Streaming Weighted Sampling over Join Queries

Michael Shekelyan*
shekelyan@compsci.science
King's College London

Graham Cormode
g.cormode@warwick.ac.uk
University of Warwick

Peter Triantafillou
p.triantafillou@warwick.ac.uk
University of Warwick

Qingzhi Ma
quincyma@tencent.com
University of Warwick

Ali Mohammadi Shanghooshabad
shanghooshabad@gmail.com
University of Warwick

ABSTRACT

Join queries are a fundamental database tool, capturing a range of tasks that involve linking heterogeneous data sources. However, with massive table sizes, it is often impractical to keep these in memory, and we can only take one or few streaming passes over them. Moreover, building out the full join result (e.g., linking heterogeneous data sources along quasi-identifiers) can lead to a combinatorial explosion of results due to many-to-many links. Random sampling is a natural tool to boil this oversized result down to a representative subset with well-understood statistical properties, but turns out to be a challenging task due to the combinatorial nature of the sampling domain. Existing techniques in the literature focus solely on the setting with tabular data residing in main memory, and do not address aspects such as stream operation, weighted sampling and more general join operators that are urgently needed in a modern data processing context. The main contribution of this work is to meet these needs with more lightweight practical approaches. First, a bijection between the sampling problem and a graph problem is introduced to support weighted sampling and common join operators. Second, the sampling techniques are refined to minimise the number of streaming passes. Third, techniques are presented to deal with very large tables under limited memory. Finally, the proposed techniques are compared to existing approaches that rely on database indices and the results indicate substantial memory savings, reduced runtimes for ad-hoc queries and competitive amortised runtimes. All pertinent code and data can be found at: https://github.com/shekelyan/weightedjoinsampling

1 INTRODUCTION

Joins are best known for their central role in relational databases (to splice tables back together after decomposing them to reduce data redundancy), but joins in the more general sense link sets of entities [14] and are ubiquitous across many fundamental analytics tasks such as entity matching [5, 74], record linkage [13, 15, 35] and similarity joins [78, 79]. As modern applications link together more and more data sources comprised of more and more entities, computing the join becomes very costly. The cost becomes prohibitive in the presence of many-to-many links that inflate the join cardinality to astronomical sizes (cf. Table 2). Many-to-many links occur routinely in record linkage with non-unique quasi-identifiers.

Handling such massive joins is therefore a challenging problem for modern data management systems. As data sizes only increase, prior work makes a compelling case for working with *approximate representations of data*, chief among which is sampling. That is, we seek some representative sample of the result of a join query which can be used as a surrogate for the full result in downstream processing and analytics. For example, approximate query processing engines, like QuickR [33] and VerdictDB [54] depend on such samplers to handle joins. Similarly, ML-based Approximate Query Processing (AQP) engines like DBEst [46] and DBEst++ [45] as well as Learned Selectivity Estimation methods, such as NeuroCard [75] explicitly depend on such sampling algorithms to handle joins. The questions we study in this paper are whether sampling can avoid the high costs of computing join queries, and how sampling can be achieved with data-dependant sampling probabilities (weights). Although efficiently collecting random samples for a single base table is trivial, collecting a weighted random sample over a join of tables remains a formidable challenge.

Random sampling is a common answer in knowledge discovery and mining tasks [16, 32, 44, 68, 72] to reduce these astronomical sizes to something manageable without having to sacrifice inference capabilities over the data population. However, it is not possible to obtain a random sample over joins by leveraging standard sampling methods. Applying sampling to the result of the full join is impractical, due to the high cost of performing the join, while trying to apply the join to samples of the data sources does not yield a meaningful distribution over join tuples [10]. In many cases, it is desirable to draw a sample according to weights defined by the data. Sampling with probabilities proportional to weights is inherent in a number of applications that cannot be solved with uniform sampling, such as

- Stratified Sampling / Join over Selections. Join rows violating column-wise selection predicates can be assigned a weight of zero via column-based weights. Hence, weighted sampling can be used to collect stratified samples, e.g., as needed by AQP systems [9, 33, 54].
- Probability proportional to size (PPS) sampling. A common sampling design that samples records that relate to larger groups proportionately more often [53, 58, 64].
- Data exploration. Weighted sampling enable smoother types of selection (as a function of data values) without a hard cut-off,which allows to prioritise more relevant entries, e.g., featuring larger sales volumes, more recent sales or sales closer to a location. Similarly, one can sample outliers by weighting expected entries lower. Thus,

*While at University of Warwick: development/analysis/evaluation of proposed approaches, writing/editing the manuscript and writing/running all the implemented code. While at King's College London: revising the manuscript.

weighted sampling is utilised for scientific discovery in observational data [63].

- Aggregations. Weighted sampling can be more accurate for aggregations on attribute values. For instance, a random sample biased towards larger values may be more accurate for aggregate functions such as SUM. Unequal sampling probabilities are widely used for statistical estimators [7, 61].
- Privacy-Protection. In privacy-preservation frameworks for query-answering such as differential privacy [19] with numerous database applications [18, 39, 48, 51], differentially private selection over joins requires weighted sampling over joins (cf. Appendix C for details).

Weighted sampling is a common requirement across many applications. For instance, optimal sub-sampling [67, 69–71, 77] for regression tasks, weighted bootstrap [37, 47], proportional-to-size sampling designs [53, 58, 64], statistical estimators [7, 61] and data analytics [63, 67, 69–71, 77] require support for weighted random sampling where probabilities are proportional to weights. Supporting weighted sampling also immediately enables support for stratified sampling and filtering entities based on selection predicates.

Streaming access to data is a desirable way to handle the large volumes of data that can arise in data processing. Ideally, we would process each stream of data once only with a single linear pass. This applies particularly to data consisting of very large data items, e.g., multimedia content, high-dimensional vectors, tabular data or documents. As we show below, guaranteeing one pass for every stream is not possible, and so we seek methods which take as few passes over the data as possible, so the data must be stored. Note that streaming passes over data can be much more efficient than allowing random access via indices. In our empirical study we compare against methods that do not operate in the streaming setting, and show that they incur higher computational costs.

As a specific motivating example, consider a common scenario in commerce where we process a very large number of transactions within orders that are linked to items with various properties (price, weight, tax band etc.) and their corresponding suppliers, and separately to customers with location and demographic information. Several analytic queries can be expressed as joins over these inputs, and a weighted sample is needed. For instance, we might want to understand the patterns of demand for goods between different pairs of countries, where the sampling probability is weighted by the value of the order. In this setting, it is reasonable to assume that we can make multiple accesses to the tables with details of customers and suppliers, but due to its size we should only make streaming access to the transactions as they arrive.

Existing join sampling methods do not address these requirements. Most approaches from the relevant literature employ some sampling heuristic that does not follow any statistically useful sampling design [2, 11, 27, 28, 30, 34, 42, 43] and those that follow a sampling design [10, 80] only support equal probability sampling, presume the data to reside pre-indexed in main memory and only support basic join operators. As the basic ideas rely on long-known structural properties of acyclic joins [76], one can either try to extend these equal-probability approaches, or revisit the question from first principles. Our main contributions are:

- Weighted sampling over linked data via a multipartite graph formulation of joins (Section 3) that provides support for a variety of join operators.
- A one-pass/two-pass sampler via an online multinomial sampler (Section 4)
- A multi-pass sampler enabling sub-linear memory footprints via *internal sampling*, *hashing gambit* and *rejection sampling for many-to-one links* (Section 5)
- A comprehensive experimental comparison with non-stream methods (Section 7), which only support equal probability sampling.

In summary, our streaming-friendly algorithms are lightweight to implement, and do not require indices or other preprocessing. They take one or a few passes over the input data, and produce an unbiased weighted sample. Our experiments show that these are fast in practice, making it practical to draw fixed-sized samples over arbitrarily large joins.

2 FORMAL PROBLEM SETTING

2.1 Weighted Sampling over Linked Data

In what follows, we describe our results in terms of data that is accessed as a stream of tuples, and analyze the number of passes through the streams that are needed. We are particularly interested in what is (or is not) possible to achieve with a constant number of passes through the data.

Definition 2.1 (data stream join). Let A and B be two data streams and $\textsc{link}(\alpha,\beta)$ be a link predicate $\textsc{link} : A \times B \rightarrow \{0,1\}$.

$$A \bowtie B = \{(\alpha,\beta) \mid \alpha \in A,\ \beta \in B,\ \textsc{link}(\alpha,\beta)\}$$
$$A \ltimes B = \{\alpha \mid \alpha \in A,\ \exists\, \beta \in B : \textsc{link}(\alpha,\beta)\}$$
$$A \triangleright B = \{\alpha \mid \alpha \in A,\ \nexists\, \beta \in B : \textsc{link}(\alpha,\beta)\}$$

($\bowtie$, $\ltimes$, $\triangleright$ are inner, semi, anti joins)

$$A ⟕ B = A \bowtie B \cup \{(a,\emptyset_B) \mid \alpha \in A,\ \nexists\, \beta \in B : \textsc{link}(\alpha,\beta)\}$$
$$A ⟗ B = A ⟕ B \cup \{(\emptyset_A,\beta) \mid \beta \in B,\ \nexists\, \alpha \in A : \textsc{link}(\alpha,\beta)\}$$
$$A ⟖ B = A \bowtie B \cup \{(\emptyset_A,\beta) \mid \beta \in B,\ \nexists\, \alpha \in A : \textsc{link}(\alpha,\beta)\}$$

($⟕$, $⟗$, $⟖$ are left, full, right outer joins)

The link functions are presumed to take the form $\textsc{link}(\alpha,\beta) = (\,f(\alpha) = g(\beta)\,)$ with functions $f : A \rightarrow \mathbb{R}$, $g : B \rightarrow \mathbb{R}$. The function values of f and g are for instance often simply join attribute values, but we will refer to them more generally as "link values". The equality operator (=) can also be replaced with some inequality operator ($<,>,\leq,\geq,\neq$). This corresponds to equi-joins and theta joins in relational databases.

A special case of interest is if $|\{\beta \mid \beta \in B, \textsc{link}(\alpha,\beta)|\}| = 1$ for any $\alpha \in A$. It allows to treat the two item sets A and B as a single entity set C that pairs each $a \in A$ with the single link partner in B. Such one-to-one or many-to-one linkages are most common in relational databases, where they occur routinely to splice the normalized tables back together, i.e., joins along foreign keys.

Definition 2.2. A weighted sample over the data stream join $((A \otimes B) \otimes \ldots) \otimes G$ with $\otimes \in \{\bowtie, ⟕, ⟗, ⟖, \ltimes, \triangleright\}$ is a multinomial sample with $\Pr(\alpha,\beta,\ldots,\gamma) \propto w(\alpha) \cdot w(\beta) \cdot \ldots \cdot w(\gamma)$ for any item tuple $(\alpha,\beta,\ldots,\gamma)$ with $\alpha \in A,\ \beta \in B,\ \ldots, \gamma \in G$.

In this work, weights are per-item, probabilities are proportional to weights and with-replacement sampling is employed, i.e., multinomial sampling [6, 17, 22, 29, 36, 60]. As in Poisson sampling, elements are drawn independently from each other, but unlike poisson sampling the sampling size is fixed and repeated samples may occur.

Our results apply to any acyclic join query that can be specified. As a limitation, cyclic joins (allowing links to form cycles) are left as an open problem. All prior works that claim to handle cyclic joins relax the join C to some suitable acyclic join $\mathcal{A}$ [62, 80] akin to Chow-Liu trees [12] which means only a fraction $\frac{|C|}{|\mathcal{A}|}$ comprises valid samples. As $\frac{|C|}{|\mathcal{A}|}$ can be arbitrarily small, it can require arbitarily large sample sizes that may be difficult to support in a streaming setting.

2.2 Lower Bounds for Data Stream Joins

In this section, we study how efficiently we can obtain a join sample. Ideally, we would like to do so with only a single pass over each relation. However, we show that this is not possible: joining a pair of one-pass streams typically requires any algorithm to fully record one of the streams.

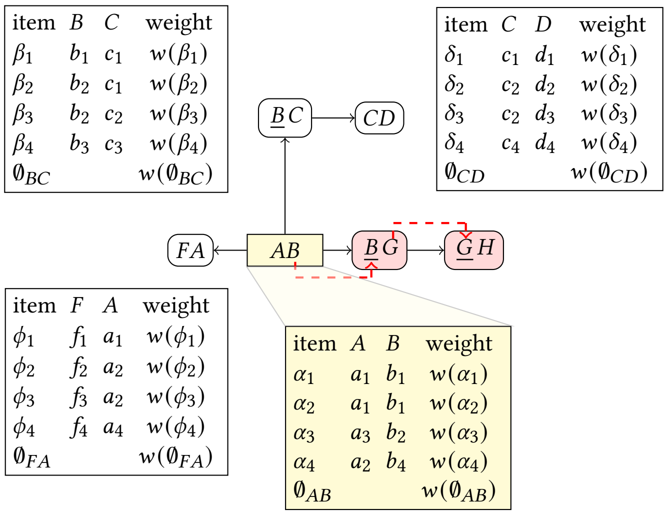
Theorem 2.3 (one-pass memory lower bound). *Let A and B be two one-pass data streams, i.e., both stream by just once. Let* distinct *be the minimal number of distinct link values of A and B,* |sample| *be the sample size. Then collecting a multinomial sample (with non-zero probabilities) of the data stream join $A \bowtie B$ requires storing $\Omega\,(\min\{\,|A| + |B|,\ \textsc{distinct} \cdot |\textsc{sample}|\,\})$ stream items.*

The proof can be found in Appendix B.1 and relies on the well-known communication complexity lower bound for the index problem [40]. The proof follows by expressing the task as a communication between Alice and Bob [59] (representing here the two streams A and B). Instead of the bit string from the index problem, Alice has a weighted random sample of items associated with each link value. Instead of an index, Bob has one item with a specific link value. Then, instead of selecting the bit with Bob's index, the task is to select the weighted random sample for the link value that matches Bob's item. Following this mapping, in place of sending Bob $\Omega(1)$ bits per index, Alice has to send Bob $\Omega(1)$ weighted random samples for each link value (or simply all stream items). The proof follows since if Alice and Bob could solve the sampling problem, they would also solve the index problem, and so is subject to the corresponding lower bound.

When we seek a small sample, and there are only a few link values, Theorem 5.1 shows that we can obtain all relevant stream items for the join in just one pass. But in general the sampling lower bound presents a limit on what is achievable. Nevertheless, with a slight relaxation we can achieve space-efficient algorithms that take few streaming passes. We propose a compromise where we take **two** passes through most streams (or store them in memory), but only a single pass through one of the streams – typically, we will choose this to be the largest data stream. We refer to this as the one*-pass setting, and define it formally as follows:

Definition 2.4 (one*-pass sampler). Let A be a data stream that passes by just once and $B, \ldots, G$ be streams that pass by once before A and once after A. Then a one*-pass sampler collects a multinomial sample of the data stream join $A \bowtie B \bowtie \ldots \bowtie G$.

Note that working in this setting is not a restriction: if we can show algorithms that adhere to the one*-pass model, it only means that they require very few passes over the data. In our experiments, we compare to methods outside of this model, and show that our one*-pass and few-pass algorithms can be much



item	B	C	weight
β_1	b_1	c_1	$w(\beta_1)$
β_2	b_2	c_1	$w(\beta_2)$
β_3	b_2	c_2	$w(\beta_3)$
β_4	b_3	c_3	$w(\beta_4)$
$\emptyset_{BC}$			$w(\emptyset_{BC})$

item	C	D	weight
δ_1	c_1	d_1	$w(\delta_1)$
δ_2	c_2	d_2	$w(\delta_2)$
δ_3	c_2	d_3	$w(\delta_3)$
δ_4	c_4	d_4	$w(\delta_4)$
$\emptyset_{CD}$			$w(\emptyset_{CD})$

item	F	A	weight
ϕ_1	f_1	a_1	$w(\phi_1)$
ϕ_2	f_2	a_2	$w(\phi_2)$
ϕ_3	f_3	a_2	$w(\phi_3)$
ϕ_4	f_4	a_4	$w(\phi_4)$
$\emptyset_{FA}$			$w(\emptyset_{FA})$

item	A	B	weight
α_1	a_1	b_1	$w(\alpha_1)$
α_2	a_1	b_1	$w(\alpha_2)$
α_3	a_3	b_2	$w(\alpha_3)$
α_4	a_2	b_4	$w(\alpha_4)$
$\emptyset_{AB}$			$w(\emptyset_{AB})$

Figure 1: Running example: Join of six data streams $(FA \mathbin{⟗} AB \mathbin{⟕} BC \ltimes CD) \bowtie BG \bowtie GH$ linked along attributes A, B, C, D, F, G, H.

more efficient. We next give a lower bound that places a precise limit on what can be achieved in the one*-pass model.

Theorem 2.5 (one*-pass memory lower bound). *A one*-pass sampler over $A \bowtie B \bowtie \ldots \bowtie G$ requires $\Omega(\textsc{distinct})$ bits of memory where* distinct *is the number of distinct link values appearing in the streams $B, \ldots, G$.*

A full proof can be found in Appendix B.2. The gap between Theorems 2.3 and 2.5 can be understood as follows. If we consider a stream containing N items each representable with $O(b)$ bits, where all link values are distinct, Theorem 2.3 means that pure one-pass streaming setting requires $\Omega(Nb)$ bits of storage. Meanwhile, in the one*-pass setting we will be able to work with only $O(N)$ bits, which can be a big difference in practice.

2.3 Running Example throughout this Work

We will use a running example of a join across six different relations, shown in Figure 1. The streams are named after the item properties/quasi-identifiers, e.g., AB is named after the item properties A and B and BC after properties B and C such that one can join both streams using a link function that requires the values to match for B. In the example, the two relations BG and GH have many-to-one relations and all items are chosen to have weight 1. Such cases can be easier to handle, and so may be omitted in some discussions.

Why join sampling? Think of FA, CD, GH as information sources about entities such as people, accounts, institutions and financial transactions. The sources can only be linked via AB, BC and BG that serve as imperfect "translation tables" that relate the various pseudo-identifiers (A,B,C,G) with each other (e.g., names and key dates/locations relating to those entities). Materialising the full join is not feasible, because the join contains all possible linkage combinations, i.e., the join result becomes astronomically large and impossible to enumerate. An easy to interpret random sample can be obtained without materialising the full join.

Why join sampling over streams? As we are joining three very large information sources (FA, CD, GH) from different parties, we are strongly motivated to minimise the access to the data: it may be unfeasible or even disallowed to create local copies of them. We therefore seek methods that require one or at most a small constant number of streaming accesses to the data sources.

Why weighted join sampling over streams? A uniform sample over the join is often not meaningful, because it weighs entities more heavily just because they are linked to more entities, which does not necessarily correlate with their true importance. Instead, we can specify sampling probabilities based on other factors of the input items, such as their value or confidence (which if based on sensitive user data calls for privacy-preserving methods as in Appendix C).

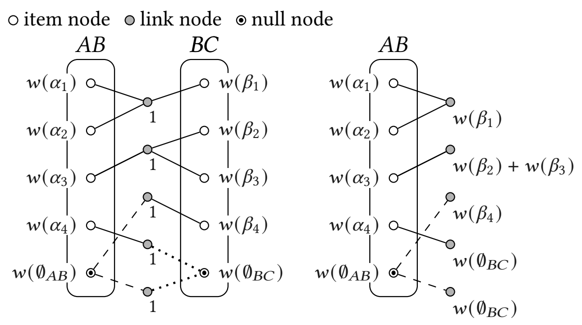
3 MULTIPARTITE GRAPH FORMULATION

Algorithm 1: Group Weights

```
Let $G = (V, E)$ be a multipartite graph corresponding to the join query and $w(\tau)$ be a node $\tau$'s label.
Let ROOT$(V) = \mathcal{A} \subseteq V$ be the item node partition of the join query's main table.
Let PARENT$_{\mathcal{A}}(\mathcal{T}, V')$ be all neighbouring link node partitions in $V'$ closer to $\mathcal{A}$ than $\mathcal{T}$.
Let CHILDREN$_{\mathcal{A}}(\mathcal{T}, V')$ be all neighbouring link node partitions in $V'$ further away from $\mathcal{A}$ than $\mathcal{T}$.
Let LEAFS$_{\mathcal{A}}(V')$ be all item node partitions in $V'$ furthest away from $\mathcal{A}$, excluding $\mathcal{A}$.
Initiate $V'$ as $V$.
while $|LEAFS_{\mathcal{A}}(V')| > 0$ do
    Pick an item node partition $\mathcal{T} \in$ LEAFS$_{\mathcal{A}}(V')$
    Let $\mathcal{J}_0 =$ PARENT$_{\mathcal{A}}(\mathcal{T}, V')$.
    Set the label of all nodes in $\mathcal{J}_0$ to $-\infty$
    foreach item node $\tau \in \mathcal{T}$ do
        Initiate $W$ as 1
        foreach link node part. $\mathcal{J} \in CHILDREN_{\mathcal{A}}(\mathcal{T}, V')$ do
            Let $W_{\mathcal{J}}$ be the label of $j \in \mathcal{J}$ connected to $\tau$
            Multiply $W$ by $W_{\mathcal{J}}$.
        end
        Let $W_{\mathcal{J}_0}$ be the label of $j \in \mathcal{J}_0$ connected to $\tau$
        if $W_{\mathcal{J}_0} = -\infty$ then Set $W_{\mathcal{J}_0}$ to 0
        Add $w(\tau)W$ to $W_{\mathcal{J}_0}$.
    end
    Remove item node partition $\mathcal{T}$ from $V'$
    foreach link node partition $\mathcal{J} \in CHILDREN_{\mathcal{A}}(\mathcal{T}, V')$ do
        Remove link node partition $\mathcal{J}$ from $V'$
    end
    Replace all labels $-\infty$ with $w(\emptyset_{\mathcal{T}})$ for nodes in $\mathcal{J}_0$
end
```

3.1 Bijection between Join Results and Partition-Spanning Trees in Multipartite Graphs.

In order to more easily reason about not just inner joins, but also outer joins, it is useful to formulate the join operation through a graph, where items and links form a multipartite graph with one partition of nodes per stream and one partition of nodes between any linked pair of streams.



(a) Each link node is the root of a sub-tree (left to right). **(b) Weights of each sub-tree are transferred into a link node.**

Figure 2: Graph formulation for join of two streams.

An example of such a multipartite graph for a two stream join can be found in Figure 2. The nodes of the graph are:

- *Item nodes*: Each stream is a set of nodes, one node for each stream item, e.g., there is a node for item $\alpha_1 \in AB$.
- *Null nodes*: null nodes for outer joins are an additional "dummy" stream item, e.g., $\emptyset_{AB}$ for stream AB.
- *Item node labels*: Each item and null node is labelled with a weight, e.g., $w(\alpha_1)$ and $w(\emptyset_{AB})$.
- *Link nodes*: Each link attribute is a set of nodes, one node for each attribute value, e.g., b_1.
- *Link node labels*: Each link node is initially labelled with the value 1, e.g., the node for b_1 has a label 1.

As join operators are not necessarily symmetrical, there is a "left" and a "right" side. The edges of the graph potentially connect link nodes with item nodes that satisfy the link predicate, but it depends on the join operator if they do:

- Each link node for ⟕, ⟖, ⟗ and ⋈ is connected to each matching item node.
- Each link node for ⟖ and ⟗ that is not connected to any item node on the left side is connected to the left null node.
- Each link node for ⟕ and ⟗ that is not connected to any item node on the left side is connected to the right null node.
- Each link node for ⋉ has label 0 if there is *no* match on the right side.
- Each link node for ▷ has label 0 if there *is* a match on the right side.

This simply means left outer joins (⟕) allow null items on the right side and right outer joins (⟖) on the left side, while full outer joins allow them on either side (⟗). Semi-joins (⋉) and anti-joins (▷) are filters that change the weight of a link node if there is a match on the right side. For semi-joins and anti-joins the right side partitions are removed after determining the link node labels.

The main motivation for this formulation is that each tree that spans all partitions corresponds to a linked set of items:

Definition 3.1. A partition-spanning tree of a multipartite graph is a connected subgraph that contains exactly one node in each partition.

The weighted sampling problem can then be posed via the multipartite graph formulation:

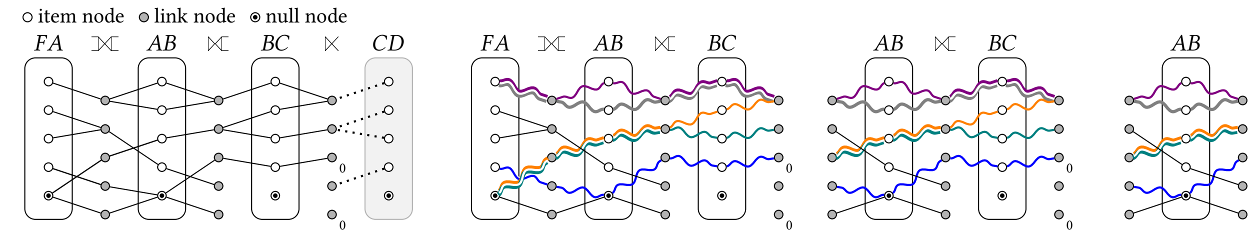

○ item node ● link node ⊙ null node
FA
AB
BC
CD
0
0
(a) Graph Formulation for Running Example
(b) after 1 iteration
(c) after 2 iterations
(d) completed

Figure 3: Algorithm 1's iterations (Lines 7-26) preserve weights of linked item trees, but simplify the graph by removing partitions.

Definition 3.2. Weighted Sampling over Joins independently draws n partition-spanning trees with probabilities proportional to the product of their node weights if they contain at least one item node and probability 0 if they only contain null nodes.

3.2 Sampling partition-spanning trees via Ancestral Sampling.

As there are as many partition-spanning trees as join results, some technique is needed to reduce the complexity. As each partition-spanning tree is sampled with probabilities proportional to their weight, any group of trees will be sampled with probabilities proportional to their sum of weights. Thus, one can sample the partition-spanning tree step-by-step rather than in one go, which makes it possible to achieve a sub-linear complexity in the number of partition-spanning trees. The step-by-step sampling corresponds to ancestral sampling [24, 38] from a Bayesian Network (BN) where each item node corresponds to a BN node and the transition probability of two BN nodes is proportional to the sum of weights of their ancestors (here extensions of the partition-spanning tree to additional partitions). For any acyclic join, one can choose any partition as a root, build a directed acyclic graph (DAG) with edges pointing away from the root (as in Figure 1) and traverse the partitions in a breadth-first manner. Due to the lack of cycles in the join, only directly connected partitions (parent and child nodes) can have join conditions with each other, which for the sampling translates to conditional independence between all partitions that are not in parent-child relationships. Thus, after sampling a parent partition (e.g., *AB*), its child partitions (e.g., *FA*, *BC*, *BG*) are sampled independently. While in Figure 1 there is just one partition with multiple children, the approach works similarly when multiple partitions have more than one child. Once all partitions on the same level (partitions with same distance to the root) are sampled, their ancestors (partitions closer to the root) do not need to be revisited.

3.3 Group Weight Algorithm.

The basic idea of the efficient algorithm is to group all partition-spanning trees that have the same root and then sample such a root-group. The challenge left to solve is how to compute the sum of weights of each root-group. The right side of Figure 2 shows the most crucial primitive for this purpose. The operation sums the weights of all sub-trees of each link node and adds it as a new label of the link node. Such an operation can be implemented through a linear-time sequential scan of *BC* and generating a hash table of *B*-values contained in *BC* where each entry holds the sum of observed weights of each *B*-value. As a second step, one can scan *AB*, which holds the root nodes. Now, for each item $\alpha_i \in AB$ with link value b_j one can look up the sum of sub-tree weights $W(b_j)$ in the hash table entry of the key b_j. The total weight of all groups that contain α_i is then $w(\alpha_i)W(b_j)$. If *BC* is not a leaf node, one simply recursively continues this procedure to compute the weights of *BC* until the leaf nodes are reached. Once a group with α_i as a root node is sampled with probability proportional to its total weight, one can continue in a similar fashion with the children nodes (adjacent nodes in the graph facing away from the main stream), until the full tree is obtained. Semi-joins, anti-joins and selections can be supported through weights, which is detailed later.

Algorithm 1 describes how the group weights can be obtained for general multi-way joins. Figure 3 applies the algorithm for the running example. *AB* serves as the main stream (see Line 2), such that the parent moves closer to *AB* (Line 3) and child nodes move further away from *AB* (Line 4), while leaves such as *CD* are on the outskirts furthest away from *AB* (Line 5). The algorithm processes a new stream in each iteration (Lines 7-26). It makes a single stream pass over the items of the new stream (Lines 11-20), computes the total weight of the sub-trees rooted at the item (Lines 12-16) and adds this sub-tree weight multiplied by its own weight $w(\tau)$ to the parent link node (Lines 17-19). After the new stream has been processed it is removed from further consideration (Lines 21-23) and the algorithm goes back to Line 7 and terminates when no more streams are left for consideration. After termination, each item node τ of the main stream is linked to multiple link nodes and the product of the link node labels multiplied by $w(\tau)$ yields the total weight of all partition-spanning trees containing τ.

3.4 Hash Table Implementation.

The algorithm can be implemented with a hash table $H_{\mathcal{J}}$ for each link node partition $\mathcal{J}$ to allow an efficient lookup of the adjacent link node for a item node based on the link value. Note that if all item nodes except a few have the same label, one only keeps entries for the exceptions and maintains a default value for the rest. Hash table entries for a stream can be computed in one scan that skips any items that do not satisfy the selection predicates. In case of a semi-join, the default value is 0 and entries are only equal to 1 for results of the semi join. While an anti-join can be as large as a stream, for sampling it can be supported via semi-join: for anti-joins the default value is 1 and entries are only equal to 0 for results of the semi join. Theta/non-equi-joins can also be easily supported. If the link condition is $\neq$, then in addition to the hash-table one needs to maintain the total weight of all hash-table entries. Then the weight for equi-joins can simply be subtracted from the total weight, to obtain the $\neq$-join weight. For theta joins with a binary operator $\odot \in \{<, \leq, \geq, >\}$ for the link condition, one can first obtain the equi-join hash table and then replace the values with cumulatives. This means that the entry for each join attribute value x holds the sum of weights of any equi-join entry y that satisfies $y \odot x$. Additionally a binary search tree needs to be constructed to efficiently find the last value $y \odot x$ where x is the queried value from a joining stream. Then the queried value can be rewritten as the last hash table value.

4 ONE*-PASS SAMPLER

In the previous section weighted join sampling has been mapped to the problem of sampling partition-spanning trees with probabilities proportional to their weight. In this section, we show how the sampling can be performed with few streaming passes over the input tables. Recall that in our notion of one*-pass sampling, we have one distinguished stream A over which we take exactly one pass. We will make use of concepts from graphical models to express the sample probabilities. Specifically, for each other stream in the query, we can use conditional probabilities to express the desired sampling probabilities. Based on that, one can for each stream linked with the one-pass stream derive a Bayesian Network that can be used with ancestral sampling. For instance in the running example, we get for $\alpha \in AB,\ \beta \in BC,\ \delta \in CD,\ \phi \in FA$:

$$\Pr(\beta, \delta, \phi \mid \alpha) = \Pr(\phi \mid \alpha) \Pr(\beta \mid \alpha) \Pr(\delta \mid \beta).$$

For the many-to-one relations with uniform weights one can first independently select a random item $\gamma \in BG$ that links with α and then a random item $\chi \in GH$ that links with γ. As the task is to sample with probabilities proportional to $W(\tau)$, one can in a first stage perform a stream pass over the main table and collect a sample using the proposed online multinomial sampler from Section 4. After collecting the main table sample, this yields the sampled groups of result trees grouped by the main table row and it is left sampling within the groups. Thus, in each stage the rows in the sample are extended by the row of another table until all tables that participate in the join have been reached. The main table due to the online multinomial sampler from Section 4 is only scanned once, while all other tables are scanned twice. The remaining challenge is then to support weights over the one-pass stream. For this purpose an online multinomial sampler is needed.

4.1 Online Multinomial Sampler

The online multinomial sampling problem (equivalent to the online weighted with-replacement problem) has been alluded to in the literature [20], but appears to not have been adequately solved. Note that unlike in [31] the weights here are real-valued and unlike in [65] only weights proportional to probabilities are available. The baseline technique from the literature would be to maintain an independent sampler for each element of the with-replacement sample, which clearly does not scale with larger sample sizes. Algorithm 2 proposes an online multinomial sampler based on an adaption of an existing sampling technique [21]:

THEOREM 4.1. *Let $n, N \in \mathbb{N}$ with $n \le N$, $w_1, \dots, w_N \in \mathbb{R}_{\ge 0}$ with $W = \sum_{i=1}^N w_i$. Then Algorithm 2 with weights $w_1, \dots, w_N$ returns a multinomial sample with probabilities proportional to weights, i.e., each M_j has $\Pr(M_j = i) = w_i / W$ for any $i \in \{1, \dots, N\}$.*

Specifically, Lines 1-3 of Algorithm 2 can be implemented as weighted reservoir sampling by Efraimidis & Spirakis [21] using

Algorithm 2: General Online Multinomial Sampler

```
  Input: Let $n, N \in \mathbb{N}$ with $n \le N$ and $w_1, \dots, w_N \in \mathbb{R}_{\ge 0}$.
1 Let $W = \sum_{i=1}^N w_i$ and $K_1, \dots, K_N$ be i.i.d. random
    variables with $K_i \sim Unif(0,1)^{(1/w_i)}$ for any
    $i \in \{1, \dots, N\}$.
2 Let $K_{[1]} \le K_{[2]} \le \dots \le K_{[N]}$ be the order statistics of $K$.
3 Let $S_1, \dots, S_n$ be the indices of $K$'s first $n$ order statistics,
    i.e., $K_{S_1} = K_{[1]},\ K_{S_2} = K_{[2]},\ \dots,\ K_{S_n} = K_{[n]}$.
4 Select $M_1 = S_1$ and initiate $m = 2$ and $j = 1$
5 while $j \le n$ do
6     Select $M_j$ according to $\Pr(M_j = i) = \frac{w_{S_i}}{W}$ for any
        natural number $i < j$ and
        $\Pr(M_j \ge i) = W - \sum_{i=1}^{j-1} w_{S_i}$
7     if $M_j \ge i$ then
8         Select $M_j = S_m$ and update $m = m + 1$.
9     Update $j = j + 1$
  Output: Multinomial sample $M_1, \dots, M_n$
```

$w_1, \dots, w_N$ as item weights. The proof can be found in Appendix B.3 and is based on the idea that one can use the ordered weighted sample without replacement as a pool for random items that have not been previously drawn. One then draws an independent element with probabilities proportional to weights at each step. Observe that after the first item is drawn, the probability of drawing one of the previous items again is simply proportional to the weights of those previous items. Thus, one needs to consider two cases. In one case the previously selected item is drawn again and in the other case a new random element is selected from the remaining population without the previously selected items. For the first case only the previously selected items are needed and for the second case only items are needed from an equally small ordered sample that can be obtained using existing techniques [21] which serves as a proxy for the population items.

4.2 Join Sampler for One*-Pass Stream Setting

Algorithm 3 then employs the online multinomial sampler to achieve a one*-pass join sampler, i.e., the main stream passes by once, whereas all other streams pass by once before and once after the main stream. The first step is to prepare the Bayesian Network probabilities based on the group weights using Algorithm 1. The second step is to collect a sample over the main stream (Lines 2-12). The third step is to use the Bayesian Network to extend the sample over the main stream. More specifically, in the graph formulation (see Figure 2), each item τ in the main stream sample is on the “left” side, connected to a link node in the “middle” that links to multiple item nodes on the “right” side. Then for each sampled main stream item, the task is to sample an item node on the “right” side with probabilities proportional to the group weight. Luckily, the total weights of the sub-trees have been previously computed and are readily available in the hash maps for the link nodes. As the total weight W of the right side is known, one can for instance employ inversion sampling, i.e., draw a random number u between 0 and W and go through the right side stream until the total weight of observed items is more than u and then pick the preceding item. This allows to collect all sample continuations of the main stream sample in one stream pass.

The runtime complexity of the one*-pass sampler is $O(kN \log N)$ for k streams of length N, as each stream is passed a constant

Algorithm 3: One*-Pass Sampler

```
1  Prepare Bayesian Network probabilities for streams
     linked with one-pass stream by doing one pass over
     those streams
2  Initiate $S_1, S_2, \ldots, S_n$ as weighted reservoir sample of size
     $n$
3  Initiate total weight $W_\mathcal{P}$ as 0 and multiset $M$ as $\{\}$
4  foreach stream item $x$ with weight $w(x)$ do
5      Consider $x$ for inclusion in $S$ and add $w(x)$ to $W_\mathcal{P}$
6  Initiate total sample weight $W_M$ as 0, $j$ as 1 and $\ell$ as 1
7  foreach $j \in \{1, \ldots, n\}$ do
8      Draw $u \in [0, 1]$ with $Pr(u) \propto 1$
9      if $u < W_M/W$ then
10         Set $M_j$ to a randomly drawn $M_i \in \{M_1, \ldots, M_{i-1}\}$
             with $\Pr(M_i) \propto w(M_i)$
11     else
12         Add $w(S_\ell)$ to $W_M$, set $M_j$ to $S_\ell$ and set $\ell$ to $\ell + 1$
13 Extend multinomial sample $M_1, \ldots, M_n$ via ancestral
     sampling over Bayesian Network in another pass over
     those streams
```

number of times (at most twice) and each stream item is processed in logarithmic time in the worst case (although, hash-based structures may yield performance on average that is constant time). The memory complexity is $O(D \cdot k)$ where $D \leq N$ is the maximal number of distinct link values per stream.

5 MULTI-PASS SAMPLER

In this section the one*-pass streaming is relaxed to achieve lower memory usage at the cost of potentially requiring additional stream passes.

5.1 Internal sampling for large streams with few distinct link values

Large streams with few link value combinations can be replaced by a sample per link value using the following theorem:

Theorem 5.1 (Internal Sampling). *Let $A, B, \ldots, G$ be data streams. Let $A', B', \ldots, G'$ be modified data streams, where any set of m stream items with the same link values may be replaced by a with-replacement sample of min(n, m) stream items with probabilities proportional to weights. The weight of each sampled item x is modified from $w(x)$ to $\frac{m}{n}w(x)$. Then a weighted sample over the join of $A' \otimes B' \otimes \ldots \otimes G'$ with $\otimes \in \{\bowtie, ⟖, ⟗, ⟕, \ltimes, \triangleright\}$ has the same distribution as a weighted sample over $A \otimes B \otimes \ldots \otimes G$.*

Proof. The modifications to the streams have the same effect as first drawing a sample from $A \otimes B \otimes \ldots \otimes G$ and then replacing all values with certain link value combinations (link value combinations pre-selected before drawing the sample) with random items with the similar link values proportional to weights. As items with similar link values are interchangeable with regards to what they link to and are still selected with probabilities proportional to weights, the probability distribution over all possible samples remains the same. □

We can apply this technique when we know that there are few such link values – via domain knowledge, or from simply checking the data with an extra initial streaming pass. In the next section, we consider how to reduce the number of values by hashing, at the expense of increasing the number of samples needed.

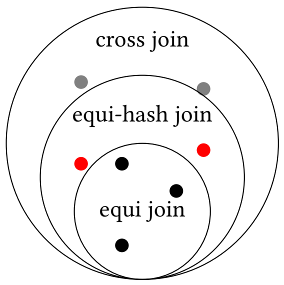


Figure 4: Hierarchy of join results. Samples from any equi-hash join with superfluous elements purged (depicted in red) are ordinary samples from the equi join.

5.2 Hashing gambit for small samples over many distinct link values

In the one*-pass sampler the memory complexity is proportional to the number of distinct link values. Thus, the basic idea is to reduce the number of distinct link values via hashing, which implements the same sampling method with potentially more stream passes (if the sample has to be large).

If the link condition is an equality and only a small sample is needed, one can efficiently deal with many distinct link values by reducing them to a smaller number of hash values. We call this the "hashing gambit", as it introduces superfluous join results that need to be rejected in order to gain the benefit of a smaller number of link values. The link function is for this purpose replaced from full equality to equality of hash values for a random hashing function.

Clearly, hash collisions link items together that are not actually linked, but those wrongly linked items can be simply purged at the end. In order to achieve a desired sample size, it can be necessary to generate more samples than before. A common source for a large number of distinct link values is when they are unique identifiers, in which case one can predict:

Lemma 5.2. *Let $T_1, T_2, \ldots, T_k$ be k streams with unique link values. Replacing the equality link with a hash-equality link using a universal hash function with universe of size u is expected to have at most $2m(\frac{m}{u})^{k-1}$ superfluous results that are not present in $T_1 \bowtie T_2 \bowtie \ldots \bowtie T_k$ where $m = \max(|T_1|, |T_2|, \ldots, |T_k|)$.*

Thus, as a heuristic, the hashing gambit will collect a $2(\frac{m}{u})^{k-1}$ times larger sample for the hash-relaxed superset (m and u as defined in Lemma 5.2), provided that it does not exceed the memory limit, as the join size is expected to be at least as large as the streams. As the parameter k counts the streams where u is much smaller than m, it is dependent on the choice of u. Thus, different choices of u can be tried out (numerically) to find the best choice within the formula before the sampling commences.

While each stream pass of the multi-pass sampler has the same runtime complexity as the one*-pass sampler, the number of needed stream passes can vary to collect the full sample. The memory complexity of the multi-pass sampler is reduced to $O(u \cdot k)$ where u is the chosen number of distinct hash values and k is the number of tables.

5.3 Rejection sampling for joins with many-to-one links

A simple case to handle is that of many-to-one relations: this allows us to first sample from the "many" stream and then look-up the "one" entity in the other stream(s). If the weights are all equal, then this reduces to sampling from one stream and then joining the sample with the other streams. If the weights are not equal, one can either first proceed as if they were equal and afterwards employ rejection sampling to rectify inclusion probabilities, or one needs to find the group weights and treat the many-to-one join like a many-to-many join. The former is more memory efficient, whereas the latter is reliably fast. For the multi-pass sampler, we advocate the more memory efficient variant.

6 RELATED WORK

As discussed in the introduction, previous works on join sampling are proposed for simple random sampling (equi-weighted) in a database context [10, 62, 80] or employ heuristics (lacking a principled statistical model) to sample from static joins [2, 11, 27, 28, 30, 34, 42, 43, 55] or joins of streams [3, 23, 57, 73]. Samples collected using heuristics do not follow any well-defined distribution and are either intended for targeted aggregations or selectivity estimation to more accurately predict the cost of query plans in database engines. Some also focus solely on simple many-to-one relationships between tables, i.e., foreign key joins that reunite tables from a normalised schema. Such approaches are equivalent to sampling from one table and then extending the sample using the other tables [1, 25]. This sampling for many-to-one schemata corresponds to conventional sampling when uniform weights are applied. However, in the presence of weights or selection predicates it requires novel methods as presented in this work.

While prior join samplers [10, 49, 62, 80] internally use weighted sampling, externally the weights cannot be controlled and only uniform sampling is supported. The internal weighted sampling arises due to the structure of acyclic joins that makes it natural to sample each relation one by one (see next paragraph for more details). For instance, [49] internally uses Poisson sampling, i.e., a coin is flipped for each item and only items with successful flips are kept and each item can have varying success probabilities. Externally, [49] employs Bernoulli sampling, which is a special case that mandates all items to have the same success probability. As Poisson/Bernoulli sampling can make it difficult to control the sample size and merge/subsample the samples, we opt in this work instead for multinomial sampling [6, 17, 22, 29, 36, 60] that draws each sample independently. Similarly, random join order enumeration methods [8] internally use non-uniform weights (that cannot be controlled), but externally only support uniformly drawn permutations that only help with uniform join samples. While we show that a limited streaming setting is achievable for join sampling, this seems unlikely for enumeration methods, but since they offer interesting applications beyond sampling, it would be interesting to see if ideas from this work could inspire extending it to weighted random permutations.

Acyclic joins have a tree-like structure where each node corresponds to a tuple's (sub)join with the tables downstream, which allows decompositions akin to Yannakakis's work on acyclic conjunctive queries [76]. This is useful when reasoning about join sampling algorithms. Zhao *et al.* [80] were first to uncover the constraint that the extension of a uniformly sampled tuple *must* be drawn with probability proportional to the size of the (sub)join of a tuple with the tables downstream, which generalises the insights from [10] from binary joins to multi-way joins. The key idea of the algorithm in [80] is a tuple-oriented approach that first approximates this probability and then later rectifies it via rejection sampling, which is inspired by Olken's method [52]. Rejections are avoided if approximations are exact, which empirical results indicate to work better for larger sample sizes [62, 80]. As the approach extends each tuple individually table-by-table, it mandates an index.Unlike in this work, the authors of [80] did not focus on limiting memory usage.

For cyclic joins it has been shown to be necessary to either build indices [80] or other intermediate data structures [62], but while relying heavily on breaking the cycles near-optimally, it has not been rigorously explored how such an optimisation problem be solved efficiently. Finally, there have been many works that operate over some approximation of the stream join, e.g., join of continuous stream with non-retroactive relations [26] (stream tuples joined with relations in current state, as changes to relations would require another pass over stream).

In conclusion, to our knowledge none of the prior works explicitly support weighted sampling over linked data streams. We also do not find any precedent for the idea of one*-pass sampling, or the hashing gambit.

7 EXPERIMENTAL STUDY

This section compares the novel one*-pass sampler (minimising stream passes) and multi-pass sampler (minimising memory) to existing non-streaming approaches from the database literature. Very large numbers that may be troublesome in practice are highlighted in red and the best achieved number is highlighted in yellow to draw the attention of the reader. Query times are reported either as *average times* to answer one query ad-hoc (data arrives during query time) and *amortised times* for pre-built index structures, i.e., how long does it take to run only the query excluding index building times that are are amortised when running many queries.

7.1 Experimental Setup

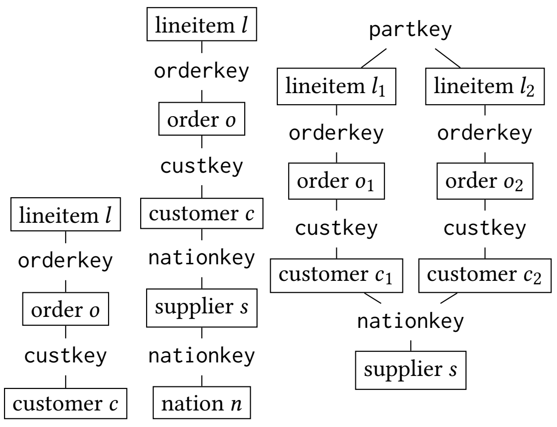
All experiments are performed on a single thread of a dedicated machine using Ubuntu 18.04.4 LTS, an Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz with 16 cores and 512GB RAM. Memory measurements are taken using the Unix primitive `/usr/bin/time -v` that provides the "maximum resident set size". All code has been written in C++11 by the same author and was compiled using GCC 7.5.0 with the compiler flags `-O3` and `-std=c++11`. The main baselines used for the experiments have been selected based on the results of Table 1 and an overview of the TPC-H queries can be found in Figure 5.

7.2 Join Queries

The experiments feature join queries over three datasets with join sizes shown in Table 2. The dataset used in the TPC-H benchmark, a social network of twitter users and a citation network using records from DBLP. Over the TPC-H benchmark the same queries are used as in [80] (cf. Figure 5), but additionally define weights. The scale factor corresponds roughly to the size of the dataset in GBs. As an application-provided weighting function `o_totalprice (1-l_discount) l_extendedprice` is used. For QY, the values of both instances of `lineitem` and `order` are multiplied with each other. Weighted queries are referred to as WQ3

Table 1: How baselines were selected: Comparison of potential baselines

	considered baselines		
	naive	[80] code	non-stream
Q3 $_{SF=100}$ memory	26.0 GB	142.0 GB	88.2 GB
QX $_{SF=100}$ memory	52.0 GB	169.7 GB	95.2 GB
QY $_{SF=100}$ memory	77.6 GB	384 GB	277.5 GB
Q3 $_{SF=100}$ average time	20.2 mins	64.2 mins	18.2 mins
QX $_{SF=100}$ average time	1236.2 mins	91.5 mins	20.3 mins
QY $_{SF=100}$ average time	895.1 mins	116 mins	102.7 mins
Q3 $_{SF=100}$ amortised time	20.2 mins	5.7 mins	7.1 mins
QX $_{SF=100}$ amortised time	1236.2 mins	5.7 mins	7.9 mins
QY $_{SF=100}$ amortised time	895.1 mins	42 mins	52.9 mins



(a) Q3 (b) QX (c) QY

Figure 5: Join queries over TPC-H data.

Table 2: Join sizes of queries used in the experiments . Q3 and DBLP are joins with many-to-one links and QX and QF are joins with many-to-many links.

TPC-H	$SF{=}10$	$SF{=}100$	Real-World Data Sets	
(W)Q3	$6.0 \cdot 10^7$	$6.0 \cdot 10^8$	DBLP	$4.5 \cdot 10^7$
(W)QX	$2.4 \cdot 10^{12}$	$2.4 \cdot 10^{13}$	Twitter QF	$2.7 \cdot 10^{21}$

(foreign-key join), WQX (many-to-many join) and WQY (cyclic join). Over the (raw) twitter dataset [41] the snowflake query QF is used as posed in [80] (cf. Table 4).

7.3 Compared Approaches

Naive (Join-Then-Sample straw-man). This straw-man approach is a slightly more sophisticated version of first joining and then sampling. The approach joins together tables in a greedy fashion that reduces the sizes of the remaining tables until only one join column remains. Then a merge-sort inspired approach is used to retrieve the inverse of uniform variates. While this approach is not explicitly proposed in the literature, it seems fairly straightforward and is a clearly superior baseline to naive join-then-sample as it avoids materialising the full join result. As shown in Table 1 this merge-sort based approach is still extremely slow and it is therefore not used as a general baseline.

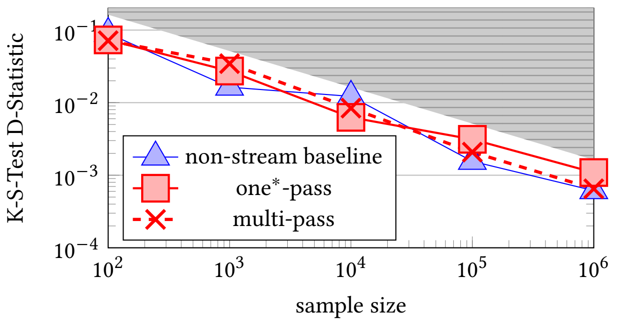


Figure 6: K-S goodness-of-fit-test (Section A) on WQY query (SF = 1) with over 10^{12} result rows.

Non-Stream (Index-based Baseline). This approach implements an index-based approach generalised to weighted sampling that is mostly based on the equal-probability sampling approach [80]. As several details and parameter settings are not fully specified in the paper [80], several aspects are reverse engineered from the published code or simplified and generalised[1]. The main goal of this implementation is to exploit indices in the same way as [80], but ideas from this work were crucial to fill in the gaps and streamline the approach.

One-Pass (Proposed).* The one*-pass sampler implements the proposed approach from Section 3 and prioritises a stream-like access over the data and limited number of scans. To only achieve a single pass over one stream (usually the largest) it uses the online multinomial sampler from Section 4.

Multi-Pass (Proposed). The multi-pass sampler implements the ideas from Section 5 and prioritises low memory usage. For many-to-one joins it first generates a uniform sample and then uses rejection sampling. For general joins, it uses the hashing gambit technique to deal with high-cardinality link values.

7.4 Correctness via Goodness-of-Fit Testing

Figure 6 shows that all implementations generate samples that follow the instructed multinomial distributions. The Kolmogorov-Smirnov (KS) test statistic is reported for each returned sample, and shading is used to indicate the critical region. The approaches all stay with 99% probability below the shaded region. Appendix A shows how to apply the KS test in the discrete setting.

7.5 Joins with Many-To-One Links

For many-to-one joins, one can first sample from the "many" side of the relationships and will have exactly one extension on the other side. This makes the problem a lot easier to solve, but still poses some challenges in the weighted sampling case. To support weighted sampling, we can upper bound the sampling weights by computing the product of maximal base table weights and then accept uniformly sampled rows with probability equal to the ratio with the weight upper bound. As can be seen in Table 3, this works well for linear weight functions. Here, the multi-pass sampler uses less memory by assuming group weights being equal to one, which is then rectified through rejections. Anticipating some rejections, it collects a ten times larger sample.

[1]The original published code for [80] was not suitable for our experiments in this work as it only supports integer-valued data without non-join columns and does not output/validate samples.

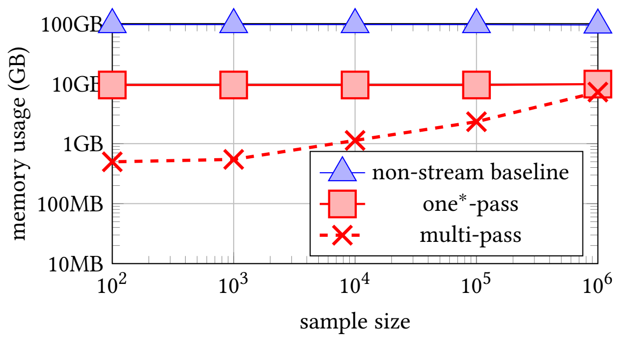


Figure 7: Memory for varied sample size over $WQX_{SF=100}$.

7.6 Joins with Many-To-Many Links

The query QX over the TPC-H dataset used in Figure 7 does not have many distinct values, yet the multi-pass sampler vastly reduces the memory footprint for smaller samples. The intuition is that for small samples one can choose to collect a larger set of tuples for the hashing gambit technique, and then discard the false positives. For the query QF over the Twitter dataset (cf. Table 4), the multi-pass sampler also achieves vast memory savings for a target sample size of 1 million items.

8 CONCLUSION

The main objective of this work are lightweight approaches for weighted sampling to support data-mining tasks over linked data streams. As in a pure one-pass streaming setting all streams would require storing all stream items, instead a one*-pass setting and a multi-pass setting is considered. In the one*-pass setting one main stream is passed once, whereas all other streams are passed once before and once after the main stream. In the multi-pass setting this is relaxed to allow for substantial memory savings. Being able to support weighted sampling despite just a single pass required advancing online samplers with-replacement and being able to reduce memory required new ideas like the hashing gambit to collect more compact statistics over the streams. In the experiments these approaches outperformed approaches based on indices if the data comes in at query-time, but the smaller memory footprint comes at the price of slightly larger query times when the build cost are presumed to be amortised. As an open problem remain joins with cyclic links and complex link functions that cannot be expressed via equations or inequalities between link values.

Table 3: Runtime and memory comparison for a million samples over foreign-key joins.

	baseline	proposed	
	non-stream	one*-pass	multi-pass
WQ3 $_{SF=10}$ memory	10.6 GB	1.6 GB	1.9 GB
WQ3 $_{SF=100}$ memory	94.6 GB	9.8 GB	2.6 GB
WQ3 $_{SF=10}$ average time	1.8 mins	1.3 mins	1.4 mins
WQ3 $_{SF=100}$ average time	20.5 mins	12.3 mins	7.5 mins
WQ3 $_{SF=10}$ amort. time	0.7 mins	1.3 mins	1.4 mins
WQ3 $_{SF=100}$ amort. time	7.9 mins	12.3 mins	7.5 mins

Table 4: Runtime and memory comparison for a million samples over joins with many-to-many links.

	baseline	proposed	
	non-stream	one*-pass	multi-pass
WQX $_{SF=10}$ memory	10.7 GB	1.6 GB	2.2 GB
WQX $_{SF=100}$ memory	95.0 GB	9.8 GB	7.2 GB
QF memory	428.4 GB	91.5 GB	16.2 GB
WQX $_{SF=10}$ average time	1.9 mins	1.4 mins	1.8 mins
WQX $_{SF=100}$ average time	20.6 mins	13.1 mins	14.0 mins
QF avg. time	126.5 mins	36.0 mins	47.0 mins
WQX $_{SF=10}$ amort. time	0.7 mins	1.4 mins	1.8 mins
WQX $_{SF=100}$ amort. time	7.9 mins	13.1 mins	14.0 mins
QF amortised time	63.1 mins	36.0 mins	47.0 mins

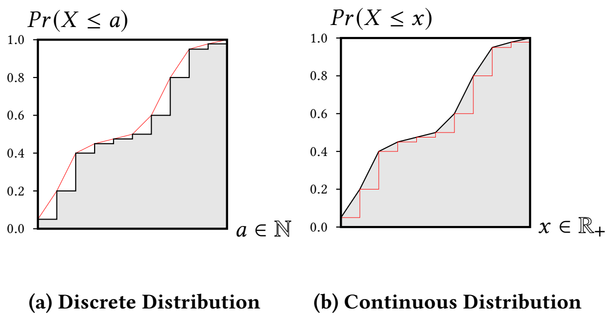


Figure 8: Converting discrete distribution into continuous one by adding uniform variates (cf. Lemma A.1) to apply continuous goodness-of-fit tests such as the K-S Test.

A APPENDIX: GOODNESS-OF-FIT TESTING FOR MULTINOMIAL DISTRIBUTIONS

For sample validation, Zhao *et al.* [80] propose to utilise conventional KS-testing to validate samples over join rows. This is at odds with the statistics literature on (conventional) KS-testing, which requires a *continuous* reference distribution. While KS-testing can be adopted to the discrete setting [4, 56], the work [80] reports distribution-free critical values which do not exist in the discrete case and appears to confuse type-I and type-II errors. The range of the reported D-statistics would also (almost certainly) not occur for the reported sample sizes, if the join result had sufficiently many distinct results to approximate a continuous distribution.

Thus, in the following a correct way is presented how to apply Kolmogorov-Smirnov (KS) testing when the reference distribution is discrete, as an alternative to goodness-of-fit tests for multinomial distributions such as the likelihood ratio test [17, 29]. Multinomial tests ignore the ordering unlike KS-tests that were originally designed for continuous distributions. There exist discrete variants of KS-tests [4, 56], but they lose a lot of desirable properties of the continuous one such as distribution-free statistics, which is why it is here instead proposed to use the (conventional) continuous KS-test after turning the discrete distribution into a continuous one in a straightforward way:

LEMMA A.1 (CONTINUOUS CONVERSION). *Let $[x_1, x_2, \ldots, x_N]$ be a multinomially distributed frequency vector with event probabilities $p_1, \ldots, p_N$. Let X be a set obtained by drawing for the i-th* *event x_i independent uniform variates between $(i-1)$ and i. Then $\mathcal{X}$'s distribution is continuous with a piecewise-linear cumulative distribution function (CDF), i.e., $F(x) = p_{\lceil x \rceil}(x - \lceil x \rceil) + \sum_{i=1}^{\lceil x \rceil} p_i$.*

Proof. As shown in Figure 8, the steps of the discrete CDF are smoothed by adding uniform variates, which have a continuous uniform distribution with a linear CDF. Thus one obtains a CDF that interpolates linearly between the steps without any jumps. □

The suggested continuous conversion describes how both samples and reference distributions have to be modified and is a clean way of applying continuous goodness-of-fit tests. While this comes with the disadvantage of slightly diluting the original distribution with additional randomness, this is needed to extend the distribution's finite support to an infinite one. Luckily, the impact of the added randomness quickly dissipates with larger discrete support size in this application, which means that the power of the test is asymptotically similar to conventional KS-testing.

B APPENDIX: PROOFS

B.1 Proof for Theorem 2.3

Let A and B be two one-pass data streams, i.e., both stream by just once. Let DISTINCT be the minimal number of distinct link values of A and B, |SAMPLE| be the sample size. Then collecting a multinomial sample (with non-zero probabilities) of $A \bowtie B$ requires storing Ω (min{ $|A| + |B|$, DISTINCT · |SAMPLE| }) stream items.

Proof. The order in which the streams are interleaved can be arbitrary; for simplicity, assume that that stream A has higher cardinality than stream B and the entirety of stream A is seen first before stream B. Producing the join is hard even if stream B contains only a single item β: if β links with an item in A, then the output should contain β, but not otherwise. Determining whether β links with an item in A essentially requires storing A in full. It is straightforward to let A encode a bit string x of length $n = |A|$, where item $\alpha_i \in A$ if and only if the corresponding bit $x_i = 1$. This captures the Index problem [40], whose communication complexity gives a lower bound of $\Omega(n)$ for the space needed to solve the join problem. In this case, the join is either empty or $\{\beta\}$, and so any sample must contain the full join results. In the case when A has a small set of distinct item identifiers, then the same argument shows that the space required must be proportional to this cardinality, DISTINCT. □

B.2 Proof for Theorem 2.5

A one*-pass sampler over $A \bowtie B \bowtie \ldots \bowtie G$ requires Ω(DISTINCT) bits of memory where DISTINCT is the number of distinct link values appearing in the streams $B, \ldots, G$.

Proof. The basic idea is to consider an adversarial setting where the content of stream A depends on which statistics are collected for stream B (simulating unpredictable data distributions). Clearly, if for an item in A we do not have any statistics available about how items in B link with it, it is impossible to sample the item from A with the correct probability. If B is only streamed once, clearly one would not only need the statistics but also the items, which means "one*-pass" is indeed the strictest setting that is still feasible without having to record the full stream.

The formal proof is by contradiction. If fewer than M values are stored for stream B, then there may exist a link value for which no statistics are available when A is streamed. The contradictory claim is then that an item from A is sampled with the correct probability without any knowledge of how many items in B link with it. Similarly, as B is streamed before A, there is no way of knowing which link values appear only in B. Note that this presumes an adversarial setting, where the content of streams is allowed to be reactive to which link values are stored (simulating unpredictable data distributions). If B is only streamed once, it would furthermore require storing all stream items as outlined in the previous theorem. □

B.3 Proof for Theorem 4.1

Ordered sampling brings the population items into an order such that one can pick the first items as the sample. Using this idea, one can generate for each item with weight w_i an independent exponential variate $V_i \sim \text{Exp}(w_i)$ and order them by these variates. It follows then from well-known properties of exponential variates that $\Pr(V_i = \min(\{V_1, V_2, \ldots, V_N\}) = \frac{w_i}{w_1 + w_2 + \ldots + w_N}$. Thus, the minimum is chosen with probability proportional to weights, the second-smallest item with probability proportional to weights if the minimum item did not exist, and so on. Efraimidis & Spirakis [21] (E&S) analogously use $K_i = e^{-V_i} \sim U(0,1)^{1/w_i}$ as random "keys", which flips the order by applying the strictly decreasing function e^{-x}, but otherwise yields the same properties and probabilities just for the maximum. E&S then define the n items with the largest K_i as a "weighted sample" of size n, although only the largest-key item is chosen proportional to weights in the final sample [66]. Thus, E&S do not use the weights to decide probabilities of the overall sampling process, but only in relation to rounds of without-replacement draws in the urn model. Meanwhile, here the n largest-key items are only used as an intermediary for the population. Weighted with-replacement sampling has a very simple interpretation, i.e., each item is independently selected into the sample with probability proportional to its weight. This approach is described in Algorithm 3. The first step is to find the total weight $W_{\mathcal{P}}$ of all population items $\mathcal{P}$ and the n largest-key items $S_1, S_2, \ldots, S_n$ from $\mathcal{P}$ still sorted by the keys, i.e., $k_1 \geq k_2 \geq \ldots \geq k_n$ (Lines 2-5). From that, it follows that all other population items $\mathcal{P}/\{S_1, S_2, \ldots, S_n\}$ have a smaller key than k_n and will not make it into the multinomial sample. To find the largest-key items one can employ the weighted reservoir sampler by E&S [21] with the exponential jump algorithm. In the next steps of the approach (Lines 8-12), one draws in each step an independent element M_j with $j \geq 1$ and maintains at each moment the total weight W_M of all distinct elements $\{M_1, M_2, \ldots, M_{j-1}\}$ that were previously selected. Note that for $M_j = M_1$ the "abuse of notation" $\{M_1, M_2, \ldots, M_{j-1}\}$ is used to denote an empty set and for $M_j = M_2$ it is equal to $\{M_1\}$. With probabilities proportional to weights, M_j is either in the set of previously chosen items or not yet chosen items:

$$M_j \in \begin{cases} \{M_1, M_2, \ldots, M_{j-1}\} & \text{with probability } W_M / W_{\mathcal{P}} \\ \mathcal{P}/\{M_1, M_2, \ldots, M_{j-1}\} & \text{otherwise} \end{cases}$$

A (biased) coin flip determines according to these probabilities from which set one random element is selected with probabilities according to weights. Observe that it is exploited here that $\{S_1\}$ is a one-item weighted sample from $\mathcal{P}$, $\{S_2\}$ is a one-item weighted sample from $\mathcal{P}/\{S_1\}$, $\{S_3\}$ is a one-item weighted sample from $\mathcal{P}/\{S_1, S_2\}$ and so on.

C DIFFERENTIALLY PRIVATE SELECTION OVER JOINS

Suppose join rows are scored in terms of interest based on sensitive user data and the impact of a single user on the scores is limited. Then differentially private join sampling allows to select the join rows of interest a lot more likely, but not too likely as to reveal with certainty which users participated in the scores. Formally:

Lemma C.1. *Let $A, B, \ldots, G$ be streams. Let $\alpha \in A, \beta \in B, \gamma \in G$ be stream items and u be a real-valued utility function defined over them. Let $u(\alpha) + u(\beta) + \ldots + u(\gamma)$ for any combination of linked stream items change at most by $\pm\Delta$ due to a single added/removed user. Then it is ε-differentially private to release a multinomial sample of size n with $\Pr(\alpha, \beta, \ldots, \gamma) \propto w(\alpha) \cdot w(\beta) \cdot \ldots \cdot w(\gamma)$ and $w(x) = \exp(\frac{\varepsilon u(x)}{2n\Delta})$ for any item x.*

Proof. The proof follows via the well-known exponential mechanism [50] and sequential composition results [19] that the probability of each multinomial sample of size n changes at most by a multiplicative factor $\exp(\varepsilon)$ due to a single user being added or removed when using this weight function. Per definition $u(\alpha) + u(\beta) + \ldots + u(\gamma)$ has sensitivity Δ, i.e., changes at most by $\pm\Delta$ due to a single added/removed user, and one can rewrite the probability as ($\propto$ hides normalisation constants):

$$\Pr(\alpha, \beta, \ldots, \gamma) \propto \exp\left(\frac{\varepsilon}{2n\Delta}(u(\alpha) + u(\beta) + \ldots + u(\gamma))\right)$$

Via the exponential mechanism [50] then follows that $\Pr(\alpha, \beta, \ldots, \gamma)$ changes due to a single added/removed user at most by a multiplicative factor $\frac{\varepsilon}{n}$, i.e., drawing one sample is ε/n-differentially private. Via sequential composition [19] then follows that drawing n independent samples is ε-differentially private. □

ACKNOWLEDGMENTS

This work is supported by European Research Council grant ERC-2014-CoG 647557 and The Alan Turing Institute under the EPSRC grant EP/N510129/1.

REFERENCES

[1] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. Join Synopses for Approximate Query Answering. In *ACM SIGMOD Conference*. 275–286. https://doi.org/10.1145/304182.304207

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Eurosys Conference*. 29–42. https://doi.org/10.1145/2465351.2465355

[3] Mohammed Al-Kateb, Byung Suk Lee, and Xiaoyang Sean Wang. 2007. Reservoir Sampling over Memory-Limited Stream Joins. In *International Conference on Scientific and Statistical Database Management*. 23. https://doi.org/10.1109/SSDBM.2007.40

[4] Taylor B Arnold and John W Emerson. 2011. Nonparametric goodness-of-fit tests for discrete null distributions. *R Journal* 3, 2 (2011).

[5] Kedar Bellare, Suresh Iyengar, Aditya G. Parameswaran, and Vibhor Rastogi. 2012. Active sampling for entity matching. In *ACM SIGKDD Conference*. 1131–1139. https://doi.org/10.1145/2339530.2339707

[6] Daniel A Bloch and Geoffrey S Watson. 1967. A Bayesian study of the multinomial distribution. *The Annals of Mathematical Statistics* (1967), 1423–1435.

[7] Ken RW Brewer and Muhammad Hanif. 2013. *Sampling with unequal probabilities*. Vol. 15. Springer Science & Business Media.

[8] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. 2020. Answering (Unions of) Conjunctive Queries using Random Access and Random-Order Enumeration. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, Dan Suciu, Yufei Tao, and Zhewei Wei (Eds.). ACM, 393–409. https://doi.org/10.1145/3375395.3387662

[9] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2007. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)* 32, 2 (2007), 9–es.

[10] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. 1999. On Random Sampling over Joins. In *ACM SIGMOD Conference*. 263–274. https://doi.org/10.1145/304182.304206

[11] Yu Chen and Ke Yi. 2017. Two-Level Sampling for Join Size Estimation. In *ACM SIGMOD Conference*. 759–774. https://doi.org/10.1145/3035918.3035921

[12] CKCN Chow and Cong Liu. 1968. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory* 14, 3 (1968), 462–467.

[13] Peter Christen. 2008. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *ACM SIGKDD Conference*. 151–159. https://doi.org/10.1145/1401890.1401913

[14] William W. Cohen and Haym Hirsh. 1998. Joins that Generalize: Text Classification Using WHIRL. In *ACM SIGKDD Conference*. 169–173. http://www.aaai.org/Library/KDD/1998/kdd98-027.php

[15] Michael D. Conover, Matthew Hayes, Scott Blackburn, Pete Skomoroch, and Sam Shah. 2018. Pangloss: Fast Entity Linking in Noisy Text Environments. In *ACM SIGKDD Conference*. 168–176. https://doi.org/10.1145/3219819.3219899

[16] Graham Cormode and Nick G. Duffield. 2014. Sampling for big data: a tutorial. In *ACM SIGKDD Conference*. 1975. https://doi.org/10.1145/2623330.2630811

[17] Noel Cressie and Timothy RC Read. 1984. Multinomial goodness-of-fit tests. *Journal of the Royal Statistical Society: Series B (Methodological)* 46, 3 (1984), 440–464.

[18] Wei Dong and Ke Yi. 2021. Residual Sensitivity for Differentially Private Multi-Way Joins. In *ACM SIGMOD Conference*. 432–444.

[19] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.

[20] Pavlos S Efraimidis. 2015. Weighted random sampling over data streams. In *Algorithms, Probability, Networks, and Games*. 183–195.

[21] Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted random sampling with a reservoir. *Inf. Process. Lett.* 97, 5 (2006), 181–185. https://doi.org/10.1016/j.ipl.2005.11.003

[22] Ted H Emigh. 1983. On the number of observed classes from a multinomial distribution. *Biometrics* (1983), 485–491.

[23] Raphaël Féraud, Fabrice Clérot, and Pascal Gouzien. 2010. Sampling the join of streams. In *Classification as a Tool for Research*. 307–314.

[24] Nir Friedman, Dan Geiger, and Moises Goldszmidt. 1997. Bayesian network classifiers. *Machine learning* 29, 2 (1997), 131–163.

[25] Rainer Gemulla, Philipp Rösch, and Wolfgang Lehner. 2008. Linked Bernoulli Synopses: Sampling along Foreign Keys. In *Scientific and Statistical Database Management*. 6–23. https://doi.org/10.1007/978-3-540-69497-7_4

[26] Lukasz Golab and M. Tamer Özsu. 2005. Update-Pattern-Aware Modeling and Processing of Continuous Queries. In *ACM SIGMOD Conference*. 658–669. https://doi.org/10.1145/1066157.1066232

[27] Peter J. Haas and Joseph M. Hellerstein. 1999. Ripple Joins for Online Aggregation. In *ACM SIGMOD Conference*. 287–298. https://doi.org/10.1145/304182.304208

[28] Marios Hadjieleftheriou, Xiaohui Yu, Nick Koudas, and Divesh Srivastava. 2008. Hashed samples: selectivity estimators for set similarity selection queries. *Proc. VLDB Endow.* 1, 1 (2008), 201–212. https://doi.org/10.14778/1453856.1453883

[29] Wassily Hoeffding. 1965. Asymptotically optimal tests for multinomial distributions. *The Annals of Mathematical Statistics* (1965), 369–401.

[30] Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. 2019. Join on Samples: A Theoretical Guide for Practitioners. *Proc. VLDB Endow.* 13, 4 (2019), 547–560. https://doi.org/10.14778/3372716.3372726

[31] Rajesh Jayaram, Gokarna Sharma, Srikanta Tirthapura, and David P Woodruff. 2019. Weighted reservoir sampling from distributed streams. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 218–235.

[32] George H. John and Pat Langley. 1996. Static Versus Dynamic Sampling for Data Mining. In *ACM SIGKDD Conference*. 367–370. http://www.aaai.org/Library/KDD/1996/kdd96-069.php

[33] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex AdHoc Queries in BigData Clusters. In *ACM SIGMOD Conference*. 631–646.

[34] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex AdHoc Queries in BigData Clusters. In *ACM SIGMOD Conference*. 631–646. https://doi.org/10.1145/2882903.2882940

[35] Alexandros Karakasidis, Georgia Koloniari, and Vassilios S. Verykios. 2015. Scalable Blocking for Privacy Preserving Record Linkage. In *ACM SIGKDD Conference*. 527–536. https://doi.org/10.1145/2783258.2783290

[36] Harry Kesten and Norman Morse. 1959. A property of the multinomial distribution. *The Annals of Mathematical Statistics* 30, 1 (1959), 120–127.

[37] Ivan Kojadinovic and Jun Yan. 2012. Goodness-of-fit testing based on a weighted bootstrap: A fast large-sample alternative to the parametric bootstrap. *Canadian Journal of Statistics* 40, 3 (2012), 480–500.

[38] Wouter Kool, Herke van Hoof, Max Welling, et al. 2020. Ancestral Gumbel-Top-k Sampling for Sampling Without Replacement. *J. Mach. Learn. Res.* 21 (2020), 47–1.

[39] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1371–1384.

[40] Ilan Kremer, Noam Nisan, and Dana Ron. 1999. On randomized one-round communication complexity. *Computational Complexity* 8, 1 (1999), 21–49.

[41] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue B. Moon. 2010. What is Twitter, a social network or a news media?. In *International Conference on*

World Wide Web. 591–600. https://doi.org/10.1145/1772690.1772751

[42] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. 2017. Cardinality Estimation Done Right: Index-Based Join Sampling. In *Conference on Innovative Data Systems Research*. http://cidrdb.org/cidr2017/papers/p9-leis-cidr17.pdf

[43] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *ACM SIGMOD Conference*. 615–629. https://doi.org/10.1145/2882903.2915235

[44] Ping Li. 2015. 0-Bit Consistent Weighted Sampling. In *ACM SIGKDD Conference*. 665–674. https://doi.org/10.1145/2783258.2783406

[45] Qingzhi Ma, A.M. Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. 2021. Dbest++: Learned Approximate Query Processing: Make it Light, Accurate and Fast. In *Conference on Innovative Data Research*. 1553–1570.

[46] Qingzhi Ma and Peter Triantafillou. 2019. Dbest: Revisiting approximate query processing engines with machine learning models. In *ACM SIGMOD Conference*. 1553–1570.

[47] Shuangge Ma and Michael R Kosorok. 2005. Robust semiparametric M-estimation and the weighted bootstrap. *Journal of Multivariate Analysis* 96, 1 (2005), 190–217.

[48] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. 2008. Privacy: Theory meets practice on the map. In *IEEE international conference on data engineering*. 277–286.

[49] Ben McCamish, Vahid Ghadakchi, Arash Termehchy, Behrouz Touri, and Liang Huang. 2018. The Data Interaction Game. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 83–98. https://doi.org/10.1145/3183713.3196899

[50] Frank McSherry and Kunal Talwar. 2007. Mechanism design via differential privacy. In *IEEE Symposium on Foundations of Computer Science*. 94–103.

[51] Arjun Narayan and Andreas Haeberlen. 2012. DJoin: Differentially private join queries over distributed databases. In {*USENIX*} *Symposium on Operating Systems Design and Implementation* ({*OSDI*} *12)*. 149–162.

[52] Frank Olken. 1993. *Random sampling from databases*. Ph.D. Dissertation. University of California, Berkeley.

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

[54] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. Verdictdb: Universalizing approximate query processing. In *ACM SIGMOD Conference*. 1461–1476.

[55] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In *ACM SIGMOD Conference*. 1461–1476. https://doi.org/10.1145/3183713.3196905

[56] Anthony N Pettitt and Michael A Stephens. 1977. The Kolmogorov-Smirnov goodness-of-fit statistic with discrete and grouped data. *Technometrics* 19, 2 (1977), 205–210.

[57] Jiadong Ren, Wanchang Jiang, and Cong Huo. 2007. Random sampling over streaming window joins. In *International Symposium on Data, Privacy, and E-Commerce*. 53–55.

[58] Bengt Rosén. 1997. On sampling with probability proportional to size. *Journal of statistical planning and inference* 62, 2 (1997), 159–191.

[59] Tim Roughgarden. 2015. Communication complexity (for algorithm designers). *arXiv preprint arXiv:1509.06257* (2015).

[60] M Ruiz, FJ Girón, CJ Pérez, J Martín, and C Rojano. 2008. A Bayesian model for multinomial sampling with misclassified data. *Journal of Applied Statistics* 35, 4 (2008), 369–382.

[61] MR Sampford. 1967. On sampling without replacement with unequal probabilities of selection. *Biometrika* 54, 3-4 (1967), 499–513.

[62] Ali Mohammadi Shanghooshabad, Meghdad Kurmanji, Qingzhi Ma, Michael Shekelyan, Mehrdad Almasi, and Peter Triantafillou. 2021. PGMJoins: Random Join Sampling with Graphical Models. In *ACM SIGMOD Conference*. 1610–1622.

[63] Lefteris Sidirourgos, Martin Kersten, and Peter Boncz. 2013. Scientific discovery through weighted sampling. In *IEEE International Conference on Big Data*. 300–306.

[64] Utkarsh Srivastava and Jennifer Widom. 2004. Memory-limited execution of windowed stream joins. In *VLDB*, Vol. 4. 324–335.

[65] Michał Startek. 2016. An asymptotically optimal, online algorithm for weighted random sampling with replacement. *arXiv preprint arXiv:1611.00532* (2016).

[66] Yves Tillé. 2019. A general result for selecting balanced unequal probability samples from a stream. *Inform. Process. Lett.* 152 (2019), 105840.

[67] Daniel Ting and Eric Brochu. 2018. Optimal subsampling with influence functions. *Advances in neural information processing systems* 31 (2018).

[68] Chi Wang and Bailu Ding. 2019. Fast Approximation of Empirical Entropy via Subsampling. In *ACM SIGKDD Conference*. 658–667. https://doi.org/10.1145/3292500.3330938

[69] HaiYing Wang. 2019. More efficient estimation for logistic regression with optimal subsamples. *Journal of machine learning research* 20 (2019).

[70] Haiying Wang and Yanyuan Ma. 2021. Optimal subsampling for quantile regression in big data. *Biometrika* 108, 1 (2021), 99–112.

[71] HaiYing Wang and Jiahui Zou. 2021. A comparative study on sampling with replacement vs Poisson sampling in optimal subsampling. In *International Conference on Artificial Intelligence and Statistics*. 289–297.

[72] Mingxi Wu and Chris Jermaine. 2006. Outlier detection by sampling with accuracy guarantees. In *ACM SIGKDD Conference*. 767–772. https://doi.org/10.1145/1150402.1150501

[73] Junyi Xie and Jun Yang. 2007. A Survey of Join Processing in Data Streams. In *Data Streams - Models and Algorithms*. 209–236. https://doi.org/10.1007/978-0-387-47534-9_10

[74] Yang Yang, Yizhou Sun, Jie Tang, Bo Ma, and Juan-Zi Li. 2015. Entity Matching across Heterogeneous Sources. In *ACM SIGKDD Conference*. 1395–1404. https://doi.org/10.1145/2783258.2783353

[75] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *Proc. VLDB Endow.* 14, 1 (2020), 61–73. https://doi.org/10.14778/3421424.3421432

[76] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Very Large Data Bases*. 82–94.

[77] Yaqiong Yao and HaiYing Wang. 2019. Optimal subsampling for softmax regression. *Statistical Papers* 60, 2 (2019), 585–599.

[78] Haoyu Zhang and Qin Zhang. 2017. EmbedJoin: Efficient Edit Similarity Joins via Embeddings. In *ACM SIGKDD Conference*. 585–594. https://doi.org/10.1145/3097983.3098003

[79] Haoyu Zhang and Qin Zhang. 2019. MinJoin: Efficient Edit Similarity Joins via Local Hash Minima. In *ACM SIGKDD Conference*. 1093–1103. https://doi.org/10.1145/3292500.3330853

[80] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *ACM SIGMOD Conference*. 1525–1539. https://doi.org/10.1145/3183713.3183739