Acyclic Join Sampling under Selections: Dichotomy, Union Sampling, and Enumeration

Jinchao Huang, Yufei Tao, and Sibo Wang

The Chinese University of Hong Kong
{*jchuang@link*, *taoyf@cse*, *swang@se*}*.cuhk.edu.hk*

August 8, 2026

Abstract

Previous research on *join sampling* has focused on joins without selection conditions, even though such conditions are prevalent in everyday queries. Motivated by this, we investigate the complexity of sampling from the result of an acyclic join under equality conditions given at runtime. When conditions are conjunctive, the goal is to understand when it is possible to precompute a *feasible* structure that uses $\tilde{O}(\mathrm{IN})$ space and supports sampling in $\tilde{O}(1)$ time, where IN is the input size. We present a dichotomy to characterize (subject to a widely-accepted conjecture) the existence of such structures based on the conditions supplied and, in every feasible scenario, give an optimal structure of $O(\mathrm{IN})$ space and $O(1)$ sampling time. We then extend our investigation to conditions expressed in disjunctive normal form (DNF), where the core challenge reduces to the fundamental *set union sampling* problem. We resolve the problem with an optimal algorithm and utilize it to develop optimal sampling structures for DNF conditions. Our findings also lead to new results on the closely-related problem of *random enumeration*.

A preliminary version of this article appeared in ICDT'26.

1 Introduction

Join evaluation on massive datasets often incurs prohibitive computation. A primary cause of this phenomenon is the sheer volume of the join result — whose size grows rapidly as the number of participating relations increases — because reporting each result tuple requires at least constant time. However, in many scenarios (ranging from approximate query processing and query optimization to interactive data exploration and machine learning), the goal is not to enumerate the entire join result, but rather to obtain a small representative sample. Such samples can be used to approximate aggregates, estimate selectivities, support interactive analysis, or train a learning model without materializing the full join. These practical needs have motivated the development of many join sampling algorithms; see [2, 8, 13, 15, 16, 19, 28, 34, 37] and the references therein.

Previous research has focused on sampling from joins without selection conditions. This stands in stark contrast to reality, where queries nearly always include such conditions, typically in the form of equality predicates like $(\mathtt{A} = a \wedge \mathtt{B} = b) \vee (\mathtt{C} = c)$. Here, the values a, b, and c are unknown in advance and are supplied by a query only at runtime. Essential in narrowing down the data of interest, these predicates prompt the question of how join sampling can remain effective in the presence of conjunctive and disjunctive selection conditions. This article addresses the question in a systematic manner.

Mathematical Conventions and Computation Model. The notation $\mathbb{N}$ represents the set of integers. Given an integer $x \geq 1$, we define $[x] = \{1, 2, ..., x\}$; for any real value x, we write $\exp(x) = e^x$. All logarithms have base 2 by default. Our analysis assumes the standard word-RAM model [17] where a word has length $\Theta(\log \mathrm{IN})$ with IN being the input size. Given an integer x representable using a word, we assume that a uniformly random number can be drawn from $[x]$ in constant time.

1.1 Problem Definitions

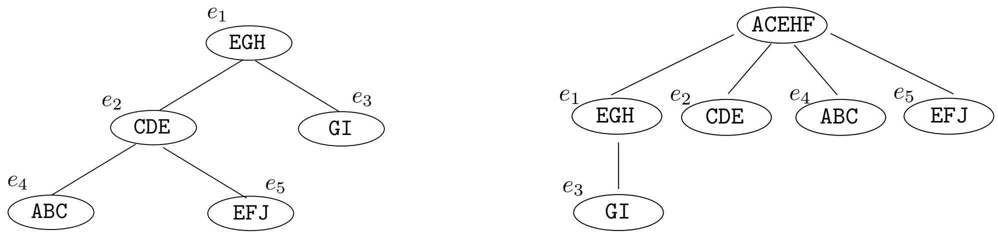
Let **att** be a set whose elements are called *attributes*. Given a subset $U \subseteq \mathbf{att}$, a *tuple* over U is a function $\boldsymbol{u} : U \to \mathbb{N}$. For each attribute $X \in U$, we call $\boldsymbol{u}(X)$ "the value of $\boldsymbol{u}$ under X" or simply the "X-value" of $\boldsymbol{u}$ — such a value is assumed to fit in a word. Given a subset U' of U, we use $\boldsymbol{u}[U']$ (note: square bracket here) to represent the tuple $\boldsymbol{v}$ over U' satisfying $\boldsymbol{v}(X) = \boldsymbol{u}(X)$ for all $X \in U'$; the tuple $\boldsymbol{u}[U']$ is called the *projection* of $\boldsymbol{u}$ onto U'. We define a *relation* as a set R of tuples over an identical set U of attributes, where U is called the *schema* of R, written as $schema(R) = U$.

Let $\mathcal{V}$ be a finite subset of **att** and $\mathcal{E}$ be a subset of $2^{\mathcal{V}}$. We call the hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ a *schema graph* if every attribute of $\mathcal{V}$ appears in at least one element — a hyperedge — of $\mathcal{E}$. The hypergraph is *acyclic* [1, 35] if there exists a tree $\mathcal{T}$ having two properties:

- [*the one-one property*] every node of $\mathcal{T}$ corresponds to one distinct hyperedge in $\mathcal{E}$;
- [*the connectedness property*] for each attribute $X \in \mathcal{V}$, the nodes in $\mathcal{T}$ (a.k.a. hyperedges in $\mathcal{E}$) containing X form a connected subtree in $\mathcal{T}$.

We refer to $\mathcal{T}$ as a *join tree* of $\mathcal{G}$.

Example 1.1. *Consider the hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{\mathtt{A}, \mathtt{B}, ..., \mathtt{J}\}$, and $\mathcal{E}$ has 5 hyperedges: $e_1 = \mathtt{EGH}$ (shortform for set $\{\mathtt{E}, \mathtt{G}, \mathtt{H}\}$), $e_2 = \mathtt{CDE}$, $e_3 = \mathtt{GI}$, $e_4 = \mathtt{ABC}$, and $e_5 = \mathtt{EFJ}$. The hypergraph is acyclic, with a join tree $\mathcal{T}$ given in Figure 1(a). The hyperedges containing attribute, for instance, $\mathtt{E}$ are e_1, e_2, and e_5, and they form a connected subtree of $\mathcal{T}$. As another example, the hyperedges containing $\mathtt{G}$ are e_1 and e_3, which again are connected.* □



(a) Join tree of an acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (b) Join tree of the graph $(\mathcal{V}, \mathcal{E} \cup \{\texttt{ACEHF}\})$

Figure 1: Acyclicity and connexity

Fix a schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. A *join instance* (or simply, a *join*) of $\mathcal{G}$ is a set $\mathcal{Q}$ of relations such that (i) $|\mathcal{Q}| = |\mathcal{E}|$, and (ii) for each hyperedge $e \in \mathcal{E}$, there is a unique relation $R_e \in \mathcal{Q}$ with $schema(R_e) = e$. We call $\mathcal{G}$ the "schema graph of $\mathcal{Q}$". The *result* of the join is a relation over $\mathcal{V}$ defined as:

$$Join(\mathcal{Q}) \quad = \quad \{\text{tuple } \boldsymbol{u} \text{ over } \mathcal{V} \mid \forall R \in \mathcal{Q}, \boldsymbol{u}[schema(R)] \in R\}. \tag{1}$$

The *input size* of $\mathcal{Q}$ is defined as

$$\text{IN} = \sum_{R \in \mathcal{Q}} |R|.$$

If $\mathcal{G}$ is acyclic, we refer to $\mathcal{Q}$ as an *acyclic join*; otherwise, it is a *cyclic join*.

If the relations in $\mathcal{Q}$ can be listed as $R_1, R_2, ..., R_{|\mathcal{Q}|}$, we may also represent $Join(\mathcal{Q})$ as $R_1 \bowtie R_2 \bowtie ... \bowtie R_{|\mathcal{Q}|}$. If a relation, say, R_1 consists of a single tuple $\boldsymbol{u}$, we may replace it with $\boldsymbol{u}$ in a join expression, e.g., $\boldsymbol{u} \bowtie R_2 \bowtie ... \bowtie R_{|\mathcal{Q}|}$.

$(\mathcal{G}, \mathcal{Z})$-Sampling. Fix a schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose size is assumed to be a constant, together with a subset $\mathcal{Z} \subseteq \mathcal{V}$. Let $\mathcal{Q}$ be a join instance of $\mathcal{G}$. Given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, define

$$\sigma_{\boldsymbol{z}}(Join(\mathcal{Q})) \quad = \quad \{\boldsymbol{u} \in Join(\mathcal{Q}) \mid \forall Z \in \mathcal{Z}, \boldsymbol{u}(Z) = \boldsymbol{z}(Z)\} \tag{2}$$

namely, $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ comprises the tuples $\boldsymbol{u}$ in the join result satisfying the conjunctive condition $\bigwedge_{Z \in \mathcal{Z}} \boldsymbol{u}(Z) = \boldsymbol{z}(Z)$. Given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, a $(\mathcal{G}, \mathcal{Z})$*-sampling operation* returns a tuple chosen from $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ uniformly at random, or returns `nil` if $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q})) = \emptyset$.

> **Problem 1:** Preprocess $\mathcal{Q}$ into a data structure that can support $(\mathcal{G}, \mathcal{Z})$-sampling operations, with the requirement that the sample returned by each operation must be independent of the outputs of all previous operations.

We call the above the $(\mathcal{G}, \mathcal{Z})$*-sampling problem*. A data structure is feasible if it uses $\tilde{O}(\text{IN})$ space and supports a sampling operation in $\tilde{O}(1)$ time where $\tilde{O}(.)$ hides a factor polylogarithmic in IN.

$(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$-Sampling. Fix a schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose size is assumed to be a constant, as well as m subsets $\mathcal{Z}_1, \mathcal{Z}_2, ..., \mathcal{Z}_m$ of $\mathcal{V}$ for some $m \geq 1$. Let $\mathcal{Q}$ be a join instance of $\mathcal{G}$. Given m distinct tuples $\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_m$ where $\boldsymbol{z}_i$ is over $\mathcal{Z}_i$ for each $i \in [m]$, define

$$\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q})) \quad = \quad \bigcup_{i=1}^{m} \sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q})) \tag{3}$$

where $\sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}))$ is as defined in (2). In other words, $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ includes all and only the tuples $\boldsymbol{u}$ in the join result that satisfy a condition given in a disjunctive normal form (DNF): $\bigvee_{i=1}^{m} \left(\bigwedge_{X \in \mathcal{Z}_i} \boldsymbol{u}(X) = \boldsymbol{z}_i(X) \right)$. The value of m is permitted to be arbitrarily greater than $2^{|\mathcal{V}|}$, implying that $\mathcal{Z}_1, \mathcal{Z}_2, ..., \mathcal{Z}_m$ need not be distinct[1].

Given m such tuples $\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_m$, a $(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$*-sampling operation* returns a tuple chosen from $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ uniformly at random, or returns `nil` if $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q})) = \emptyset$.

> **Problem 2:** Preprocess $\mathcal{Q}$ into a data structure that can support $(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$-sampling operations, with the requirement that the sample returned by each operation must be independent of the outputs of all previous operations.

We call the above the $(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$*-sampling problem.*

1.2 Our Results

Problem 1 ($(\mathcal{G}, \mathcal{Z})$-Sampling). The following is a conjecture that has been extensively used in studying fine-grained complexities; see, e.g., [14, 18, 23, 24, 29, 30] and their references.

> **Strong Set Disjointness (SSD) Conjecture.** Let $S_1, S_2, ..., S_m$ be $m \geq 2$ sets whose elements are integers. Define $N = \sum_{i=1}^{m} |S_i|$. In the *set disjointness problem*, we want to preprocess these sets into a data structure of $\mathcal{S}$ space such that, given any distinct integers $a, b \in [m]$, we can report in $\mathcal{T} = o(N)$ time whether $S_a \cap S_b = \emptyset$. The SSD conjecture states that $\mathcal{S}$ must be $\Omega(N^2/(\mathcal{T}^2 \operatorname{polylog} N))$.

We investigate under the above conjecture the existence of feasible structures for Problem 1. It turns out that the answer depends on whether the schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of $\mathcal{Q}$ is "ext-$\mathcal{Z}$-connex", which is a notion introduced by Bagan, Durand and Grandjean [5]:

Definition 1.1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$. We say that $\mathcal{G}$ is* **ext-$\mathcal{Z}$-connex** *if the hypergraph $(\mathcal{V}, \mathcal{E} \cup \{\mathcal{Z}\})$ is acyclic.*

Example 1.2. *Consider again the schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in Example 1.1. The table below lists the answers to* "is $\mathcal{G}$ ext-$\mathcal{Z}$-connex?" *for several representative choices of $\mathcal{Z}$.*

$\mathcal{Z}$	$\emptyset$	*E*	*AE*	*ACE*	*ACEH*	*ACEHI*	*ACEHF*
ext-$\mathcal{Z}$-connex?	*yes*	*yes*	*no*	*yes*	*yes*	*no*	*yes*

Figure 1(b) shows a join tree of the hypergraph $(\mathcal{V}, \mathcal{E} \cup \{ACEHF\})$, which serves as evidence that $(\mathcal{V}, \mathcal{E} \cup \{ACEHF\})$ is acyclic; hence, $\mathcal{G}$ is ext-ACEHF-connex. □

Our dichotomy result is:

Theorem 1.1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an acyclic schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$. Subject to the SSD-conjecture, the $(\mathcal{G}, \mathcal{Z})$-sampling problem admits a feasible structure if and only if $\mathcal{G}$ is ext-$\mathcal{Z}$-connex.*

[1] For example: `SELECT * FROM Payment P, TaxPayer T WHERE P.ssn = T.ssn AND (T.job =` 'prof' `OR T.job =` 'lawyer'). Here, both $\mathcal{Z}_1$ and $\mathcal{Z}_2$ are {`job`}.

Prior to our work, ext-$\mathcal{Z}$-connexity has been used to prove dichotomies in several settings [5, 9–13, 21], all of which seem quite different from $(\mathcal{G}, \mathcal{Z})$-sampling. Interestingly, the dichotomy in Theorem 1.1 implies an approach to implement feasible $(\mathcal{G}, \mathcal{Z})$-sampling via "direct-access" queries. To explain, let L be an ordering of the attributes in $\mathcal{V}$: $X_1, X_2, ..., X_{|\mathcal{V}|}$. This L defines a lexicographic order $\prec$ on the tuples in $Join(\mathcal{Q})$: $\boldsymbol{u_1} \prec \boldsymbol{u_2}$ if there is an $i \in [|\mathcal{V}|]$ such that $\boldsymbol{u_1}(X_i) < \boldsymbol{u_2}(X_i)$ but $\boldsymbol{u_1}(X_j) = \boldsymbol{u_2}(X_j)$ for all $1 \le j \le i-1$. Given an integer t, a *direct access* (DA) query [6, 7, 12, 13] returns the t-th tuple of $Join(\mathcal{Q})$ under $\prec$ if $t \in [|Join(\mathcal{Q})|]$, or `nil` otherwise. As shown in [12], if $\mathcal{G}$ is ext-$\mathcal{Z}$-connex, we can find an order L such that

- the attributes of $\mathcal{Z}$ form a prefix of L, i.e., $X_i \in \mathcal{Z}$ for every $i \in [|\mathcal{Z}|]$;
- one can build a structure of $O(\text{IN})$ space that answers any DA query in $O(\log \text{IN})$ time.

Fix an arbitrary tuple $\boldsymbol{z}$ over $\mathcal{Z}$. Crucially, the tuples in $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ must be *consecutive* under $\prec$ (because $\mathcal{Z}$ is a prefix of L). Thus, if $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q})) \ne \emptyset$, there exist integers t_1 and t_2 such that a tuple is in $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ if and only if it is the t-th tuple under $\prec$ for some $t \in [t_1, t_2]$. With $O(\log \text{IN})$ DA queries, one can obtain the values of t_1 and t_2 (or certify their non-existence) via binary search. Then, a sample of $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ can be drawn by generating a random $t \in [t_1, t_2]$ and issuing one more DA query. This supports a $(\mathcal{G}, \mathcal{Z})$-sampling operation in $O(\log^2 \text{IN})$ time.

In this work, we show that the sampling time can be reduced to constant:

Theorem 1.2. *Consider the $(\mathcal{G}, \mathcal{Z})$-sampling problem where $\mathcal{G}$ is ext-$\mathcal{Z}$-connex. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN, *we can build a structure of $O(\text{IN})$ space in $O(\text{IN})$ expected time such that a $(\mathcal{G}, \mathcal{Z})$-sampling operation on $\mathcal{Q}$ can be supported in $O(1)$ time.*

Problem 2 ($(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$-Sampling). Our core contribution to this problem is a new algorithm for a fundamental sampling problem:

> **Set Union Sampling:** Let $S_1, S_2, ..., S_m$ be $m \ge 2$ sets of elements drawn from a certain domain. Each S_i ($i \in [m]$) supports three operations in constant time:
>
> 1. (*size*) return $|S_i|$;
> 2. (*sampling*) return an element of S_i chosen uniformly at random;
> 3. (*membership*) check whether a given element is in S_i.
>
> The goal is to sample an element from $\bigcup_{i=1}^{m} S_i$ uniformly at random using only these operations. The sample obtained each time must be independent of all previous samples.

We will prove:

Theorem 1.3. *There is a set union sampling algorithm with $O(m)$ expected sampling time.*

Currently the fastest expected sampling time [4, 13] — as will be reviewed in Section 2 — is $O(\min\{m^2, m \log^2 N\})$ where $N = \sum_{i=1}^{m} |S_i|$, which we strictly improve. The sampling time in Theorem 1.3 is optimal (in the expected sense) because any algorithm must figure out whether $S_i = \emptyset$ for each $i \in [m]$, thus incurring at least $\Omega(m)$ operations.

By combining Theorems 1.2 and 1.3 with additional ideas, we will prove:

Theorem 1.4. *Consider $(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$-sampling where $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for all $i \in [m]$. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN, *we can build a structure of $O(\text{IN})$ space in $O(\text{IN})$ expected time such that a $(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$-sampling operation on $\mathcal{Q}$ can be supported in $O(m)$ expected time.*

The $O(m)$ time of Theorem 1.4 is optimal (again, in the expected sense) for drawing a single sample, since $\Omega(m)$ time is compulsory just to read the m given tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$. The connexity requirement is necessary for $m \leq \mathrm{IN}^{0.49}$, as formally stated below:

Theorem 1.5. *Consider $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling where $m \leq \mathrm{IN}^{0.49}$. If $\mathcal{G}$ is not ext-$\mathcal{Z}_i$-connex for some $i \in [m]$, no structures of $\tilde{O}(\mathrm{IN})$ space can guarantee $\tilde{O}(m)$ expected sampling time, subject to the SSD conjecture.*

Each $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling operation in Theorem 1.4 returns one sample from $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$. If t samples are required, a naive approach is to perform the operation t times, necessitating $O(m \cdot t)$ expected time. It turns out that the sampling time can be significantly improved:

Theorem 1.6. *Consider $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling where $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for all $i \in [m]$. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN, *we can build a structure of $O(\mathrm{IN})$ space in $O(\mathrm{IN})$ expected time such that, given any m tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ (with $\boldsymbol{z}_i$ over $\mathcal{Z}_i$ for each $i \in [m]$) and any integer $t \geq 1$, we can draw t independent tuples from $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ uniformly at random in $O(t)$ expected time, after an initial processing of $O(m)$ expected time.*

An Application: Random Enumeration. *Random enumeration* of a query result produces a (uniformly) random permutation of the elements therein. An algorithm achieves a *delay* of Δ if it can (i) output the first element or declare an empty result within Δ time, and (ii) after the previous output, produce the next element or declare "no more" within an additional Δ time. The algorithm ensures an *expected* delay Δ if it satisfies the preceding requirement, except that the two "Δ time" occurrences are replaced with "Δ expected time". By combining our sampling techniques with a new enumeration-to-sampling reduction, we prove:

Theorem 1.7. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$ such that $\mathcal{G}$ is ext-$\mathcal{Z}$-connex. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN, *we can build a structure of $O(\mathrm{IN})$ space in $O(\mathrm{IN})$ expected time such that, given any tuple $\boldsymbol{z}$ over $\mathcal{Z}$, we can randomly enumerate $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ with an expected delay of $O(1)$.*

The above theorem applies to *conjunctive* conditions. We also obtain a slightly weaker result for DNF conditions:

Theorem 1.8. *Consider $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling where $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for all $i \in [m]$. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN, *we can build a structure of $O(\mathrm{IN})$ space in $O(\mathrm{IN})$ expected time such that, given any m tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ (with $\boldsymbol{z}_i$ over $\mathcal{Z}_i$ for each $i \in [m]$) and any real value $0 < \delta < 1$, we can, with probability at least $1 - \delta$, randomly enumerate $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ with an expected delay of $O(1)$ after an initial processing of $O(m + \log \frac{1}{\delta})$ expected time.*

Remark. A preliminary version of this work appeared in ICDT 2026. The current article includes two new results — Theorems 1.6 and 1.8 — which, together with the results in the conference paper, provide a systematic treatment of the proposed sampling techniques.

2 Related Work

At a high level, the objective of join sampling is to create a data structure on the input relations of a join $\mathcal{Q}$ that can be used to extract a uniformly random tuple from $Join(\mathcal{Q})$. The samples obtained from repetitive extractions must be mutually independent. If $\mathcal{Q}$ is acyclic, Zhao et al. [37] described an $O(\mathrm{IN})$-space structure ensuring $O(1)$ sampling time. The problem becomes more challenging when $\mathcal{Q}$ is cyclic. Improving over [16, 19], Kim et al. [28] presented a structure of $O(\mathrm{IN})$ space that can guarantee a sampling time of $O(AGM/\max\{1, \mathrm{OUT}\})$, where AGM is the join's *AGM bound* [3], and $\mathrm{OUT} = |Join(\mathcal{Q})|$. For a broad class of joins with "degree constraints", Wang and Tao [34] managed to reduce the sampling time to $O(polymat/\max\{1, \mathrm{OUT}\})$, where *polymat* is the *polymatroid bound* [27] of $\mathcal{Q}$, which never exceeds but can be significantly lower than the AGM bound. More recently, several alternative methods [8, 32] have been developed to the result of [34] up to polylogarithmic factors.

We are not aware of previous work on join sampling under conjunctive or disjunctive selection predicates. Like [37], our study concentrates on acyclic joins, but the sampling problem in [37] is merely a special case of $(\mathcal{G}, \mathcal{Z})$-sampling (see Problem 1) where $\mathcal{Z} = \emptyset$; the result of [37] can be regarded as a corollary of our Theorem 1.2.

The set union sampling problem in Section 1.2 has been studied in two independent articles [4, 13]. Both articles described an algorithm to achieve an expected sampling time of $O(m \cdot (N/n))$ where $n = |\bigcup_{i=1}^{m} S_i|$ and $N = \sum_{i=1}^{m} |S_i|$ (Section 5.1 will discuss this in detail). Note that N can reach mn — this happens when $S_1 = S_2 = ... = S_m$ — in which case the sampling time becomes $O(m^2)$. In [4], Aumuller et al. presented another algorithm to extract, in $O(m \log^2 N)$ expected time, a sample that is uniform "with high probability". Our Theorem 1.3 strictly improves these results.

Given an acyclic join $\mathcal{Q}$, Carmeli et al. [13] showed how to randomly enumerate $Join(\mathcal{Q})$ with a delay of $O(\log \mathrm{IN})$, after an $O(\mathrm{IN} \log \mathrm{IN})$-time preprocessing. In Theorem 1.7, we settle a more general problem (i.e., random enumeration of $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$) with an expected delay of $O(1)$. Given m acyclic joins $\mathcal{Q}_1, \mathcal{Q}_2, ..., \mathcal{Q}_m$ whose results have the same schema, Carmeli et al. [13] described a way to randomly enumerate $\bigcup_{i=1}^{m} Join(\mathcal{Q}_i)$ with an expected delay of $O(m^2 \log \mathrm{IN}_{all})$ after an $O(\mathrm{IN}_{all} \log \mathrm{IN}_{all})$-time preprocessing, where IN_{all} is the sum of the input sizes of $\mathcal{Q}_1, ..., \mathcal{Q}_m$. Our set union sampling algorithm in Section 5.1 can be used to reduce their expected delay to $O(m \log \mathrm{IN}_{all})$. Random enumeration of $\bigcup_{i=1}^{m} Join(\mathcal{Q}_i)$ for general (cyclic) joins $\mathcal{Q}_1, ..., \mathcal{Q}_m$ was considered in [19]; we do not delve into their results further because they are subsumed by those of [13] on acyclic joins (i.e., this work's focus).

Join reporting (rather than sampling) under selection has been investigated in [18, 36]. To explain their findings, fix a schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of a constant size (note: $\mathcal{G}$ can be cyclic), together with a non-empty subset $\mathcal{Z} \subseteq \mathcal{V}$. Consider the task of creating a data structure on a join instance $\mathcal{Q}$ of $\mathcal{G}$ such that, when supplied with a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, the structure can report $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ efficiently. The focus of [18, 36] is to study the relationships between the space consumption — denoted as $\mathcal{S}$ — of the structure and the time of computing $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$. Targeting "output-sensitive" algorithms that report $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ in $\mathcal{T} + O(|\mathrm{OUT}|)$ time where $\mathrm{OUT} = |\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))|$, Zhao et al. [36] derived smooth tradeoffs between $\mathcal{S}$ and $\mathcal{T}$. In Section 7, we complement their results by identifying scenarios where $\mathcal{S} = O(\mathrm{IN})$ and $\mathcal{T} = O(1)$ are possible, provided that the time of reporting $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ is allowed to be $\mathcal{T} + O(|\mathrm{OUT}|)$ *expected*. The work of [18], on the other hand, concerns a different form of tradeoffs that do not bear direct relevance to our results.

3 Preliminaries

This section reviews several fundamental results that will be needed in our technical discussion.

3.1 Weighted Sampling

Let S be a set of n elements, denoted as $e_1, e_2, ..., e_n$, where each e_i ($i \in [n]$) carries a positive integer *weight* $w(e_i)$. Set $W = \sum_{i=1}^{n} w(e_i)$. A *weighted sampling* operation returns a random element X

such that $\mathbf{Pr}[X = e_i] = w(e_i)/W$ for each $i \in [n]$. The *alias method* [33] supports such an operation in $O(1)$ time, after creating a data structure of $O(n)$ space over S in $O(n)$ time. We will refer to the structure as the *alias structure*.

3.2 Acyclic Join Sampling without Selections

This subsection gives a self-contained explanation on how to build a structure over an acyclic join $\mathcal{Q}$ that uses $O(\text{IN})$ space and can extract a uniformly random tuple from $Join(\mathcal{Q})$ in $O(1)$ time.

Take a join tree $\mathcal{T}$ of the schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of $\mathcal{Q}$. Root $\mathcal{T}$ at an arbitrary node (recall that each node of $\mathcal{T}$ is a hyperedge in $\mathcal{E}$) — doing so enables us to speak about the "parents" and "children" of the nodes in $\mathcal{T}$.

Subjoins. Given a node e in $\mathcal{T}$, we denote by $\mathcal{T}_e$ the subtree of $\mathcal{T}$ induced by all the descendants of e (note: e is a descendant of itself). Define

$$\mathcal{Q}_e = \{R_{e'} \mid \text{node } e' \text{ is in } \mathcal{T}_e\} \tag{4}$$

where $R_{e'}$ is the (only) relation in $\mathcal{Q}$ whose schema is e'. We will refer to $\mathcal{Q}_e$ as a *subjoin*. The schema graph of $\mathcal{Q}_e$ is $\mathcal{G}_e = (\mathcal{V}_e, \mathcal{E}_e)$ where $\mathcal{E}_e = \{e' \in \mathcal{E} \mid \text{node } e' \text{ is in } \mathcal{T}_e\}$ and $\mathcal{V}_e = \bigcup_{e' \in \mathcal{E}_e} e'$.

Subjoin Weights. Let e be an internal node of the rooted $\mathcal{T}$, and R_e be the relation in $\mathcal{Q}$ whose schema is e. Given a tuple $\boldsymbol{u} \in R_e$, let us examine $\boldsymbol{u} \bowtie Join(\mathcal{Q}_e)$, namely, the result tuples of the subjoin $\mathcal{Q}_e$ to which $\boldsymbol{u}$ "contributes". Define the *subjoin weight* of $\boldsymbol{u}$ as

$$\textit{subj-w}_e(\boldsymbol{u}) \;\; = \;\; \Big|\boldsymbol{u} \bowtie Join(\mathcal{Q}_e)\Big| \tag{5}$$

where $\mathcal{Q}_e$ is given in (4). The following lemma is implicit from the arguments in [13, 37]. In Appendix A, we present an explicit proof for completeness.

Lemma 3.1. *In $O(\text{IN})$ expected time, we can compute the subjoin weights of all the tuples in R_e for all $e \in \mathcal{E}$.*

Sampling. The next lemma is implicit in [37] and explicitly proved in Appendix A.

Lemma 3.2. *In $\tilde{O}(\text{IN})$ expected time, we can build a structure over every relation in $\mathcal{Q}$ such that, given any hyperedge $e \in \mathcal{E}$ and any tuple $\boldsymbol{u} \in R_e$, we can draw a uniformly random tuple from $\boldsymbol{u} \bowtie Join(\mathcal{Q}_e)$ in $O(1)$ time.*

It is now easy to design the overall structure for sampling from $Join(\mathcal{Q})$. Let e^* be the root of $\mathcal{T}$; note that $Join(\mathcal{Q}) = Join(\mathcal{Q}_{e^*})$. Notice that $Join(\mathcal{Q})$ can be decomposed into a collection of sets: $\{\boldsymbol{u} \bowtie Join(\mathcal{Q}) \mid \boldsymbol{u} \in R_{e^*}\}$. Random sampling from $Join(\mathcal{Q})$ can be performed in two steps.

- First, obtain a random tuple $\boldsymbol{X} \in R_{e^*}$ such that $\mathbf{Pr}[\boldsymbol{X} = \boldsymbol{u}] = |\boldsymbol{u} \bowtie Join(\mathcal{Q}_{e^*})|/|Join(\mathcal{Q})| = \textit{subj-w}_{e^*}(\boldsymbol{u})/|Join(\mathcal{Q})|$ for each $\boldsymbol{u} \in R_{e^*}$. As $|Join(\mathcal{Q})| = \sum_{\boldsymbol{u} \in R_{e^*}} |\boldsymbol{u} \bowtie Join(\mathcal{Q}_{e^*})|$, this is an instance of weighted sampling once the $\textit{subj-w}_{e^*}(\boldsymbol{u})$ value of every $\boldsymbol{u} \in R_{e^*}$ has been calculated from Lemma 3.1. An alias structure on R_{e^*} allows us to obtain $\boldsymbol{X}$ in $O(1)$ time.
- Second, apply Lemma 3.2 to draw a random tuple from $\boldsymbol{X} \bowtie Join(\mathcal{Q}_{e^*})$ in $O(1)$ time.

3.3 Properties of Ext-$\mathcal{Z}$-Connexity

In this subsection, we first introduce several concepts related to ext-$\mathcal{Z}$-connexity that will aid our exposition in later sections. Then, we review two key properties of ext-$\mathcal{Z}$-connexity essential for our technical development.

Definition 3.1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an acyclic schema graph, $\mathcal{Z}$ be a subset of $\mathcal{V}$, and $\mathcal{T}$ be a join tree of $\mathcal{G}$. We say that $\mathcal{G}$ is* $\mathcal{Z}$**-canonical** *if there is a hyperedge $e \in \mathcal{E}$ satisfying $\mathcal{Z} \subseteq e$.*

Definition 3.2. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an acyclic schema graph, $\mathcal{Z}$ be a subset of $\mathcal{V}$, and $\mathcal{T}$ be a join tree of $\mathcal{G}$. An edge $\{e_1, e_2\}$ of $\mathcal{T}$ is* $\mathcal{Z}$**-breakable** *if $e_1 \cap e_2 \subseteq \mathcal{Z}$ (note that e_1 and e_2 are hyperedges of $\mathcal{E}$; and $\{e_1, e_2\}$ represents an undirected edge in $\mathcal{T}$).*

Suppose that removing all the $\mathcal{Z}$-breakable edges of $\mathcal{T}$ partitions $\mathcal{T}$ into s subtrees $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_s$. For each $i \in [s]$, define a hypergraph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ with $\mathcal{E}_i = \{e \in \mathcal{E} \mid \text{node } e \text{ exists in } \mathcal{T}_i\}$ and $\mathcal{V}_i = \bigcup_{e \in \mathcal{E}_i} e$. Each $\mathcal{G}_i$ is a $(\mathcal{Z}, \mathcal{T})$**-component** *of $\mathcal{G}$.*

Given a schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we call two distinct vertices $X, Y \in \mathcal{V}$ *neighbors* if they appear together in at least one hyperedge. A *path* in $\mathcal{G}$ is a sequence of distinct vertices $X_1, X_2, ..., X_t$ such that X_i and X_{i+1} are neighbors for all $i \in [t-1]$.

Definition 3.3. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an acyclic schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$. A* $\mathcal{Z}$**-path** *is a path $Z_1, X_1, X_2, ..., X_\ell, Z_2$ (where $\ell \geq 1$) such that (i) Z_1 and Z_2 belong to $\mathcal{Z}$, but they are not neighbors; (ii) $X_i \notin \mathcal{Z}$ for every $i \in [\ell]$.*

Example 3.1. *Consider the schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in Figure 1(a). Among the choices of $\mathcal{Z}$ in Example 1.2, $\mathcal{G}$ is $\mathcal{Z}$-canonical only for $\mathcal{Z} = \emptyset$ and $\mathcal{Z} = \mathtt{E}$.*

Set $\mathcal{Z}$ to $\mathtt{ACEHF}$. The edge $\{\mathtt{ABC}, \mathtt{CDE}\}$ of $\mathcal{T}$ is $\mathcal{Z}$-breakable because $\mathtt{ABC} \cap \mathtt{CDE} = \mathtt{C}$ is a subset of $\mathcal{Z}$. On the other hand, the edge $\{\mathtt{EGH}, \mathtt{GI}\}$ is not $\mathcal{Z}$-breakable because $\mathtt{EGH} \cap \mathtt{GI} = \mathtt{G}$ is not a subset of $\mathcal{Z}$. By removing the $\mathcal{Z}$-breakable edges, we obtain four subtrees of $\mathcal{T}$: the first contains nodes e_1 and e_3, while every other node (i.e., e_2, e_4, and e_5) forms a subtree by itself. Accordingly, $\mathcal{G}$ has four $(\mathcal{Z}, \mathcal{T})$-components: $\mathcal{G}_1 = (\mathtt{EGHI}, \{\mathtt{EGH}, \mathtt{GI}\})$, $\mathcal{G}_2 = (\mathtt{ABC}, \{\mathtt{ABC}\})$, $\mathcal{G}_3 = (\mathtt{CDE}, \{\mathtt{CDE}\})$, and $\mathcal{G}_4 = (\mathtt{EFJ}, \{\mathtt{EFJ}\})$. There are no $\mathcal{Z}$-paths in $\mathcal{G}$.

Set $\mathcal{Z}$ instead to $\mathtt{ACEHI}$. The path $\mathtt{E}$, $\mathtt{G}$, $\mathtt{I}$ is a $\mathcal{Z}$-path. □

Lemma 3.3 ([5, Lemma 23])**.** *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an acyclic schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$. The following statements are equivalent:*

1. *$\mathcal{G}$ is ext-$\mathcal{Z}$-connex.*
2. *$\mathcal{G}$ does not have a $\mathcal{Z}$-path.*
3. *Let $\mathcal{T}$ be an arbitrary join tree of $\mathcal{G}$. Denote by s the number of $(\mathcal{Z}, \mathcal{T})$-components of $\mathcal{G}$, and by $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ the i-th component. For every $i \in [s]$, $\mathcal{G}_i$ is $(\mathcal{V}_i \cap \mathcal{Z})$-canonical.*

Example 3.2. *Continuing Example 3.1, for $\mathcal{Z} = \mathtt{ACEHF}$, as mentioned $\mathcal{G}$ has $(\mathcal{Z}, \mathcal{T})$-components: $\mathcal{G}_1 = (\mathtt{ABC}, \{\mathtt{ABC}\})$, $\mathcal{G}_2 = (\mathtt{CDE}, \{\mathtt{CDE}\})$, $\mathcal{G}_3 = (\mathtt{EFJ}, \{\mathtt{EFJ}\})$, and $\mathcal{G}_4 = (\mathtt{EGHI}, \{\mathtt{EGH}, \mathtt{GI}\})$. For every $1 \leq i \leq 4$, $\mathcal{G}_i$ is $(\mathcal{V}_i \cap \mathcal{Z})$-canonical; hence, $\mathcal{G}$ is ext-$\mathcal{Z}$-connex. However, for $\mathcal{Z} = \mathtt{ACEHI}$, as mentioned there is a $\mathcal{Z}$-path in $\mathcal{G}$, which is thus not ext-$\mathcal{Z}$-connex.* □

4 A Dichotomy on $(\mathcal{G}, \mathcal{Z})$-Sampling

Theorem 1.1 involves a negative result (the "only-if direction") and a positive result (the "if direction"). We will prove the negative result in Section 4.1. The positive result is a corollary of Theorem 1.2, whose proof is presented in Section 4.2.

4.1 The Only-If Direction of Theorem 1.1

As explained in Section 1.2, the input to the set disjointness problem comprises $m \geq 2$ sets $S_1, ..., S_m$. Given distinct integers $a, b \in [m]$, a *disjointness query* returns whether $S_a \cap S_b = \emptyset$. The SSD-conjecture states that any structure promising to answer such a query in $O(\text{polylog}\, N)$ time must use $\Omega(N^2/\text{polylog}\, N)$ space, where $N = \sum_{i=1}^{m} |S_i|$.

Suppose that the only-if claim of Theorem 1.1 is false. Thus, there exist a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a subset $\mathcal{Z} \subseteq \mathcal{V}$ such that $\mathcal{G}$ is not ext-$\mathcal{Z}$-connex but a feasible $(\mathcal{G}, \mathcal{Z})$-sampling structure exists. We will show how to build a set-disjointness structure of $\tilde{O}(N)$ space that answers a disjointness query in $\tilde{O}(1)$ time, thus breaking the SSD-conjecture. Since $\mathcal{G}$ is not ext-$\mathcal{Z}$-connex, it must have a $\mathcal{Z}$-path by Lemma 3.3. Pick an arbitrary $\mathcal{Z}$-path: $A, X_1, X_2, ..., X_\ell, B$ for some $\ell \geq 1$. Remember that no vertex on the path belongs to $\mathcal{Z}$ except A and B.

From $S_1, ..., S_m$ (the input to set disjointness), next we create a join $\mathcal{Q}$ whose schema graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For each hyperedge $e \in \mathcal{E}$, construct a relation R_e with schema e as follows. For each integer $i \in [m]$ and every element $x \in S_i$, insert a tuple $\boldsymbol{u}$ into R_e (provided that $\boldsymbol{u}$ is not already in R_e) such that

- if $A \in e$, then $\boldsymbol{u}(A) = i$;
- if $B \in e$, then $\boldsymbol{u}(B) = i$;
- for every attribute $Z \in (e \cap \mathcal{Z}) \setminus \{A, B\}$, set $\boldsymbol{u}(Z) = \perp$ where $\perp$ is a special symbol;
- for every attribute $X \in e \setminus \mathcal{Z}$, set $\boldsymbol{u}(X) = x$.

By Definition 3.3, no hyperedge of $\mathcal{E}$ covers both A and B; hence, at most one of the first two bullets applies. The relation R_e thus designed has a size at most N.

Lemma 4.1. *Given distinct $a, b \in [m]$, define a tuple $\boldsymbol{z}$ over $\mathcal{Z}$ with $\boldsymbol{z}(A) = a$, $\boldsymbol{z}(B) = b$, and $\boldsymbol{z}(Z) = \perp$ for every $Z \in \mathcal{Z} \setminus \{A, B\}$. Then, $S_a \cap S_b \neq \emptyset$ if and only if $\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q})) \neq \emptyset$.*

Proof. Let us first prove the "if direction" ($\Leftarrow$). Consider an arbitrary tuple $\boldsymbol{u} \in \sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))$. Clearly, $\boldsymbol{u}(A) = a$ and $\boldsymbol{u}(B) = b$. Since $A, X_1, X_2, ..., X_\ell, B$ is a $\mathcal{Z}$-path, there exist hyperedges $e_0, e_1, ..., e_\ell$ in $\mathcal{E}$ such that (i) $\{A, X_1\} \subseteq e_0$, (ii) $\{X_i, X_{i+1}\} \subseteq e_i$ for each $i \in [\ell - 1]$, and (iii) $\{X_\ell, B\} \subseteq e_\ell$. We argue:

$$\boldsymbol{u}(X_1) = \boldsymbol{u}(X_2) = ... = \boldsymbol{u}(X_\ell). \tag{6}$$

To see why, fix any $i \in [\ell - 1]$. As $\boldsymbol{u} \in \sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))$, we know $\boldsymbol{u}[e_i] \in R_{e_i}$. As X_i and X_{i+1} are attributes outside $\mathcal{Z}$, our construction of R_{e_i} ensures $\boldsymbol{u}(X_i) = \boldsymbol{u}(X_{i+1})$. The fact that this holds for every $i \in [\ell - 1]$ proves (6). By the construction of $\mathcal{Q}$, $\boldsymbol{u}(X_1)$ is an element of S_a and $\boldsymbol{u}(X_\ell)$ is an element of S_b. Therefore, (6) tells us $S_a \cap S_b \neq \emptyset$.

Next, we prove the "only if direction" ($\Rightarrow$). Fix any $x \in S_a \cap S_b$. Construct a tuple $\boldsymbol{u}$ over $\mathcal{V}$ where $\boldsymbol{u}(A) = a, \boldsymbol{u}(B) = b, \boldsymbol{u}(Z) = \perp$ for every $Z \in \mathcal{Z} \setminus \{A, B\}$, and $\boldsymbol{u}(X) = x$ for every $X \in \mathcal{V} \setminus \mathcal{Z}$. It suffices to prove that $\boldsymbol{u} \in \sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))$. Indeed, this is true because, for every $e \in \mathcal{E}$, our construction explicitly inserts tuple $\boldsymbol{u}[e]$ into R_e. □

The above lemma can be used to break the SSD-conjecture. Note that the input size of $\mathcal{Q}$ satisfies $\text{IN} \leq N \cdot |\mathcal{E}| = O(N)$, thus allowing us to build a structure of $\tilde{O}(\text{IN}) = \tilde{O}(N)$ space on $\mathcal{Q}$ to support a $(\mathcal{G}, \mathcal{Z})$-sampling operation in $\tilde{O}(1)$ time. A disjointness query parameterized by integers $a, b \in [m]$ can be answered as follows. First, create a tuple $\boldsymbol{z}$ over $\mathcal{Z}$ such that $\boldsymbol{z}(A) = a$,

$\boldsymbol{z}(B) = b$, and $\boldsymbol{z}(Z) = \perp$ for every $Z \in \mathcal{Z} \setminus \{A, B\}$. Then, issue a $(\mathcal{G}, \mathcal{Z})$-sampling operation and declare $S_a \cap S_b = \emptyset$ if and only if the operation returns nothing. This correctly answers the query in $\tilde{O}(1)$ time.

4.2 Proof of Theorem 1.2 (the If-Direction of Theorem 1.1)

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$, we want to build a structure of $O(\text{IN})$ space in $O(\text{IN})$ expected time to support a $(\mathcal{G}, \mathcal{Z})$-sampling operation on $\mathcal{Q}$ in constant time.

4.2.1 Case 1: $\mathcal{G}$ Is $\mathcal{Z}$-Canonical

By Definition 3.1, in this case there is a hyperedge $e^* \in \mathcal{E}$ satisfying $\mathcal{Z} \subseteq e^*$. Root $\mathcal{T}$ at e^*, apply Lemma 3.1 to calculate the subjoin weights of all tuples in R_{e^*} (the relation in $\mathcal{Q}$ with schema e^*), and apply Lemma 3.2 to build a sampling structure for each relation in $\mathcal{Q}$.

Given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, define

$$R_{e^*}(\boldsymbol{z}) \quad = \quad \{\boldsymbol{u} \in R_{e^*} \mid \boldsymbol{u}[\mathcal{Z}] = \boldsymbol{z}[\mathcal{Z}]\}. \tag{7}$$

It is easy to verify

$$\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q})) \quad = \quad \bigcup_{\boldsymbol{u} \in R_{e^*}(\boldsymbol{z})} \boldsymbol{u} \bowtie \mathit{Join}(\mathcal{Q}). \tag{8}$$

Random sampling from $\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))$ can be done in a two-step approach similar to what was described at the end of Section 3.2. First, obtain a random tuple $\boldsymbol{X} \in R_{e^*}(\boldsymbol{z})$ such that, for each $\boldsymbol{u} \in R_{e^*}(\boldsymbol{z})$, we have

$$\mathbf{Pr}[\boldsymbol{X} = \boldsymbol{u}] = \frac{|\boldsymbol{u} \bowtie \mathit{Join}(\mathcal{Q}_{e^*})|}{|\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))|} = \frac{\mathit{subj}\text{-}w_{e^*}(\boldsymbol{u})}{|\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))|}$$

where $\mathit{subj}\text{-}w_{e^*}(\boldsymbol{u})$ is the subjoin weight of $\boldsymbol{u}$ (with respect to the rooted $\mathcal{T}$); see (5). As $|\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))| = \sum_{\boldsymbol{u} \in R_{e^*}(\boldsymbol{z})} |\boldsymbol{u} \bowtie \mathit{Join}(\mathcal{Q}_{e^*})| = \sum_{\boldsymbol{u} \in R_{e^*}(\boldsymbol{z})} \mathit{subj}\text{-}w_{e^*}(\boldsymbol{u})$, this is an instance of weighted sampling since the $\mathit{subj}\text{-}w_{e^*}(\boldsymbol{u})$ value of every $\boldsymbol{u} \in R_{e^*}(\boldsymbol{z})$ is already available. Hence, an alias structure on $R_{e^*}(\boldsymbol{z})$ allows us to obtain $\boldsymbol{X}$ in constant time. Second, apply Lemma 3.2 to draw a uniformly random tuple from $\boldsymbol{X} \bowtie \mathit{Join}(\mathcal{Q}_{e^*})$ in constant time.

The alias structure on $R_{e^*}(\boldsymbol{z})$ occupies $O(|R_{e^*}(\boldsymbol{z})|)$ space and can be built in $O(|R_{e^*}(\boldsymbol{z})|)$ time, as explained in Section 3.1. We do this for every $\boldsymbol{z} \in \Pi_{\mathcal{Z}}(R_{e^*})$, where Π is projection in relational algebra. For distinct $\boldsymbol{z}, \boldsymbol{z}' \in \Pi_{\mathcal{Z}}(R_{e^*})$, the sets $R_{e^*}(\boldsymbol{z})$ and $R_{e^*}(\boldsymbol{z}')$ are disjoint. Hence, in total, all the alias structures occupy $O(|R_{e^*}|) = O(\text{IN})$ space and can be built in $O(\text{IN})$ expected time (the time is expected because hashing is required to obtain the sets $R_{e^*}(\boldsymbol{z})$ of each $\boldsymbol{z} \in \Pi_{\mathcal{Z}}(R_{e^*})$).

Remark: Finding the Size $|\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))|$. We make an observation here that will be useful later. For each $\boldsymbol{z} \in \Pi_{\mathcal{Z}}(R_{e^*})$, as mentioned $|\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))| = \sum_{\boldsymbol{u} \in R_{e^*}(\boldsymbol{z})} \mathit{subj}\text{-}w_{e^*}(\boldsymbol{u})$. Since the subjoin weights of all tuples in R_{e^*} are available, we can compute the sizes $|\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))|$ of all $\boldsymbol{z} \in \Pi_{\mathcal{Z}}(R_{e^*})$ by scanning R_{e^*} once in $O(\text{IN})$ time. Storing all those sizes takes $O(|R_{e^*}|) = O(\text{IN})$ extra space. In $O(|R_{e^*}|)$ expected time, we can build a perfect hashing structure [17] that, given any tuple $\boldsymbol{z}$ over $\mathcal{Z}$, returns $|\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))|$ in $O(1)$ time.

4.2.2 Case 2: $\mathcal{G}$ Is Not $\mathcal{Z}$-Canonical

Let $\mathcal{T}$ be an arbitrary join tree of $\mathcal{G}$. As $\mathcal{G}$ is ext-$\mathcal{Z}$-connex, by Lemma 3.3, $\mathcal{T}$ defines a number $s \geq 1$ of $(\mathcal{Z}, \mathcal{T})$-components of $\mathcal{G}$ — denoted as $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), ..., \mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$, respectively — such that $\mathcal{G}_i$ is $(\mathcal{V}_i \cap \mathcal{Z})$-canonical for every $i \in [s]$. For each $i \in [s]$, define $\mathcal{Z}_i = \mathcal{V}_i \cap \mathcal{Z}$ and $\mathcal{Q}_i = \{R_e \in \mathcal{Q} \mid e \in \mathcal{E}_i\}$. Note that $\mathcal{Q}_i$ is a join instance of the schema graph $\mathcal{G}_i$, which is $\mathcal{Z}_i$-canonical. Denote by IN_i the input size of $\mathcal{Q}_i$; we have $\sum_{i=1}^{s} \mathrm{IN}_i = \mathrm{IN}$ because every relation of $\mathcal{Q}$ appears in exactly one of $\mathcal{Q}_1, ..., \mathcal{Q}_s$. For each $i \in [s]$, we build a $(\mathcal{G}_i, \mathcal{Z}_i)$-sampling structure on $\mathcal{Q}_i$ in the way explained for Case 1. All the s structures occupy $O(\mathrm{IN})$ space in total and can be built in $O(\mathrm{IN})$ expected time.

Consider now a $(\mathcal{G}, \mathcal{Z})$-sampling operation, which is given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$. For each $i \in [s]$, defining $\boldsymbol{z}_i = \boldsymbol{z}[\mathcal{Z}_i]$, we perform a $(\mathcal{G}_i, \mathcal{Z}_i)$-sampling operation using $\boldsymbol{z}_i$; let us assume that this operation returns $\boldsymbol{u}_i$. If $\boldsymbol{u}_i$ is `nil` for any i, we return `nil` for the original $(\mathcal{G}, \mathcal{Z})$-sampling operation. Otherwise, return $\boldsymbol{u}_1 \bowtie \boldsymbol{u}_2 \bowtie ... \bowtie \boldsymbol{u}_s$. The whole operation takes $O(s) = O(1)$ time overall.

Next, we prove the correctness of our algorithm (for Case 2). Consider the following procedure:

component-product
1. $S_\times \leftarrow \emptyset$
2. $S_i \leftarrow \sigma_{\boldsymbol{z}_i}(\mathit{Join}(\mathcal{Q}_i))$ for each $i \in [s]$ /* recall that $\boldsymbol{z}_i = \boldsymbol{z}[\mathcal{Z}_i]$ */
3. **for** each $(\boldsymbol{v}_1, ..., \boldsymbol{v}_s) \in S_1 \times ... \times S_s$ **do**
4.   add $\boldsymbol{v}_1 \bowtie \boldsymbol{v}_2 \bowtie ... \bowtie \boldsymbol{v}_s$ to $S_\times$

Lemma 4.2. *Both statements below are true about* **component-product***:*

1. *Line 4 always adds a new tuple to $S_\times$;*
2. *$\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q})) = S_\times$.*

Proof. Let us first prove Statement 1. For each $i \in [s]$, take an arbitrary tuple $\boldsymbol{v}_i \in S_i$ (note: S_i is defined at Line 2 of **component-product**). We will first show $\boldsymbol{v}_1 \bowtie \boldsymbol{v}_2 \bowtie ... \bowtie \boldsymbol{v}_s \neq \emptyset$. For this purpose, it suffices to prove that, for any distinct $i, j \in [s]$ with $\mathcal{V}_i \cap \mathcal{V}_j \neq \emptyset$, it must hold that $\boldsymbol{v}_i(Y) = \boldsymbol{v}_j(Y)$ for any attribute $Y \in \mathcal{V}_i \cap \mathcal{V}_j$.

As $Y \in \mathcal{V}_i$ (resp., $Y \in \mathcal{V}_j$), there is a hyperedge $e_i \in \mathcal{E}_i$ (resp., $e_j \in \mathcal{E}_j$) containing Y. As e_i and e_j are in different $(\mathcal{Z}, \mathcal{T})$-components — recall that $\mathcal{T}$ is a join tree of $\mathcal{Q}$ — the (unique) simple path between them on $\mathcal{T}$ must cross at least one $\mathcal{Z}$-breakable edge, denoted as $\{e_\circ, e_\bullet\}$. The connectedness property of $\mathcal{T}$ ensures that $Y \in e_\circ \cap e_\bullet$. By the definition of $\mathcal{Z}$-breakable edge, Y must be an attribute in $\mathcal{Z}$, indicating that the tuple $\boldsymbol{z}$ has a Y-value (recall that $\boldsymbol{z}$ is the input to a $(\mathcal{G}, \mathcal{Z})$-sampling operation). As $\boldsymbol{v}_i \in S_i$ and $\boldsymbol{v}_j \in S_j$, we must have $\boldsymbol{v}_i(Y) = \boldsymbol{z}(Y) = \boldsymbol{v}_j(Y)$.

To prove Statement 1, it remains to show that Line 4 never adds to $S_\times$ the same tuple twice. Consider any two executions of Line 4: the first with $\boldsymbol{v}_i = \boldsymbol{v}_i^1$ for $i \in [s]$, and the second with $\boldsymbol{v}_i = \boldsymbol{v}_i^2$ for $i \in [s]$. There exists at least one $j \in [s]$ such that $\boldsymbol{v}_j^1 \neq \boldsymbol{v}_j^2$. This further indicates the existence of an attribute $Y \in \mathcal{V}_j$ such that $\boldsymbol{v}_j^1(Y) \neq \boldsymbol{v}_j^2(Y)$. We can now assert that $\boldsymbol{v}_1^1 \bowtie ... \bowtie \boldsymbol{v}_s^1$ and $\boldsymbol{v}_1^2 \bowtie ... \bowtie \boldsymbol{v}_s^2$ must differ in their Y-values.

Next, we prove Statement 2. We first show

$$\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q})) \subseteq S_\times. \tag{9}$$

Take any tuple $\boldsymbol{v} \in \sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))$. Define $\boldsymbol{v}_i = \boldsymbol{v}[\mathcal{V}_i]$ for each $i \in [s]$. To show (9), it suffices to prove $\boldsymbol{v}_i \in \sigma_{\boldsymbol{z}_i}(\mathit{Join}(\mathcal{Q}_i))$ for every $i \in [s]$. We achieve the purpose by arguing that $\boldsymbol{v_i}[e] \in R_e$ for any $e \in \mathcal{E}_i$. As every relation of $\mathcal{Q}_i$ belongs to $\mathcal{Q}$, we know $R_e \in \mathcal{Q}$. We can thus infer $\boldsymbol{v}[e] \in R_e$ from $\boldsymbol{v} \in \sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))$. On the other hand, $\boldsymbol{v}[e] = \boldsymbol{v}_i[e]$ because $e \subseteq \mathcal{V}_i$. It thus follows that $\boldsymbol{v}_i[e] \in R_e$.

It remains to show

$$S_\times \subseteq \sigma_{\boldsymbol{z}}(Join(\mathcal{Q})). \tag{10}$$

For each $i \in [s]$, take an arbitrary tuple $\boldsymbol{v}_i \in S_i$. Define $\boldsymbol{w} = \boldsymbol{v}_1 \bowtie ... \bowtie \boldsymbol{v}_s$. Our proof of Statement 1 has explained that $\boldsymbol{w}$ cannot be empty. To prove (10), our goal is to show that $\boldsymbol{w} \in \sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$. We first argue that $\boldsymbol{w} \in Join(\mathcal{Q})$, i.e., $\boldsymbol{w}[e] \in R_e$ for every $e \in \mathcal{E}$. Indeed, as e must belong to $\mathcal{E}_i$ for some $i \in [s]$, we know $\boldsymbol{w}[e] = \boldsymbol{v}_i[e]$, which is in R_e because $\boldsymbol{v}_i \in \sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}_i))$. To prove $\boldsymbol{w} \in \sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$, we still have to show $\boldsymbol{w}(Z) = \boldsymbol{z}(Z)$ for every $Z \in \mathcal{Z}$. For this purpose, simply identify any $i \in [s]$ satisfying $Z \in \mathcal{V}_i$ (this i exists because $\bigcup_{i=1}^s \mathcal{V}_i = \mathcal{V}$). That $\boldsymbol{w}(Z) = \boldsymbol{z}(Z)$ follows from the fact $\boldsymbol{v}_i \in \sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}_i))$. $\square$

The above lemma shows that there is a one-one correspondence between the tuples of $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ and $S_\times$. It is easy to verify that our sampling algorithm (for Case 2) draws a tuple from $S_\times$ uniformly at random and, hence, also a tuple from $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ uniformly at random.

Remark: Finding the Size $|\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))|$. We make an observation that will be useful in Section 5.2: it is possible to build a structure of $O(\text{IN})$ space in $O(\text{IN})$ expected time that, given any tuple $\boldsymbol{z}$ over $\mathcal{Z}$, allows us to find $|\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))|$ in constant time. In (a remark in) Section 4.2.1, we have explained how to do so when $\mathcal{G}$ is $\mathcal{Z}$-canonical.

When $\mathcal{G}$ is not $\mathcal{Z}$-canonical, consider the $(\mathcal{Z}, \mathcal{T})$-components of $\mathcal{G}$ obtained earlier, i.e., $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), ..., \mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$. As before, for each $i \in [s]$, define $\mathcal{Z}_i = \mathcal{V}_i \cap \mathcal{Z}$ and $\mathcal{Q}_i = \{R_e \in \mathcal{Q} \mid e \in \mathcal{E}_i\}$; remember that $\mathcal{Q}_i$ is a join instance of $\mathcal{G}_i$, which is $\mathcal{Z}_i$-canonical. For each $i \in [s]$, apply the remark in Section 4.2.1 to build a structure Υ_i that, upon given any tuple $\boldsymbol{z}_i$ over $\mathcal{Z}_i$, returns $|\sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}_i))|$ in constant time. If IN_i is the input size of $\mathcal{Q}_i$, the structure occupies $O(\text{IN}_i)$ space and can be built in $O(\text{IN}_i)$ expected time. Thus, all the s structures together occupy $\sum_{i=1}^s O(\text{IN}_i) = O(\text{IN})$ space; and their construction takes $O(\text{IN})$ expected time in total.

Given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, we obtain $|\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))|$ as follows. For each $i \in [s]$, define $\boldsymbol{z}_i = \boldsymbol{z}[\mathcal{Z}_i]$ and use Υ_i to obtain the size $|\sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}_i))|$ in $O(1)$ time. Then, we return $|\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))| = \prod_{i=1}^s |\sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}_i))|$, whose correctness follows from Lemma 4.2.

The lemma below summarizes the above discussion.

Lemma 4.3. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$ such that $\mathcal{G}$ is ext-$\mathcal{Z}$-connex. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN*, we can build a structure of $O(\text{IN})$ space in $O(\text{IN})$ expected time such that, given any tuple $\boldsymbol{z}$ over $\mathcal{Z}$, we can report $|\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))|$ in $O(1)$ time.*

5 $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-Sampling

The crux of our solution to Problem 2 is an algorithm optimally settling the set union sampling problem (see Section 1.2). We will present this algorithm in Section 5.1 and explain in Section 5.2 how it leads to a structure for Problem 2 that establishes Theorem 1.4. Finally, Section 5.3 gives the proof of Theorem 1.5.

5.1 Set Union Sampling

Recall from Section 1.2 that, in this problem, we have a collection of $m \geq 2$ sets $S_1, S_2, ..., S_m$, each supporting three $O(1)$-time operations: *size*, *membership*, and *sample*. The objective is to draw an element from $\bigcup_{i=1}^m S_i$ uniformly at random.

As before, set $n = |\bigcup_{i=1}^m S_i|$ and $N = \sum_{i=1}^m |S_i|$. For each element $y \in \bigcup_{i=1}^m S_i$, define its *inverted set* as $\mathrm{InvS}(y) = \{i \in [m] \mid y \in S_i\}$, i.e., the "ids" of the sets containing y. The *degree* of y is $\deg(y) = |\mathrm{InvS}(y)|$. Let us first describe a baseline method, which was given explicitly in [4] and implicitly in [13], before presenting our new ideas.

Baseline. Draw a random value $X \in [m]$ such that $\mathbf{Pr}[X = i] = |S_i|/N$ for each $i \in [m]$. Then, use the sampling operation to draw an element Y from S_X. Finally, carry out an *acceptance step*:

Acceptance Step: Accept Y with probability $1/\deg(Y)$.

If accepted, Y is returned; otherwise, the algorithm repeats from scratch.

The algorithm correctly returns a uniform sample of $\bigcup_{i=1}^m S_i$. To see why, fix an arbitrary element $y \in \bigcup_{i=1}^m S_i$. This element is returned if and only if three conditions are satisfied: (i) $X \in \mathrm{InvS}(y)$, which occurs with probability $|S_X|/N$ for each $X \in \mathrm{InvS}(y)$, (ii) $Y = y$, which occurs with probability $1/|S_X|$ conditioned on X, and (iii) y is accepted, which occurs with probability $1/\deg(y)$ conditioned on y. Hence, the algorithm outputs y with probability

$$\sum_{X \in \mathrm{InvS}(y)} \frac{|S_X|}{N} \cdot \frac{1}{|S_X|} \cdot \frac{1}{\deg(y)} = \frac{1}{N} \tag{11}$$

which is identical for all $y \in \bigcup_{i=1}^m S_i$. As a corollary, in each repeat, the algorithm *succeeds* in returning a sample with probability n/N. Thus, N/n repeats are needed in expectation.

The random value X can be easily obtained in $O(m)$ time. One (logically simple) way to do so is to build an alias structure (see Section 3.2) on $|S_1|, |S_2|, ..., |S_m|$ — namely, the m set sizes — on the fly in $O(m)$ time (using the size operation), after which X can be extracted from the structure in $O(1)$ time. The time to obtain Y is $O(1)$ (using the sampling operation). The troublemaker, however, is the acceptance step. The standard approach [4, 13] is to query the membership of Y in each S_i ($i \in [m]$), which costs $O(m)$ time. This renders the overall sampling time $O(mN/n)$ in expectation.

New Idea: Law of Total Expectation. Our objective is to implement the acceptance step in $O(mn/N)$ expected time — note that this is faster than $O(m)$ by a factor of N/n, which eventually allows us to bring the expected sampling time from $O(mN/n)$ down to $O(m)$.

Let us start with a fundamental fact:

Proposition 5.1. *Let Γ be a random variable taking values from $[m]$, and U be a uniformly random variable over $[m]$. If U and Γ are independent, then $\mathbf{Pr}[U \le \Gamma] = \frac{1}{m}\,\mathbf{E}[\Gamma]$.*

Proof. For any random variables X and Y, the law of total expectation states $\mathbf{E}[X] = \mathbf{E}[\mathbf{E}[X \mid Y]]$.

To prove the proposition, define X to be 1 if $U \le \Gamma$ or 0 otherwise; furthermore, set $Y = \Gamma$. Clearly, $\mathbf{E}[X \mid Y] = \mathbf{Pr}[U \le \Gamma \mid \Gamma] = \Gamma/m$ by the independence of U and Γ. We can now derive $\mathbf{Pr}[U \le \Gamma] = \mathbf{E}[X] = \mathbf{E}[\mathbf{E}[X \mid Y]] = \mathbf{E}[\Gamma/m] = \frac{1}{m}\,\mathbf{E}[\Gamma]$. □

Equipped with the above, implementing the acceptance step boils down to:

Given $x \in [m]$ and $y \in S_x$, generate a random variable $\Gamma \in [m]$ with $\mathbf{E}[\Gamma] = m/\deg(y)$.

Later, we will explain how to achieve the above purpose in $O(m/\deg(y))$ expected time. Once done, we can perform the acceptance step as follows (recall that, prior to this, the baseline method has obtained the values of two random variables X and Y):

- generate a uniformly random variable U over $[m]$ in constant time;
- generate the aforementioned random variable Γ (setting x and y to the values of X and Y, respectively) in $O(m/\deg(Y))$ expected time;
- accept if $U \leq \Gamma$.

Correctness follows from Proposition 5.1 (the acceptance probability is $\frac{1}{m}\mathbf{E}[\Gamma] = 1/\deg(Y)$, as desired). For a particular $x \in [m]$ and a particular $y \in S_x$, we have $\mathbf{Pr}[X = x, Y = y] = \frac{|S_x|}{N}\frac{1}{|S_x|} = 1/N$. Thus, the expected cost of the acceptance step is at the order of

$$\begin{aligned}\sum_{x\in[m]}\sum_{y\in S_x}\mathbf{Pr}[X = x, Y = y]\frac{m}{\deg(y)} &= \sum_{x\in[m]}\sum_{y\in S_x}\frac{1}{N}\frac{m}{\deg(y)} \\ &= \frac{m}{N}\sum_{y\in\cup_{i=1}^m S_i}\;\sum_{x\in\mathrm{InvS}(y)}\frac{1}{\deg(y)} \\ &= \frac{mn}{N} \end{aligned} \tag{12}$$

where the last step used $\deg(y) = |\mathrm{InvS}(y)|$ and $n = |\bigcup_{i=1}^m S_i|$.

Generation of Γ. Next, we will concentrate on the Γ-generating task defined earlier. Recall that the generation is based on a set "id" $x \in [m]$ and an element $y \in S_x$. Consider the procedure below:

find-2nd (x, y)
1. $F \leftarrow 0$ and $I \leftarrow [m] \setminus \{x\}$
2. **while** $I \neq \emptyset$ **do**
3. sample without replacement (WoR) an integer $i \in I$
   /* note: the WoR-sampling operation removes i from I */
4. **if** $y \in S_i$ **then return** F **else** increase F by 1
5. **return** F /* note: F must be $m - 1$ here */

In plain words, if $\deg(y) \geq 2$, the value F returned is a random variable giving the number of *failed* WoR-sampling operations before finding *another* set $S_i \neq S_x$ containing y; otherwise, $F = m - 1$.

Proposition 5.2. $\mathbf{E}[F] = \frac{m}{\deg(y)} - 1$.

Proof. If $\deg(y) = 1$, then F is deterministically $m - 1$, in which case the lemma clearly holds.

The rest of the proof considers $\deg(y) \geq 2$. In that scenario, F can be rephrased in the following conventional WoR-sampling setup. Suppose that we have $m - 1$ balls, among which $\deg(y) - 1$ ones are red and the rest are white. Uniformly sample a ball WoR until seeing a red ball, and F gives the number of white balls sampled in this process. It is well known (e.g., see [25]) that F follows the Negative Hypergeometric distribution with $\mathbf{E}[F] = \frac{m}{\deg(y)} - 1$. □

Combining the above with the obvious fact that $F \in [0, m - 1]$, we can now define

$$\Gamma = F + 1$$

as the desired random variable satisfying $\Gamma \in [m]$ and $\mathbf{E}[\Gamma] = m/\deg(y)$. The WoR-sampling operation at Line 3 can be implemented in $O(1)$ time; see, e.g., [13]. Thus, the cost of **find-2nd** is proportional to the value F returned. It follows from Proposition 5.2 that the expected cost of **find-2nd** is $O(\mathbf{E}[F]) = O(m/\deg(y))$.

Total Cost of Set-Union Sampling. Recall that, in the (original) baseline algorithm, each repeat takes $O(m)$ time; as the number of repeats is N/n expected, the total cost of taking one sample from $\bigcup_{i=1}^{m} S_i$ is $O(mN/n)$. By applying our remedy, one repeat of the baseline algorithm is now carried out in $O(mn/N)$ expected time (see the analysis in (12)). The remedy does not affect the expected number of repeats (i.e., N/n), which suggests that the overall expected sampling time should be $O(\frac{mn}{N} \cdot \frac{N}{n}) = O(m)$. Indeed, if $\mathbf{E}[T_{total}]$ represents the expected time for our algorithm to acquire a sample from $\bigcup_{i=1}^{m} S_i$, we can write

$$\mathbf{E}[T_{total}] = O(mn/N) + (1 - n/N) \cdot \mathbf{E}[T_{total}]$$

which solves to $\mathbf{E}[T_{total}] = O(m)$. This completes the proof of Theorem 1.3.

Remarks. The proposed algorithm is reminiscent of that of Karp, Luby, and Madras [26], which was designed to estimate $|\bigcup_{i=1}^{m} S_i|$ rather than to sample from $\bigcup_{i=1}^{m} S_i$. Our key novelty lies in employing Proposition 5.1 for the acceptance step, while Karp, Luby, and Madras [26] used a more sophisticated method that is tailored for estimation and does not extend to sampling.

Our algorithm, in fact, also implies a simple method for estimating the value of n. Recall that one repeat of our algorithm runs in $O(mn/N)$ expected time and *succeeds* in returning a sample with probability n/N. To estimate n, we keep repeating our algorithm until getting

$$\gamma \;=\; \lceil 36 \ln(2/\delta) \rceil \tag{13}$$

successes, where δ is a real value satisfying $0 < \delta < 1$ (the purpose of δ will be clear shortly). If ρ is the number of repeats that have been performed, we calculate

$$\hat{n} \;=\; \frac{3\gamma}{2\rho} \cdot N. \tag{14}$$

We prove in Appendix B:

Lemma 5.3. *With probability at least $1 - \delta$, it holds that $n \le \hat{n} \le 2n$.*

As proved earlier, our algorithm uses $O(m)$ expected time to draw a sample from $\bigcup_{i=1}^{m} S_i$. The above method for estimating n essentially draws γ samples. We thus obtain:

Corollary 5.4. *For any $0 < \delta < 1$, we can obtain in $O(m \log \frac{1}{\delta})$ expected time a value of $\hat{n}$ that satisfies $n \le \hat{n} \le 2n$ with probability at least $1 - \delta$.*

The corollary will be useful in Section 6.

5.2 A Structure for Problem 2

Denote by $\mathcal{Q}$ a join instance of a schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. We prove Theorem 1.4 by reducing the $(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$-sampling problem to set union sampling.

Recall that $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for every $i \in [m]$. Hence, we can create for each $i \in [m]$

- a structure of Theorem 1.2 — denoted as $\Upsilon_i^{\mathrm{sam}}$ — that supports $(\mathcal{G}, \mathcal{Z}_i)$-sampling operations in $O(1)$ time;

- a structure of Lemma 4.3 — denoted as Υ_i^{size} — that, given any tuple $\boldsymbol{z}_i$ over $\mathcal{Z}_i$, returns $|\sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}))|$ in $O(1)$ time;

As each structure occupies $O(\text{IN})$ space, it may appear *as if* the total space would be $O(m \cdot \text{IN})$. This is not true. To see why, note that every $\mathcal{Z}_i$ is a subset of $\mathcal{V}$; as $\mathcal{V}$ has a constant size, the number of its subsets, i.e., $2^{|\mathcal{V}|}$, is a constant. This means that there can be only a constant number of *distinct* subsets among $\mathcal{Z}_1, \mathcal{Z}_2, ..., \mathcal{Z}_m$. Physically, it suffices to build a structure of Theorem 1.2 and a structure of Lemma 4.3 for each distinct subset; hence, the total space is $O(\text{IN})$. For similar reasons, all the structures can be constructed in $O(\text{IN})$ expected time.

For each relation $R \in \mathcal{Q}$, we create a perfect-hashing structure that allows us to decide whether a given tuple exists in R in $O(1)$ time. All the perfect-hashing structures can be built in $O(\text{IN})$ expected time [17] and occupy $O(\text{IN})$ space.

A $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling operation is given m tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ over $\mathcal{Z}_1, ..., \mathcal{Z}_m$, respectively. For each $i \in [m]$, define

$$S_i = \sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q})).$$

The operation, essentially, aims to return a uniformly-random tuple from $\bigcup_{i=1}^m S_i$. To cast this as an instance of set union sampling, we need to implement each of the size, membership, and sampling operations in $O(1)$ time on each S_i. We achieve the purpose as follows:

- size: use Υ_i^{size} to extract $|S_i|$;
- sampling: use Υ_i^{sam} to perform a $(\mathcal{G}, \mathcal{Z}_i)$-sampling operation;
- membership: given a tuple $\boldsymbol{u}$ over $\mathcal{V}$, this operation checks whether $\boldsymbol{u} \in \sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}))$. This requires checking if $\boldsymbol{u}[e]$ is in the relevant relation of $\mathcal{Q}$ for each $e \in \mathcal{E}$ and if $\boldsymbol{u}(Z) = \boldsymbol{z}_i(Z)$ for each $Z \in \mathcal{Z}_i$. All the checking can be done in $O(1)$ time.

Our algorithm in Theorem 1.3 can now be utilized to extract a uniformly-random tuple from $\bigcup_{i=1}^m S_i$ in $O(m)$ expected time, thus completing the proof of Theorem 1.4.

5.3 Proof of Theorem 1.5

The SSD conjecture given in Section 1.2 concerns data structures that solve the set disjointness problem with *deterministic* query time (here, a "query" is given $a, b \in [m]$ and reports whether $S_a \cap S_b = \emptyset$). To prove Theorem 1.5, we will first argue that a similar conjecture still stands even on set-disjointness structures with *expected* query time.

> **Expected SSD Conjecture.** Let $S_1, S_2, ..., S_m$ be $m \geq 2$ sets whose elements are integers. Define $N = \sum_{i=1}^m |S_i|$. We want to preprocess these sets into a data structure of $\mathcal{S}$ space such that, given any distinct integers $a, b \in [m]$, we can report in $\mathcal{T} = o(N)$ expected time whether $S_a \cap S_b = \emptyset$. The *expected SSD conjecture* states that $\mathcal{S}$ must be $\Omega(N^2/(\mathcal{T}^2 \operatorname{polylog} N))$.

Lemma 5.5. *The SSD conjecture implies the expected SSD conjecture.*

Proof. Suppose that we can build a structure Υ of $\mathcal{S}$ space and $\mathcal{T}$ expected query time for the set disjointness problem. We will prove the existence of a set-disjointness structure of $\mathcal{S} + O(\mathcal{T} \log N)$ space and $O(\mathcal{T} \log N)$ deterministic query time. This establishes the claim in Lemma 5.5 because $\mathcal{T} \leq N \leq \mathcal{S}$, implying $\mathcal{S} + O(\mathcal{T} \log N) = O(\mathcal{S} \log N)$.

Fix any distinct integers $a, b \in [m]$. Let $\mathcal{A}$ be the algorithm associated with $\mathfrak{T}$ for deciding whether $S_a \cap S_b = \emptyset$. Denote by X the cost for $\mathcal{A}$. Note that X is a random variable with $\mathbf{E}[X] \le \mathfrak{T}$. By Markov's inequality, $\mathbf{Pr}[X \ge 2\mathfrak{T}] \le 1/2$. Imagine running $\mathcal{A}$ for $3 \log N$ times. The probability for all those runs to take at least $2\mathfrak{T}$ time to terminate is at most $1/N^3$. In other words, with probability at least $1 - 1/N^3$, at least one run finishes within $2\mathfrak{T}$ time.

In general, a randomized algorithm becomes deterministic once all the random bits are fixed. "Running $\mathcal{A}$" is equivalent to (i) first fixing a sequence σ of random bits, and (ii) then executing *deterministically* the instructions of $\mathcal{A}$ based on σ. If $\mathcal{A}$ finishes within $2\mathfrak{T}$ time, it consumes at most $2\mathfrak{T}$ words of random bits. Now, take $t = 3 \log N$ random bit sequences $\sigma_1, ..., \sigma_t$, each of which is $2\mathfrak{T}$ words long. For each $i \in [t]$, define $\mathcal{A}(\sigma_i)$ as the deterministic algorithm that runs $\mathcal{A}$ based on σ_i, with the modification that $\mathcal{A}(\sigma_i)$ terminates itself after having run $\mathcal{A}$ for a duration of $2\mathfrak{T}$ time. As per our earlier discussion, with probability at least $1 - 1/N^3$, at least one of $\mathcal{A}(\sigma_1), ..., \mathcal{A}(\sigma_t)$ manages to report whether $S_a \cap S_b = \emptyset$.

Now, let us consider all $m(m-1)/2$ distinct pairs of $a, b \in [m]$. With probability at least $1 - m^2/N^3 \ge 1 - 1/N$, for every pair of a and b, at least one of $\mathcal{A}(\sigma_1), ..., \mathcal{A}(\sigma_t)$ manages to report whether $S_a \cap S_b = \emptyset$ — let us call $\{\sigma_1, ..., \sigma_t\}$ a *working set*. As the probability $1 - 1/N$ is greater than 0 (because $N \ge m \ge 2$), a working set must exist.

Our final set-disjointness structure consists of Υ and a working set of random-bit sequences $\{\sigma_1, ..., \sigma_t\}$, the total space of which is $\mathcal{S} + O(\mathfrak{T} \log N)$. Given two distinct integers $a, b \in [m]$, we run all of $\mathcal{A}(\sigma_1), ..., \mathcal{A}(\sigma_t)$, at least one of which manages to report whether $S_a \cap S_b = \emptyset$. The total query time is $O(\mathfrak{T} \cdot t) = O(\mathfrak{T} \log N)$. □

Our argument in Section 4.1 is a reduction from set disjointness to $(\mathcal{G}, \mathcal{Z})$-sampling where $\mathcal{G}$ is not ext-$\mathcal{Z}$-connex, and works regardless of whether the $(\mathcal{G}, \mathcal{Z})$-sampling algorithm is randomized or not. The reduction shows that if there is a structure of $O(\text{IN})$ space that can support a $(\mathcal{G}, \mathcal{Z})$-sampling operation in $O(\text{IN}^{0.49})$ expected time, then there is a set-disjointness structure of $O(N)$ space and $O(N^{0.49})$ expected query time — this will break the expected SSD conjecture.

Let us return to the context of Theorem 1.5 where $m \le \text{IN}^{0.49}$. W.l.o.g., assume that $\mathcal{G}$ is not ext-$\mathcal{Z}_1$-connex, and yet there is a structure Υ of $O(\text{IN})$ space that can support a $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling operation in $O(m)$ expected time. We will show how to use Υ to support a $(\mathcal{G}, \mathcal{Z}_1)$-sampling operation in $O(\text{IN}^{0.49})$ expected time, which, as pointed out earlier, breaks the expected SSD conjecture. In fact, this is fairly obvious. Suppose that we are given a tuple $\boldsymbol{z}_1$ over $\mathcal{Z}_1$. For each $i \in [2, m]$, construct a dummy tuple $\boldsymbol{z}_i$ over $\mathcal{Z}_i$ such that, for each $Z \in \mathcal{Z}_i$, $\boldsymbol{z}_i(Z)$ is a value that does not appear in the relations of $\mathcal{Q}$. Use Υ to perform a $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling operation and simply return the output of this operation.

6 Batch Sampling

The sampling times in Theorems 1.2 and 1.4 apply to only *one* sampling operation. In practice, one often needs to extract a *batch* of samples under the same selection conditions. Specifically:

- For Problem 1, this means that, given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, we want to draw $t \ge 1$ independent tuples uniformly at random from $\sigma_{\boldsymbol{z}}(\mathit{Join}(\mathcal{Q}))$.

- For Problem 2, this means that, given m distinct tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ over $\mathcal{Z}_1, ..., \mathcal{Z}_m$, respectively, we want to draw $t \ge 1$ independent tuples from $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(\mathit{Join}(\mathcal{Q}))$ uniformly at random.

Batch sampling under Problem 1 is easy: the structure of Theorem 1.2 can be used to get t samples in $O(t)$ time, provided that the schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is ext-$\mathcal{Z}$-connex; clearly, the time $O(t)$ is asymptotically optimal. For Problem 2, however, applying Theorem 1.4 to draw t samples incurs $O(m \cdot t)$ expected time, provided that $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for all $i \in [m]$. This is substantially higher than the straightforward $\Omega(m + t)$ lower bound on the (expected) sampling time. Next, we present a *predicate grouping* approach to reduce the sampling time from $O(m \cdot t)$ to $O(m + t)$.

Predicate Grouping. Each of $\mathcal{Z}_1, ..., \mathcal{Z}_m$ in the definition of Problem 2 is a subset of $\mathcal{V}$ (recall that $\mathcal{V}$ consists of all the attributes in the relations of $\mathcal{Q}$). As noted in Section 5.2, even though m can be arbitrarily large, the number of distinct subsets among $\mathcal{Z}_1, ..., \mathcal{Z}_m$ cannot exceed $2^{|\mathcal{V}|}$, which is a constant. To facilitate presentation, we introduce several notations:

- Let $g = O(1)$ be the number of distinct subsets among $\mathcal{Z}_1, ..., \mathcal{Z}_m$.
- Let $\mathcal{Z}_1^*, \mathcal{Z}_2^*, ..., \mathcal{Z}_g^*$ be the g distinct subsets among $\mathcal{Z}_1, ..., \mathcal{Z}_m$.
- For each $i \in [g]$, let m_i be the number of occurrences of $\mathcal{Z}_i^*$ among $\mathcal{Z}_1, ..., \mathcal{Z}_m$.

A $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling operation is given m tuples $\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_m$ that are over $\mathcal{Z}_1, ..., \mathcal{Z}_m$, respectively. For each $i \in [g]$ and $j \in [m_i]$, we define

$$\boldsymbol{z}_{i,j}^* = \text{the tuple among } \boldsymbol{z}_1, ..., \boldsymbol{z}_m \text{ that is over the } j\text{-th occurrence of } \mathcal{Z}_i^* \text{ in } \mathcal{Z}_1, ..., \mathcal{Z}_m. \quad (15)$$

Example 6.1. *Consider the schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ shown in Figure 1(a). Suppose $m = 6$ and $\mathcal{Z}_1 = \texttt{EGH}$, $\mathcal{Z}_2 = \mathcal{Z}_3 = \texttt{CDE}$, and $\mathcal{Z}_4 = \mathcal{Z}_5 = \mathcal{Z}_6 = \texttt{GI}$. Thus, $g = 3$ and $\mathcal{Z}_1^* = \texttt{EGH}$, $\mathcal{Z}_2^* = \texttt{CDE}$, and $\mathcal{Z}_3^* = \texttt{GI}$. If $\boldsymbol{z}_1, ..., \boldsymbol{z}_6$ are the tuples given to a $(\mathcal{G}, \bigvee_{i=1}^6 \mathcal{Z}_i)$-sampling operation, we write $\boldsymbol{z}_{1,1}^* = \boldsymbol{z}_1$, $\boldsymbol{z}_{2,1}^* = \boldsymbol{z}_2$, $\boldsymbol{z}_{2,2}^* = \boldsymbol{z}_3$, $\boldsymbol{z}_{3,1}^* = \boldsymbol{z}_4$, $\boldsymbol{z}_{3,2}^* = \boldsymbol{z}_5$, and $\boldsymbol{z}_{3,3}^* = \boldsymbol{z}_6$.* □

For each $i \in [g]$, define

$$S_i = \bigcup_{j=1}^{m_i} \sigma_{\boldsymbol{z}_{i,j}^*}(\mathit{Join}(\mathcal{Q})). \quad (16)$$

It is clear that

$$\bigcup_{i=1}^{g} S_i = \sigma_{\boldsymbol{z}_1 \vee \boldsymbol{z}_2 \vee ... \vee \boldsymbol{z}_m}(\mathit{Join}(\mathcal{Q})). \quad (17)$$

In other words, sampling from $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(\mathit{Join}(\mathcal{Q}))$ is equivalent to sampling from $\bigcup_{i=1}^g S_i$.

Data Structures. For every $i \in [g]$, since $\mathcal{G}$ is ext-$\mathcal{Z}_i^*$-connex, we can create

- a structure of Theorem 1.2 — denoted as Υ_i^{sam} — that supports $(\mathcal{G}, \mathcal{Z}_i^*)$-sampling operations in $O(1)$ time;
- a structure of Lemma 4.3 — denoted as Υ_i^{size} — that, given any tuple $\boldsymbol{z}_i^*$ over $\mathcal{Z}_i^*$, returns $|\sigma_{\boldsymbol{z}_i^*}(\mathit{Join}(\mathcal{Q}))|$ in $O(1)$ time.

The $2g$ structures can be built in $O(g \cdot \text{IN}) = O(\text{IN})$ expected time and occupy $O(\text{IN})$ space in total.

For each relation $R \in \mathcal{Q}$, we also create a perfect-hashing structure that, given a tuple $\boldsymbol{u}$ over $\mathit{schema}(R)$, can decide whether $\boldsymbol{u} \in R$ in constant time. All the perfect-hashing structures can be constructed in $O(\text{IN})$ expected time [17] and use $O(\text{IN})$ space.

Sampling. Suppose that we are given tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ over $\mathcal{Z}_1, ..., \mathcal{Z}_m$, respectively — or equivalently, the tuple set $\{\boldsymbol{z}^*_{i,j} \mid i \in [g], j \in [m_i]\}$ as defined in (15). Our objective is to sample $t \geq 1$ tuples uniformly at random from $\bigcup_{i=1}^{g} S_i$, where S_i is defined in (16).

We aim to reduce the operation of drawing one uniform sample from $\bigcup_{i=1}^{g} S_i$ to an instance of set union sampling — solved in Section 5.1 — that is defined on $S_1, S_2, ..., S_g$. This means that, for each S_i ($1 \leq i \leq g$), we must implement its size, sampling, and membership operations in such a way that each takes only constant time to finish. If we manage to achieve the purpose, Theorem 1.3 assures us that one sample can be drawn from $\bigcup_{i=1}^{g} S_i$ in $O(g) = O(1)$ expected time, implying $O(t)$ expected time for drawing t samples.

Next, we show how to complete the reduction in $O(m)$ expected time. For each $i \in [g]$, we can use Υ_i^{size} to find the size of S_i in $O(m_i)$ time. The crucial observation is that (16) implies

$$|S_i| \quad = \quad \sum_{j=1}^{m_i} |\sigma_{\boldsymbol{z}^*_{i,j}}(Join(\mathcal{Q}))| \tag{18}$$

which holds true because $\boldsymbol{z}^*_{i,1}, ..., \boldsymbol{z}^*_{i,m_i}$ are distinct tuples and, thus, $\sigma_{\boldsymbol{z}^*_{i,1}}(Join(\mathcal{Q})), ..., \sigma_{\boldsymbol{z}^*_{i,m_i}}(Join(\mathcal{Q}))$ are mutually disjoint. Hence, the values of $|S_1|, |S_2|, ..., |S_g|$ can be obtained in $O(\sum_{i=1}^{g} m_i) = O(m)$ total time, after which a size operation on S_i for any $i \in [g]$ can be trivially supported in $O(1)$ time.

For each $i \in [g]$, we also build an alias structure for generating a random value X satisfying

$$\mathbf{Pr}[X = j] \quad = \quad \frac{|\sigma_{\boldsymbol{z}^*_{i,j}}(Join(\mathcal{Q}))|}{|S_i|} \tag{19}$$

for each $j \in [m_i]$. As the values $|\sigma_{\boldsymbol{z}^*_{i,1}}(Join(\mathcal{Q}))|, ..., |\sigma_{\boldsymbol{z}^*_{i,m_i}}(Join(\mathcal{Q}))|$ have already been attained, it takes $O(m_i)$ time to build the alias structure (see Section 3.1). The total construction time of all the g alias structures is $\sum_{i=1}^{g} O(m_i) = O(m)$. For any $i \in [g]$, a sampling operation on S_i can now be performed in $O(1)$ time as follows: first draw a random value X according to (19) and then use Υ_i^{sam} to draw a tuple from $\sigma_{\boldsymbol{z}^*_{i,X}}(Join(\mathcal{Q}))$ uniformly at random.

Finally, for each $i \in [g]$, we build a perfect-hashing structure on the set $\{\boldsymbol{z}^*_{i,j} \mid 1 \leq j \leq m_i\}$. The total time of building all such structures is $\sum_{i=1}^{g} O(m_i) = O(m)$ expected [17]. For any $i \in [g]$, a membership operation on S_i, which decides whether a tuple $\boldsymbol{u}$ over $\mathcal{V}$ is in S_i, can now be performed in $O(1)$ time as follows. First, check for each $e \in \mathcal{E}$ whether $\boldsymbol{u}[e]$ is in the relation $R \in \mathcal{Q}$ with $schema(R) = e$. Then, check whether $\boldsymbol{u}[\mathcal{Z}_i^*]$ is in the set $\{\boldsymbol{z}^*_{i,j} \mid 1 \leq j \leq m_i\}$. The tuple $\boldsymbol{u}$ is in S_i if and only if it passes all the checking.

We therefore have obtained an algorithm to draw t independent tuples from $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ uniformly at random in $O(t)$ expected time, after an initial processing (i.e., the reduction) of $O(m)$ expected time. This completes the proof of Theorem 1.6.

Remark: Estimating $|\boldsymbol{\sigma_{z_1 \vee ... \vee z_m}(Join(\mathcal{Q}))}|$. By Corollary 5.4, after the reduction is complete, we can estimate $|\bigcup_{i=1}^{g} S_i| = |\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))|$ by deriving in $O(g \log \frac{1}{\delta}) = O(\log \frac{1}{\delta})$ time an estimate $\hat{n}$ that satisfies $|\bigcup_{i=1}^{g} S_i| \leq \hat{n} \leq 2|\bigcup_{i=1}^{g} S_i|$ with probability at least $1 - \delta$. Taking into account the fact that the reduction itself takes $O(m)$ expected time, we have arrived at:

Corollary 6.1. *Consider $(\mathcal{G}, \bigvee_{i=1}^{m} \mathcal{Z}_i)$-sampling where $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for all $i \in [m]$. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN*, we can build a structure of $O(\text{IN})$ space in $O(\text{IN})$ expected time such that, given any m tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ (with $\boldsymbol{z}_i$ over $\mathcal{Z}_i$ for each $i \in [m]$) and any real value $0 < \delta < 1$, we can obtain in $O(m + \log \frac{1}{\delta})$ expected time a value $\hat{n}$ that satisfies $|\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))| \leq \hat{n} \leq 2|\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))|$ with probability at least $1 - \delta$.*

7 Random Enumeration from Sampling

This section will establish a connection between random enumeration and uniform sampling and then leverage the connection to prove Theorems 1.7 and 1.8.

7.1 A Reduction

We consider a general setup. Let S be a set of elements whose size n may be unknown. We are given an estimate $\hat{n}$ satisfying

$$n \leq \hat{n} \leq 2n. \tag{20}$$

In addition, two subroutines are at our disposal:

- The first lists the elements of S (in an order we cannot control) within $n \cdot \lambda_{rep}$ time;
- The second samples a uniformly random element of S with replacement within λ_{sam} time.

The rest of the section serves as a proof of the following lemma.

Lemma 7.1. *The elements of S can be randomly permuted with an expected delay of $O(\lambda_{rep} + \lambda_{sam})$.*

Let us first describe an algorithm that produces a random permutation of S but does not ensure a small delay. The algorithm runs in three phases. In the first one, we use dynamic perfect hashing [20] to maintain a set S_{seen} of elements that have been found. Initially, $S_{seen} = \emptyset$. Then, we carry out *iterations*, each of which adds a new element to S_{seen}. Specifically, an iteration starts by randomly sampling an element e from S. If $e \notin S_{seen}$, we add it to S_{seen} and output it. Otherwise, the iteration re-samples from S until getting an unseen element. As long as $|S_{seen}| \leq \hat{n}/4 \leq n/2$, an unseen element is sampled with probability at least $1/2$. Hence, two samples suffice in expectation, and the cost of an iteration is $O(\lambda_{sam})$ expected. The first phase finishes after $\hat{n}/4$ iterations. The second phase finds the entire S — note: the value of n becomes known at this time — and extracts (with hashing) and randomly permutes (with Fisher-Yates shuffle [22]) $S \setminus S_{seen}$. The phase performs at most $c \cdot n \cdot \lambda_{rep}$ atomic instructions of the RAM model for some constant c. Finally, the third phase outputs $S \setminus S_{seen}$ by the permuted order.

We apply a *de-amortization* approach to implement the above algorithm with a small expected delay. In the first phase, every time an element is added to S_{seen}, we do not output it immediately, but instead append it to a buffer queue S_{buf}. We remove and output the head element of S_{buf} every

$$\alpha = \left\lceil 1 + \frac{\lambda_{rep}}{\lambda_{sam}} \right\rceil$$

iterations so that S_{buf} still has $\frac{\hat{n}}{4}(1 - 1/\alpha)$ elements left at the end of the first phase. In the second phase, we remove and enumerate the head of S_{buf} after every

$$\frac{c \cdot n \cdot \lambda_{rep}}{(\hat{n}/4) \cdot (1 - \frac{1}{\alpha})} \leq 4c \cdot \lambda_{rep} \cdot \frac{\alpha}{\alpha - 1}$$

atomic instructions. The third phase simply enumerates the remaining elements in S_{buf} (by their queued order) and $S \setminus S_{seen}$ (by their permuted order) with a constant delay. Overall, the expected delay, which is determined by the first two phases, is bounded by

$$\alpha \cdot O(\lambda_{sam}) + \frac{4c \cdot \lambda_{rep} \cdot \alpha}{\alpha - 1}.$$

It is rudimentary to show that the bound is $O(\lambda_{sam} + \lambda_{rep})$, completing the proof of Lemma 7.1.

7.2 Random Enumeration of $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ (Proof of Theorem 1.7)

Let $\mathcal{Q}$ be a join with schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and let $\mathcal{Z}$ be a subset of $\mathcal{V}$ such that $\mathcal{G}$ is ext-$\mathcal{Z}$-connex. Our goal in this subsection is to preprocess $\mathcal{Q}$ into a data structure such that, given any tuple $\boldsymbol{z}$ over $\mathcal{Z}$, we can enumerate the tuples of $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ with a constant expected delay (see Section 1.2 for the definition of "expected delay"). More specifically, we aim to apply Lemma 7.1 to establish Theorem 1.7.

We create a structure Υ^{size} of Lemma 4.3, a structure Υ^{sam} of Theorem 1.2, and a structure Υ^{rep} of the lemma below.

Lemma 7.2. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a schema graph and $\mathcal{Z}$ be a subset of $\mathcal{V}$ such that $\mathcal{G}$ is ext-$\mathcal{Z}$-connex. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN*, we can build a structure of $O(\text{IN})$ space in $O(\text{IN})$ expected time such that, given any tuple $\boldsymbol{z}$ over $\mathcal{Z}$, we can report $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ in $O(1+|\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))|)$ time.*

The proof is presented in Appendix C. Given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, let us define $S = \sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$. The structures built earlier provide all the necessary ingredients for us to apply Lemma 7.1 to enumerate S:

- Use Υ^{size} to obtain the size of $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ and set $\hat{n}$ to it (by the terminology in Section 7.1, the estimate $\hat{n}$ is simply $n = |S|$).
- Use Υ^{sam} to sample a tuple from S in $\lambda_{sam} = O(1)$ time.
- Use Υ^{rep} report S in $O(1 + |S|)$ time, implying $\lambda_{rep} = O(1)$.

Immediately, Lemma 7.1 yields an algorithm to randomly permute $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ with an $O(1)$ expected delay. This completes the proof of Theorem 1.7.

7.3 Random Enumeration of $\sigma_{\boldsymbol{z}_1 \vee \ldots \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ (Proof of Theorem 1.8)

Let $\mathcal{Q}$ be a join with schema graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and let $\mathcal{Z}_1, ..., \mathcal{Z}_m$ ($m \geq 1$) be subsets of $\mathcal{V}$ such that $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for all $i \in [m]$. Our goal in this subsection is to preprocess $\mathcal{Q}$ into a data structure such that, given any tuple $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ over $\mathcal{Z}_1, ..., \mathcal{Z}_m$, respectively, we can enumerate the tuples of $\sigma_{\boldsymbol{z}_1 \vee \ldots \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ with a constant expected delay. More specifically, we will establish Theorem 1.8 by putting together the techniques developed in Sections 4-6.

The starting point of our solution is the predicate grouping technique introduced in Section 6. As before, let g be the number of distinct subsets among $\mathcal{Z}_1, ..., \mathcal{Z}_m$; remember that g is bounded by $2^{|\mathcal{V}|}$, which is a constant. Denote by $\mathcal{Z}_1^*, ..., \mathcal{Z}_g^*$ the distinct subsets among $\mathcal{Z}_1, ..., \mathcal{Z}_m$; and for each $i \in [g]$, let m_i represent the number of occurrences of $\mathcal{Z}_i^*$ among $\mathcal{Z}_1, ..., \mathcal{Z}_m$. Furthermore, given the tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$, we rename them into $\{\boldsymbol{z}_{i,j}^* \mid i \in [g], j \in [m_i]\}$ according to the definition in (15). Now, set

$$S = \bigcup_{i=1}^{g} S_i \tag{21}$$

where S_i ($1 \leq i \leq g$) is as defined in (16). The task of enumerating $\sigma_{\boldsymbol{z}_1 \vee \ldots \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ has been converted to that of enumerating S.

We create a structure Υ^{size} of Corollary 6.1, a structure Υ^{sam} of Theorem 1.6, and a structure Υ^{rep} of the lemma below.

Lemma 7.3. *Consider $(\mathcal{G}, \bigvee_{i=1}^m \mathcal{Z}_i)$-sampling where $\mathcal{G}$ is ext-$\mathcal{Z}_i$-connex for all $i \in [m]$. Given a join instance $\mathcal{Q}$ of $\mathcal{G}$ with input size* IN*, we can build a structure of $O(\mathrm{IN})$ space in $O(\mathrm{IN})$ expected time such that, given any m tuples $\boldsymbol{z}_1, ..., \boldsymbol{z}_m$ (with $\boldsymbol{z}_i$ over $\mathcal{Z}_i$ for each $i \in [m]$), we can report $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ in $O(|\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))|)$ time, after an initial processing of $O(m)$ expected time.*

The proof is presented in Appendix D. The structures built earlier provide all the necessary ingredients for us to apply Lemma 7.1 to enumerate the set S in (21):

- Use Υ^{size} to obtain an estimate $\hat{n}$ of $|S|$ in $O(m + \log \frac{1}{\delta})$ time. With probability at least $1 - \delta$, it holds that $|S| \le \hat{n} \le 2|S|$.
- After an initial processing of $O(m)$ expected time, use Υ^{sam} to sample a tuple from S in $\lambda_{sam} = O(1)$ time.
- After an initial processing of $O(m)$ expected time, use Υ^{rep} report S in $O(|S|)$ time, implying $\lambda_{rep} = O(1)$.

Immediately, Lemma 7.1 yields an algorithm to randomly permute S with an $O(1)$ expected delay, after an initial processing of $O(m + \log \frac{1}{\delta})$ expected time. This completes the proof of Theorem 1.8.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, pages 275–286, 1999.

[3] Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *SIAM J. of Comp.*, 42(4):1737–1767, 2013.

[4] Martin Aumuller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. Sampling a near neighbor in high dimensions - who is the fairest of them all? *TODS*, 47(1):4:1–4:40, 2022.

[5] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic*, pages 208–222, 2007.

[6] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frederic Olive. Computing the jth solution of a first-order query. *RAIRO Theor. Informatics Appl.*, 42(1):147–164, 2008.

[7] Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. *TODS*, 50(1):1:1–1:44, 2025.

[8] Florent Capelli, Oliver Irwin, and Sylvain Salvati. A simple algorithm for worst case optimal join and sampling. In *ICDT*, pages 23:1–23:19, 2025.

[9] Nofar Carmeli and Markus Kroll. Enumeration complexity of conjunctive queries with functional dependencies. *Theory Comput. Syst.*, 64(5):828–860, 2020.

[10] Nofar Carmeli and Markus Kroll. On the enumeration complexity of unions of conjunctive queries. *TODS*, 46(2):5:1–5:41, 2021.

[11] Nofar Carmeli and Luc Segoufin. Conjunctive queries with self-joins, towards a fine-grained enumeration complexity analysis. In *PODS*, pages 277–289, 2023.

[12] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *TODS*, 48(1):1:1–1:45, 2023.

[13] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Alessio Conte, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. *TODS*, 47(3):9:1–9:49, 2022.

[14] Timothy M. Chan, Saladi Rahul, and Jie Xue. Range closest-pair search in higher dimensions. *Computational Geometry*, 91:101669, 2020.

[15] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. On random sampling over joins. In *SIGMOD*, pages 263–274, 1999.

[16] Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In *ICDT*, pages 7:1–7:18, 2020.

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[18] Shaleen Deep and Paraschos Koutris. Compressed representations of conjunctive query results. In *PODS*, pages 307–322, 2018.

[19] Shiyuan Deng, Shangqi Lu, and Yufei Tao. On join sampling and the hardness of combinatorial output-sensitive join algorithms. In *PODS*, pages 99–111, 2023.

[20] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. of Comp.*, 23(4):738–761, 1994.

[21] Arnaud Durand. Fine-grained complexity analysis of queries: From decision to counting and enumeration. In *PODS*, pages 331–346, 2020.

[22] Ronald Aylmer Fisher and Frank Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. Edinburgh and London: Oliver and Boyd., 1953.

[23] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *WADS*, pages 421–436. Springer, 2017.

[24] Isaac Goldstein, Moshe Lewenstein, and Ely Porat. On the hardness of set disjointness and set intersection with bounded universe. In *ISAAC*, pages 7:1–7:22, 2019.

[25] Norman L Johnson, Adrienne W Kemp, and Samuel Kotz. *Univariate Discrete Distributions*. John Wiley & Sons, 2005.

[26] Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

[27] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *PODS*, pages 429–444, 2017.

[28] Kyoungmin Kim, Jaehyun Ha, George Fletcher, and Wook-Shin Han. Guaranteeing the $\tilde{O}$(AGM/OUT) runtime for uniform sampling and size estimation over joins. In *PODS*, pages 113–125, 2023.

[29] Shangqi Lu and Yufei Tao. Indexing for keyword search with structured constraints. In *PODS*, pages 263–275, 2023.

[30] Mihai Patrascu and Liam Roditty. Distance oracles beyond the Thorup-Zwick bound. *SIAM J. of Comp.*, 43(1):300–311, 2014.

[31] Cheng Sheng, Yufei Tao, and Jianzhong Li. Exact and approximate algorithms for the most connected vertex problem. *TODS*, 37(2):12:1–12:39, 2012.

[32] Yufei Tao and Ru Wang. Split theorems for join reporting and sampling. *SIGMOD Rec.*, 54(4):7–18, 2025.

[33] Alastair J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974.

[34] Ru Wang and Yufei Tao. Join and subgraph sampling under degree constraints. *JCSS*, 155, 2026.

[35] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

[36] Hangdong Zhao, Shaleen Deep, and Paraschos Koutris. Space-time tradeoffs for conjunctive queries with access patterns. In *PODS*, pages 59–68, 2023.

[37] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *SIGMOD*, pages 1525–1539, 2018.

A Supplementary Proofs for Section 3.2

A.1 Primitive Data Structures

Group-By Sum. Let R be a relation over a set $\mathcal{X}$ of attributes such that each tuple $\boldsymbol{u} \in R$ carries a positive-integer weight. Let $\mathcal{Y}$ be a non-empty subset of $\mathcal{X}$. Given a tuple $\boldsymbol{v}$ over $\mathcal{Y}$, a *group-by sum* query returns the total weight of all tuples $\boldsymbol{u} \in R$ satisfying $\boldsymbol{u}[\mathcal{Y}] = \boldsymbol{v}$ — we refer to the query result as the *group-by sum of* $\boldsymbol{v}$.

Lemma A.1. *In $O(|R|)$ expected time, we can create a structure of $O(|R|)$ space that answers group-by sum queries on R in constant time.*

Proof. Group the tuples of R by their projections on $\mathcal{Y}$ and compute, for each group, the total sum of the weights of its tuples. In addition, create a perfect-hashing structure [20] that, given any tuple $\boldsymbol{v}$ over $\mathcal{Y}$, returns in constant time the group whose $\mathcal{Y}$ projection is $\boldsymbol{v}$. The construction takes $O(|R|)$ expected time in total (note: the time is expected due to [20]). □

Group-By Weighted Sampling. Let R be a relation over a set $\mathcal{X}$ of attributes such that each tuple $\boldsymbol{u} \in R$ carries a positive-integer weight. Let $\mathcal{Y}$ be a non-empty subset of $\mathcal{X}$. Given a tuple $\boldsymbol{v}$ over $\mathcal{Y}$, a *group-by weighted sampling* query returns a random variable $\boldsymbol{X}$ such that, for each tuple $\boldsymbol{u} \in R$ satisfying $\boldsymbol{u}[\mathcal{Y}] = \boldsymbol{v}$, we have:

$$\mathbf{Pr}[\boldsymbol{X} = \boldsymbol{u}] \quad = \quad \frac{\text{weight of } \boldsymbol{u}}{\text{the group-by sum of } \boldsymbol{v}}.$$

Lemma A.2. *In $O(|R|)$ expected time, we can create a structure of $O(|R|)$ space that answers group-by weighted sampling queries in constant time.*

Proof. Group the tuples of R by their projections on $\mathcal{Y}$ and create an alias structure (see Section 3.1) on each group. In addition, create a perfect-hashing structure [20] that, given any tuple $\boldsymbol{v}$ over $\mathcal{Y}$, returns in constant time the alias structure for the group whose $\mathcal{Y}$ projection is $\boldsymbol{v}$. The construction of all the alias structures and the perfect-hashing structure takes $O(|R|)$ expected time in total. $\square$

A.2 Proof of Lemma 3.1

We first review a property of acyclic joins that can be used to compute, for each tuple $\boldsymbol{u} \in R_e$ where e is a node of $\mathcal{T}$ (a.k.a. a hyperedge in $\mathcal{E}$), the result of $\boldsymbol{u} \bowtie Join(\mathcal{Q}_e)$ in a manner reminiscent of Cartesian products. W.l.o.g., suppose that e has $t \geq 1$ child nodes in $\mathcal{T}$ denoted as $e_1, e_2, ..., e_t$. Define $\mathcal{X}_i = e \cap e_i$ for each $i \in [t]$. Consider the set $S_\times$ of tuples created by the procedure below:

children-prod ($\boldsymbol{u}$) /* $\boldsymbol{u}$ is a tuple in R_e */
1. $S_\times \leftarrow \emptyset$
2. $S_i \leftarrow \boldsymbol{u}[\mathcal{X}_i] \bowtie Join(\mathcal{Q}_{e_i})$ for each $i \in [t]$
   /* recall that $\boldsymbol{u}[\mathcal{X}_i]$ is the projection of $\boldsymbol{u}$ onto $\mathcal{X}_i$ */
3. **for** each $(\boldsymbol{v}_1, ..., \boldsymbol{v}_t) \in S_1 \times ... \times S_t$ **do**
4. add $\boldsymbol{u} \bowtie \boldsymbol{v}_1 \bowtie \boldsymbol{v}_2 \bowtie ... \bowtie \boldsymbol{v}_t$ to $S_\times$

The next lemma is a well-known property of Yannakakis' algorithm [35].

Lemma A.3. [1,35] *Both statements below are true regarding* **children-prod***: (i) Line 4 always adds a new tuple to $S_\times$, and (ii) $\boldsymbol{u} \bowtie Join(\mathcal{Q}_e) = S_\times$.*

To prove Lemma 3.1, we process the relations of $\mathcal{Q}$ — namely, computing the subjoin weights of their tuples — in a bottom-up manner: a relation $R_e \in \mathcal{Q}$ is processed only after the relation of every $R_{e'}$, where e' is a proper descendant of e in $\mathcal{T}$, has been processed. If e is a leaf node of $\mathcal{T}$, the processing is trivial: its subjoin $\mathcal{Q}_e$ includes only one relation, which is R_e. Thus, $Join(\mathcal{Q}_e) = R_e$ and $subj\text{-}w_e(\boldsymbol{u}) = |\boldsymbol{u} \bowtie R_e| = 1$ for each $\boldsymbol{u} \in R_e$. Hence, the subjoin weights of all the tuples in R_e can be obtained in $O(|R_e|)$ time.

Now, consider e to be an internal node of $\mathcal{T}$. Suppose that e has $t \geq 1$ child nodes in $\mathcal{T}$ denoted as $e_1, ..., e_t$; for each $i \in [t]$, define $\mathcal{X}_i = e \cap e_i$ (as in **children-prod**). Let $\boldsymbol{u}$ be an arbitrary tuple in R_e. According to Lemma A.3, we have

$$subj\text{-}w_e(\boldsymbol{u}) = \prod_{i=1}^{t} |S_i(\boldsymbol{u})| \tag{22}$$

where $S_i(\boldsymbol{u}) = \boldsymbol{u}[\mathcal{X}_i] \bowtie Join(\mathcal{Q}_{e_i})$, as stated at Line 2 of **children-prod**. Define for each $i \in [t]$

$$R_{e_i}(\boldsymbol{u}) = \{\boldsymbol{v} \in R_{e_i} \mid \boldsymbol{v}[\mathcal{X}_i] = \boldsymbol{u}[\mathcal{X}_i]\}. \tag{23}$$

We can now write

$$\begin{aligned} |S_i(\boldsymbol{u})| = |\boldsymbol{u}[\mathcal{X}_i] \bowtie Join(\mathcal{Q}_{e_i})| &= \Big| \bigcup_{\boldsymbol{v} \in R_{e_i}(\boldsymbol{u})} \boldsymbol{v} \bowtie Join(\mathcal{Q}_{e_i}) \Big| \\ &= \sum_{\boldsymbol{v} \in R_{e_i}(\boldsymbol{u})} |\boldsymbol{v} \bowtie Join(\mathcal{Q}_{e_i})| \end{aligned}$$

$$= \sum_{\boldsymbol{v} \in R_{e_i}(\boldsymbol{u})} \textit{subj-w}_{e_i}(\boldsymbol{v}).$$

Hence, $|S_i(\boldsymbol{u})|$ is the result of a group-by sum query (see Section A.1).

Because the subjoin weights of the tuples in R_{e_i} have been computed (due to the bottom-up processing order), we can construct a structure of Lemma A.1 on R_{e_i}, which allows us to obtain $|S_i(\boldsymbol{u})|$ in constant time for any $\boldsymbol{u} \in R_e$. The structure occupies $O(|R_{e_i}|)$ space and its construction takes $O(|R_{e_i}|)$ expected time. Because of (22), these structures allow us to calculate $\textit{subj-w}_e(\boldsymbol{u})$ in $O(t) = O(1)$ time. This way, we obtain the subjoin weights of all tuples in R_e in $O(|R_e|)$ time.

The overall computation time is $O(\sum_{e \in \mathcal{E}} |R_e|) = O(\text{IN})$ expected. All the structures occupy $O(\sum_{e \in \mathcal{E}} |R_e|) = O(\text{IN})$ space.

A.3 Proof of Lemma 3.2

We first apply Lemma 3.1 to compute the subjoin weights of all the tuples in each relation of $\mathcal{Q}$.

If e is a leaf of the (rooted) $\mathcal{T}$, then $\textit{Join}(\mathcal{Q}_e) = R_e$ and $\boldsymbol{u} \bowtie R_e$ is simply $\{\boldsymbol{u}\}$. Hence, no special structure is needed to sample from $\boldsymbol{u} \bowtie R_e$.

Now, consider e to be an internal node of $\mathcal{T}$. W.l.o.g., suppose that e has $t \geq 1$ child nodes in $\mathcal{T}$ denoted as $e_1, ..., e_t$. For each $i \in [t]$, define $\mathcal{X}_i = e \cap e_i$. Assuming inductively that a structure has been created to draw a sample from $\boldsymbol{v} \bowtie \textit{Join}(\mathcal{Q}_{e_i})$ in $O(1)$ time for any $i \in [t]$ and $\boldsymbol{v} \in R_{e_i}$, next we describe how to build the corresponding structure on R_e.

Let $\boldsymbol{u}$ be an arbitrary tuple in R_e. According to Lemma A.3, there is a one-one correspondence between $\boldsymbol{u} \bowtie \textit{Join}(\mathcal{Q}_e)$ and $S_1(\boldsymbol{u}) \times S_2(\boldsymbol{u}) \times ... \times S_t(\boldsymbol{u})$ where $S_i(\boldsymbol{u}) = \boldsymbol{u}[\mathcal{X}_i] \bowtie \textit{Join}(\mathcal{Q}_{e_i})$, as defined at Line 2 of procedure **children-prod**. Hence, to take a uniform random tuple from $\boldsymbol{u} \bowtie \textit{Join}(\mathcal{Q}_e)$, we can take a uniformly random tuple from $S_i(\boldsymbol{u})$ for each $i \in [t]$.

Using the notation $R_{e_i}(\boldsymbol{u})$ in (23), we can write

$$S_i(\boldsymbol{u}) = \boldsymbol{u}[\mathcal{X}_i] \bowtie \textit{Join}(\mathcal{Q}_{e_i}) \quad = \bigcup_{\boldsymbol{v} \in R_{e_i}(\boldsymbol{u})} \boldsymbol{v} \bowtie \textit{Join}(\mathcal{Q}_{e_i}).$$

Random sampling from $S_i(\boldsymbol{u})$ can be performed in two steps. First, obtain a random tuple $\boldsymbol{X} \in R_{e_i}(\boldsymbol{u})$ such that $\mathbf{Pr}[\boldsymbol{X} = \boldsymbol{v}] = |\boldsymbol{v} \bowtie \textit{Join}(\mathcal{Q}_{e_i})|/|S_i(\boldsymbol{u})| = \textit{subj-w}_{e_i}(\boldsymbol{v})/|S_i(\boldsymbol{u})|$ for each $\boldsymbol{v} \in R_{e_i}(\boldsymbol{u})$. Second, draw a uniformly random tuple from $\boldsymbol{X} \bowtie \textit{Join}(\mathcal{Q}_{e_i})$.

The second step can be supported in $O(1)$ time by induction. The operation in the first step is a "group-by weighted sampling" query in Section A.1. As all the subjoin weights of the tuples in R_{e_i} have been computed, we can build a structure of Lemma A.2 on R_{e_i}. The structure occupies $O(|R_{e_i}|)$ space and the construction takes $O(|R_{e_i}|)$ expected time. This structure enables us to perform the first step in constant time. Therefore, a uniform random sample from $\boldsymbol{u} \bowtie \textit{Join}(\mathcal{Q}_e)$ can be drawn in $O(t) = O(1)$ time.

The overall computation time is $O(\sum_{e \in \mathcal{E}} |R_e|) = O(\text{IN})$ expected. All the structures occupy $O(\sum_{e \in \mathcal{E}} |R_e|) = O(\text{IN})$ space.

B Proof of Lemma 5.3

Let X be the sum of a finite number of independently and identically distributed Bernoulli random variables. For any $0 < \epsilon < 1$, the following two inequalities — known as *Chernoff bounds* — are always correct:

$$\begin{aligned}\mathbf{Pr}[X \le (1-\epsilon)\,\mathbf{E}[X]] &\le \exp\left(-\epsilon^2 \cdot \mathbf{E}[X]/3\right) \quad (24)\\ \mathbf{Pr}[X \ge (1+\epsilon)\,\mathbf{E}[X]] &\le \exp\left(-\epsilon^2 \cdot \mathbf{E}[X]/3\right). \quad (25)\end{aligned}$$

The above bounds can be found in many sources, e.g., [31].

Let us now return to the context of Lemma 5.3. Set $p = n/N$. By the definition of $\hat{n}$ in (14), $n \le \hat{n} \le 2n$ holds if and only if

$$\frac{3\gamma}{4p} \le \rho \le \frac{3\gamma}{2p} \quad (26)$$

We will establish the following two inequalities:

$$\begin{aligned}\mathbf{Pr}\left[\rho > \frac{3\gamma}{2p}\right] &\le \exp(-\gamma/36) \quad (27)\\ \mathbf{Pr}\left[\rho < \frac{3\gamma}{4p}\right] &\le \exp(-\gamma/36) \quad (28)\end{aligned}$$

which imply that (26) fails to hold with probability at most $2\exp(-\gamma/36)$, which is at most δ by the definition of γ in (13).

For any integer $t \ge 1$, define

$$X_t \;=\; \text{the number of successes in the first } t \text{ repeats of our algorithm.}$$

Remember that each repeat of our algorithm succeeds with probability $p = n/N$. It thus follows that $\mathbf{E}[X_t] = p \cdot t$. Define

$$\begin{aligned}t_1 &= \lceil 3\gamma/(2p) \rceil\\ t_2 &= \lfloor 3\gamma/(4p) \rfloor\\ \epsilon_1 &= 1 - \frac{\gamma}{\mathbf{E}[X_{t_1}]} = 1 - \frac{\gamma}{t_1 \cdot p}\\ \epsilon_2 &= \frac{\gamma}{\mathbf{E}[X_{t_2}]} - 1 = \frac{\gamma}{t_2 \cdot p} - 1.\end{aligned}$$

It is easy to verify that $1/3 \le \epsilon_1 < 1$ and $1/3 \le \epsilon_2 < 1$ (for the latter, we need $\gamma > 4$, which holds given the definition in (13)).

As the event $\rho > 3\gamma/(2p)$ implies $X_{t_1} \le \gamma$, we can derive

$$\begin{aligned}\mathbf{Pr}\left[\rho > \frac{3\gamma}{2p}\right] &\le \mathbf{Pr}\left[X_{t_1} \le \gamma\right] = \mathbf{Pr}\left[X_{t_1} \le (1-\epsilon_1) \cdot \mathbf{E}[X_{t_1}]\right]\\ \text{(by (24))} \quad &\le \exp\left(-\frac{\epsilon_1^2 \cdot \mathbf{E}[X_{t_1}]}{3}\right) = \exp\left(-\frac{\epsilon_1^2 \cdot t_1 \cdot p}{3}\right)\\ &< \exp(-\gamma/36)\end{aligned}$$

where the last step used $\epsilon_1 \ge 1/3$ and $t_1 \ge 3\gamma/(2p)$. This proves (27).

As the event $\rho < 3\gamma/(4p)$ implies $X_{t_2} \ge \gamma$, we can derive

$$\begin{aligned}\mathbf{Pr}\left[\rho < \frac{3\gamma}{4p}\right] &\le \mathbf{Pr}\left[X_{t_2} \ge \gamma\right] = \mathbf{Pr}\left[X_{t_2} \ge (1+\epsilon_2) \cdot \mathbf{E}[X_{t_2}]\right]\\ \text{(by (25))} \quad &\le \exp\left(-\frac{\epsilon_2^2 \cdot \mathbf{E}[X_{t_2}]}{3}\right) = \exp\left(-\frac{\epsilon_2^2 \cdot t_2 \cdot p}{3}\right)\\ &\le \exp(-\gamma/36)\end{aligned}$$

where the last step used $\epsilon_2 \ge 1/3$. This proves (28).

C Proof of Lemma 7.2

Yannakakis' algorithm [35] outputs $Join(\mathcal{Q})$ in $O(\text{IN} + |Join(\mathcal{Q})|)$ time. The algorithm allows one to specify a *source relation* $R \in \mathcal{Q}$. In an $O(\text{IN})$-time preprocessing stage, it

- converts R into a *fully-reduced* state, i.e., eliminating every tuple $\boldsymbol{u} \in R$ with $\boldsymbol{u} \bowtie Join(\mathcal{Q}) = \emptyset$ (i.e., $\boldsymbol{u}$ does not "contribute" to the join result);
- creates a structure that, given any tuple $\boldsymbol{u} \in R$, can report $\boldsymbol{u} \bowtie Join(\mathcal{Q})$ in $O(|\boldsymbol{u} \bowtie Join(\mathcal{Q})|)$ time.

Let $\mathcal{Z}$ be a subset of $\mathcal{V}$ such that $\mathcal{G}$ is ext-$\mathcal{Z}$-connex. It should have become straightforward to combine the above and the discussion in Section 5.2 to obtain a structure with the following guarantees: it uses $O(\text{IN})$ space, can be built in $O(\text{IN})$ expected time, and when given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, can report $\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))$ in $O(1 + |\sigma_{\boldsymbol{z}}(Join(\mathcal{Q}))|)$ time.

Case 1: $\mathcal{G}$ is $\mathcal{Z}$-Canonical. By Definition 3.1, there is a hyperedge $e^* \in \mathcal{E}$ satisfying $\mathcal{Z} \subseteq e^*$. Run the preprocessing stage of Yannakakis's algorithm to create the aforementioned structure Υ by specifying R_{e^*} as the source relation (recall that this is the relation in $\mathcal{Q}$ whose schema is e^*). The preprocessing leaves R_{e^*} in a fully-reduced state. Given a tuple $\boldsymbol{z}$ over $\mathcal{Z}$, we identify with hashing in constant time the set $R_{e^*}(\boldsymbol{z})$ as defined in (7). Then, for each $\boldsymbol{u} \in R_{e^*}(\boldsymbol{z})$, use Υ to report $\boldsymbol{u} \bowtie Join(\mathcal{Q})$ (which must be non-empty).

Case 2: $\mathcal{G}$ Is Not $\mathcal{Z}$-Canonical. Let $\mathcal{T}$ be an arbitrary join tree of $\mathcal{G}$, which defines a number s of $(\mathcal{Z}, \mathcal{T})$-components of $\mathcal{G}$ — denoted as $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), ..., \mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$, respectively — such that $\mathcal{G}_i$ is $(\mathcal{V}_i \cap \mathcal{Z})$-canonical for every $i \in [s]$ (see Lemma 3.3). For each $i \in [s]$, define $\mathcal{Z}_i = \mathcal{V}_i \cap \mathcal{Z}$ and $\mathcal{Q}_i = \{R_e \in \mathcal{Q} \mid e \in \mathcal{E}_i\}$. Note that $\mathcal{Q}_i$ is a join instance of the schema graph $\mathcal{G}_i$, which is $\mathcal{Z}_i$-canonical. For each $i \in [s]$, build a structure Υ_i on $\mathcal{Q}_i$ in the way explained for Case 1. Consider now a tuple $\boldsymbol{z}$ over $\mathcal{Z}$. For each $i \in [s]$, defining $\boldsymbol{z}_i = \boldsymbol{z}[\mathcal{Z}_i]$, we use Υ_i to extract $S_i = \sigma_{\boldsymbol{z}_i}(Join(\mathcal{Q}_i))$. Then, for every $(\boldsymbol{u}_1, ..., \boldsymbol{u}_s) \in S_1 \times ... \times S_s$, output $\boldsymbol{u}_1 \bowtie \boldsymbol{u}_2 \bowtie ... \bowtie \boldsymbol{u}_s$.

D Proof of Lemma 7.3

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the schema graph of $\mathcal{Q}$. Define g, $\mathcal{Z}_1^*, ..., \mathcal{Z}_g^*$, $m_1, ..., m_g$, and $\{\boldsymbol{z}_{i,j}^* \mid i \in [g], j \in [m_i]\}$ as in Section 6. For each $i \in [g]$, let S_i be as defined in (16), and set $S = \bigcup_{i=1}^g S_i$, which is $\sigma_{\boldsymbol{z}_1 \vee ... \vee \boldsymbol{z}_m}(Join(\mathcal{Q}))$ as shown in (17).

For every $i \in [g]$, since $\mathcal{G}$ is ext-$\mathcal{Z}_i^*$-connex, we can create a structure of Lemma 7.2 — denoted as Υ_i^{rep} — that, given any tuple $\boldsymbol{z}_i^*$ over $\mathcal{Z}_i^*$, reports $\sigma_{\boldsymbol{z}_i^*}(Join(\mathcal{Q}))$ in $O(1 + |\sigma_{\boldsymbol{z}_i^*}(Join(\mathcal{Q}))|)$ time. The $g = O(1)$ structures can be built in $O(g \cdot \text{IN}) = O(\text{IN})$ expected time and occupy $O(\text{IN})$ space in total.

For each $i \in [g]$ and $j \in [m_i]$, we use Υ_i^{rep} to retrieve $\sigma_{\boldsymbol{z}_{i,j}^*}(Join(\mathcal{Q}))$ in $O(1 + |\sigma_{\boldsymbol{z}_{i,j}^*}(Join(\mathcal{Q}))|)$ time. The total cost is

$$
\begin{aligned}
& O\left(\sum_{i=1}^{g} \sum_{j=1}^{m_i} (1 + |\sigma_{\boldsymbol{z}_{i,j}^*}(Join(\mathcal{Q}))|)\right) \\
\text{(by (18))} \quad = \quad & O\left(m + \sum_{i=1}^{g} |S_i|\right) = O(m + g \cdot |S|) = O(m + |S|)
\end{aligned}
$$

time. The remaining issue is that every tuple in S can be reported up to g times, once from each of $S_1, ..., S_g$. We need to remove all duplicates and retain only one copy for each tuple. Naively, one could resort to hashing for this purpose, but this would make the overall running time $O(m + |S|)$ expected. In contrast, we aim to report the distinct elements of S in $O(|S|)$ deterministic time (the reduction in Section 7.1 requires listing S with a deterministic time bound), after an initial processing taking $O(m)$ expected time.

We use another idea to remove duplicates: for each tuple $\boldsymbol{u} \in S$, we report it only when it is fetched from $S_{i_{min}}$, where i_{min} is the smallest $i \in [g]$ such that $\boldsymbol{u} \in S_i$. To implement the idea, we need a "membership structure" for each $i \in [g]$ that, given a tuple $\boldsymbol{u}$ over $\mathcal{V}$, tells us whether $\boldsymbol{u} \in S_i$ in constant time. In Section 6, we have already explained how to build these structures, which include

- a perfect-hashing structure on each relation of $\mathcal{Q}$;
- a perfect-hashing structure on $\{\boldsymbol{z}^*_{i,j} \mid j \in [m_i]\}$ for each $i \in [g]$.

The structures in the first bullet are created offline (i.e., when $\mathcal{Q}$ is preprocessed); they occupy $O(\text{IN})$ space and can be built in $O(\text{IN})$ expected time. Those in the second bullet are created online (after receiving $\{\boldsymbol{z}^*_{i,j} \mid i \in [g], j \in [m_i]\}$) in $O(m)$ expected time.

We can now report S without duplicates as follows. As before, for each $i \in [g]$ and $j \in [m_i]$, we use Υ_i^{rep} to retrieve $\sigma_{\boldsymbol{z}^*_{i,j}}(Join(\mathcal{Q}))$. For every tuple $\boldsymbol{u}$ found, we check whether $\boldsymbol{u} \in S_{i'}$ for at least one $i' \in [1, i-1]$ — this checking requires $O(g) = O(1)$ time — and add $\boldsymbol{u}$ to S only if the answer is negative. This completes the proof of Lemma 7.3.