# Sampling over Union of Joins

Anonymous Author(s)

## ABSTRACT

Data scientists often draw on multiple relational data sources to collect training data. A standard assumption in machine learning is that training data is an i.i.d sample of the underlying distribution. Given a set of joins, we consider 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 and disjoint union of chain, acyclic, and cyclic joins, with sample uniformity and independence guarantees. We study the novel problem of union size approximation of joins and propose a direct way and an online aggregation way of approximating the overlap size of joins. 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.

**ACM Reference Format:**
Anonymous Author(s). 2018. Sampling over Union of Joins. In *SIGMOD '23: ACM Symposium on Neural Gaze Detection, June 18–23, 2023, Seattle, WA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/1122445.1122456

## 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 [10]. Moreover, data may be collected from distributed sources, each described as a join over internal databases, data lakes, or web data [3, 11]. Therefore, the target data is the union of the results of joins.

Because 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 [10, 20, 21, 32]. Fortunately, an important result [33] from the learning theory suggests that learning and approximate query answering [24] do not require the full results and an i.i.d sample can achieve a bounded error. Since this holds for any model, when collecting data from multiple relational sources, the question to ask is how to sample over the union of joins to enable a general solution to learning over the union. This is the central problem we study in this paper. 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.


*SIGMOD '23, June 18–23, 2023, Seattle, WA*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

**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 required features, these queries need to join data from various relations. Note that join queries $J_W$, $J_E$, and $J_{MW}$ are acyclic, cyclic, and chain joins, respectively. In $J_W$, relation* `Orders` *is self-joined to obtain the information of items in the same order (bundle purchases). All three joins have unionable result schemas, i.e., there is a one-to-one mapping between their attributes. The attribute names have been renamed for better readability. 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 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, 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 [16]. For approximate query answering, some techniques such as RippleJoin [14] and WanderJoin [23] 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 [9, 30]. The most recent work by Zhao et al. proposes a framework for sampling over one join that handles general multi-way joins [35]. 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, 9, 31].

**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 $r \in S_E \cap S_W \cap$*

$J\_W$ : **Customer_W**$(CustKey, CName, CNationKey, CAcctBal, \cdots) \bowtie$ **Part1_W**$(PartKey1, PName1, RetailPrirce1, \cdots) \bowtie$
**Supplier1_W**$(SuppKey1, SName1, \cdots) \bowtie$ **PartSupp1_W**$(PartKey1, SuppKey1, AvailQty1, \cdots) \bowtie$
**Part2_W**$(PartKey2, PName2, RetailPrirce2, \cdots) \bowtie$ **Supplier2_W**$(SuppKey2, SName2, \cdots) \bowtie$
**PartSupp2_W**$(PartKey2, SuppKey2, AvailQty1, \cdots) \bowtie$ **LineItem1_W**$(OrderKey1, PartKey1, SuppKey1, Quantity1, ReturnFlag1, \cdots) \bowtie$
**Orders1_W**$(OrderKey1, CustKey, OrderStatus1, TotalPrice1, \cdots) \bowtie$ **Orders2_W**$(OrderKey1, CustKey, OrderStatus2, TotalPrice2, \cdots)$
**LineItem2_W**$(OrderKey1, PartKey2, SuppKey2, Quantity2, ReturnFlag2, \cdots)$

$\bigcup$ $J\_E$ : **Customer_E**$(CustKey, CName, CNationKey, CAcctBal, \cdots) \bowtie$ **PartSupplier_E1**$(PartKey1, PName1, SuppKey1, SName1, \cdots) \bowtie$
**PartSupplier_E2**$(PartKey2, PName2, SuppKey2, SName2, AvailQty2, \cdots) \bowtie$
**LineItem_E1**$(OrderKey1, PartKey1, SuppKey1, Quantity1, ReturnFlag1, \cdots) \bowtie$
**DoubleOrders_E**$(OrderKey1, CustKey, OrderStatus1, OrderKey2, OrderStatus2, \cdots) \bowtie$
**LineItem_E2**$(OrderKey2, PartKey2, SuppKey2, Quantity2, ReturnFlag2, \cdots)$

$\bigcup$ $J\_MW$ : **Customer_MW**$(CustKey, CName, CNationKey, CAcctBal, \cdots) \bowtie$
**DoublePartSupplier_MW**$(PartKey1, SuppKey1, AvailQty1, PartKey2, SuppKey2, AvailQty2, \cdots) \bowtie$
**DoubleOrdersLineItem_MW**$(CustKey, OrderKey1, PartKey1, SuppKey1, \cdots, OrderKey2, PartKey2, SuppKey2, \cdots)$

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

$S_{MW}$, *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 § 3 and § 4, 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 union 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 union of joins. In particular, we consider two types of union: disjoint union and set union. This is also consistent with query language operators. There are several challenges to address 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 for the disjoint and set union 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, 19] and its online extension to streams [8, 13, 19, 26] has been extensively studied in the approximate counting literature, to the best of our knowledge, there is no study that addresses the problem of bounding the union size of joins without performing the full join and overlap. Third, direct union estimation requires knowing the overlap of an exponential number of sets of joins, each set in the powerset of join paths. We reduce the space of calculation by reformulating the problem to use $k$-overlaps of each join, i.e., the subset of a join result that is shared with *exactly* $(k-1)$ other joins. Next, we propose a direct way and online way for upper-bounding the overlap of joins with an arbitrary number of relations and all join types (chain, cyclic, and acyclic). The direct technique is fast, requires knowing the statistics of joins, and may incur a loose bound under circumstances. The online technique is accurate, needs warm-up sampling, and has theoretical guarantees. Finally, the warm-up phase of our online techniques incur some overhead on the algorithm. We extend the union sampling algorithm to enable reusing the tuples obtained from the warm-up.

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 disjoint and set union of joins (§ 3 and § 4). Any instantiation of the framework always returns uniform and independent samples from the full result (Theorems 1 and 3), but with different sampling efficiency (§ 8.2).
- We present the first study of the size approximation of union of joins and an indirect formulation of union approximation (§ **??**).
- We design direct (§ 7) and online (§ 8) techniques to bound the size of overlap of any collection of chain, acyclic, or cyclic joins.
- We extend our set union sampling technique to reuse the samples collected in a warm-up phase (§ 8.2).
- We perform extensive experimental evaluations using TPC-H benchmark to investigate the performance of the proposed union size estimation and sampling techniques and compare them with other baselines (§ 10). The results demonstrate the accuracy of union size estimation, as well as the efficiency and scalability of our union sampling techniques.

## 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 unionable join paths $S = \{J_1, \ldots, J_n\}$. A join path $J_j$ 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. Two join paths $J_i$ and $J_j$ are unionable if there is an alignment between attributes in every pair of $\mathcal{A}(J_i)$ and $\mathcal{A}(J_j)$. An alignment between two sets

$R_{1,1}$

| $A_1$ |
|---|
| 1 |
| 2 |
| 3 |

$R_{1,2}$

| $A_1$ | $B_1$ |
|---|---|
| 1 | 2 |
| 2 | 2 |
| 2 | 3 |
| 3 | 5 |
| 3 | 6 |
| 4 | 6 |

$R_{1,3}$

| $B_1$ | $C_1$ |
|---|---|
| 3 | 1 |
| 3 | 2 |
| 3 | 3 |
| 6 | 4 |

$R_{2,1}$

| $A_2$ |
|---|
| 1 |
| 3 |
| 5 |
| 6 |

$R_{2,2}$

| $A_2$ | $B_2$ |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 5 |
| 3 | 6 |
| 5 | 5 |
| 6 | 6 |

$R_{2,3}$

| $B_2$ | $C_2$ |
|---|---|
| 2 | 1 |
| 2 | 2 |
| 5 | 3 |
| 6 | 4 |
| 7 | 5 |

$J_1 = R_{1,1} \bowtie R_{1,2} \bowtie R_{1,3}$

| $A_1$ | $B_1$ | $C_1$ |
|---|---|---|
| 2 | 3 | 2 |
| 2 | 3 | 3 |
| 3 | 6 | 4 |

$J_2 = R_{2,1} \bowtie R_{2,2} \bowtie R_{2,3}$

| $A_2$ | $B_2$ | $C_2$ |
|---|---|---|
| 1 | 2 | 1 |
| 3 | 6 | 4 |
| 1 | 2 | 2 |
| 5 | 5 | 3 |
| 6 | 6 | 4 |

$J_1 \uplus J_2 =$
$\{(2,3,2), (2,3,3), (3,6,4), (1,2,1), (3,6,4), (1,2,2), (5,5,3), (6,6,4)\}$
$J_1 \cup J_2 = \{(2,3,2), (2,3,3), (3,6,4), (1,2,1), (1,2,2), (5,5,3), (6,6,4)\}$

**Table 1: Join path results**

of attributes is a one-to-one mapping between attributes, where attribute mapping can be defined based on a variety of attribute unionability measures [6, 28]. Indeed, this assumption is implicit in the input, otherwise, the input join paths would not be meaningful. For simplicity, similar to database query languages, we assume all join paths have the same output schema after performing the join in terms of the number and name of attributes. Note that join paths can still have different length 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 problem of sampling over 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 join paths $S = \{J_1, \ldots, J_n\}$, return $N$ independent samples from $V = J_1 \uplus \ldots \uplus J_n$ such that each sampled tuple is returned with probability $\frac{1}{|V|} = \frac{1}{J_1+\ldots+J_n}$.*

The set union operation eliminates duplicate tuples from a result set. 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 join paths $S = \{J_1, \ldots, J_n\}$, 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 $\frac{1}{|J_1 \cup \cdots \cup J_n|}$.*

**Example 3.** *Consider relations and join paths $J_1$ and $J_2$ of Table 1. Note that $R_{1,1}$ is joined with $R_{1,2}$ on attribute $A_1$. Suppose the user knows that attributes $R_{1,1}.A_1$ and $R_{2,1}.A_2$ are from the same semantic domain (e.g., domain of countries), thus, they are unionable. Similarly, $R_{1,2}.B_1$ is unionable with $R_{2,2}.B_2$ and $R_{1,3}.C_1$ is unionable with $R_{2,3}.C_2$. As a result, the union of $J_1$ and $J_2$ is a meaningful union. The set union and disjoint union of these joins are shown in Table 1. A set union sampling algorithm and a disjoint union algorithm return a tuple from $J_1 \cup J_2$ with probability $1/7$ and $1/15$, respectively.*

## 3 SAMPLING DISJOINT UNION

We first solve the disjoint union case which is simpler. Algorithm 1 shows the pseudocode for sampling a disjoint union. Given the disjoint union $V = J_1 \uplus \ldots \uplus J_n$, we first select a join path $J_j$ with probability $P(J_j) = \frac{|J_j|}{|J_1+\ldots+J_n|}$, then, we select a random tuple from $J_j$. We repeat the process until $N$ sampled tuples are obtained.

**Algorithm 1** Disjoint Union of Joins Sampling

**Input:** Join paths $S = \{J_j, 1 \le j \le n\}$, Tuple count $N$, join path sizes $\{|J_j|, 1 \le j \le n\}$,
**Output:** Sample $T = \{t_i, 1 \le i \le N\}$
1: $\{|J_j|, 1 \le j \le n\} \leftarrow warmup(S)$ ▷ direct (§ 5.1) or online (§ 8.1)
2: $T \leftarrow \{\}$ ▷ target sample
3: **while** $n < N$ **do**
4: $J \leftarrow$ a random join path $J_j$ with probability $\frac{|J_j|}{\sum_{j=1}^{n}|J_j|}$
5: $t \leftarrow$ sample from $J$, add $t$ to $T$, $n \leftarrow n+1$ ▷ using § 5.2
6: **return** $T$

Methods of sampling a tuple from a single join path has long been a popular problem [9, 9, 30, 34, 35]. We revisit random sampling over join in § 5.2.

**Theorem 1.** *Given join paths $S = \{J_1, \ldots, J_n\}$, Algorithm 1 returns each result tuple $t$ with probability $\frac{1}{|J_1+\ldots+J_n|}$.*

Proof. The algorithm selects join path $J_j$ is selected with probability $|J_j|/|V|$ and a tuple $t$ in $J_j$ has probability $1/|J_j|$ of being selected at random. Therefore, we have $P(t) = \frac{|J_j|}{|V|} \cdot \frac{1}{|J_j|} = \frac{1}{|V|}$. □

Algorithm 1 always returns independent samples because a returned sample is always uniform regardless of the previous sampling iterations. In order to calculate the probability distribution of join paths, we need to estimate $|J_j|$ for all $j$'s, during the preprocessing phase. There are various ways of upper bounding the size of a join of two relations in the database theory community [29]. Regardless of the choice of algorithms for join estimation and random sampling over join, Algorithm 1 always returns independent samples since returned sample is always uniform regardless of the history of samples.

## 4 SAMPLING SET UNION

Let $U$ be the universe of the set union of tuples of join paths. We assume there is no duplicates in each join path. 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 4.** *Consider $t_1 = (3,6,4) \in J_1$ and $t_2 = (3,6,4) \in J_2$ of Table 1, which have same value for all attributes in the union schema. The value of each tuple $t$, namely $t.val$, can be obtained by concatenating its attribute values using a standard convention. Then,*

| | |
|---|---|
| $J_j$ | chain join path $J_j = R_{j,1} \bowtie_{A_{j,1}} R_{j,2} \bowtie_{A_{j,2}} \cdots \bowtie_{A_{j,n_1-1}} R_{j,n_1}$ |
| $R_{j,i}$ | relation $i$ in join path $J_j$ |
| $A_i$ | set of attributes $R_i$ and $R_{i+1}$ join on in $J_j$ |
| $S$ | set of join paths $S = \{J_1, \ldots, J_n\}$ |
| $V$ | disjoint union of join paths $V = J_1 \sqcup \ldots \sqcup J_n$ |
| $U$ | set union of join paths $U = J_1 \cup \ldots \cup J_n$ |
| $\mathcal{A}_j^k$ | set of tuples of $k$-th overlapped in $J_j$ |
| $O_\Delta$ | set of overlap tuples of joins in set $\Delta \subset S$ |
| $M_A(R_i)$ | maximum degree of values in $R_i$ on attribute $A$ |
| $\mathcal{K}(i)$ | upper bound of number of overlapping tuples after the $i$th join in join paths |
| $\mathcal{K}(v, i)$ | upper bound of number of overlapping tuples with $v$ on $A_{i,1}$ after the $i$th join in join paths |
| $\mathcal{D}_A(R)$ | domain of values of attribute $A$ in relation $R$ |

**Table 2: Notations**

*by the definition of set, we consider $t_1$ and $t_2$ as the same tuple, say $u$, in $J_1 \cup J_2$. We want $P(u)$, the probability of selecting a tuple with value $u$ from $J_1 \cup J_2$, to be $\frac{1}{|U|}$. Tuples $t_1$ and $t_2$ are distributed in different join paths. Hence, when we sample from each join path, we want $P(t = u) = P(t_1) + P(t_2) = \frac{1}{|U|}$.*

Note that to generate a sample of size $N$, we continuously sample until the desired sample size is reached. Therefore, the final sample may include tuples with duplicate value. We guarantee each such tuple is in the sample with $1/|U|$ probability. If the goal is to obtain $N$ unique tuples, we can discard duplicate tuples and resume sampling until $N$ unique tuples are obtained. Next, we propose two ways of obtaining i.i.d samples from set union $U$.

## 4.1 Bernoulli Case

Algorithm 2 shows the pseudocode for sampling from a set union of joins using a Bernoulli distribution. At every round, the algorithm iterates through all join paths, and selects a path with probability $P(J_j) = |J_j|/|U|$. In § 5.1, we described ways of estimating a join path size, $|J_j|$. In § **??**, we will introduce a suite of techniques for estimating the size of the set union of joins, $|U|$.

Upon selecting $J_j$, we randomly sample a tuple $t$ from $J_j$ with replacement. Recall $u = t.val$ denotes the value of tuple $t$. Let $J(u)$ be the join path from which $u$ is sampled for the first time. Suppose it is the first time some tuple with value $u$ is obtained, then we assign $J(u) = J_j$ and accept the tuple; if $u$ has been sampled before, we only accept the tuple if $J(u) = J_j$, and reject it otherwise. We now prove that this algorithm returns every tuple value $u$ in $U$ with probability $\frac{1}{|U|}$.

**Theorem 2.** *Given join paths $S = \{J_1, \ldots, J_n\}$, Algorithm 2 returns each result tuple $t$ with value $u$ with probability $\frac{1}{|J_1 \cup \ldots \cup J_n|}$.*

Proof. Let $J(u)$ be the join path from which a tuple with value $u$ is selected for the first time. Let $\Delta(u)$ be the set of all join paths that contain tuples with value $u$. Since a tuple with value $u$ can be obtained from any join paths in $\Delta(u)$, we have the following.

$$P(t = u) = \sum_{J_j \in \Delta(u)} P(J_j) \cdot P(t = u | J_j)$$

The algorithm only accepts $t = u$ if it is sampled from the same join path it is sampled from the first time, i.e. $J(u)$. Therefore, we can rewrite the probability of $P(t = u)$ according to the three cases.

$$P(t = u) = \frac{|J(u)|}{|U|} \cdot \frac{1}{|J(u)|} + \sum_{J_j \in \Delta(u) \setminus J(u)} \frac{|J_j|}{|U|} \cdot \frac{1}{|J_j|} \cdot 0 + \sum_{J_j \in S \setminus \Delta(u)} 0$$
$$= \frac{1}{|U|} + 0 + 0 = \frac{1}{|U|}$$

□

**Algorithm 2** Bernoulli Set Union Sampling

**Input:** Join paths $S = \{J_j, 1 \le j \le n\}$, tuple count $N$
**Output:** Tuples $\{t_i, 1 \le i \le N\}$

1: $\{|J_j|, 1 \le j \le n\}, |U| \leftarrow warmup(S)$ ▷ direct (§ 7) or online (§ 8)
2: $T \leftarrow \{\}$ ▷ target sample
3: **while** $n < N$ **do**
4: $round\_recs \leftarrow \{\}$
5: **for** $J_j \in S$ select $J_j$ with probability $\frac{|J_j|}{|U|}$ **do**
6: **while** $t \in round\_recs$ **do**
7: $t \leftarrow$ a random sample from $J_j$ ▷ § 5.2
8: **if** $t \in S$ and $J(t.val) \neq J_j$ **then** reject $t$
9: **else**
10: accept and $round\_recs \leftarrow round\_recs \cup \{t\}$
11: $J(t.val) = J_j, n \leftarrow n + 1$
12: **if** $n > N$ **then** ▷ control the final steps
13: $n \leftarrow n - |round\_recs|, round\_recs \leftarrow \{\}$
14: $T \leftarrow T \cup round\_recs$
15: **return** $T$

Since in every round of Algorithm 2, we define a Bernoulli distribution on join paths, there exists a possibility that more than one join paths are selected and more than one tuple with value $u$ is sampled from different join paths for the first time. As a result, the sampled value $u$ would have two values for $J(u)$, therefore, we select one of these join paths and discard all sampled tuples with $u$ in this round except the one from the selected join path and re-sample a new tuple from each of other join paths (line 7). For instance, if $t_1$ and $t_2$ are sampled from $J_1$ and $J_2$, respectively, and $t_1.val = t_2.val = u$, we let $J(u) = J_j$, we discard $t_2$ and re-sample a new tuple from $J_2$. We use $round\_recs$ to keep track of this information. Another situation worth discussing is when we get more than $N$ tuples in the sample after the last round. For example, we have already gathered $N - k$ tuples, but in the last iteration, we sample more than $k$ tuples. In this case, we discard all sampled tuples in this round (line 11-12) and continuously run new rounds until we exactly sample $k$ new tuples from the join paths in a round.

## 4.2 Non-Bernoulli Case

An alternative way to the above approach is to define a probability distribution over join paths and select only one join path in each round. This way we can obtain exactly $N$ samples and can avoid discarding samples and repeating the last round. The idea mimics the union operation in practice. The goal is to keep samples from an overlap area only when they are sampled from exactly one predetermined join path and discard otherwise. Consider two joins $J_1$ and $J_2$ with overlapping data region $B$ in Fig. 2a. We select and

keep sample $t_1 \in J_1$, where $t_1 \in B$. Then, if we sample a $t_2 \in B$ from $J_2$ later, we can simply reject $t_2$. This way, we keep the $B$ from $J_1$ as the only source we need. Since we reject tuples in $B$ from $J_2$, we can define $P(J_1) = \frac{|A+B|}{|A+B+C|} = \frac{|J_1|}{|U|}$, and $P(J_2) = \frac{|C|}{|A+B+C|} = \frac{|J_2|-|B|}{|U|}$. Prior to sampling, the algorithm needs to decide which overlapping region is restrictively sampled from which join path. We call this division of join paths a *cover*. For example, in Fig. 2a, the cover indicates that tuples in region $B$ are only accepted if they are sampled from $J_1$.

We can generate a cover over the join paths $\{J_i\}$, namely $\{J'_i\}$, by randomly keeping or removing overlapping parts. Fig. 2b illustrates an example of a cover for three overlapping join paths. Based on this cover, $J_i$ is selected with $P(J_i) = \frac{|P(J'_i)|}{|U|}$. 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, since we do not have overlap information apriori, a non-trivial case is when we first sample $t \in J_i$ while in fact $t \notin J'_i$. However, if we sample $t$ later from another $J_j$ whose $J'_j$ covers the overlapping part we should do a critical operation, *revision*. This means we remove $t \in J_j$ from the sample and re-sample $J_j$, while keeping the $t$ from $J_i$.

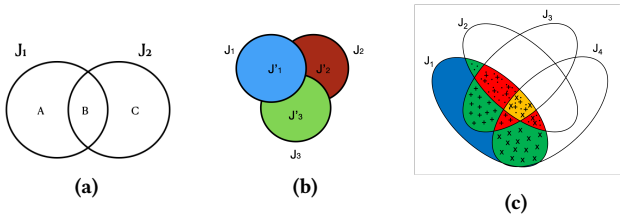



**Figure 2: Union operation on (a) two and (b) three join, (c) $\mathcal{A}_j^k$ of four join**

Note that, in order to define join selection probabilities, both algorithms need to estimate join size, $|J_i|$, and union size, $|U|$. Our framework computes these statistics during a warm-up phase. For join size estimation, we draw on the rich body of work in the literature. While any join size estimation technique can be plugged into our framework, § 5.1 revisits some of the existing techniques. Set union size estimation is a core problem to our paper (§ **??**). To the best of our knowledge, we are the first to consider the problem of estimating the union of join paths. Finally, upon selecting a join path, the algorithm needs to obtain a random sample. Our framework incorporates the state-of-the-art algorithm on random sampling over join [35], which we overview in § 5.2.

Now we prove that Algorithm 3 guarantees uniformity.

**Theorem 3.** *Given join paths $S = \{J_1, \ldots, J_n\}$, Algorithm 3 returns each result tuple $t$ with value $u$ with probability $\frac{1}{|J_1 \cup \ldots \cup J_n|}$.*

Proof. Given any tuple $t$, we will show:

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

where $u$ is any value in the domain of $U$. Based on our design of probability for selecting a join path, there is a map from $\{u\}$ to $\{J_i\}$, named f. We define a map on $\{J_i\}$ to the quotient space of $\{u\}$ over $\alpha$, where $\alpha$ is a probability assignment operation, and $g(J(u)) = [\{u\}]$, s.t. $f(\alpha \cdot u) = J(u)$, where $J(u)$ is the join path value $u$ is assigned. $g$ is an 1-to-1 map, so g sends $J_i$ to $J'_i$, namely. Therefore, we obtain the probability of $P(t = u)$ as following:

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

□

**Algorithm 3** Non-Bernoulli Set Union Sampling

**Input:** Join paths $S = \{J_j, 1 \le j \le n\}$, tuple count $N$
**Output:** Tuples $\{t_i, 1 \le i \le N\}$

```
{|J_j|, 1 ≤ j ≤ n}, |U| ← warmup(S)      ▷ direct (§ 7) or online (§ 8)
{|J'_j|, 1 ≤ j ≤ n} ← order(S)
T ← {}                                    ▷ target sample
record_set_i ← {}
while n < N do
    select J_j with probability |J'_j| / |U|
    t ← a random sample from J_j
    if t ∈ record_set_i for any i < j then reject t
    if t ∈ record_set_i for any i > j then
        remove t from record_set_i
        randomly re-sample t' ≠ t from J_i
        add t to record_set_j
T ← T ∪_i record_set_i
return T
```

## 5 JOIN SIZE AND SAMPLING REVISITED

### 5.1 Join Size: Extending Olken's Algorithm

Our sampling framework allows us to plug in any of join size estimations. For example, Olken proposed a way of estimating the size of join paths of two relations [30]. Here, we adopt part of the algorithm proposed in Ngo et al. [29] and extend Olken's algorithm to calculate the upper bound on the size of join paths of an arbitrary number of relations. Assume a join path $J = R_1 \bowtie_{A_1} R_2 \bowtie_{A_2} \cdots \bowtie_{A_{n-1}} R_n$. 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 path $J$: $|J| \le |R_1| \cdot \prod_{i=1}^{n-1} M_{A_i}(R_{i+1})$. In this paper, we consider the above extension of Olken's algorithm for the subroutine of sampling over join in Algorithm 1 and the following algorithms.

### 5.2 Random Sampling over Join

For sampling a single join, we consider the work by Zhao et al. [35], which is a generic framework considering various ways for instantiating join size [35], including a generalization of the classical Olken's algorithm [31] and Chaudhuri et al.'s algorithm [9] to multiple relations. 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. For weight instantiation, we use all three proposed techniques in [35]: extended olken's, exact, and Wander

Join [23]. This framework requires index structures over relations to know which tuples can be joined together. We use hash tables for relations to maintain tuples joinability information. 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.

## 6 SIZE ESTIMATION OF UNION OF JOINS

Since executing the join paths and unioning the results is expensive, we evaluate the set union size directly by estimating the size of joins and the size of overlap of joins. To do so, we first separate each join path $J_j$ 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 join paths. 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 a non-trivial information, which requires estimating and combining the overlap size of $k$-combinations of joins. There are two challenges for computing $\mathcal{A}_j^k$. The first one is that there is no relationship between the pairwise overlapping information and the $\mathcal{A}_j^k (k > 2)$. The second reason is that estimating the pairwise overlapping size without full join is more challenging than estimating a single join path size.

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

**Example 5.** *Consider the join paths $S = \{J_1, \cdots, J_4\}$ 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 join paths 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 path 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 $O_\Delta$'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.*

Using the intuition of this example, we now introduce a theorem for computing $\mathcal{A}_j^k$.

**Theorem 4.** *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 $1 \le k \le n$, we have*

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

*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 $\Delta$. 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 $k < r \le n$, 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 calculate $|\mathcal{A}_j^k|$'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 § 7.2. 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, § 7.2 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.

As shown in Eq. 1, evaluating the size of set union requires estimating the overlap of all $k$-subsets of join paths, 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 speedup by reusing some of the computation. In the next two sections, we describe ways of estimating the overlap size of a set of joins $\Delta$, namely $|O_\Delta|$, in direct and online manners.

## 7 OVERLAP ESTIMATION OF JOINS

We start with the simple case of estimating the overlap of two chain join paths of the same length (number of relations). Then, we extend it to bounding the overlap of more than two chain join paths. Building on this bound, we relax the assumption on the length and

relation schema of join paths and solve the problem for the generic case of chain join overlap. Then, we focus on the case of having acyclic and cyclic joins.

## 7.1 Overlap of Two Equi-length Chain Joins

Given two join paths $J = R_1 \bowtie_{A_1} R_2 \bowtie_{A_2} \cdots \bowtie_{A_{n_1-1}} R_{n_1}$, with $n_1$ tables, and $J' = R'_1 \bowtie_{A'_1} R'_2 \bowtie_{A'_2} \cdots \bowtie_{A'_{n_2-1}} R'_{n_2}$, with $n_2$ tables, we want to find an upper bound for the size of $|J \cap J'|$. Consider the simplest case where $J$ and $J'$ are both chain joins of the same length, i.e., $n_1 = n_2$, and for each relation $R_i$ in $J$ there is a corresponding unionable relation $R'_i$ in $J'$. This allows us to order relations in join paths. In other words, for now, we consider the special case where join paths are constructed from relations with unionable schemas. We also assume that join attributes on the first two relations of the join paths are unionable. For example, for the join $R_1 \bowtie_{A_1} R_2$ on relations $R_1$ and $R_2$ of $J$ and the join $R'_1 \bowtie_{A'_1} R'_2$ on respectively corresponding unionable relations $R'_1$ and $R'_2$ in $J'$, we assume join attributes $A_1$ and $A'_1$ are unionable. We will relax these assumptions in § 7.3. Trivially, a loose upper bound for the overlap of $J$ and $J'$ is $\min\{|J|, |J'|\}$. Let $J_1 = R_1 \bowtie_{A_1} R_2$ and $J'_1 = R'_1 \bowtie_{A'_1} R'_2$ denote the results of the first joins in $J$ and $J'$. We define $\mathcal{K}(1) = \min\{|J_1|, |J'_1|\}$. Then, we look at $J_2$ and $J'_2$ defined as $J_2 = R_1 \bowtie_{A_1} R_2 \bowtie_{A_2} R_3 = J_1 \bowtie_{A_2} R_3$ and $J'_2 = R'_1 \bowtie_{A'_1} R'_2 \bowtie_{A'_2} R'_3 = J'_1 \bowtie_{A'_2} R'_3$, respectively. Similarly, a loose upper bound for $|J_2 \cap J'_2|$ is $\min\{|J_1 \bowtie_{A_2} R_3|, |J'_1 \bowtie_{A'_2} R'_3|\}$. Let $M_{A_l}(R_i)$ be the maximum degree of values in relation $R_i$ on the join attribute $A_l$. We define $\mathcal{K}(2)$ as follows.

$$\begin{aligned}\mathcal{K}(2) &= \min\{|J_1 \bowtie_{A_2} R_3|, |J'_1 \bowtie_{A'_2} R'_3|\} \\ &\le \min\{\mathcal{K}(1) \cdot M_{A_2}(R_3), \mathcal{K}(1) \cdot M_{A'_2}(R'_3)\} \\ &= \mathcal{K}(1) \cdot \min\{M_{A_2}(R_3), M_{A'_2}(R'_3)\}\end{aligned}$$

For the general case, we have the following.

$$\begin{aligned}\mathcal{K}(i) &= \min\{|J_{i-1} \bowtie_{A_i} R_{i+1}|, |J'_{i-1} \bowtie_{A'_i} R'_{i+1}|\} \\ &\le \min\{\mathcal{K}(i-1) \cdot M_{A_i}(R_{i+1}), \mathcal{K}(i-1) \cdot M_{A'_i}(R'_{i+1})\} \\ &= \mathcal{K}(i-1) \cdot \min\{M_{A_i}(R_{i+1}), M_{A'_i}(R'_{i+1})\}\end{aligned}$$

This gives us $\mathcal{K}(n-1) = \min\{|R_1 \bowtie \cdots \bowtie R_n|, |R'_1 \bowtie \cdots \bowtie R'_n|\}$. Hence, the overlap of two join paths with the same number of relations can be evaluated as follows.

$$|J \cap J'| \le \mathcal{K}(n-1) = \mathcal{K}(n-2) \cdot \min\{M_{A_{n-1}}(R_n), M_{A'_{n-1}}(R'_n)\}$$

This allows us to compute the pairwise join overlap recursively using the maximum degree of join attributes as well as the overlap between the join results of the first two relations in each join path. Next, we introduce two methods of estimating $\mathcal{K}(1)$.

One way is to follow Olken's algorithm [30]. Let $\mathcal{N}(i)$ and $\mathcal{N}'(i)$ be the upper bound of $|J_i|$ and $|J'_i|$, respectively. For each tuple $t \in R_1$, a loose bound for the number of tuples in $R_2$ with which $t$ could join is $M_{A_1}(R_2)$. Since there are $|R_1|$ tuples in $R_1$, we have $\mathcal{N}(1) = |R_1| \cdot M_{A_1}(R_2)$. Similarly, we have $\mathcal{N}'(1) = |R'_1| \cdot M_{A'_1}(R'_2)$. Therefore, we get $\mathcal{K}(1) = \min\{\mathcal{N}(1), \mathcal{N}'(1)\}$.

Note that Olken's method is tuple-based. Now, we propose our value-based approach. Let $\mathcal{D}_A(R)$ be the domain of values of attribute $A$ in relation $R$. Let $d_{A_l}(v, R_i)$ denote the degree of value $v \in \mathcal{D}_{A_l}(R_i)$. Given $R_1 \bowtie_{A_1} R_2$, since we may have many-to-many joins, we have the number of tuples with join attribute value $v$ obtained from the join to be $d_{A_1}(v, R_1) \cdot d_{A_1}(v, R_2)$. Similarly, $d_{A'_1}(v, R'_1) \cdot d_{A'_1}(v, R'_2)$ bounds the number of tuples with join attribute value $v$ obtained from $R'_1 \bowtie_{A'_1} R'_2$. Now we introduce an upper bound $\mathcal{K}(v, 1)$ for the number of overlapping tuples with value $v$ after the first joins $J_1$ and $J_2$. For each Let $\mathcal{D} = \mathcal{D}_{A_1}(R_1) \cap \mathcal{D}_{A_1}(R_2) \cap \mathcal{D}_{A'_1}(R'_1) \cap \mathcal{D}_{A'_1}(R'_2)$. For each $v \in \mathcal{D}$, we have the following.

$$\mathcal{K}(v, 1) = \min\{d_{A_1}(v, R_1) \cdot d_{A_1}(v, R_2), d_{A'_1}(v, R'_2) \cdot d_{A'_1}(v, R'_2)\}$$

An aggregation over all values gives us $\mathcal{K}(1) = \sum_{v \in \mathcal{D}} \mathcal{K}(v, 1)$. Let $\mathcal{D}_{A_l}$ be the domain of values of attribute $A_l$ in the intersection of relations $R_i$ and $R'_i$. Let $M_{\mathcal{D}_{A_l}}(R_i)$ be the maximum degree of values in the intersection of the two relations in two join paths in relation $R_i$ on the join attribute $A_l$. If we want to further improve the accuracy of the bound, we can replace $M_{A_l}(R_i)$ with $M_{\mathcal{D}_{A_l}}(R_i)$ in all the equations above.

## 7.2 Overlap of Multiple Equi-length Chain Joins

Now, we turn our attention to computing the overlap of a set $\Delta$ of multiple equi-length chain join paths. We denote $J_j \in \Delta$ with $R_{j,1} \bowtie_{A_{j,1}} R_{j,2} \bowtie_{A_{j,2}} \cdots \bowtie_{A_{j,n_1-1}} R_{j,n_1}$. We make the same assumptions as pairwise overlaps. That is, relations are unionable and the join attribute of the first two relations in all join paths are unionable. Let $C = \bigcap_{J_j \in \Delta} \mathcal{D}_{A_{j,1}}(R_{j,1}) \cap \mathcal{D}_{A_{j,2}}(R_{j,2})$ be the overlap of the domain of the join attributes in the first join of each path. Following the idea that a loose bound for $O_\Delta$ is $\min_{J_j \in \Delta}\{|J_j|\}$, for each $v \in C$ we define the following.

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

which takes the minimum degree of value $v$ in all join paths in $\Delta$. Therefore, we have $\mathcal{K}(1) = \sum_{v \in C} \mathcal{K}(v, 1)$. Similarly, $\mathcal{K}(i)$ can be extended from the pairwise case.

**Theorem 5.** *Given a collection of join paths $S$ 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 $J_j \in \Delta$. 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 path $J_j$ and let $d_{A_l}(v, R_{j,i})$ be the degree of value $v$ in the domain of $A_l$. We obtain an upper bound for $|O_\Delta|$ dynamically as follows.*

$$\begin{aligned}|O_\Delta| &\le \mathcal{K}(n-1) = \mathcal{K}(n-2) \cdot \min_{J_j \in \Delta}\{M_{A_{n-1}}(R_{j,n})\} \\ \mathcal{K}(1) &= \sum_{v \in C} \mathcal{K}(v, 1) = \sum_{v \in C} \min_{J_j \in \Delta}\{d_{A_1}(v, R_{j,1}) \cdot d_{A_1}(v, R_{j,2})\} \\ \mathcal{K}(i) &= \mathcal{K}(i-1) \cdot \min_{J_j \in \Delta}\{M_{A_i}(R_{j,i+1})\}\end{aligned}$$

The proof for Theorem 5 can be extended from the deduction process discussed earlier.

## 7.3 Chain Joins Overlap: General Case

So far, we proposed a way of estimating the overlap of multiple join paths of the same length where there is a one-to-one mapping between relations in every pair of join paths in terms of schema. We now release this assumption to accommodate join paths with arbitrary size and arbitrary relation schemas. Note that the join paths

$J_1 = (A, B, C) \bowtie (C, D, E) \Rightarrow (A, B) \bowtie' (B, C) \bowtie (C, D) \bowtie' (D, E)$

$J_2 = (A, B) \bowtie (B, C, D) \bowtie (D, E) \Rightarrow (A, B) \bowtie (B, C) \bowtie' (C, D) \bowtie (D, E)$

$J_3 = (B, C) \bowtie (A, B, D, E) \Rightarrow (A, B) \bowtie (B, C) \bowtie' (C, D) \bowtie' (D, E)$

**Table 3: Example of splitting method**

themselves should still have the same schemas after join. We introduce the "splitting method" that derives new join paths by breaking down relations into sub-relations, each sub-relation consisting of exactly two attributes. The derived join paths have the same schema, are lossless, i.e., each generate the same data as the original path, and all contain the same number of relations. Moreover, for each relation in a derived join path, there are corresponding unionable relations in other join paths. Since, the derived join paths satisfy the requirements of § 7.3 and generate the same data, we can directly apply Theorem 5 to estimate the overlap size of the original join paths. Although the input join paths may not include unionable relations, they definitely have unionable attributes and the same schema after join. As such, breaking all relations in sub-relations of two attributes and redefining join paths incurs join with the same number of relations and unionable 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 [12].

For simplicity, we rename unionable attributes of join paths to have the same unique names. For example, in the join paths of Table 3, attributes labeled with $A$ are all unionable and different than attributes labeled with $B$. Then, attributes of each relation are organized based on the lexicographical ordering. Next, a relation $R_i$, with $m$ attributes, in join path $J$, is split into a derived join path of two-attribute relations $R_{i,1} \bowtie' R_{i,2} \cdots \bowtie' R_{i,m-1}$, where $R_{i,k}$ is a relation consisting of the $k$th and $(k+1)$-th attributes of $R_i$. Here, to avoid generating spurious tuples when computing the overlap and join, we redefine the "join" performed on $R_{i,j}$'s. The operation $\bowtie'$, also referred as "fake join", in $R_{i,1} \bowtie' R_{i,2} \cdots \bowtie' R_{i,m-1}$ is a natural join and is defined such that $R_{i,j}$ and $R_{i,j+1}$ have a one-to-one relationship between tuples in join attributes. That is, $|R_{i,1} \bowtie' R_{i,2} \cdots \bowtie' R_{i,m-1}| = |J|$.

**Example 6.** *In Table 3, join paths $J_1, J_2$ and $J_3$ all have the same schema with attributes $A, B, C, D, E$. The split method decomposes each join path into two-attribute-relation chain joins. When splitting a relation with attributes $(A, B, C)$ to relations $(A, B)$ and $(B, C)$, we redefine the "join" performed on $(A, B)$ and $(B, C)$ to be a "fake" join. That is, each tuple $t(A, B, C)$ in this relation is split into $t(A, B)$ and $t(B, C)$ with the same value on $B$, hence, $(A, B)$ and $(B, C)$ always have a one-to-one relationship for the same tuple under the fake join $(A, B) \bowtie' (B, C)$. Splitting keeps the real joins intact. For example, in $J_1$, the join $(B, C) \bowtie (C, D)$ represents a "real" join connecting the first two relations of the path. Note that the splitting method does not change the result of join paths. Suppose $(A, B) \bowtie' (B, C)$ is performed as the first join of $J_1$. Since $(A, B)$ is fake joined with $(B, C)$ and has the same cardinality as $(A, B)$, the resulting relation will be exactly the original relation $(A, B, C)$, that is, $|(A, B)| = |(B, C)| = |(A, B, C)|$. Now, consider $J_2$, first $(B, C)$ is joined with $(A, B)$ then the result $(A, B, C)$ is fake joined with $(C, D)$. In order to reproduce the tuples in the original relation without generating bogus tuples, we follow the one-to-one property that each tuple $t$ in $(A, B, C)$ can only be mapped to one tuple with $t.C$ on attribute $C$ in relation $(C, D)$. Hence, $|(A, B, C) \bowtie' (C, D)| = |(A, B, C)|$. Note that $J_3$ can also be decomposed into the same form as $J_1$ and $J_2$, since $J_3$ is a chain join and the order of tables does not matter.*

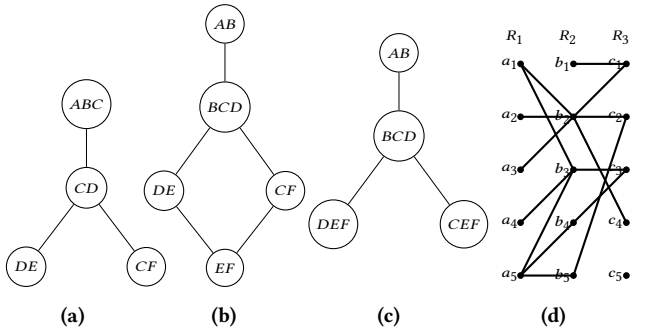



**Figure 3: (a) Acyclic join, (b) Cyclic join, (c) Equivalent acyclic join to (b), (d) Join data graph of $J$**

The split method transforms original join paths into join paths of relations of the same size (two attributes) and schema across all paths. We call these join paths *split join paths*. This allows us to apply Theorem 5 to estimate $O$'s by adjusting the maximum degree of join attribute for fake joins.

**Corollary 1.** *Given a collection of split join paths $S$ 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 path $J_j$ and let $d_{A_l}(v, R_{j,i})$ be the degree of value $v$ in the domain of $A_l$. Define*

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

*We then obtain an upper bound for $|O_\Delta|$ dynamically as follows.*

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

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

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

Note that when all join paths have common subset of attributes with length more than 2, we can keep the subset instead of still separating them into pairs.

## 7.4 Extension to Cyclic and Acyclic Joins

An acyclic join can be represented as a join tree, where each node refers to a relation in the join and is labeled by attributes of the relation. Each edge denotes a join operation between the relations. First, we apply the algorithm proposed by Zhao et al. [35] to break up all the cycles in a cyclic join path to make the join path connected and acyclic. For example, Fig. 3c shows the equivalent join tree to the cyclic join of Fig. 3b.

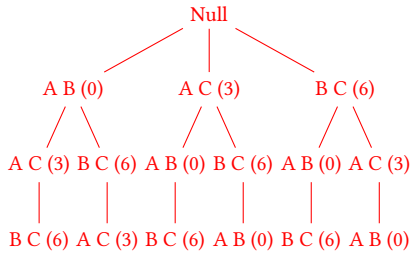
Recall, when all joins are chain joins, to make sure all paths result in the same schema, we order the attributes and relations

lexicographically. Here, we would like all acyclic joins to incur the same schema. This is an important condition as both the union operation and the join paths overlap algorithm require the unification of schemas. Therefore, we construct the equivalent join trees to acyclic joins such that a breadth-first traversal, always starting from the left-most node in each level, gives us join paths of the same schema for all trees.

Corollary 1 presents a way of estimating the overlap of split join paths. To be able to apply this technique on acyclic joins, we need to transform acyclic joins into a sequence of joins on two-attribute relations. In Figure 3a, after the join $(A, B, C) \bowtie (C, D)$, we can continue with either $(D, E)$ or $(C, F)$. Our solution involves first building a standard template of joins. A template is a join query structure to which the structure of every join path can be converted. The reason we need the template is that the degree-based comparison which is necessary for the size estimation could only be done when relations have exactly the sample structure.

Recall that Corollary 1 can be applied only when all the joins have the same length and schema (with fake join included). Therefore, we want to decompose every acyclic join to the same form similar to what we did for chain join. We still formalize the standard as a chain join that contains relations of two attributes, and we choose this standard before decomposing. A good standard is important in the estimation process. A bad standard form can lead us to the worst bound results from this approach is $\min_{j\in[n]} |J_j|$. Take join in Fig. 3a as an example. Suppose we choose the standard as $(A, D) \bowtie (A, C) \bowtie (B, C) \bowtie (B, E) \bowtie (E, F)$. 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 standard $(A, B) \bowtie (B, C) \bowtie (C, D) \bowtie (D, E) \bowtie (E, F)$ gives us a better solution 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 would like the standard to contain relations with attributes that were originally in the same relations in the join path. Therefore, we design our standard choosing approach based on pairwise attribute scores. Choosing the optimal template could be hard, and we define two heuristics. Therefore, we introduce our standard choosing approach based on pairwise attributes score in detail.

*7.4.1 Pairwise attributes score.* Suppose all $J$'s in $S$ result in tables with attributes $\mathcal{D}$. For any pair of attributes $A, A' \in \mathcal{D}$, let $Dist_j(A, A')$ be the distance between node(relation)s of $A$ and $A'$ in join tree for $J_j$. Note that distance between two attributes $A$ and $A'$ actually represents the number of joins we need for estimating to obtain $(A, A')$ in the standard. Then, we define the score between $A$ and $A'$ as $score(A, A') = \sum_{j\in[n]} Dist_j(A, A')$. Again take Figure 3a as an example, $score(A, B) = 0 + 0 + 0 = 0$, which has the highest priority when we select a table for the standard. $score(A, F) = 2+3+2 = 7$ represents that $A$ and $F$ are far from each other and has small possibility to appear together in the original tables. Thus, pairs with lower score have higher possibility of originally being in the same table, and hence We sort all the pairs based on the score from low to high. The lower the score is, the higher the priority. We form all the pairs as a tree, where the root is an empty node and each path from root to leaf is an eligible path after eliminating the empty root node. For example, if the resulting table has schema $\mathcal{D} = \{A, B, C\}$, and $(A, B) = 0, (A, C) = 3, (B, C) = 6$, the tree will be formed as shown in Fig. 7.4.1.




We want the standard path to have the lowest score, so we can convert the problem to finding the minimum cost path, and it can be done recursively.

*7.4.2 Alternating score.* Another thing worth noticing is that split relations and joins without estimating sub-join size preserve most information, so we may give weights to the case with $Dist_j(A, A') = 0$. We can view the score for this case as a hyper-parameter that can be tuned for finding the tightest bound.

Given a standard template, we now introduce how acyclic and cyclic joins can be converted while preserving information for "fake join"s. Consider the tree structure acyclic join. Suppose node for $R_i$ has $k$ number of children, $R_{i_1}, R_{i_2}, \ldots R_{i_k}$, and we have an extreme case of the template where each table $R_{i_j}$ has one attribute that's paired with an attribute in $R_i$. In this case, we do "fake join" on each $R'_{i,j} = R_i \bowtie' R_{i_j} \bowtie' Childs(R_{i_j})$ and estimate $|R'_{i,j}|$ using the method in § 5.1. We let In this step, we also record the estimated maximum degree in each attribute $A$ in $R'_{i_j}$ as follows:

$$M_A(R'_{i_j}) = \begin{cases} M_A(R_i) \cdot M_A(R_{i_j}) & \text{if } A \text{ is join attribute} \\ \max\{M_A(R_i), M_A(R_{i_j})\} & \text{otherwise} \end{cases}$$

Through this way, we can split $R'_{i_j}$ according to the standard template and with information on both cardinality and maximum degrees, and we are able to estimate the overlap size accordingly. Note that we don't necessarily need to "fake join" all the child nodes with their parent for transformation, as in real scenarios, we select the children based on the schemes of relations in the standard template.

Now consider cyclic joins. Following the method proposed in Zhao et al. [35], we remove a subset of the relations in the query, break the cycle and make the join connected and acyclic. We denote the set of removed relations as $\mathcal{S}_R$, and the set of relations in the main acyclic join as $\mathcal{S}_M$. Let attributes in $\mathcal{S}_R$ be $Attr(\mathcal{S}_M)$, and attributes in $\mathcal{S}_R$ as $Attr(\mathcal{S}_R)$. Now we treat $\mathcal{S}_R$ as a single relation in the new acyclic join with maximum degree $M(\mathcal{S}_R)$ for any attribute in $\mathcal{S}_R$ as

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

As mentioned in Zhao et al. [35], the choice of set or relations to remove can have a significant influence on performance, and we follow their methods to find where to break the cycle carefully in practice.

# 8 ONLINE JOIN AND OVERLAP ESTIMATION

The techniques proposed in § 6 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.

## 8.1 Online 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 [23]. 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 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 $p(t) = \frac{1}{|R_1|} \cdot \frac{1}{|d_2(t_1)|} \cdot \cdots \cdot \frac{1}{|d_m(t_{m-1})|}$, where $d_i(t_{i-1})$ is the number of tuples in $R_i$ than join with $t_{i-1}$.

**Example 7.** *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 [15], the estimated join size of $J$ based on sample $S$, namely $|J|_S$ can be evaluated as $|J|_S = \sum_{t \in S} \frac{1}{p(t_k)} \cdot \frac{1}{m}$ [23]. We can update this estimation 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 online overlap. Hence, a real-time approximate answer is returned with some confidence level, and the accuracy improves as sample size grows larger. Extending from wander join, we have two methods to estimate the overlap sizes.

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

## 8.2 Overlap of Joins

We described an online algorithm 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 $\Delta$, namely $O_\Delta$. 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 occurrence 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 $i \neq j$ 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 4 to compute the union size of joins in $\Delta$. Next, we compute the confidence interval for $|O_\Delta|$. The variance of $|\bigcap_{J_i \in \Delta} S'_i|/|S'_j|$, denoted by $\sigma_j^2$, can be computed by a binomial sampling, with a variance of $\hat{p_j}(1-\hat{p_j})$ and mean of $\hat{p_j}$ [22]. Li et al. showed the mean and variance of $|J_j|$, denoted by $\phi_j^2$, 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 [23]. Assuming these terms are independent, we have the variance of $|O_\Delta|$ as follows.

$$\sigma^2_{|O_\Delta|} = T^j_{n,2}(u) \cdot \hat{p_j} \cdot (1-\hat{p_j}) + T^j_{n,2}(u) \cdot \hat{p_j} + T^j_n(u) \cdot \hat{p_j} \cdot (1-\hat{p_j})$$

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

$$E = z \cdot \sqrt{\frac{1}{n}\sum_{J_j \in \Delta} (T^j_{n,2}(u) \cdot \hat{p_j} \cdot (1-\hat{p_j}) + T^j_{n,2}(u) \cdot \hat{p_j} + (T^j_n(u) \cdot \hat{p_j} \cdot (1-\hat{p_j}))} \tag{3}$$

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 online estimator for overlap is unbiased. We first guarantee uniformity by adding $\frac{1}{p(t)}$ number of tuple $t$ to the collection $S_j$. Then since

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

we 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 online estimator for values used in our algorithms.

## 8.3 Reuse of Samples

Although the online estimation technique gives us a more accurate estimation of join size and union size, it requires sampling during the warm-up phase. Note that the sampled tuples join and union estimation are not uniform, however, we can reuse them in the main sampling over the union phase with one extra step. Algorithm 4 illustrates the step of union sampling by reusing warm-up samples. Suppose we store for each join path every tuple $t$ and its sampling probability $p(t)$ collected during the warm-up phase. During the main sampling phase, after selecting a join path, using any of the Algorithm 1 and 2, we use the stored tuples instead of sampling. However, we need to add a rejection rate. Suppose we sampled $S = \{t_1, t_2, \ldots, t_l\}, t_i \in J_j$, for estimating $J_j$. 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 $t_1, t_2, \ldots, t_l$ with probability $\frac{1}{l}$, but with probability $1-\frac{l}{p(t)\cdot|J_j|}$ we reject it. In this way, we obtain $t$ with probability $p(t) \cdot \frac{1}{l} \cdot \frac{l}{p(t)\cdot|J_j|} = \frac{1}{|J_j|}$ and ensure uniformity. The algorithm is described in Algorithm 4

**Algorithm 4** Set Union Sampling with Sample Reuse

**Input:** Join paths $S = \{J_j, 1 \le j \le n\}$, tuple count $N$
**Output:** Tuples $\{t_i, 1 \le i \le l\}$

```
{|J_j|, S_j = {(t_l, p(t_l)) | t_l ∈ J_j}, 1 ≤ j ≤ n}, |U| ← warmup(S)  ▷ online sampling, join size, and union estimation (§ 8)
T ← {}                                                                  ▷ target sample
while n < N do
    round_recs ← {}
    for J_j ∈ S select J_j with probability |J_j|/|U| do
        while t ∈ round_recs do
            if S ≠ ∅ then
                randomly sample t_i ∈ S, accept with l/(p(t)·|J_j|)
                remove t_i from S
        if t ∈ S and J(t.val) ≠ J_j then reject t
        else
            accept and round_recs ← round_recs ∪ {t}
            J(t.val) = J_j, n ← n + 1
    if n > N then                                                       ▷ control the final steps
        n ← n − |round_recs|, round_recs ← {}
    T ← T ∪ round_recs
return T
```

Another way is to work with unique pre-sampled tuples which is consistent with our assumption of join paths containing no duplicate. Suppose we obtained a set of samples $Q = \{t_1, t_2, \ldots, t_l\}, t_i \in J_j$ during warm-up. Upon selecting $J_j$, we randomly choose a tuple from $Q$ and reject with probability $1 - |Q|/|J_j|$. This guarantees i.i.d because the probability of a tuple would be $1/l \cdot |Q|/|J_j|$. Note that for both approaches, if we accept $t$, we do 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 § 5.2.

Note that the acceptance rate can be greater or equal to 1. Here we formalize it to a mathematical expression. Let $R$ be the acceptance rate, and we define the $r_i$ as the prob that $i$ such sample be accepted in a certain round. Then we have:

$$\sum_i^n r_i \cdot i = R$$

$$\sum_i^n r_i = 1$$

$$0 \le r_i \le 1$$

where $n \in N^+$. When $n > R$, this system has infinitely many solutions. Any valid solution could be taken.

## 8.4 Discussion: Direct vs. Online Estimation

We proposed a way of offsetting the cost of online estimation by reusing samples taken during the warm-up phase. The online estimation algorithm improves the efficiency of sampling over the union of joins, by finding a tighter bound on join size and union size and having a lower rejection rate. However, it heavily relies on accessing the whole data and building a join graph which requires having index structures built in advance. Therefore, online estimation is more applicable to settings of datasets residing in disk-based and in-memory database management systems, while direct estimation can be applied to data in the wild or scenarios, such as data markets, when access to the whole data is infeasible.

## 8.5 Direct Initialization with Online Backtracking

**Algorithm 5** Backtracking

```
procedure BACKTRACK(T, P)
    {|J_j|', 1 ≤ j ≤ n}, |U|' ← ONLINEUPDATE(P)
    for each tuple t in T do
        remove t from T with probability 1 − (|J(t.val)|'/|U|') / (|J(t.val)|/|U|)
    return {|J_j|', 1 ≤ j ≤ m}, |U|', T
end procedure
procedure ONLINEUPDATE(P)
    for each J_j do
        |J_j| ← |J_j| = Σ_{p∈P[j]} p · 1/m
    |U| ← CALCU(P)                                                      ▷ Follow § 6 and § 8
    return {|J_j|, 1 ≤ j ≤ m}, |U|
end procedure
```

Since upper bounds calculated using direct method in § 7 for overlap and join estimations may not be unbiased estimators, and online estimation heavily relies on the warm-up phase, we now combine our two methods and propose an algorithm with direct initialization with online backtracking.

Without loss of generality, we introduce the trick on Algorithm 2 and modify it as shown in Algorithm 6. Aside from $T$, set for target samples, we now initialize a list $P$ that stores the probabilities of tuples obtained from the join path either it's accepted, rejected, or when the random walk fails. We also specify a parameter $\phi$, which indicates when we should do backtracking. During the sampling process, we record $t$ only when $t$ is a successfully returned and accepted tuple but record $p$ even when $t$ is rejected or the random walk fails (in this case $p = 0$). Each time when we collect $\phi$ number

**Algorithm 6** Set Union Sampling with Backtracking

**Input:** Join paths $S = \{J_j, 1 \le j \le m\}$, tuple count $N$, backtrack para $\phi$
**Output:** Tuples $\{t_i, 1 \le i \le N\}$

```
1: {|J_j|, 1 ≤ j ≤ n}, |U| ← warmup(S)                    ▷ direct (§ 7)
2: T ← {}                                                  ▷ target sample
3: P ← [m][] ▷ record probability of selected tuples from each join path
4: while n < N do
5:     round_recs ← {}
6:     for J_j ∈ S select J_j with probability |J_j|/|U| do
7:         while t ∈ round_recs do
8:             t, p ← a random sample from J_j with probability p
9:             add p to P[j]
10:        if t ∈ S and J(t.val) ≠ J_j then reject t
11:        else
12:            accept and round_recs ← round_recs ∪ {t}
13:            J(t.val) = J_j, n ← n + 1
14:    if n > N then                                       ▷ control the final steps
15:        n ← n − |round_recs|, round_recs ← {}
16:    T ← T ∪ round_recs
17:    if Σ_{j∈[m]} |P[j]| % φ == 0 then
18:        {|J_j|, 1 ≤ j ≤ m}, |U|, T ← Backtrack (T)
19: return T
```

of $p$'s, we do a backtracking following Algorithm 5. The general idea is that we use the collected probabilities, which are similar to what we obtain in wander join, to estimate the join sizes and the overlap sizes following methods in § 8, and then calculate union size following § 6. After all the values and hence probabilities are updated, we iterate over all the sampled tuples in the target set and adjust their probabilities by removing tuple $t$ with probability with $\frac{|J(t.val)'|/|U|'}{|J(t.val)|/|U|}$ where $|J(t.val)|$ and $|U|$ are original values, and $|J(t.val)|'$ and $|U|$ are updated values. Hence we guarantee that each tuple in the target set is sampled with $\frac{1}{|U|'}$. We can also keep track of the confidence level of the estimated sizes and stop backtracking when the accuracy is beyond the predefined threshold.

# 9 COMPLEXITY ANALYSIS

In this section, we discuss the time complexity analysis of Algorithm 1, 2, and 3. Let the time of sampling from a single join path be $t$ following the method proposed by Zhao et al. [35], the target number of samples be $N$, and the number of join paths be $m$. Note that in set union sampling, some iteration rejects the tuple if it's not from the assigned join path. We define the delay between two successful answers as $D$ for now and will provide a full analysis of it in our future work.

*Disjoint Union of Joins Sampling.* Since every tuple successfully returned by sampling over a single join Zhao et al. [35] is accepted, the runtime of Algorithm 1 is simply $O(N \cdot t)$

*Bernoulli Set Union Sampling.* In each round, we first iterate over all the join paths, and in the worst case, we need to select a tuple from each of them. Since we need to keep track of the *round_recs*, whose size is bounded by $m$, and iterate over it to ensure uniformity, we have that each round has runtime $O(m^2 \cdot t)$. We let the upper bound of the number of iterations be $N + c$, where $c$ is a constant obtained from the analysis on the last few rounds, which we will also conduct in the future. In addition to those, other operations can all be done in constant time. Together, we have the runtime for Algorithm 2 be $O((N + c) \cdot m^2 \cdot t \cdot D)$

*Non-Bernoulli Set Union Sampling.* The Non-Bernoulli case is simpler. Within $N$ rounds, we sample a tuple with time $t$ from the selected join path. Aside from it, other operations can all be done in constant time. Hence considering the delay between accepted tuples, the runtime for Algorithm 3 is $O(N \cdot t \cdot D)$

# 10 EVALUATION

## 10.1 Experimental Setup

**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 5 chain joins; UQ2 consists of 3 chain joins; and, UQ3 has 1 acyclic join and 2 chain joins. For both UQ1 and UQ2, we use 5 tables: `nation`, `supplier`, `customer`, `orders`, and `lineitem`. We use UQ2 only for the evaluation of our online techniques of § 8. For UQ3, we derive queries of different types and length from 3 tables: `supplier`, `customer`, and `orders`. We split them vertically and horizontally, so tables in each join path have different schemas. Therefore, working with UQ3 involves the application of splitting method of § 7.4. 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 $K\% \cdot N \cdot 3$. To study the impact of overlap in our sampling framework, we vary the overlap scale $P$% between joins. 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 queries that involve cyclic joins since they can be easily converted to acyclic joins according to [35].

**Algorithms** We proposed a direct and online way of parameters estimation during the warm-up phase. For union size approximation, we plug in techniques for DirectOverlap of § 7 and OnlineOverlap of § 8.2, in Theorem 4. The join estimation of DirectOverlap can be instantiated by baselines EW (Exact Weight) [35], 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) [35], which we described in § 5.1. The join estimation of the online technique uses our OnlineJoin of § 8.1. 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 size more than 5GB.

For sampling over union phase, we consider DisjoinUnion and SetUnion, which are the implementations of Algorithm 1 and 2, with the Direct instantiation techniques explained above. We do not evaluate the non-Bernoulli set union sampling of § 4.2, since it is a slightly different variation of the Bernoulli by defining a meaningful order on joins. We also evaluate OnlineUnion which

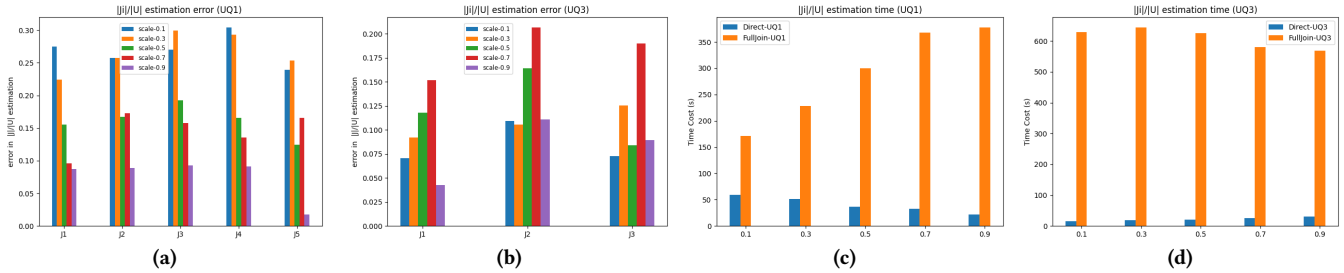



**Figure 4: The error of join to union size ratio estimation using DirectOverlap+EO on (a) UQ1 and (b) UQ3; runtime of union size estimation using DirectOverlap and FullJoin on (c) UQ1 and (d) UQ3**

is the implementation of Algorithm 4 with online instantiation of join and overlap size parameters.

**Implementation:** The framework is implemented in Python. Relations in joins are stored in hash tables with a linear search. Acyclic joins are implemented in 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).

## 10.2 Join and Union Size Approximation

To evaluate the accuracy of our join and union approximation techniques, we consider 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 instantiations of our direct algorithm. First, we observe that 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, when extracting the overlap information, a small size of samples will have a large effect on the estimation performance. However, when we have a large scale of overlap, which is the case of our motivation, 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 operation in the path, our Direct method gains higher accuracy on joins with smaller length. Given that UQ3 is smaller both in length and numbers, this explains why the estimation is relatively more accurate for UQ3.

We also report the runtime of our parameter estimation algorithms, in Fig. 4c and 4d. First, DirectOverlap is significantly faster than the brute-force full join. Second, for UQ1, we observe that when the cost for full join goes higher, the time our method needs is 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, higher overlap scale instead accelerates our method's speed of finding the tuple with max degree at that's in the intersection of two tables' join attribute. Additionally, we employ split method on UQ3. In practice, splitting relations does not affect the accuracy of estimating overlaps. Nevertheless, for acyclic and cyclic joins, integrating relations, viewing them as a whole, and splitting them according to the standard loses some information. In most cases, if we can preserve the main branch of the join and pre-process on smaller branches, and decrease the number of integration, split method will have little impact on estimating the overlap.

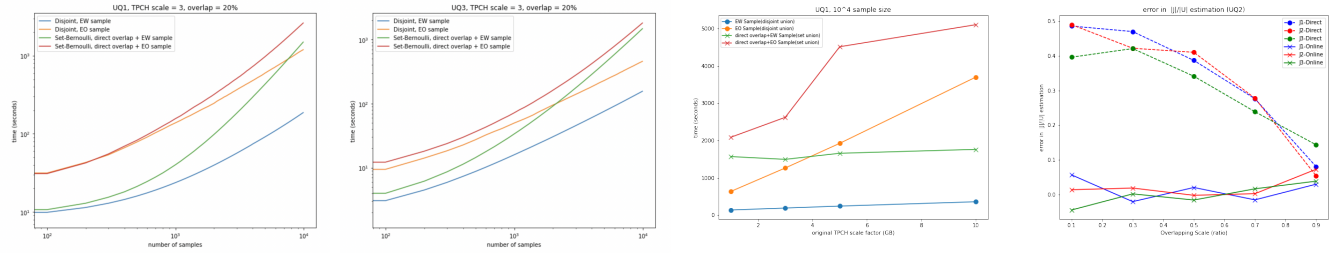
## 10.3 Direct Disjoint and Set Union Sampling

In the first set of experiments, we evaluate our algorithms with direct parameter estimation: DisjoinUnion and SetUnion and their corresponding instantiations. Fig. 5a and 5b show how DisjoinUnion and SetUnion scale in terms of sampling time with respect to samples size. First, DisjoinUnion performs better with EW than with EO, since with exact weights calculated, we obtain a rejection rate of zero. For similar reasons, SetUnion with EW outperforms EO. Second, overall, SetUnion instantiations are slower than DisjoinUnion counterparts, for both datasets, due to the rejection ratio to guarantee uniformity. Finally, although the sampling time for DisjoinUnion stays relatively stable, we observe that SetUnion sampling time increases with the sample size. This can be explained by the rejection setting in Bernoulli algorithm 2. As more and more tuples have been sampled and are "assigned" to join paths they belong with, we get a higher chance to reject tuples that already been seen in other join paths.

Next, we evaluate the impact of overlap ratio among joins on DisjoinUnion and SetUnion. Fig. 5c reports sampling time for various data scales. 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, scale has a much larger effect on EO than EW. As the size of each table grows larger, a tuple has higher rejection rate due to the growth of number of tuples in the relation to be joined with. Finally, from both plots we can conclude that SetUnion is slightly slower than DisjoinUnion, given the fact that we adjust the probabilities by rejecting some of the tuples in SetUnion.

## 10.4 Online Disjoint and Union Sampling

Unlike the direct technique, our online technique collects sampling statistics on the fly during the warm-up phase. We compare the performance of DirectOverlap with EO [30, 35], as join size instantiation, with OnlineOverlap used with OnlineJoin, in terms




(a) (b) (c) (d)

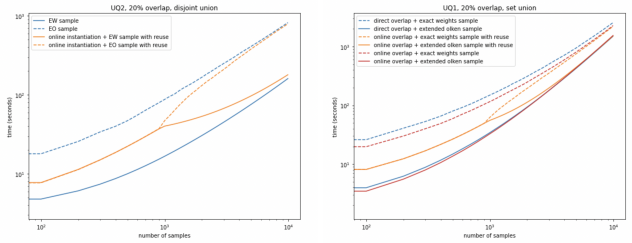
**Figure 5: Runtime vs. sample size for** DisjoinUnion **and** SetUnion **on (a) UQ1 and (b) UQ3; (c) data scale vs.** DisjoinUnion **and** SetUnion**, using** DirectOverlap**; (d) the error of join to union size ratio using** Online Estimation **on UQ2**

of the error of join to union size ratio estimation on UQ2. We used data scale of 3GB for each query. The results are reported for various overlap scales in Fig. 5d. First, OnlineOverlap outperforms DirectOverlap; in fact, OnlineOverlap is extremely accurate and has an error close to zero for all joins. This is because the nature of indexing will give us extra information of 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 DirectOverlap 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, and hence obtain a higher accuracy more efficiently. Nevertheless, though OnlineOverlap has better performance, DirectOverlap is much faster, and can be applied on database without index structure.

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., $\frac{|\bigcap_{i\in\Delta} S'_i|}{|S'_j|}$. 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 [23]. We terminate online sampling when the confidence level reaches 90% or we obtain 1,000 samples.

In the next experiment, we evaluate the runtime of the OnlineUnion sampling using the idea of reusing samples collected during warm-up. We compare OnlineOverlap with reuse with DirectOverlap on DisjoinUion. We also compare OnlineOverlap with or without reuse with DirectOverlap on SetUnion. Results are reported in Fig. 6. Since OnlineOverlap is more accurate than DirectOverlap, we observe that both SetUnion and DisjointUnion outperform the EO instantiation. But, they are not quite as fast as EW which has the reject rate of 0. As shown in Fig. 6b, the OnlineReuse is slightly slower than its counterparts that do not re use samples, due to the additional rejection ratio. However, the algorithm catches up when all warm-up samples are reused.

As we can see from Figure. 6, in terms of the effect of applying Algorithm 4, there is a bifurcation point for Online methods. Before the bifurcation point, "reuse" methods are sampling from the




(a) (b)

**Figure 6: (a) time vs. sample size for** DisjoinUnion **on** UQ2 **using** DirectOverlap **with reuse (b) time vs. sample size for** SetUnion **on** UQ2 **using** DirectOverlap **with and without reuse**

same pool (presampled data), so EO and EW are similar in performance. After the the bifurcation point, presampled samples are all used, and two methods will converge to their original performance respectively. Moreover, for EO, reuse will lead to a lower rejection rate, improving the efficiency. However, for EW, since it doesn't have rejection rate, reuse doesn't make any difference.

## 11 RELATED WORK

**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 [16]. To solve this, Olken proposed the idea of rejecting join of two samples with specific probabilities for two-table join [30]; Chaudhuri et al. proposed techniques that are applicable to linear joins but not to arbitrary joins [9]. 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 [9]. However, all the above three methods only work for 2-table Joins. Ripple join returns dependent and uniform samples [14]. Wander join [23] 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 [35]. This

algorithm can be plugged in our framework for random sampling over a single join path.

**Join Size Estimation** results on join size upper bounds [6, 18, 22] There has always been an interest in the database theory community in upper bounding a join size, due to the connection of this problem to query optimization. We described Chaudhuri et al.'s algorithm [9] and Olken's algorithm [31], which we extended for multi-relation joins, as well as Zhao et al.'s [35] algorithm in § 5.1. Another well-known algorithm is the AGM bound [4]. On a general query, it requires solving a linear program. AGM returns the optimal bound on the join size when the only information is the size of relations.

**Union of Sets** The union-of-sets problem has been studied in approximate counting literature [19]. 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. [19]. 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 [18], including designing hashing-based algorithms [8, 13, 19, 26]. Another popular line of work is on estimating the union of sets where each set arrives in a streaming fashion [5, 17, 25, 27].

## 12 CONCLUSION

This paper studies two novel problems: sampling over union of joins and size approximation of union of joins. A general union sampling framework is proposed with online and direct techniques that produce an upper bound on the union size of joins of arbitrary multi-way acyclic and cyclic. We explore different direct and online variations of the proposed framework. Extensive experiments have demonstrated the efficiency and effectiveness of the proposed techniques. Interesting future work directions include considering selection predicates and integrating the proposed work 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] 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, Vol. 2483)*, José D. P. Rolim and Salil P. Vadhan (Eds.). Springer, 1–10.

[6] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *ICDE*. 709–720.

[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, Vol. 5369)*, Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga (Eds.). Springer, 436–447.

[8] 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.

[9] 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

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

[11] 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.

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

[13] 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.

[14] 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

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

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

[23] 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

[24] 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.

[25] 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.

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

[27] 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.
[28] 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.
[29] 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.
[30] Frank Olken. 1993. *Random Sampling from Databases.* Ph. D. Dissertation. University of California at Berkeley.
[31] Frank Olken and Doron Rotem. 1995. Random sampling from databases: a survey. *Statistics and Computing* 5, 1 (1995), 25–42.
[32] 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.
[33] 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.
[34] Swarup Acharya Phillip B Gibbons Viswanath and Poosala Sridhar Ramaswamy. 1998. Join Synopses for Approximate Query Answering. (1998).
[35] 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