] and extend Olken's algorithm [33] to calculate the upper bound on the size of joins of an arbitrary number of relations.

Assume a join $J = R_1 \bowtie_{A_1} R_2 \bowtie_{A_2} \cdots \bowtie_{A_{n-1}} R_n$. Let $M_{A_i}(R_{i+1})$ be the maximum value frequency in attribute A_i of relation R_{i+1}. Since each tuple in R_2 with value v for A_i can be matched with maximum $M_{A_i}(R_{i+1})$ tuples of R_{i+1} on A_i, we have the following upper bound for the size of a join J: $|J| \leq |R_1| \cdot \prod_{i=1}^{n-1} M_{A_i}(R_{i+1})$. In our framework, we consider the above extension of Olken's algorithm for join size estimation in all algorithms.

3.3 Cost Analysis

Since the subroutine of sampling from a join in Algorithm 1 is based on the existing algorithms, for the cost analysis, we decouple the delay of random sampling over join from our algorithm and consider the total number of samples obtained from the join subroutine as our total cost.

Theorem 2. *Given joins $S = \{J_1, \ldots, J_n\}$, the expected total sampling cost of Algorithm 1 for returning N uniform and independent samples is $N + N \log N$.*

PROOF. Given a cover $C = \{J'_j \mid j \in [1, n] \cap \mathbf{Z}\}$, Algorithm 1 samples each join J_j with probability $|J'_j|/|U|$. Let N_j be the number of tuples from J_j that are in the final result. Based on Algorithm 1, we know a tuple from J_j is in the final sample if it is obtained from J'_j. Therefore, we have $N_j = \frac{|J'_j|}{|U|} \cdot N$, in expectation. Let ψ_j be the number of tuples Algorithm 1 ever obtains from J_j. A tuple may be a rejected, accepted, or revised sample, because the set of tuples from different joins may intersect. Based on the union bound, the number of iterations of Algorithm 1 is bounded by the sum of the number of tuples sampled from each join. Using this principle, we have the expected total number of iterations of $\psi \leq \sum_{j=1}^{n} \psi_j$. Given N_j coupons, the coupon collector's problem provides a bound for the number of samples we expect we need to draw with replacement before having drawn each coupon at least once [30]. This result allows us to obtain the expected value of $\psi_j = N_j \log N_j$. Therefore, we have the following expected number of iterations.

$$\psi \leq \sum_{j=1}^{n} N_j \log N_j = \sum_{j=1}^{n} N \cdot \frac{|J'_j|}{|U|} \log \left(N \cdot \frac{|J'_j|}{|U|} \right)$$

Let $\alpha_j = \frac{|J'_j|}{|U|}$. We have the following.

$$\psi \leq \sum_{j=1}^{n} \alpha_j . N \log(\alpha_j . N) = N \left(\sum_{j=1}^{n} \alpha_j \log \alpha_j + \sum_{j=1}^{n} \alpha_j \log N \right)$$

From the definition of cover, we know $\sum_{j=1}^{n} \frac{|J'_j|}{|U|} = 1$. Therefore, we have the following bound on the expected total time.

$$\psi \leq N(\log(H(n)) + \log N) \leq N + N \log N$$

□

We remark that although our algorithm does not have a strict and deterministic guarantee on the delay between samples, our total time is on par with the $O(N \log N)$ time of the algorithm proposed by Carmeli et al., for the random enumeration of the result of the union of conjunctive queries, where N is the number of answers [8].

4 SIZE OF SET UNION OF JOINS

Executing full joins and computing set union is costly. We propose a novel way of computing the set union size by using the size of joins and the size of the overlap of joins. To do so, we first separate each join J_j into n disjoint parts, denoted as $J_j = \bigcup_{k=1}^{n} \mathcal{A}_j^k$, where $\mathcal{A}_j^k$ is the set of tuples of k-th overlap in J_j, i.e., each tuple in $\mathcal{A}_j^k$ belongs to J_j and appears in exactly $k - 1$ other joins. The base case $\mathcal{A}_j^1$ includes the tuples in J_j that are the set complement of all overlaps. Fig. 2c represents the $\mathcal{A}_j^k$ areas for a join J_1. Since for each J_j, $\mathcal{A}_j^k$'s are disjoint, we can define the size of the set union U as follows.

$$|U| = \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{1}{k} |\mathcal{A}_j^k| \tag{1}$$

Note that $\mathcal{A}_j^k$ is non-trivial information, which requires combining the overlap size of k-combinations of joins. There are two challenges for computing $\mathcal{A}_j^k$. First, there is no relationship between

the pairwise overlap information and higher order k-th overlap, $\mathcal{A}_j^k (k > 2)$. Second, computing a pairwise overlap size without a full join is more challenging than computing a single join size.

Suppose we have a way of computing the overlap for any set of joins. More formally, given a collection $\Delta \in S$ of joins, O_Δ denotes the overlap of joins in Δ. In § 4 and 7, we describe various algorithms for overlap estimation of all join types (chain, cyclic, and acyclic). Now, we turn our attention to computing $\mathcal{A}_j^k$ using O_Δ. We describe the intuition of our solution with an example.

Example 5. *Consider the joins $S = \{J_1, \cdots, J_4\}$ of Fig. 2c. The areas $\mathcal{A}_1^k$ for $k \in [1, 4]$ are color-coded. We would like to compute the size of $\mathcal{A}_1^2$. The dotted, +, and × areas included all pairwise overlaps. Suppose we first compute the sum of the pairwise overlap size of joins with J_1, i.e., $\sum_{\Delta \in \mathbb{P}_2 \wedge J_1 \in \Delta} |O_\Delta|$, where $\mathbb{P}_2$ is the collection of all subsets of size 2 of S. However, to determine the area of the overlap of exactly one join with J_1, $\mathcal{A}_1^2$, we need to exclude all $\mathcal{A}_1^3$ and $\mathcal{A}_1^4$ areas. In fact, each subarea of $\mathcal{A}_1^3$ counts twice in the above sum. For example, $J_1 \cap J_2 \cap J_3$ is in both $J_1 \cap J_2$ and $J_1 \cap J_3$. Similarly, $\mathcal{A}_1^4$ counts three times in the sum of O_Δ's since it is included in $J_1 \cap J_2 \cap J_3$, $J_1 \cap J_2 \cap J_4$, and $J_1 \cap J_3 \cap J_4$. To avoid over-counting, the $\mathcal{A}_j^k$'s are weighed by $1/k$, in Eq. 1.*

Theorem 3. *Let $S = \{J_1, J_2, \ldots J_n\}$ and $\mathbb{P}_k$ be all subsets of size k of S, then for any join path J_j, and for any $1 \leq k \leq n$, we have*

$$|\mathcal{A}_j^k| = \sum_{\Delta \in \mathbb{P}_k \wedge J_j \in \Delta} |O_\Delta| - \left(\sum_{r=k+1}^{n} \binom{r-1}{k-1} \cdot |\mathcal{A}_j^r| \right).$$

For $k = n$, we have $|\mathcal{A}_j^n| = |O_S|$. For $k = 1$, we have the following.

$$|\mathcal{A}_j^1| = \sum_{\Delta \in \mathbb{P}_1 \wedge J_j \in \Delta} |O_\Delta| - \sum_{r=2}^{n} \binom{r-1}{0} |\mathcal{A}_j^k| = |J_j| - \sum_{r=2}^{n} |\mathcal{A}_j^r|$$

PROOF. When $k = n$, $\mathbb{P}_n$ is the set representing the universe S including J_j. Therefore, it is trivial that $|\mathcal{A}_j^n| = O_S$, which can be evaluated with $\bigcap_{J_j \in S} J_j$. Then, for $k \in [2, n - 1] \cap \mathbf{Z}$, we calculate $|\mathcal{A}_j^k|$ dynamically. Now, suppose we know $|\mathcal{A}_j^{k+1}|$. Recall $\mathcal{A}_j^k$ consists of all tuples in J_j that appear in exactly $k - 1$ other join paths. That is, tuples in J_j that are in some $\Delta \in \mathbb{P}_k$ but are not in any higher order overlap $\Delta' \in \mathbb{P}_r$, where $r \in [k + 1, n]$. Therefore, we first add up all the k-th overlap for sets $\Delta \in \mathbb{P}_k$, where $J_j \in \Delta$. Since J_j is confirmed, we have $\binom{n-1}{k-1}$ number of such sets Δ. Note that a tuple $t \in \mathcal{A}_j^k$ may appear in multiple $\Delta \in \mathbb{P}_r, r \in [k + 1, n]$. Therefore, to get the exact value of $|\mathcal{A}_j^k|$, for each $r \in [k + 1, n]$, we need to count the number of $\Delta \in \mathbb{P}_r$ where $J_j \in \Delta$. Starting with $r = k + 1$, each such combination of $\Delta \in \mathbb{P}_{k+1}$ contains J_j, therefore, it appears once in remaining $\binom{k}{k-1}$ number of $\Delta' \in \mathbb{P}_k$'s. Hence, we need to deduct $(k - 1) \cdot |\mathcal{A}_j^{k+1}|$ from the sum. Now for the general case r, where $k < r \leq n$, after J_j is confirmed, each combination of $\Delta \in \mathbb{P}_r$ has its other $k - 1$ paths chosen in $\binom{r-1}{k-1}$ number of $\Delta' \in \mathbb{P}_k$, so a total number of $\binom{r-1}{k-1} |\mathcal{A}_j^r|$ needs to be deducted from the sum for each r. Therefore, we can organize the formula of calculating $|A_j^k|$ as shown in the theorem. □

Using this theorem to compute $|\mathcal{A}_j^k|$'s for a given J_j and all $k \in [1, n]$, we start by initializing $|\mathcal{A}_j^n|$ with $|O_S|$ using the method proposed in § 4. Then, $|\mathcal{A}_j^{n-1}|$ requires evaluating $|\mathcal{A}_j^n|$ that have been already computed as well as $|O_\Delta|$ for each subset of size $n-1$ of S. Again, § 4 is used to compute a $|O_\Delta|$. In general, iterating from $n-1$ to 1, each $|\mathcal{A}_j^k|$ can be computed from $|\mathcal{A}_j^r|$'s, where $r \in (k, n]$, that have been already evaluated and $|O_\Delta|$'s that can be computed from our method for the pairwise join path overlap.

Computing the size of a set union requires computing the overlap of all k-subsets of joins, which is exponential in the number of input joins. We remark that in practice the number of input joins is small. However, when S is large, if we compute $|O_\Delta|$'s in the order of the bottom-up traversal of the powerset lattice of S, we can speed up by reusing some of the computation.

Warm-up Phase: Note that computing the exact values of k-overlaps and overlaps for an arbitrary number of joins and relations is computationally expensive or infeasible. Next, we present two instantiations of the framework for approximating these parameters. We consider two cases: centralized and decentralized [18]. In a centralized setting, relations are accessible through direct access to data, such as relations within databases. We propose RANDOM-WALK for this setting. In a decentralized setting, data is private or expensive to sample. Examples include data markets or large relations in databases. Our HISTOGRAM-BASED method is suitable for this setting. Different instantiations of the framework only differ in how the union size bound, and join overlap bounds are computed during the warm-up phase. We remark that both methods guarantee uniformity. There is a tradeoff between efficiency and cost of estimation: tighter upper bounds are more costly to set up, but once in place, can generate samples more efficiently. On the other hand, looser upper bounds are easier to compute but lead to low sampling efficiency (due to potentially higher rejection rates). We propose a modified version of union sampling based on the RANDOM-WALK method that does not require warm-up and strikes a better tradeoff between upper-bound computation and sampling efficiency.

5 INSTANTIATION WITH HISTOGRAMS

Database management systems often maintain histograms as a special type of column statistic that provides more detailed information about the data distribution in a table column during query optimization. These histograms are useful for cardinality estimation, particularly if the data in a column is skewed. In this section, we present ways of estimating join overlap and union size using these histograms and even more minimalistic statistics such as maximum degrees of tuples in relations. Here, we propose a solution for the case of chain join, inspired by Olken's seminal work on join size estimation[34]. In § 8.2, we extend our framework to more generic cyclic and acyclic joins.

5.1 Overlap of Equi-length Chain Joins

We start with estimating the overlap of multiple chain joins. Suppose all joins consist of the same number of relations and there is a one-to-one mapping between relations of each pair of joins such that mapped relations have the same schema. Given a collection

of joins S and a subset $\Delta \subseteq S$, let $O_\Delta = \bigcap_{J_j \in \Delta} J_j$ be the set of tuples that appear in all $J_j \in \Delta$. Trivially, a loose upper bound for the overlap is $\min\{|J_j| : J_j \in \Delta\}$. We first partition the joins on relations consistently. At each step, we estimate the overlap size of each sub-join dynamically from the overlap of smaller sub-joins by multiplying the overlap size of a smaller sub-joins by the minimum of the maximum degree of values join attributes. For example, for joins of three relations, we first evaluate the overlap of the first relations in all joins. Then, we evaluate the overlap of the first two relations in all joins by multiplying the overlap of the first relations by the minimum of the maximum degree of values of the first relations, and so on.

More formally, let $\mathcal{K}(i)$ be the upper bound of the number of overlapping tuples after the i-th join. Hence, $|O_\Delta| \leq \mathcal{K}(n-1)$. Let $M_{A_l}(R_{j,i})$ be the maximum degree of values in the domain of a join attribute A_l of relation $R_{j,i}$ of join J_j and let $d_{A_l}(v, R_{j,i})$ be the degree of value v in the domain of A_l. Note that the statistics of the degree of values are available from the histograms on join attributes. We can obtain an upper bound dynamically as $\mathcal{K}(i) = \mathcal{K}(i-1) \cdot \min_{J_j \in \Delta}\{M_{A_i}(R_{j,i+1})\}$. Note that for $\mathcal{K}_1$ we calculate the bounds based on values, i.e., $\mathcal{K}(1) = \sum_{v \in C} \min_{J_j \in \Delta}\{d_{A_1}(v, R_{j,1}) \cdot d_{A_1}(v, R_{j,2})\}$. So far, this bound requires the full histogram of the first relations in all joins and the maximum degree of values in the remaining relations. If the histograms are available for all join attributes in the relations, we can further refine the bound by replacing the term of the minimum of maximum degrees, $M_{A_i}(R_{j,i+1})$, with the minimum of the average degree of values in the join attributes.

5.2 Overlap of Chain Joins

We now release this assumption to accommodate joins with arbitrary length and arbitrary relation schemas. Note that the joins themselves should still have the same schemas after joining. We introduce the *splitting method* that aims to reorganize joins into joins on relations of the same size, so that the results of § 5.1 can be applied. The splitting method derives new joins by breaking down relations into sub-relations, each sub-relation consisting of exactly two attributes. The derived joins have the same schema and are lossless, i.e., each generates the same data as the original join, and all contain the same number of relations. Moreover, for each relation in a derived join, there are corresponding relations in other joins. Since the derived joins satisfy the requirements of § 5.1 and generate the same data, we can directly apply § 5.1 to estimate the overlap size of the original joins. Although the input joins may not include relations with the same schemas, they definitely have corresponding attributes and the same schema after joining. As such, breaking all relations in sub-relations of two attributes and redefining joins incurs join with the same number of same-schema relations.

Note that our splitting method is different than the normalization in the database theory which aims to decompose relations into sub-relations based on functional dependencies to avoid anomalies [13]. *Split* relations keep a record of their original sizes for the estimation steps. We call the join between two relations split from the same original relation *fake join*. The following theorem describes a generic way of bounding the overlap of chain joins.

Theorem 4. *Given a collection of split joins S and a subset $\Delta \subset S$, let $O_\Delta = \bigcap_{J_j \in \Delta} J_j$. Let $M_{A_l}(R_{j,i})$ be the maximum degree of values in the domain of a join attribute A_l of relation $R_{j,i}$ of join J_j and let $d_{A_l}(v, R_{j,i})$ be the degree of value v in the domain of A_l. We define the following.*

$$M_{j,i} = \begin{cases} M_{A_i}(R_{j,i+1}) & \text{if } R_{j,i} \bowtie R_{j,i+1} \\ \\ 1 & \text{if } R_{j,i} \bowtie' R_{j,i+1} \end{cases}$$

Let $\mathcal{K}(i)$ be the upper bound of the number of overlapping tuples after the i-th join and let $d_{A_l}(v, R_{j,i})$ be the degree of value v in the domain of A_l. We then obtain an upper bound for the overlap size of joins in Δ, $|O_\Delta|$, dynamically as follows.

$$|O_\Delta| \le \mathcal{K}(n-1) = \mathcal{K}(n-2) \cdot \min_{J_j \in \Delta}\{M_{j,n}\}$$

$$\mathcal{K}(1) = \sum_{v \in C} \min_{J_j \in \Delta}\{d_{A_1}(v, R_{j,1}) \cdot d_{A_1}(v, R_{j,2})\}$$

$$\mathcal{K}(i) = \mathcal{K}(i-1) \cdot \min_{J_j \in \Delta}\{M_{j,i}\}$$

The proof of this theorem is available in the extended version in the supplementary material.

We remark that Theorem 4 can become a biased estimator of join overlap if the data is skewed. Here, we present a solution with the least statistics available. We can extend the theorem, to become an unbiased estimator, in a straightforward way to use the histogram information of all join attributes and compute the expected value and upper bound of overlap.

6 INSTANTIATION WITH RANDOM WALKS

The techniques proposed in § 4 perform join union size estimation in a direct manner. In this section, we consider an alternative and more accurate way of estimating join overlap size in an online manner. The idea is to update the join size and overlap size on the fly, during the warm-up phase, by obtaining tuples from join paths and reusing these tuples during the main sampling step.

6.1 Join Size Estimation Revisited

To solve the online aggregation problem over join, wander join proposes an algorithm by performing random walks over the underlying join data graph [25]. This solution can be applied to join size estimation by computing the COUNT operation over the join. A join data graph models the join relationships among the tuples as a graph, where nodes are tuples and there is an edge between two tuples if they can join. Using a join graph, we can easily obtain successfully joined tuples by performing random walks. The probability of a tuple sampled from a join can be computed on the fly using the join graph. Given a join $J = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m$, the probability of a result tuple $t = t_1 \bowtie t_2 \bowtie \ldots \bowtie t_m$ is computed as $p(t) = \frac{1}{|R_1|} \cdot \frac{1}{|d_2(t_1)|} \cdot \ldots \cdot \frac{1}{|d_m(t_{m-1})|}$, where $d_i(t_{i-1})$ is the number of tuples in R_i than join with t_{i-1}.

Example 6. *Consider the index graph of J in Fig. 3d. The probability of choosing a_1 is $\frac{1}{5}$. Then among the three joinable tuples with a_1, the probability of selecting b_2 is $\frac{1}{2}$. Similarly, the probability of selecting c_1 is $\frac{1}{3}$. Therefore, the probability of obtaining tuple $a_1 \bowtie b_2 \bowtie c_1$ is $p(a_1 \bowtie b_2 \bowtie c_1) = \frac{1}{5} \times \frac{1}{2} \times \frac{1}{3}$.*

Suppose we have obtained a sample S of size m from a join path J. Following Horvitz-Thompson estimator [17], the estimated join size of J based on sample S, namely $|J|_S$ can be evaluated as $|J|_S = \sum_{t \in S} \frac{1}{p(t_k)} \cdot \frac{1}{m}$ [25]. We can update this estimation in real-time as new join samples are obtained. Suppose a new tuple t_0 is added to S, we can update the join size estimation as follows.

$$|J|_{S \cup t_0} = \frac{\sum_{t \in S} \frac{1}{p(t_k)} + \frac{1}{p(t_0)}}{(m+1)} = \frac{\sum_{t \in S} \frac{1}{p(t_k)}}{m} + \frac{\frac{m}{p(t_0)} - \sum_{t \in S}\frac{1}{p(t_k)}}{(m+1)m}$$
$$= |J|_S + \frac{1}{m+1}\left(\frac{1}{p(t_0)} - |J|_S\right)$$

We revisit the mean and variance of $|J|$ later in the discussion of random walk overlap. Hence, a real-time approximate answer is returned with some confidence level, and the accuracy improves as the sample size grows larger. Extending from wander join, we have two methods to estimate the overlap sizes.

First, we set an α as a parameter, which is the confidence level we want to achieve. There is a confidence level value z_α corresponding to the α. The half-width of the confidence interval is $\frac{z_\alpha \cdot \sigma}{\sqrt{n}}$, where n is the sample size and σ, is the standard deviation of the sample set. We terminate the sampling when the half-width becomes less than the threshold we defined.

6.2 Overlap of Joins

We described an algorithm based on random walks for sampling a join and estimating a join size. Given a set $\Delta \in S$ of join paths, we would like to estimate the overlap of joins in Δ, namely O_Δ. Let $S_j = \{t_1, t_2, \ldots, t_m\}$ denote a collection of sampled tuples from join $J_j \in \Delta$. Let $count(t)$ be the number of occurrences of tuple t in a set. We define S'_j such that for each tuple t in S_j, S'_j contains exactly $\frac{1}{p(t)}$ number of such tuple t, i.e., $S'_j = \{t \in S_j \mid count(t) = \frac{1}{p(t)}\}$. Thus, sample S'_j preserves the distribution of J_j. We assume uniformity, over overlap, and non-overlap regions among join paths, that is we sample tuples and estimate join sizes by performing random walks, for any $J_j \in \Delta$, we have $\frac{|O_\Delta|}{|J_j|} = \frac{|\bigcap_{J_j \in \Delta} S'_j|}{|S'_j|}$. Therefore, a join overlap size is estimated on the fly as follows.

$$|O_\Delta| = |\bigcap_{J_j \in \Delta} J_j| = |J_j| \cdot \frac{|\bigcap_{J_i \in \Delta} S'_i|}{|S'_j|} \tag{2}$$

How to get the $|\bigcap_{J_i \in \Delta} S'_i|$? We fix a $J_j \in \Delta$ and continually sample from this single source, forming the S_j. In each round, if we accept the sample t, then we check every $J_i \in \Delta$, where $i \ne j$ to see where t is contained in J_i. Since we already have the index for each J_i (stored in hash tables), this operation could be cheap since it just requires $(N-1) \times (M-1)$ queries with key, where $N = |\Delta|$ and M is the number of tables in a join path. If t is in every J_i, we include it into $\bigcap_{J_i \in \Delta} S'_i$. We can now plug in this estimation in Theorem 3 to compute the union size of joins in Δ. Next, we compute the confidence interval for $|O_\Delta|$. The variance of $|\bigcap_{J_i \in \Delta} S'_i|/|S'_j|$, denoted by σ_j^2, can be computed by a binomial sampling, with a variance of $\hat{p_j}(1 - \hat{p_j})$ and mean of $\hat{p_j}$ [24]. Li et al. showed the mean and variance of $|J_j|$, denoted by ϕ_j^2, are $T_n^j(u)(= \frac{1}{n-1}\sum_{i=1}^n f^j(i))$ and

$T_{n,2}^j(u)(= \frac{1}{n-1}\sum_{i=1}^n (f^j(i) - T_n^j(f))^2)$, respectively [25]. Assuming these terms are independent, we have the variance of $|O_\Delta|$ as follows.

$$\sigma_{|O_\Delta|}^2 = T_{n,2}^j(u) \cdot \hat{p}_j \cdot (1 - \hat{p}_j) + T_{n,2}^j(u) \cdot \hat{p}_j + T_n^j(u) \cdot \hat{p}_j \cdot (1 - \hat{p}_j)$$

This gives us the following confidence interval for $|O_\Delta|$ of Eq. 2.

$$E = z \cdot \sqrt{\frac{1}{n}\sum_{J_j \in \Delta}(T_{n,2}^j(u) \cdot \hat{p}_j \cdot (1 - \hat{p}_j)+ \atop T_{n,2}^j(u) \cdot \hat{p}_j + (T_n^j(u) \cdot \hat{p}_j \cdot (1 - \hat{p}_j))} \tag{3}$$

This means to obtain a 90% confidence on overlap estimation, the algorithm requires a sample size of $(\frac{1.96 \cdot z}{E} \cdot \sigma_{|O_\Delta|})^2$, on average.

Note that our estimator for overlap, using random walks, is unbiased. We first guarantee uniformity by adding $\frac{1}{p(t)}$ number of tuple t to the collection S_j. We know we have the following.

$$\lim_{|S_j| \to \infty} \frac{|\bigcap_{J_j \in \Delta} S_j'|}{|S_j'|} = \frac{\lim_{|S_j| \to |\Delta|} |\bigcap_{J_j \in \Delta} S_j'|}{\lim_{|S_j| \to |\Delta|} |S_j'|} = \frac{|\bigcap_{J_j \in \Delta} J_j|}{|J_j|}$$

Therefore, we can show that our result gets more and more accurate when $|S_j|$ gets larger and equals the exact result when $|S_j| = |\Delta|$. As the accuracy of overlap estimation gets closer to the true values, we also obtain a better estimation for the union size, which shows that our estimator improves for values used in our algorithms.

7 ONLINE UNION SAMPLING

The HISTOGRAM-BASED method has almost zero setup cost but low sampling efficiency, while the RANDOM-WALK method requires some sampling cost during the warm-up phase, but yields better estimation and efficiency. To design a sampling algorithm with a minimal setup cost and high sampling efficiency, we introduce an online union sampling algorithm as illustrated in Algorithm 2. At a high level, join and union size estimation is performed in an online manner as the union of joins is being sampled. Algorithm 2 extends Algorithm 1 with two optimizations: sample reuse and backtracking with parameter update. It initializes join and union parameters using the HISTOGRAM-BASED method, then, continues with selecting joins and sampling joins using the RANDOM-WALK method. At each iteration, obtained samples are used to further refine estimations using the join and union estimation proposed in § 6.1.

Sample Reuse (lines 8-10 of Algorithm 2) This makes up for the overhead of the RANDOM-WALK. Recall the tuples sampled by RANDOM-WALK are not uniform, however, with an extra accept/reject step we can reuse them in the main sampling phase. For each join, we keep track of every tuple t and its probability $p(t)$, computed during join sampling as described in § 6.1. Suppose we have already sampled $S = \{t_1, t_2, \ldots, t_l\}, t_i \in J_j$, from J_j. Recall S may have duplicates, i.e., there exists i, j s.t. $t_i = t_j$. Then, if we choose J_j, we can first randomly choose a tuple t from $t_1, t_2, \ldots, t_l$, but we only accept it with probability $\frac{l}{p(t) \cdot |J_j|}$. In this way, the algorithm guarantees that the reused t is sampled from J_j with probability $p(t) \cdot \frac{1}{l} \cdot \frac{l}{p(t) \cdot |J_j|} = \frac{1}{|J_j|}$ which ensures uniformity of sampling over the union. Note that if we accept t, we do not return t to the pool, i.e., it is a sample without replacement process and l is changing. Once we use all the tuples we stored, the next time J_j is selected, we simply sample over join using the techniques of § 3.2.

Note that the acceptance rate, namely R, can be equal to and greater than 1. This means the algorithm may return more than one instance of t in a certain round, while still ensuring the uniforming condition. We define the r_i as the probability that i instances of t be accepted in a certain round. That is, $\sum_i^n r_i \cdot i = R$, where $\sum_i^n r_i = 1$, $0 \le r_i \le 1$. Then, we have to choose the number of instances, $n \in N^+$, by choosing one of the many valid solutions of this system.

Algorithm 2 Set Union Sampling with Reuse and Backtracking

Input: Join paths $\{J_j, 1 \le j \le m\}$, tuple count N, backtrack para ϕ, target confidence level γ
**Output:** Tuples $\{t_i, 1 \le i \le N\}$
1: $\{|J_j|, 1 \le j \le n\}, |U|, \leftarrow warmup(S), \{|J_j'|\} \leftarrow cover(S)$
2: $conf\_level \leftarrow 0, T \leftarrow \{\}$ ▷ result sample
3: $P \leftarrow [j][]$ ▷ record probability of selected tuples from each join path
4: $orig\_join_i \leftarrow \{\}$ ▷ record of original join of seen tuples
5: **while** $n < N$ **do**
6:    select J_j with probability $\frac{|J_j'|}{|U|}$
7:    **if** $S_j \ne \emptyset$ **then**
8:       sample $t \in S_j$, accept with $\frac{l}{p(t) \cdot |J_j|}$, remove t from S_j
9:    **if** $S_j = \emptyset$ or t from S_j is rejected **then**
10:       $t \leftarrow$ a random sample from J_j
11:    **if** $t \in orig\_join_i$ for any $i < j$ **then** reject t
12:    **else**
13:       **if** $t \in orig\_join_i$ for any $i > j$ **then** ▷ revision
14:          remove t from $orig\_join_i$ and add t to $orig\_join_j$
15:          remove all t's from T and delete $P[i][t]$'s
16:       **if** $t \notin orig\_join_i$ **then** add t to $orig\_join_j$
17:       $T \leftarrow T \cup \{t\}$ and update $P[j][t]$
18:       **if** $\sum_{j \in [m]} |P[j]| \% \phi == 0$ and $conf\_level < \gamma$ **then**
19:          $\{|J_j|, 1 \le j \le m\}, |U|, T$
20:          $conf\_level \leftarrow Update(T, P)$ ▷ backtrack (§ 7)
21: **return** T

Backtracking with Parameter Update In § 4, we show that despite the small overhead of the HISTOGRAM-BASED method and its usefulness, the HISTOGRAM-BASED method may not be an unbiased estimator of our sampling parameters. Moreover, in § 6, we proved that the RANDOM-WALK is an unbiased estimator whose parameter estimations converge to the true values after infinitely many numbers of samples. Algorithm 2 initializes the framework with the estimation of the HISTOGRAM-BASED method and refines the parameters by applying RANDOM-WALK. The caveat is that, with this refinement strategy, although at each round the probability of sampled tuples is uniform and equal to $1/|U|$, the uniformity of tuples sampled across rounds is not guaranteed since the estimation of $|U|$ changes from one round to another with more random walks. To mitigate this non-uniformity, we introduce a backtracking trick which is an accept/reject strategy for all already sampled tuples in previous rounds.

Algorithm 2 initializes T to be the set of result samples and initializes a list P to store, $p(t)$'s, the probabilities of tuples obtained from a join either it is accepted, rejected, or when the random walk fails. We also specify a parameter ϕ, which indicates how often we backtrack. During the sampling process, we record all $p(t)$'s regardless of t being a rejected or reused tuple or being the
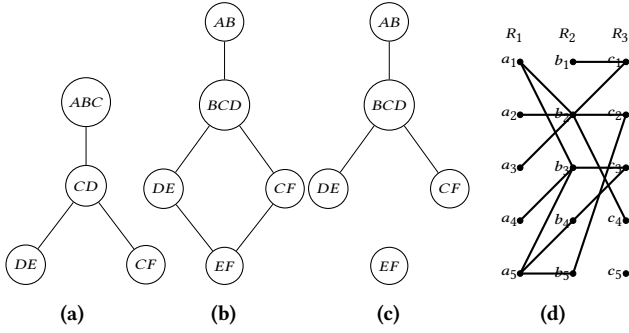
Figure 3: (a) Acyclic join, (b) Cyclic join, (c) Tree structure for overlap estimation of (b), (d) Skeleton join and residual joins for random sampling (d') Join data graph of J

result of a failed random walk (in this case, $p(t) = 0$). Every ϕ iterations, i.e., ϕ recorded $p(t)$'s, we update join, overlap, and union estimations following the RANDOM-WALK method then perform backtracking following Algorithm ?? to adjust the probability of previously sampled tuples based on the new estimation of $|U|$.

During backtracking, we iterate over all previously sampled tuples in the result and adjust their probabilities by rejecting tuple t with probability $\frac{|J(t.val)'|/|U|'}{|J(t.val)|/|U|}$, where $|J(t.val)|$ and $|U|$ are original values, and $|J(t.val)|'$ and $|U|$ are updated values. It is not hard to see that the backtracking algorithm guarantees that each tuple in the result is sampled with $\frac{1}{|U|'}$. We also keep track of the confidence level γ of the estimated sizes and stop backtracking when the accuracy is beyond a predefined threshold.

8 OTHER TYPES OF JOINS

In this section, we show how to generalize our sampling framework to acyclic joins. The join subroutine of our algorithm relies on an existing algorithm. The work by Zhao et al. provides a way of random sampling over join of all types: chain, acyclic, and acyclic [38]. The two discussed instantiations of our framework propose different ways of estimating join overlap size parameters. The RANDOM-WALK method relies on samples obtained from joins for estimation and handles acyclic and cyclic joins in the subroutine of join sampling. For brevity, we do not repeat the algorithm of Zhao et al. and describe how we extend the HISTOGRAM-BASED method to acyclic and cyclic joins.

8.1 Acyclic Joins

We organize the relations in a join tree, where each node refers to a relation and each edge denotes a join. Figure 3c illustrates an example of a join tree. The basic idea in extending our sampling algorithm to acyclic joins is to transform all acyclic joins and chain joins in the union to the base case of equi-length chain joins and use the results of § 5.1 to estimate join overlaps. Our solution involves first building a *standard template* of joins. A template is a join tree structure to which the structure of every join can be converted. We formalize the standard template as a chain join that contains relations of two attributes. The reason we need the template is that the degree-based comparison which is necessary for the size estimation of § 5.1 can only be applied when relations have exactly the sample structure.

To rewrite an acyclic join as a base chain join, we first construct the equivalent join tree such that a breadth-first traversal, always starting from the left-most node in each level, gives us joins of the same schema for all trees. A chain join is indeed a join tree with one branch. Joins may result in different tree structures. Therefore, we next need to choose a standard tree structure (template) before decomposing them into base chain joins. A good template is important in the estimation process. A bad template can lead us to the worst bound results of $\min_{j \in [n]} |J_j|$.

Example 7. *Consider the join in Fig. 3a. Suppose we choose the template of $(A, D) \bowtie (A, C) \bowtie (B, C) \bowtie (B, E) \bowtie (E, F)$. To obtain (B, E), we need to estimate the size of $(A, B, C) \bowtie (C, D) \bowtie (D, E)$; to obtain (E, F), we need to estimate the size of $(D, E) \bowtie (C, D) \bowtie (C, F)$. Since we also need to estimate the fake join size, these two estimations between relations lose lots of information. However, the template $(A, B) \bowtie (B, C) \bowtie (C, D) \bowtie (D, E) \bowtie (E, F)$ gives us a better bound as we only use the pre-estimation for relations once to obtain (E, F).*

It is not hard to notice that if we want to preserve most of the structure of the original relations, we prefer templates that put attributes in their original relations. We formulate the problem of finding a standard template for a collection of chain and cyclic joins as the problem of splitting joins into two-attribute relations such that the total pairwise distance of attributes in the same relation, in the tree of the template, is minimized. In the extended version of the paper, we introduce heuristics for constructing an optimal standard template.

8.2 Cyclic Joins

In this section, we extend our sampling algorithm to cyclic queries. Following the method proposed in [38], we break all the cycles in the join hyper-graph by removing a subset of relations so that the join becomes a connected and acyclic join. The residual join, namely $\mathcal{S}_R$, is the set of removed relations and the skeleton join, namely $\mathcal{S}_M$, is the set of relations in the main acyclic join. Fig. 3c shows the equivalent skeleton join tree and residual join to the cyclic join of Fig. 3b. Let attributes in $\mathcal{S}_R$ be $Attr(\mathcal{S}_R)$, and attributes in $\mathcal{S}_M$ as $Attr(\mathcal{S}_M)$. We treat $\mathcal{S}_R$ as a single relation in the new acyclic join. We can even materialize $\mathcal{S}_R$ by performing joins in $\mathcal{S}_R$. Note that some attributes in $Attr(\mathcal{S}_R)$ from the residual $\mathcal{S}_R$ may be joined with $Attr(\mathcal{S}_M)$. This means we have an acyclic join (the skeleton join) and a residual that can be joined with two or more relations in the skeleton. Now the maximum degree $M(\mathcal{S}_R)$ of any attribute in $\mathcal{S}_R$ is defined as follows.

$$M(\mathcal{S}_R) = \max_{v_i \in A_i} |t : t \in \mathcal{S}_R, \pi_{A_i}(t) = v_i, \forall A_i \in Attr(\mathcal{S}_M) \cap Attr(\mathcal{S}_R)|$$

Since we treat the residual as one relation, with the degree information, we can estimate the join size and overlap size by breaking $\mathcal{S}_R$ into the base chain join structure, as described in § 8.1. Note that the choice of set or relations to remove can have a significant influence on performance. We follow the methods used by Zhao et al. [38] to decide where to break the cycle in practice.

8.3 Selection Predicates

Our sampling algorithms can support selection predicates in two ways. The first alternative is to push down the predicates to relations, i.e., we filter each relation with the predicates, during the preprocessing, and work with filtered relations during sampling. This paradigm works for both HISTOGRAM-BASED and RANDOM-WALK. Another alternative is to enforce the selection predicate during the sampling process. This paradigm works with only RANDOM-WALK, unless the HISTOGRAM-BASED method has access to the selectivity degree of the predicate and can adjust the degree statistics. Since this paradigm adds an additional rejection factor, it is most appropriate for selection predicates that are not very selective.

9 EVALUATION

Datasets: We use three datasets consisting of different types of joins tailored from the TPC-H benchmark. Each query workload is to sample from the union of joins in a dataset. UQ1 consists of five chain joins, where each has five relations: `nation`, `supplier`, `customer`, `orders`, and `lineitem`; UQ2 consists of three chain joins which use: `region`, `nation`, `supplier`, `partsupp`, and `part`, where we also add selection predicates following $Q_2^N \cup Q_2^P \cup Q_2^S$ in [8]; and, UQ3 has one acyclic join and two chain joins. UQ3 is derived from relations: `supplier`, `customer`, and `orders`. We split them vertically and horizontally to get relations with different schemas. Therefore, working with UQ3 involves the application of the splitting method.

To experiment with the scale of data, we use TPCH-DBGen to generate relations with various scales. For example, with TPC-H scale factor N-gb, and $K\%$ scale ratio, UQ3 is a dataset of size $K\% \cdot N \cdot 3$. For UQ2, we have the same data for three joins but have different constraints for selection predicates. Hence, UQ2 has a large overlap scale. We also vary the overlap scale $P\%$ between joins of UQ1. When generating different queries, we keep $P\%$ of the data the same in the original corresponding relations. This way, although we cannot ensure that the overlap ratio in queries is exactly $P\%$, given unknown information between relations, we can guarantee that the overlap ratio between queries is proportional to the overlap scale. Note that we did not perform experiments on cyclic joins queries, particularly because transforming cyclic to acyclic joins and online sampling from cyclic join is done based on an existing work [38].

Algorithms We evaluate the HISTOGRAM-BASED and a RANDOM-WALK instantiations by plugging in techniques of § 4 and § 6, respectively, in Theorem 3. The join estimation of HISTOGRAM-BASED can be instantiated by baselines EW (Exact Weight) [38], which is the ground truth for weights by calculating the exact weight of each tuple in the join data graph, or EO (Extended Olken's) [38], which we described in § 3.2. The join estimation of the online technique uses our RANDOM-WALK of § 6. We also consider FULLJOINUNION as the ground truth for our join size and union size estimations. This algorithm performs the full join and computes the union. Note that FULLJOINUNION is extremely expensive on large datasets. Our experiments timed out on data sizes of more than 5GB (per relation). We do not evaluate DISJOINUNION since it is consistent with sampling over one join path as it has no extra delays. we do not evaluate the Bernoulli set union sampling since it is a slightly

different variation of the Non-Bernoulli and has lower efficiency theoretically.

Implementation: The framework is implemented in Python. Relations in joins are stored in hash relations with a linear search. Acyclic joins are implemented in a tree structure and acyclic joins are handled by recursion. All experiments are conducted on a machine with 2 Intel® Xeon Gold 5218 @ 2.30GHz (64 cores), 512 GB DDR4 memory, a Samsung® SSD 983 DCT M.2 (2 TB), 4 GPUs - TU102 (GeForce RTX 2080 Ti).

9.1 Join and Union Size Approximation

9.1.1 Error. We evaluate the estimation error of the ratio $|J_i|/|U|$ for each join in a query, because our algorithms rely on this ratio to define probability distributions over joins. For these experiments, we use UQ1 and UQ3 with 3GB scale raw data. After preprocessing, UQ1 is 9GB and UQ3 is 5.4GB.The overlap scale is set to 0.2. Fig. 4a and 4b show the ratio estimation error for UQ1 and UQ3, with respect to overlap scale, using HISTOGRAM-BASED method.

For large overlap scales, the error tends to be small and stable. For smaller scales, the performance is unstable. This is because when the overlap scale is small, small samples will have a large effect on the estimation performance. However, when we have a large scale of overlap, which is our use case, the randomness will be removed. Besides, we observe that the average error for UQ3, in Fig. 4b, is better than UQ1, in Fig. 4a. As we take an upper bound for every join, our HISTOGRAM-BASED method gains higher accuracy on joins with a smaller length. Given that UQ3 is smaller both in length and numbers, this explains why the estimation is relatively more accurate for UQ3.

9.1.2 Runtime. We report the runtime of our parameter estimation methods, in Fig. 4c and 4d. First, HISTOGRAM-BASED *is significantly faster than the brute-force full join.* Second, for UQ1, we observe that as the cost of full join increases with overlap scale, the time HISTOGRAM-BASED method needs becomes less. This is because when the overlap scale is large and the overlapping structure is complex, it becomes harder for the full join to scan over data, but for our method, a higher overlap scale instead accelerates our method in finding the tuple with the maximum degree.

Unlike the HISTOGRAM-BASED technique, our RANDOM-WALK technique collects sampling statistics during the warm-up phase. When evaluating the confidence level of the overlap size, we are actually evaluating the ratio that the overlap part takes in the join, i.e., $\frac{|\bigcap_{J_i \in \Delta} S_i'|}{|S_j'|}$. In Eq. 2, we take $|J_j|$ as an exact value to fulfill the assumption of independence. This is in fact equivalent to having the confidence level of $|J_j|$ as 1 and confidence interval as 0, which is an approximation of the case given by Wande Join [25]. We terminate online sampling when the confidence level reaches 90% or we obtain 1,000 samples.

Fig. 5a compares the performance of HISTOGRAM-BASED with EO [33, 38], as join size instantiation, with RANDOM-WALK, in terms of the error of join to union size ratio estimation on UQ1. We used a data scale of 3GB for each query. First, RANDOM-WALK *outperforms* HISTOGRAM-BASED; *in fact,* RANDOM-WALK *is extremely accurate and stable and has an error close to zero for all joins.* This is because the
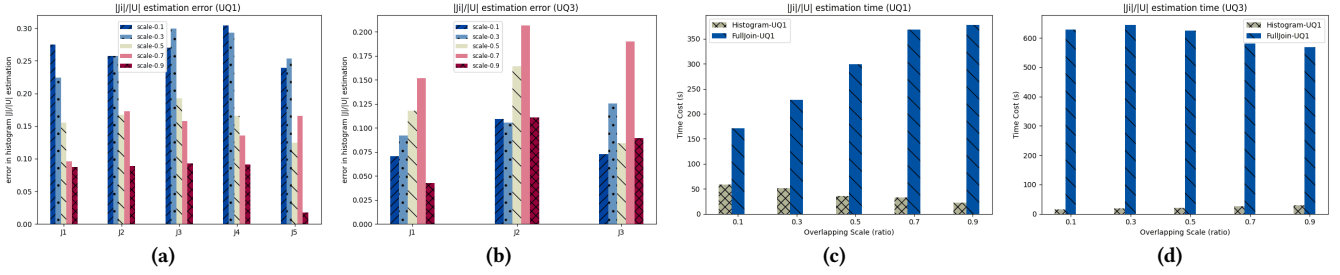
Figure 4: The error of join to union size ratio estimation using HISTOGRAM-BASED **+EO on (a) UQ1 and (b) UQ3; runtime of union size estimation using** HISTOGRAM-BASED **and** FULLJOIN **on (c) UQ1 and (d) UQ3.**

nature of indexing will give us extra information about overlapping. We remark that the accurate estimation comes at the cost of sampling during the warm-up phase. We will discuss the empirical evaluation of the sampling technique that reuses these samples, shortly. Besides, *while the estimation error is quite robust across joins, the higher the overlap, the more accurate* HISTOGRAM-BASED *becomes*. Since the accuracy of overlap size estimation heavily depends on the overlap size of samples we collect, the larger the actual overlap is, the easier we find overlap in samples, As we take the minimum in each step as the upper bound of overlap size, the bound gets tighter when overlap size approaches data size, which results in more accurate results in overlap size estimation. Nevertheless, though RANDOM-WALK has better performance, HISTOGRAM-BASED is relatively faster and can be applied to databases without index structures.

9.2 Set Union Sampling

9.2.1 Scaling with Number of Samples. For HISTOGRAM-BASED, We use both EW and EO methods for weights initialization in sampling from a single join, and we only use EW for RANDOM-WALK. First, Fig. 5c, 5d, and 5e show how SETUNION scale with number of samples. Overall, we can see that when using EW instantiation, HISTOGRAM-BASED and RANDOM-WALK have nearly no difference in performance. In other words, *the accuracy of the estimation bound has little impact on sampling efficiency.* However, for HISTOGRAM-BASED, using EW results in a much slower situation than using EO on all three queries, since with exact weights calculated, we obtain a rejection rate of zero.

9.2.2 Runtime Breakdown. Fig. 5f, 5g, and 5h shows the comparisons of time spent on parameter estimation(join size, overlap, and weights), producing accepted answers and on producing rejected answers. The reason for the decay comes from EO, as well as the fact that we need to reject duplicate tuples that are sampled from a join different from what it is assigned to. From these plots, the most significant finding is that though using EO is much less efficient than using EW, it has better performance in the warm-up phase. Moreover, since it uses the upper bound of weights for sampling from a join, it has an extra rejection phase and needs to spend much more time on rejected answers than using EW. Besides, the time spent on accepted answers is similar for three combinations of instantiations for all queries. Moreover, *our* SETUNION *algorithm*

spends minor time rejecting duplicate tuples and has very high efficiency when using EW *for join sampling.*

9.2.3 Scaling with Relation Size. Although we use *scale factor* = 5 for all three queries, we will get different sizes of unions if we perform full joins due to different numbers of relations and different levels of overlaps. From our set out, the order of union size for three queries from large to small is UQ1, UQ3, UQ2. From both sets of plots, we notice that sampling time is in proportion to the resulting union size. What's more, when the expected union size is small, as for UQ2, EO has a relatively smaller gap with EW during sampling, and has an even better advantage in the warm-up phase.

Moreover, Fig. 5b reports sampling time for various data scales for UQ1. The first observation is that using EO for join size estimation makes both algorithms slower than using EW and overall EW scales better with the size of data since with exact weights the rejection rate for sampling from single join path is 0. Second, though the sampling time of both algorithms increases with the size of data, the scale has a much larger effect on EO than EW. As the size of each relation grows larger, a tuple has a higher rejection rate due to the growth of the number of tuples in the relation to be joined with. Finally, initialization in union size using either HISTOGRAM-BASED or RANDOM-WALK has little impact on efficiency, which is consistent with the conclusion we obtained earlier.

9.3 Online Union Sampling with Sample Reuse

In the next set of experiments, we evaluate the runtime of the RANDOM-WALK sampling using the idea of reusing samples collected during warm-up. We compare RANDOM-WALK with and without reuse on all three queries. Fig. 6a shows sampling time with respect to sample size. First, we can clearly observe that we have much higher efficiency when we sample with reuse. When we sample from the pool of pre-sampled and joined tuples during the warm-up phase, we only do a fast check on rejection or acceptance and do not need to sample over each relation. Moreover, there is a slight change in slope on lines of sampling with reuse cases. When pre-sampled samples are all used, the performance of SETUNION will slowly converge to their original performance. One other interesting phenomenon is that the reuse of samples has a more apparent increase in performance when the expected union size is larger. For UQ1, there is a huge gap between with and without reuse; but for UQ2, the gap is much smaller. Fig. 6b compares to time spent on successfully accepting one tuple in the regular sampling phase and
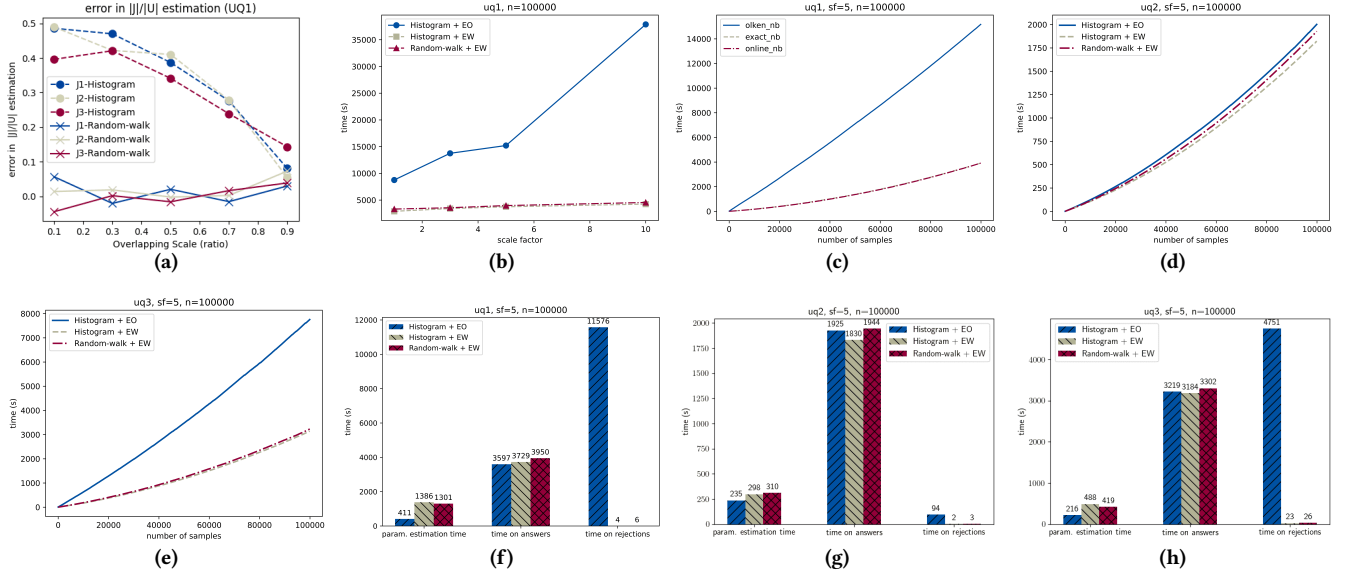
Figure 5: (a) the error of join to union size ratio; (b) SETUNION time vs. data scale on UQ1; runtime vs. sample size on (c) UQ1 and (d) UQ2; (e) UQ3; time breakdown of (f) UQ1 (g) UQ2 and (h) UQ3.
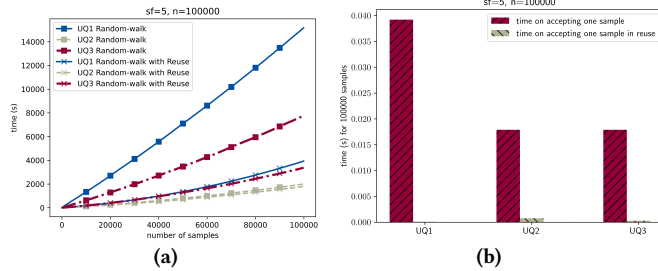

Figure 6: (a) time vs. sample size with and without reuse (b) time per sample spent in a regular phase vs. a reuse phase.

in the reuse sampling phase. We use the ratio of total time spent on sampling and the number of successfully sampled tuples for each phase for comparison, and we can see that when we reuse pre-sampled tuples, we have much higher efficiency. This shows the huge improvement in efficiency brought by our online union sampling.

10 RELATED WORK

Random Access to Query Results The closest problem to ours is random access to the results of conjunctive queries. Bagan et al. show that the free-connex acyclic conjunctive queries can be evaluated using an enumeration algorithm with a constant delay between consecutive answers, at the cost of a linear-time preprocessing phase [5]. However, because this work does not guarantee the randomness of the intermediate answers, the produced result may have extreme bias, making it unsuitable for learning tasks. Recently, Carmeli et al. studied the problem of enumerating the answers of the union of acyclic conjunctive queries in a uniformly random order [8]. The proposed algorithm requires full access to

the database, i.e., the computation of the full joins as well as a linear pre-processing time in the size of the database. As such, this algorithm is not applicable to random sampling over open data, data markets, proprietary databases, or web databases where the access model is tuple-at-a-time access. Unlike the approach of Carmeli et al. which requires computing the exact join and overlap sizes, our framework presents sampling strategies and ways of approximating these parameters using simple statistics, such as degrees, in our direct method or a subset of random samples in our online method.

Random Sampling over Joins The problem of random sampling over a single join path was posed in the 1990s [1]. Acharya et al. proposed a solution for good approximate answers using only random samples from the base relations, but accuracy still remained to be improved [1]. Joining random samples of joins produces a much smaller sample size than samples. Moreover, it is shown that join samples obtained do not satisfy the independence requirement [18]. To solve this, Olken proposed the idea of rejecting join of two samples with specific probabilities for two-table join [33]; Chaudhuri et al. proposed techniques that are applicable to linear joins but not to arbitrary joins [10]. Both methods require full information of the tables as well as the index structure. Chaudhuri et al. significantly improved the efficiency by proposing another strategy group sample algorithm that relies on only partial statistics [10]. However, all the above three methods only work for 2-table Joins. Ripple join returns dependent and uniform samples [16]. Wander join [25] extended ripple join to return independent but non-uniform samples from the join. Recently, Zhao et al. proposed a framework that handles general multi-way joins and guarantees i.i.d [38]. This algorithm can be plugged in our framework for random sampling over a single join path.

Union of Sets and Queries The union-of-sets problem has been studied in approximate counting literature [21]. The goal is to

design a randomized algorithm that can output an approximation of the size of the union of sets efficiently. Karp et al. proposed a $(1+\epsilon)$-randomized approximation algorithm for approximating the size of the union of sets with a linear running time. This algorithm requires the exact size of each set and a uniform random sample of each set. [21]. Bringmann and Friedrich later applied this algorithm in designing an algorithm for high dimensional geometric objects using uniform random sampling. They also proved that the problem is #P-hard for high dimensional boxes [7]. The computation of union of sets also has links to 0-th frequency moment estimation [2]. One line of work in this area is on DNF counting problem [20], including designing hashing-based algorithms [9, 14, 21, 28]. Another popular line of work is on estimating the union of sets where each set arrives in a streaming fashion [6, 19, 27, 29].

11 CONCLUSION

This paper studies two novel problems: sampling over the union of joins and size approximation of the union of joins. A general union sampling framework is proposed that estimates join overlap and union parameters when (1) data statistics are available in DBMSs and (2) access to the data in relations is feasible. The framework extends to the union size of joins of arbitrary multi-way acyclic and cyclic. Interesting future work directions include analyzing the impact of data skew on approximations as well as integrating a union sampling operator into a database engine.

REFERENCES

[1] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. Join Synopses for Approximate Query Answering. *SIGMOD Rec.* 28, 2 (jun 1999), 275–286. https://doi.org/10.1145/304181.304207

[2] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The Space Complexity of Approximating the Frequency Moments. In *STOC*, Gary L. Miller (Ed.). ACM, 20–29.

[3] Abolfazl Asudeh and Fatemeh Nargesian. 2022. Towards Distribution-aware Query Answering in Data Markets. *PVLDB* 15, 11 (2022), 3137–3144.

[4] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size Bounds and Query Plans for Relational Joins. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*. IEEE Computer Society, USA, 739–748. https://doi.org/10.1109/FOCS.2008.43

[5] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *CSL*, Jacques Duparc and Thomas A. Henzinger (Eds.), Vol. 4646. 208–222.

[6] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. 2002. Counting Distinct Elements in a Data Stream. In *RANDOM (Lecture Notes in Computer Science)*, José D. P. Rolim and Salil P. Vadhan (Eds.), Vol. 2483. Springer, 1–10.

[7] Karl Bringmann and Tobias Friedrich. 2008. Approximating the Volume of Unions and Intersections of High-Dimensional Geometric Objects. In *ISAAC (Lecture Notes in Computer Science)*, Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga (Eds.), Vol. 5369. Springer, 436–447.

[8] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. 2020. Answering (Unions of) Conjunctive Queries using Random Access and Random-Order Enumeration. In *PODS*. 393–409.

[9] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2016. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *IJCAI*, Subbarao Kambhampati (Ed.). 3569–3576.

[10] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1999. On Random Sampling over Joins. *SIGMOD Rec.* 28, 2 (jun 1999), 263–274. https://doi.org/10.1145/304181.304206

[11] Lingjiao Chen, Arun Kumar, Jeffrey F. Naughton, and Jignesh M. Patel. 2017. Towards Linear Algebra over Normalized Data. *PVLDB* 10, 11 (2017), 1214–1225.

[12] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David R. Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020), 1373–1387.

[13] E. F. Codd. 1971. Further Normalization of the Data Base Relational Model. *Research Report / RJ / IBM / San Jose, California* RJ909 (1971).

[14] Paul Dagum, Richard M. Karp, Michael Luby, and Sheldon M. Ross. 2000. An Optimal Algorithm for Monte Carlo Estimation. *SIAM* 29, 5 (2000), 1484–1496.

[15] Arnaud Durand and Yann Strozecki. 2011. Enumeration Complexity of Logical Query Problems with Second-order Variables. In *CSL (LIPIcs)*, Marc Bezem (Ed.), Vol. 12. 189–202.

[16] Peter J. Haas and Joseph M. Hellerstein. 1999. Ripple Joins for Online Aggregation. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (Philadelphia, Pennsylvania, USA) *(SIGMOD '99)*. Association for Computing Machinery, New York, NY, USA, 287–298. https://doi.org/10.1145/304182.304208

[17] D.G. Horvitz and DJ Thompson. 1952. A generalization of sampling without replacement from a finite universe. *J. Amer. Statist. Assoc.* 47, 260 (1952), 663–685.

[18] Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. 2019. Join on Samples: A Theoretical Guide for Practitioners. *PVLDB* 13, 4 (2019), 547–560.

[19] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. 2010. An optimal algorithm for the distinct elements problem. In *PODS*, Jan Paredaens and Dirk Van Gucht (Eds.). ACM, 41–52.

[20] Richard M. Karp and Michael Luby. 1983. Monte-Carlo Algorithms for Enumeration and Reliability Problems. In *FOCS*. 56–64.

[21] Richard M. Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo Approximation Algorithms for Enumeration Problems. *J. Algorithms* 10, 3 (1989), 429–448.

[22] Arun Kumar, Jeffrey F. Naughton, and Jignesh M. Patel. 2015. Learning Generalized Linear Models Over Normalized Data. In *SIGMOD*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1969–1984.

[23] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, and Xiaojin Zhu. 2016. To Join or Not to Join?: Thinking Twice about Joins before Feature Selection. In *SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 19–34.

[24] Yong-Gu Lee and Sam-Yong Kim. 2008. Introduction to statistics. *Yulgokbooks, Korea* (2008), 342–351.

[25] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 615–629. https://doi.org/10.1145/2882903.2915235

[26] Kaiyu Li and Guoliang Li. 2018. Approximate Query Processing: What is New and Where to Go? - A Survey on Approximate Query Processing. *Data Sci. Eng.* 3, 4 (2018), 379–397.

[27] Kuldeep S. Meel, Sourav Chakraborty, and N. V. Vinodchandran. 2022. Estimation of the Size of Union of Delphic Sets: Achieving Independence from Stream Size. In *PODS*. ACM, 41–52.

[28] Kuldeep S. Meel, Aditya A. Shrotri, and Moshe Y. Vardi. 2017. On Hashing-Based Approaches to Approximate DNF-Counting. In *FSTTCS (LIPIcs)*, Satya V. Lokam and R. Ramanujam (Eds.), Vol. 93. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 41:1–41:14.

[29] Kuldeep S. Meel, N. V. Vinodchandran, and Sourav Chakraborty. 2021. Estimating the Size of Union of Sets in Streaming Models. In *PODS*. ACM, 126–137.

[30] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.

[31] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* 11, 7 (2018), 813–825.

[32] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case optimal join algorithms. *Journal of the ACM (JACM)* 65, 3 (2018), 1–40.

[33] Frank Olken. 1993. *Random Sampling from Databases*. Ph.D. Dissertation. University of California at Berkeley.

[34] Frank Olken and Doron Rotem. 1995. Random sampling from databases: a survey. *Statistics and Computing* 5, 1 (1995), 25–42.

[35] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. 2016. Learning Linear Regression Models over Factorized Joins. In *SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 3–18.

[36] V. N. Vapnik and A. Y. Chervonenkis. 1971. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and its Applications* XVI, 2 (1971), 264–280.

[37] Swarup Acharya Phillip B Gibbons Viswanath and Poosala Sridhar Ramaswamy. 1998. Join Synopses for Approximate Query Answering. (1998).

[38] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1525–1539. https://doi.org/10.1145/3183713.3183739