Towards Output-Optimal Uniform Sampling and Approximate Counting for Join-Project Queries

XIAO HU, University of Waterloo, Canada
JINCHAO HUANG, The Chinese University of Hong Kong, Hong Kong SAR

Uniform sampling and approximate counting are fundamental primitives for modern database applications, ranging from query optimization to approximate query processing. While recent breakthroughs have established optimal sampling and counting algorithms for full join queries, a significant gap remains for join-project queries, which are ubiquitous in real-world workloads. The state-of-the-art "propose-and-verify" framework [13] for these queries suffers from fundamental inefficiencies, often yielding prohibitive complexity when projections significantly reduce the output size.

In this paper, we present the first asymptotically optimal algorithms for fundamental classes of join-project queries, including matrix, star, and chain queries. By leveraging a novel rejection-based sampling strategy and a hybrid counting reduction, we achieve polynomial speedups over the state of the art. We establish the optimality of our results through matching communication complexity lower bounds, which hold even against algebraic techniques like fast matrix multiplication. Finally, we delineate the theoretical limits of the problem space. While matrix and star queries admit efficient sublinear-time algorithms, we establish a significantly stronger lower bound for chain queries, demonstrating that sublinear algorithms are impossible in general.

CCS Concepts: • **Theory of computation → Database query processing and optimization (theory)**.

Additional Key Words and Phrases: Join-project query, Fast matrix multiplication, Yannakakis framework

1 Introduction

Joins constitute the backbone of relational database systems, serving as the fundamental primitive for combining data across multiple relations. However, in the era of big data, the sheer volume of join results often renders exact evaluation computationally prohibitive. While worst-case optimal join algorithms [32, 33] have settled the worst-case complexity up to the AGM bound [7], the join size can still be polynomially larger than the input size N (i.e., the number of tuples in the database), reaching up to N^{ρ^*}, where ρ^* is the fractional edge covering number of the underlying join query. Consequently, for many downstream applications—such as approximate query processing [2]—computing the exact result is both expensive and unnecessary. This computational bottleneck has necessitated a shift towards *uniform sampling* and *approximate counting*, which share strong algorithmic connections and complexity characteristics. The goal is to generate random samples from the query result or estimate the output size without materializing the full query result.

For full join queries (where all attributes are retained without projection), these two problems have been extensively studied by the database community [1, 12, 19, 28, 35, 36, 40] and the algorithm community [6, 20]. Acharya et al. [1] and Chaudhuri et al. [12] initiated the investigation of uniform sampling over joins in 1999, identifying the primary barrier: the sampling operator cannot be pushed down, i.e., $\mathsf{sample}(R \bowtie S) \neq \mathsf{sample}(R) \bowtie \mathsf{sample}(S)$. To overcome this, research focused on precomputing indices to guide sampling. Zhao et al. [40] demonstrated that for acyclic joins, an index can be built in $O(N)$ time to support $O(1)$ uniform sampling. More recently, Kim et al. [28] and Deng et al. [19] tackled general joins (including cyclic joins) by showing that an index can be built in $O(N)$ time to support $\tilde{O}\left(\frac{N\rho^*}{\mathsf{OUT}_\bowtie}\right)$ uniform sampling, where $\mathsf{OUT}_\bowtie$ is the size of the full join result. Parallel to uniform sampling, Kim et al. [28] showed that the same algorithmic principles apply to approximate counting. The size of the full join, $\mathsf{OUT}_\bowtie$, can be estimated with the

Authors' Contact Information: Xiao Hu, xiaohu@uwaterloo.ca, University of Waterloo, Waterloo, Ontario, Canada; Jinchao Huang, jchuang@se.cuhk.edu.hk, The Chinese University of Hong Kong, Shatin, Hong Kong SAR.

same complexity once the input is preprocessed to support some basic primitives (see Section 1.1). Moreover, for acyclic joins, this bound can be further improved to $\tilde{O}\left(\min\{N, \frac{N^{\rho^*}}{\mathsf{OUT}_{\bowtie}}\}\right)$ by leveraging the exact counting algorithm [39].

However, a significant gap remains in the literature: real-world analytical queries rarely retain all attributes; they frequently involve projections. A join-project query asks for distinct tuples projected onto a subset of attributes. Unfortunately, the optimal techniques developed for full joins do not generalize to arbitrary projections. The state-of-the-art sampling framework for join-project queries [13] relies on a "propose-and-verify" approach: it proposes a candidate tuple from the *projection-induced join* (the join query induced by projecting input relations onto the output attributes) and subsequently verifies if this candidate extends to a valid answer in the full join. This approach faces two fundamental bottlenecks: (1) *low acceptance probability*: the size of the projection-induced join space is often significantly larger than the actual projected result size OUT; and (2) *high verification cost*: verifying whether a candidate tuple extends to a full join result is computationally expensive, often as costly as evaluating the residual join query induced by non-output attributes. Consequently, this framework yields a sampling complexity of $\tilde{O}\left(\frac{N^{\rho^*}}{\mathsf{OUT}}\right)$. This also remains the state-of-the-art for approximate counting via a reduction to uniform sampling [13].

In this paper, we break the stalemate for join-project queries by developing the first asymptotically output-optimal algorithms for fundamental query classes, including matrix, star, and chain queries (formally defined in Section 1.1). Crucially, our lower bounds are derived from communication complexity, which applies to *any* algorithm, and therefore effectively rule out the possibility of circumvention by algebraic techniques such as fast matrix multiplication. Furthermore, our hardness result for chain queries rules out the possibility of sublinear algorithms for general join-project queries. These results collectively mark a significant step towards characterizing the fine-grained complexity of uniform sampling and approximate counting for general join-project queries.

1.1 Problem Definitions

Join-Project Query. A join-project query can be modeled as a triple $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, where $\mathcal{V}$ is a set of attributes, $\mathcal{E} = \{e_1, e_2, \ldots, e_k\} \subseteq 2^{\mathcal{V}}$ is a set of schemas, and $\mathbf{y} \subseteq \bigcup_{e\in\mathcal{E}} e$ is the set of output attributes. For simplicity, we assume $\mathcal{V} = \bigcup_{e\in\mathcal{E}} e$. Let $\mathrm{dom}(A)$ be the domain of an attribute $A \in \mathcal{V}$. Let $\mathrm{dom}(X) = \prod_{A\in X} \mathrm{dom}(A)$ be the domain of a subset of attributes $X \subseteq \mathcal{V}$. A *tuple* t defined over a subset of attributes $X \subseteq \mathcal{V}$ is a function that assigns a value from $\mathrm{dom}(A)$ to every attribute $A \in X$. An *instance* of Q is a set of relations $\mathcal{R} = \{R_e : e \in \mathcal{E}\}$, where each relation R_e is a set of *tuples* defined over attributes e. For a tuple defined over attributes X and any subset of attributes $Y \subseteq X$, we denote by $\pi_Y t$ the projection of t onto Y. The query result of Q on $\mathcal{R}$ is defined as:

$$Q(\mathcal{R}) = \{t \in \mathrm{dom}(\mathbf{y}) \mid \exists t' \in \mathrm{dom}(\mathcal{V}), \pi_{\mathbf{y}} t' = t, \forall e \in \mathcal{E}, \pi_e t' \in R_e\}.$$

We use $N = \sum_{e\in\mathcal{E}} |R_e|$ to denote the *input size*, i.e., the total number of tuples in input relations, and $\mathsf{OUT} = |Q(\mathcal{R})|$ to denote the *output size*, i.e., the number of tuples in the query result.

In plain language, a join-project query first computes the full join result of the underlying relations in Q, which is all combinations of tuples, one from each relation, that share the same value on all common attributes, and then computes the projection (without duplication) of the full join result onto the output attributes. Join-project queries include many commonly seen database queries as special cases. For example, if we take $\mathcal{V} = \{A, B, C\}$ with $\mathbf{y} = \{A, C\}$, and $\mathcal{E} = \{\{A, B\}, \{B, C\}\}$, it becomes the matrix query (denoted as Q_{matrix}); if we take $\mathcal{V} = \{A_1, A_2, \cdots, A_k, B\}$ with $\mathbf{y} = \{A_1, A_2, \cdots, A_k\}$, and $\mathcal{E} = \{\{A_1, B\}, \{A_2, B\}, \cdots, \{A_k, B\}\}$, it becomes the k-star query (denoted as Q_{star}); and if we take $\mathcal{V} = \{A_1, A_2, \cdots, A_{k+1}\}$ with $\mathbf{y} = \{A_1, A_{k+1}\}$, and $\mathcal{E} = \{\{A_1, A_2\}, \{A_2, A_3\}, \cdots, \{A_k, A_{k+1}\}\}$, it becomes the k-chain query (denoted as Q_{chain}).

An important operator used together with join is semi-join. For two relations R_e and $R_{e'}$, the semi-join $R_e \ltimes R'_e$ finds the set of tuples in R_e that can be joined with at least one tuple from $R_{e'}$. For some tuple $t \in R_{e'}$, the semi-join $R_e \ltimes t$ finds the set of tuples in R_e that can be joined with t.

Uniform Sampling and Approximate Counting for Join-project Queries. In this paper, we are mainly interested in the following two problems. The first concerns building a data structure for generating random samples from the query result:

Problem 1.1 (Uniform Sampling over Join-Project Query)**.** Given a join-project query Q and an instance $\mathcal{R}$, the goal is to build a data structure in linear time that supports efficient generation of uniform samples from the query result $Q(\mathcal{R})$. Additionally, the samples returned to distinct requests must be independent.

The second problem concerns estimating the size of the query result. To quantify the accuracy of our estimation, we first define the standard notion of multiplicative approximation: For a non-negative value V and a parameter $0 < \epsilon < 1$, $\widehat{V}$ is an *ϵ-approximation* of V if $(1-\epsilon)V \leqslant \widehat{V} \leqslant (1+\epsilon)V$.

Problem 1.2 (Approximate Counting for Join-Project Query)**.** Given parameters $0 < \epsilon, \delta < 1$, a join-project query Q and an instance $\mathcal{R}$, the goal is to return an ϵ-approximation $\widetilde{\mathsf{OUT}}$ of the output size OUT with probability at least $1 - \delta$.

For the uniform sampling problem, we are interested in the *sampling time* of generating a uniform sample from the preprocessed data structure. For the approximate counting problem, we are interested in the total runtime. For both problems, we study the data complexity by considering the query size (i.e., k) as constants and measuring the complexity of algorithms by data-dependent quantities, such as input size N and output size OUT, and additional parameters, such as approximation quality ϵ and the failure probability δ.

Model of Computation. We use the standard word RAM model with uniform cost measures [16]. For an instance of size N, the machine operates on words of length $w = \Omega(\log N)$ bits. Accessing a memory location takes $O(1)$ time. Performing basic arithmetical operations (such as addition, multiplication, and division) on two words takes $O(1)$ time. We assume the machine can generate a random integer uniformly from a range $[a, b]$ in $O(1)$ time, provided the range fits within a word. We denote this operation as $\texttt{RandInt}(a, b)$.

In addition, for both problems on graphs (as a special case of self-joins), researchers have adopted the *property testing model* [6], which assumes that the graph has already been stored in some standard graph data structure (e.g., adjacency lists with hash tables), so that some basic primitives are supported in $O(1)$ time. This model was subsequently extended to the relational setting [28], so that the following primitives are supported in $\tilde{O}(1)$ time:

- **(sample tuple)** For a relation $R_e \in \mathcal{R}$, it returns a uniform sample from R_e.
- **(compute degree)** For a relation $R_e \in \mathcal{R}$, a tuple t defined on attributes $e' \subseteq e$, it returns $|R_e \ltimes t|$, i.e., the number of tuples from R_e that match t on attributes e'.
- **(access neighbor)** For a relation $R_e \in \mathcal{R}$, a tuple t defined on attributes $e' \subseteq e$, and an integer j, it returns the j-th tuple in $\pi_{e-e'}(R_e \ltimes t)$ under a fixed ordering.[1]
- **(test tuple)** For a relation $R_e \in \mathcal{R}$ and a tuple t, it returns true if $t \in R_e$ and false otherwise.

Other Notations and Assumptions. We denote $q = (\mathcal{V}, \mathcal{E})$ as the full join of the join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathrm{y})$. Given a join query $q = (\mathcal{V}, \mathcal{E})$, a *fractional edge covering* is a function w that assigns a non-negative weight $w(e)$ to each schema $e \in \mathcal{E}$ such that for every attribute $A \in \mathcal{V}$, the sum of weights of all schema containing A is at least 1, i.e., $\sum_{e \in \mathcal{E}: A \in e} w(e) \geqslant 1$. The *fractional*

[1]The primitives in [28] are more powerful with respect to *access neighbor*: for a relation R_e, a tuple t defined on attributes $e' \subseteq e$, a subset $I \subseteq e - e'$, and an integer j, it returns the j-th tuple in $\pi_I(R_e \ltimes t)$ under a fixed ordering.

edge covering number ρ^* is the minimum possible total weight, $\sum_{e\in\mathcal{E}} w(e)$, over all fractional edge coverings w. Note that $\rho^* = k$ for k-star join, and $\rho^* = \lceil \frac{k+1}{2} \rceil$ for k-chain join.

For an instance $\mathcal{R}$, we denote the size of the full join results of q as $\mathsf{OUT}_{\bowtie}$. The *active domain* of an attribute $A \in \mathcal{V}$, denoted as $\mathrm{adom}_{\mathcal{R}}(A)$, is the set of values from $\mathrm{dom}(A)$ appearing in some tuple of $\mathcal{R}$, i.e., $\mathrm{adom}_{\mathcal{R}}(A) = \{a \in \mathrm{dom}(A) \mid \exists e \in \mathcal{E}, \exists t \in R_e, \pi_A t = a\}$. When the context is clear, we use $\mathrm{adom}(A)$ to denote $\mathrm{adom}_{\mathcal{R}}(A)$. For any $n \in \mathbb{Z}^+$, we use $[n]$ to denote $\{1, 2, \cdots, n\}$. Every logarithm has base 2 by default. For a pair of sets S_1 and S_2, we use $S_1 - S_2 = \{x \in S_1 : x \notin S_2\}$ to denote the set minus operation. We use $\widetilde{O}(\cdot)$ to suppress any poly-logarithmic factors.

Handling Empty Results. For clarity of presentation, we focus on the case when $\mathsf{OUT} \geqslant 1$. The edge case $\mathsf{OUT} = 0$ is handled using a standard timeout strategy [13, 28]. Since the complexities of our algorithms are monotonically non-increasing with respect to OUT (e.g., $O(\frac{N}{\sqrt{\mathsf{OUT}}})$), the worst-case runtime upper bound occurs when $\mathsf{OUT} = 1$. Consequently, the case of $\mathsf{OUT} = 0$ can be detected by running the algorithm with a time budget corresponding to a constant times the complexity for $\mathsf{OUT} = 1$. If the algorithm fails to generate a sample or return an estimate within this budget (repeated for $\tilde{O}(1)$ trials to satisfy error probability δ), we terminate and report $\mathsf{OUT} = 0$ with high probability. This timeout cost satisfies our general complexity bounds. With a slight abuse of notation, we use terms like $O(\frac{N}{\sqrt{\mathsf{OUT}}})$ to denote $O(\frac{N}{\sqrt{\max\{1,\mathsf{OUT}\}}})$ in the paper.

1.2 Previous Results

Recall that full joins are special cases of join-project queries where all attributes are output attributes. An index can be built in $O(N)$ time such that one uniform sample can be generated in $O(1)$ time for acyclic joins[2] [40] and in $\tilde{O}(\frac{N\rho^*}{\mathsf{OUT}})$ time for cyclic joins [19, 28][3]. The previous result for acyclic joins can be extended to the class of free-connex queries[4], since a free-connex query can be transformed into an acyclic full join in $O(N)$ time. However, this is a very limited class of join-project queries; for example, matrix, star, and chain queries are not free-connex.

Chen and Yi [13] first investigated this problem for join-project queries with arbitrary projections. They proposed the "propose-and-verify" framework: first, propose a uniform sample from the *projection-induced full join*; second, accept the candidate if it can be extended to a full join result of q. By integrating the state-of-the-art algorithms [19, 28] for sampling a full join result, an index can be built in $O(N)$ time, such that one uniform sample can be generated in $\tilde{O}(\frac{N\rho^*}{\mathsf{OUT}})$ time, where ρ^* is the fractional edge covering number of the underlying full join q. Hence, the best sampling time is $\tilde{O}(\frac{N^2}{\mathsf{OUT}})$ for matrix queries, $\tilde{O}(\frac{N^k}{\mathsf{OUT}})$ for k-star queries, and $\tilde{O}(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\mathsf{OUT}})$ for k-chain queries.

Chen and Yi [13] also showed a reduction from approximate counting to uniform sampling for general join-project queries. If an index can be built in $O(N)$ time and returns a uniform sample from the query results in $O(1)$ time with probability $\frac{\mathsf{OUT}}{N\rho^*}$, then a constant-approximation of the output size can be returned with constant probability in $\tilde{O}(N + \frac{N\rho^*}{\mathsf{OUT}})$ time.

Additionally, approximate counting and (near) uniform sampling[5] were previously studied by [5] for general join-project queries. Their work established that a fully polynomial-time randomized

[2] A join query $q = (\mathcal{V}, \mathcal{E})$ is α-acyclic [39] if there exists a join tree such that (i) there is a one-to-one correspondence between tree nodes and relations; and (ii) for each attribute, the set of nodes containing it forms a connected subtree.

[3] [28] does not have a preprocessing cost in the (stronger) property testing model. However, preprocessing the extra primitive they introduce still incurs an $O(N)$ cost. For fair comparison, all results remain within the standard property testing model.

[4] A join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ is free-connex [8] if $(\mathcal{V}, \mathcal{E})$ is acyclic and $(\mathcal{V}, \mathcal{E} \cup \{\mathbf{y}\})$ (i.e., by adding another relation containing all output attributes) is also acyclic.

[5] Unlike perfect uniform sampling algorithms, a near-uniform sampling algorithm may not return each query result with strictly same probability. The sampling algorithm we derive for chain queries is also near-uniform.

Problem		Matrix	Star	Chain	
				$k = 3$	$k \geqslant 4$
Uniform Sampling	Prior Work [13, 28]	$O(\frac{N^2}{\mathsf{OUT}})$	$O(\frac{N^k}{\mathsf{OUT}})$	$O(\frac{N^2}{\mathsf{OUT}})$	$O(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\mathsf{OUT}})$
	Our UB	$O(\frac{N}{\sqrt{\mathsf{OUT}}})$ Theorem 2.3	$O(\frac{N}{\mathsf{OUT}^{1/k}})$ Theorem 5.1	$O(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ Theorem 6.2, [13]	$O(N)$ Theorem 6.2
	Our LB	$\Omega(\frac{N}{\sqrt{\mathsf{OUT}}})$ Theorem 4.3	$\Omega(\frac{N}{\mathsf{OUT}^{1/k}})$ Theorem 5.5	$\Omega(N)$ for $\mathsf{OUT} \leqslant N$ Theorem 6.5	$\Omega(N)$ Theorem 6.7
Approximate Counting	Prior Work [13, 28]	$O(\frac{N^2}{\mathsf{OUT}})$	$O(\frac{N^k}{\mathsf{OUT}})$	$O(\frac{N^2}{\mathsf{OUT}})$	$O(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\mathsf{OUT}})$
	Our UB	$O(\frac{N}{\sqrt{\mathsf{OUT}}})$ Theorem 3.8	$O(\frac{N}{\mathsf{OUT}^{1/k}})$ Theorem 5.2	$O(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ Theorem 6.1, [13]	$O(N)$ Theorem 6.1
	Our LB	$\Omega(\frac{N}{\sqrt{\mathsf{OUT}}})$ Theorem 4.2	$\Omega(\frac{N}{\mathsf{OUT}^{1/k}})$ Theorem 5.4	$\Omega(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ Theorem 6.3	$\Omega(N)$ Theorem 6.4

Fig. 1. Comparison between prior results and our new results (in red) in the property testing model. N is the input size, OUT is the output size, and k is the number of relations. For uniform sampling, we assume $O(N)$ preprocessing time. For matrix and star queries, the uniform samplers always return every query result with the exact same probability, while for the chain query, the sampler is near-uniform with high probability. The lower bounds for uniform sampling over chain queries only apply to W-uniform sampling algorithms. For approximate counting, we assume ϵ is a small constant and $\delta = 1/N^{O(1)}$.

approximation scheme exists if and only if the query has bounded hypertree width. In contrast, our work focuses on the fine-grained complexity of these problems, specifically characterizing the runtime in terms of both the input size N and the output size OUT.

1.3 Our Results

Our main results are summarized in Figure 1. All our indices built for uniform sampling use $O(N)$ preprocessing time. For simplicity, the lower bound shown in Figure 1 assumes $O(N)$ preprocessing time. Below, we focus on the sample generation time and approximate counting time.

Section 2 (uniform sampling over matrix query). We begin with uniform sampling over matrix queries. We demonstrate that a uniform sample can be generated in $O\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$ expected time. This strictly outperforms the prior best bound of $O\left(\frac{N^2}{\mathsf{OUT}}\right)$, yielding an improvement factor of $O\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$. Since OUT ranges from 1 to N^2, this implies a polynomial speedup when $\mathsf{OUT} = o(N^2)$.

Section 3 (approximate counting for matrix query). We next address approximate counting. We introduce a novel algorithm based on a hybrid reduction strategy that partitions the input instance and applies tailored sampling algorithms to each partition, ensuring efficient and accurate estimation. This approach runs in $\tilde{O}\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$ time. In contrast, the previous result by the standard

[5] A sampling algorithm is called *W-uniform sampling* if given an index built during preprocessing, a uniform sample from the query result can be drawn with probability $\frac{\mathsf{OUT}}{W}$ for some known value $W > \mathsf{OUT}$. Note that all sampling algorithms presented in this paper fall into this class, as do all prior sampling algorithms for join-project queries that we are aware of.

reduction to uniform sampling [13] requires $\tilde{O}\left(\frac{N^2}{\mathsf{OUT}}\right)$ time. Consequently, our algorithm improves the existing result by a factor of $\tilde{O}\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$.

Section 4 (lower bounds for matrix query). We establish a matching lower bound of $\tilde{\Omega}\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$ for both uniform sampling and approximate counting. First, we derive this bound via a reduction from the set disjointness problem in communication complexity to approximate counting. Notably, this lower bound is information-theoretic and thus applies to *all* algorithms, including non-combinatorial approaches that might leverage fast matrix multiplication. Following the reduction from approximate counting to uniform sampling [13], it translates to the same lower bound for uniform sampling, assuming $\tilde{O}\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$ preprocessing time, for all algorithms. Second, for combinatorial algorithms, we provide an alternative result by resorting to the reduction from query evaluation to uniform sampling [19]. The intuition is that repeatedly drawing samples for $\tilde{O}(\mathsf{OUT})$ iterations would retrieve the full query result with high probability. Since any combinatorial algorithm for evaluating matrix query requires $\Omega\left(N\sqrt{\mathsf{OUT}}\right)$ time [4], a sampling algorithm (assuming $O(N)$ preprocessing time) that improves to $O\left(\frac{N}{\mathsf{OUT}^{\frac{1}{2}+\gamma}}\right)$ for any small constant $\gamma > 0$, would imply a combinatorial evaluation algorithm running in $\tilde{O}\left(N \cdot \mathsf{OUT}^{\frac{1}{2}-\gamma}\right)$ time, which is impossible.

Section 5 (star query). The upper and lower bounds derived for matrix query extend naturally to star queries. We achieve a tight time complexity of $\Theta\left(\frac{N}{\mathsf{OUT}^{1/k}}\right)$ for the k-star query.

Section 6 (chain query). We finally turn to chain queries. For both the uniform sampling and approximate counting problems, we provide a new upper bound $O(N)$ by resorting to the *k-minimum value summary* [14, 15, 24]. Together with the existing upper bound $O(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\mathsf{OUT}})$ [13], we obtain a hybrid upper bound $O(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ for the 3-chain query and $O(N)$ for general chain query with $k \geqslant 4$. For the approximate counting problem, we prove a matching lower bound $\Omega(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ for the 3-chain query and $\Omega(N)$ for general chain queries with $k \geqslant 4$. Following the reduction from approximate counting to uniform sampling [13], it translates to a lower bound of $\Omega(N)$ for general chain queries with $k \geqslant 4$ and for the 3-chain query when $\mathsf{OUT} \leqslant N$, assuming $O(N)$ preprocessing time. All these lower bounds rely on the information-theoretic hardness, thus apply to *all* algorithms, including those that might leverage fast matrix multiplication.

2 Uniform Sampling over Matrix Query

In this section, we present our uniform sampling algorithm for the basic matrix query. In Section 2.1, we show a two-step framework of the sampling algorithm and also compare it with the prior framework. In Section 2.2 and Section 2.3, we show detailed procedures to implement these two steps separately. At last, we give an analysis in Section 2.4.

2.1 Framework

As described in Algorithm 1, our framework is conceptually as simple as two steps. In **step 1** (Lines 1-3), we sample a full join result $s = (a, b, c) \in R_1 \bowtie R_2$ uniformly at random. In **step 2** (Lines 4-5), we accept (a, c) as the final sample with probability $\frac{1}{\deg(a,c)}$, where $\deg(a, c) = |\pi_B(R_1 \ltimes a) \cap \pi_B(R_2 \ltimes c)|$, i.e., the number of full join results participated by (a, c). If no sample is returned (Line 1), we repeat these two steps until a successful sample is returned. We will formally prove why this framework can produce a uniform sample for Q_{matrix} in Section 2.4. The intuition is that for each query result (a, c), it will appear in the uniform sample of full join result with probability $\frac{\deg(a,c)}{\mathsf{OUT}_{\bowtie}}$, since (a, c) will participate in $\deg(a, c)$ full join results, each with a distinct value in $(R_1 \ltimes a) \cap (R_2 \ltimes c)$. As $\deg(a, c)$

Algorithm 1: SampleMatrix($\mathcal{R}$)

Input: An instance $\mathcal{R}$ for the matrix query $Q_{\text{matrix}} = \pi_{A,C} R_1(A,B) \bowtie R_2(B,C)$, with some auxiliary indices built for $\mathcal{R}$ if needed.
Output: A uniform sample of the query result $Q_{\text{matrix}}(\mathcal{R})$.

```
while true do
    s ← JoinSampleMatrix(R);          ▸ Algorithm 2 or Algorithm 4;
    if s ≠ ⊥ then break;
if MatrixAccept(R, s) = true then return (π_A s, π_C s)    ▸ Algorithm 3;
else return ⊥;
```

Algorithm 2: JoinSampleMatrix-H($\mathcal{R}$)

Input: An instance $\mathcal{R}$ of the matrix query $Q_{\text{matrix}} = \pi_{A,C} R_1(A,B) \bowtie R_2(B,C)$
Output: A full join result of $R_1 \bowtie R_2$.

```
// Suppose the following statistics are computed in the preprocessing step:
   For each value b ∈ adom(B), W_b = |R_1 ⋉ b| · |R_2 ⋉ b|; and W = Σ_{b∈adom(B)} W_b.
b ← a random sample from adom(B) with probability W_b / W;
a ← the RandInt(1, |R_1 ⋉ b|)-th value in π_A(R_1 ⋉ b);
c ← the RandInt(1, |R_2 ⋉ b|)-th value in π_C(R_2 ⋉ b);
return (a, b, c);
```

can be different for different query pairs (a,c), the first step does not guarantee a uniform sample of the query result. Therefore, an additional acceptance step is required to rescale the probability. After the second step, every query result (a,c) is accepted with probability $\frac{\deg(a,c)}{\text{OUT}_\bowtie} \cdot \frac{1}{\deg(a,c)} = \frac{1}{\text{OUT}_\bowtie}$, which only depends on the number of full join results.

2.2 Step 1: Sample Full Join Result

Throughout the paper, we will present two different strategies of sampling a full join result from $R_1 \bowtie R_2$. For uniform sampling over matrix query, Algorithm 2 would suffice; hence, we simply focus on this strategy now. Later, when using uniform sampling for approximate counting matrix query, we will introduce another strategy in Algorithm 4 and leave all the details to Section 2.5.

Auxiliary indices. In addition to the operations supported by the property testing model, we need to build additional indices to support the following operations in $O(1)$ time:

- For each value $b \in \text{adom}(B)$, return the "weight" $W_b = |R_1 \ltimes b| \cdot |R_2 \ltimes b|$.
- Return the total weight $W = \sum_{b \in \text{adom}(B)} W_b$.
- Return a random weighted sample b from $\text{adom}(B)$ with probability $\frac{W_b}{W}$.

As described in Algorithm 2, JoinSampleMatrix-H first samples a value $b \in \text{adom}(B)$ with probability proportional to its weight W_b. It then samples a full join tuple containing b uniformly at random. More specifically, it independently samples an A-value a from the neighbors of b in R_1 and a C-value c from the neighbors of b in R_2 uniformly at random. Observe that (a,b,c) is sampled with probability $\frac{W_b}{W} \cdot \frac{1}{|R_1 \ltimes b|} \cdot \frac{1}{|R_2 \ltimes b|} = \frac{1}{W}$. Consequently, (a,b,c) is a uniform sample from $R_1 \bowtie R_2$.

[6]We can implement sampling without replacement efficiently using an online version of Fisher-Yates shuffle [22, 30], which has a cost linear only to the number of samples drawn instead of the total number of elements in the universe, which is $|S|$ in the context of Algorithm 3. The procedure uses an initially empty hash map H to store index swaps. Conceptually,

Algorithm 3: MatrixAccept($\mathcal{R}$, (a, b, c))

Input: An instance $\mathcal{R}$ of $Q_{\text{matrix}} = \pi_{A,C} R_1(A, B) \bowtie R_2(B, C)$, and a full join result $(a, b, c) \in R_1(A, B) \bowtie R_2(B, C)$.

Output: A Boolean value indicating acceptance.

$F \leftarrow 0$;
if $|R_1 \ltimes a| < |R_2 \ltimes c|$ **then** $S \leftarrow \pi_B(R_1 \ltimes a)$;
else $S \leftarrow \pi_B(R_2 \ltimes c)$;
`// We sample from` S `using uniform sampling without replacement`[6].
while *at least one element in* S *is not visited yet* **do**
  $b' \leftarrow$ a random sample from non-visited elements in S;
  if $b' = b$ **then continue**;
  if $(a, b') \notin R_1$ **or** $(b', c) \notin R_2$ **then** $F \leftarrow F + 1$;
  else break;
if `RandInt`$(1, |S|) \leqslant F + 1$ **then return true**;
return false;

2.3 Step 2: Acceptance Check

The main challenge is to implement the acceptance step. Recall that this step takes the full join tuple (a, b, c) generated in Step 1 and decides whether to accept (a, c) as a final sample. To ensure uniformity, we must accept (a, c) with probability $\frac{1}{\deg(a,c)}$, where $\deg(a, c) = |\pi_B(R_1 \ltimes a) \cap \pi_B(R_2 \ltimes c)|$. Explicitly computing $\deg(a, c)$ is too expensive ($O(N)$ worst-case). Instead, we simulate this probability using the following helper toolkit:

Proposition 2.1. *Let Y be a discrete random variable taking values in $[M]$, where M is a positive integer. Let U be a uniform random variable on $[M]$, independent of Y. Then* $\Pr[U \leqslant Y] = \frac{1}{M} \cdot \mathbb{E}[Y]$.

Proof of Proposition 2.1. By the Law of Total Probability and the independence of U and Y: $\Pr[U \leqslant Y] = \sum_{y=1}^{M} \Pr[U \leqslant y] \cdot \Pr[Y = y]$. Since U is uniform on $[M]$, $\Pr[U \leqslant y] = \frac{y}{M}$. Substituting this yields: $\Pr[U \leqslant Y] = \frac{1}{M} \sum_{y=1}^{M} y \cdot \Pr[Y = y] = \frac{\mathbb{E}[Y]}{M}$. □

To apply this proposition, we first identify the smaller set of candidate neighbors, S. Wlog, assume $|\pi_B(R_1 \ltimes a)| \leqslant |\pi_B(R_2 \ltimes c)|$, so we set $S = \pi_B(R_1 \ltimes a)$ and $M = |S|$. Note that $b \in S$ is already known to be a valid join value (a "success") because (a, b, c) was sampled from the join.

We construct the random variable Y by sampling from the *other* elements in S. Specifically, we sample without replacement from $S - \{b\}$ and count the number of "failures" (F) encountered before finding another valid join value $b' \in \pi_B(R_1 \ltimes a) \cap \pi_B(R_2 \ltimes c)$. If no other valid join value exists (i.e., b is unique), we continue until $S - \{b\}$ is exhausted. We define $Y = F + 1$.

Algorithm 3 implements this procedure. It iterates through random samples b' from S. If $b' = b$, it is skipped (ensuring we sample from $S - \{b\}$). If $b' \neq b$, we check if b' connects a and c (i.e., $(b', c) \in R_2$). If it does not, we increment the failure count F. The process stops as soon as a valid b' is found or S is exhausted.

[6] $H[i] = i$ if no value pair has been stored in H with key i. To draw the i-th sample, we generate a uniform random integer j from $\{i, i + 1, \cdots, |S|\}$, and return the $H[j]$-th element in S. Meanwhile, we need to update H by storing (key, value) pair (j, i) into H to effectively swap the chosen index with the one at the i-th position, ensuring it won't be sampled again.

As shown in Section 2.4, the expectation of this specific construction is $\mathbb{E}[Y] = \frac{|S|}{\deg(a,c)}$. Finally, we generate a random integer X uniformly from $[|S|]$. If $X \leqslant F+1$, we return **true**; otherwise, we return **false**. By Proposition 2.1, the acceptance probability is exactly $\frac{1}{|S|} \cdot \frac{|S|}{\deg(a,c)} = \frac{1}{\deg(a,c)}$.

2.4 Analysis

We now prove the correctness and analyze the time complexity of Algorithm 1. First, we focus on Algorithm 3 and introduce the concept of the *negative hypergeometric distribution*:

Definition 2.2 (Negative Hypergeometric Distribution [26])**.** Consider an experiment of drawing balls without replacement from a box containing P balls, of which K are successes. Let a random variable $Z \sim \mathtt{NHG}(r, P, K)$ count the number of failures drawn before obtaining the r-th success. Its expectation is $\mathbb{E}[Z] = \frac{r(P-K)}{K+1}$.

In Algorithm 3, when determining whether to accept the query result (a, c) that is generated from the full join result (a, b, c), we are sampling from a population of $P = |S| - 1$ distinct values (excluding b) with only $K = \deg(a, c) - 1$ successes to retrieve the first ($r = 1$) one. Thus, $F \sim \mathtt{NHG}(1, |S| - 1, \deg(a, c) - 1)$. The expectation of F is: $\mathbb{E}[F] = \frac{|S|}{\deg(a,c)} - 1$. Now we apply Proposition 2.1. Let $Y = F + 1$. We know that $Y \in [|S|]$ and have established that $\mathbb{E}[Y] = \frac{|S|}{\deg(a,c)}$. Algorithm 3 returns **true** if a random integer $U \in [|S|]$ satisfies $U \leqslant Y$. By Proposition 2.1, this occurs with probability: $\Pr[\text{accept}] = \frac{\mathbb{E}[Y]}{|S|} = \frac{1}{|S|} \cdot \frac{|S|}{\deg(a,c)} = \frac{1}{\deg(a,c)}$.

We now return to Algorithm 1. Consider an arbitrary query result $(a, c) \in Q_{\text{matrix}}(\mathcal{R})$. In **one iteration** of the loop, the probability that the sampled full join result projects to (a, c) is $\frac{\deg(a,c)}{\mathsf{OUT}_\bowtie}$. In **Step 2**, such a candidate is accepted with probability $\frac{1}{\deg(a,c)}$. Consequently, in a single iteration, (a, c) is returned with probability $\frac{\deg(a,c)}{\mathsf{OUT}_\bowtie} \cdot \frac{1}{\deg(a,c)} = \frac{1}{\mathsf{OUT}_\bowtie}$. Since this probability is the same for any $(a, c) \in Q_{\text{matrix}}(\mathcal{R})$, the final output (conditioned on acceptance) is a uniform sample of $Q_{\text{matrix}}(\mathcal{R})$.

We now analyze the time complexity of Algorithm 1. The preprocessing step takes $O(N)$ time. After preprocessing, each invocation of Algorithm 2 takes $O(1)$ time. As Algorithm 2 always returns a successful sample (without rejection), the while-loop at Lines 1-3 takes $O(1)$ time. The invocation of Algorithm 3 at Line 4 takes $O(F + 1)$ time. From our analysis above, the expectation $\mathbb{E}[F + 1] = \frac{|S|}{\deg(a,c)} = \frac{\min\{|R_1 \ltimes a|, |R_2 \ltimes c|\}}{\deg(a,c)}$. Hence, Algorithm 1 takes:

$$\mathbb{E}[\text{time per invocation}] = 1 + \sum_{(a,c) \in Q_{\text{matrix}}(\mathcal{R})} \frac{\deg(a,c)}{\mathsf{OUT}_\bowtie} \cdot \frac{\min\{|R_1 \ltimes a|, |R_2 \ltimes c|\}}{\deg(a,c)} \leqslant 1 + \frac{N\sqrt{\mathsf{OUT}}}{\mathsf{OUT}_\bowtie}.$$

expected time (omitting the big-O), where the last inequality follows from the fact that $\min\{x, y\} \leqslant \sqrt{xy}$ and the Cauchy-Schwarz inequality[7]. Algorithm 1 succeeds with probability $\frac{\mathsf{OUT}}{\mathsf{OUT}_\bowtie}$, so the expected number of invocations before getting one successful sample is $\frac{\mathsf{OUT}_\bowtie}{\mathsf{OUT}}$. The total expected cost is the product of these two terms: $O\left(\frac{\mathsf{OUT}_\bowtie}{\mathsf{OUT}} \cdot \left(1 + \frac{N\sqrt{\mathsf{OUT}}}{\mathsf{OUT}_\bowtie}\right)\right) = O\left(\frac{\mathsf{OUT}_\bowtie}{\mathsf{OUT}} + \frac{N}{\sqrt{\mathsf{OUT}}}\right) = O\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$, where the last inequality follows from the fact in [4] that for Q_{matrix}, $\mathsf{OUT}_\bowtie \leqslant N\sqrt{\mathsf{OUT}}$.

Theorem 2.3. *For Q_{matrix}, there is an algorithm that can take as input an arbitrary instance $\mathcal{R}$ of input size N and output size* OUT*, and builds an index in $O(N)$ time such that a uniform sample from the query result $Q_{\text{matrix}}(\mathcal{R})$ can be returned in $O\left(\frac{N}{\sqrt{\mathsf{OUT}}}\right)$ expected time.*

[7] $\sum_{(a,c) \in Q_{\text{matrix}}(\mathcal{R})} \min\{|R_1 \ltimes a|, |R_2 \ltimes c|\} \leqslant \sum_{(a,c) \in Q_{\text{matrix}}(\mathcal{R})} \sqrt{|R_1 \ltimes a| \cdot |R_2 \ltimes c|} \leqslant N\sqrt{\mathsf{OUT}}$.

Algorithm 4: JoinSampleMatrix-L($\mathcal{R}$)

Input: An instance $\mathcal{R}$ of the matrix query $Q_{\text{matrix}} = \pi_{A,C} R_1(A,B) \bowtie R_2(B,C)$.
Output: A full join result of $R_1 \bowtie R_2$ or $\perp$ indicating "failure".
// Suppose the following statistic is computed in the preprocessing step:
$\Delta \geqslant \max_{b \in \text{adom}(B)} |R_2 \ltimes b|$.
$(a,b) \leftarrow$ a uniform sample from R_1;
if $|R_2 \ltimes b| = 0$ **then return** $\perp$;
$j \leftarrow \texttt{RandInt}(1, |R_2 \ltimes b|)$;
$c \leftarrow$ the j-th value in $\pi_C(R_2 \ltimes b)$;
if $\texttt{RandInt}(1,\Delta) > |R_2 \ltimes b|$ **then return** $\perp$;
return (a,b,c);

Standard Amplification Techniques. Specifically, if we set a time budget of twice the expected runtime and restart the algorithm if it does not terminate within that budget, the probability of failure drops exponentially with the number of restarts. Repeating this process $O(\log N)$ times guarantees the result with high probability. This observation extends to our other sampling algorithms.

2.5 A variant of Step 1: Sample Full Join Result

To prepare for the next section on approximately counting matrix queries, we present an alternative strategy for **Step 1** to sample a full join result uniformly at random. This strategy does not require preprocessing to build auxiliary indices; instead, it relies on an upper bound Δ on the maximum degree of B-values in one of the relations. Wlog, we assume $\Delta \geqslant \max_{b \in \text{adom}(B)} |R_2 \ltimes b|$.

As outlined in Algorithm 4, JoinSampleMatrix-L employs rejection sampling. It first samples a tuple (a,b) uniformly at random from R_1, and subsequently samples a value c uniformly at random from the neighbors of b in R_2 (i.e., from $\pi_C(R_2 \ltimes b)$). A specific candidate join result (a,b,c) is initially proposed with probability $\frac{1}{|R_1|} \cdot \frac{1}{|R_2 \ltimes b|}$. However, this distribution is not uniform, as the degree $|R_2 \ltimes b|$ varies across different B-values. To correct for this bias, the algorithm accepts the candidate (a,b,c) with probability $\frac{|R_2 \ltimes b|}{\Delta}$. This is a valid probability since $\Delta \geqslant |R_2 \ltimes b|$. Consequently, the probability of returning a specific tuple (a,b,c) becomes $\frac{1}{|R_1|} \cdot \frac{1}{|R_2 \ltimes b|} \cdot \frac{|R_2 \ltimes b|}{\Delta} = \frac{1}{|R_1| \cdot \Delta}$, which is uniform across all full join results.

The algorithm performs a constant number of primitive operations, so it runs in $O(1)$ time. For any specific full join tuple $t = (a,b,c) \in R_1 \bowtie R_2$, the probability that it is returned is the product of sampling (a,b) from R_1, sampling c from $R_2 \ltimes b$, and passing the rejection check: $\Pr[\text{return } t] = \frac{1}{|R_1|} \cdot \frac{1}{|R_2 \ltimes b|} \cdot \frac{|R_2 \ltimes b|}{\Delta} = \frac{1}{|R_1| \cdot \Delta}$. Since this probability is independent of t, the output distribution is uniform conditioned on not returning $\perp$. Summing this probability over all $t \in R_1 \bowtie R_2$ gives the total success probability $\Pr[\text{success}] = \frac{\text{OUT}_\bowtie}{|R_1| \cdot \Delta}$, which implies $\Pr[\text{return } \perp] = 1 - \frac{\text{OUT}_\bowtie}{|R_1| \cdot \Delta}$.

Theorem 2.4. *Algorithm 4 runs in $O(1)$ time. It returns $\perp$ with probability $1 - \frac{\text{OUT}_\bowtie}{|R_1| \cdot \Delta}$. Conditioned on not returning $\perp$, the output (a,b,c) is a uniform sample from the full join result $R_1 \bowtie R_2$.*

To distinguish between the two sampling strategies, we denote the version utilizing Algorithm 2 as SampleMatrix-H and the version utilizing Algorithm 4 as SampleMatrix-L.

3 Approximate Counting for Matrix Query

In this section, we present our approximate counting algorithm for the basic matrix query. We show the framework of a hybrid strategy of uniform sampling algorithms in Section 3.1 and the detailed algorithms in Section 3.2. All missing materials are provided in Section 3.3.

3.1 Framework

Chen and Yi [13] showed a reduction from approximate counting to uniform sampling for join-project queries. Following Proposition 3.1, Theorem 2.3 immediately yields an algorithm that, with constant probability, returns an ϵ-approximation of OUT in expected time $O\left(N + \frac{N}{\epsilon^2 \cdot \sqrt{\mathsf{OUT}}}\right) = O(N)$, by the following facts: (1) Algorithm 1 returns a query result with probability $\frac{\mathsf{OUT}}{\mathsf{OUT}_{\bowtie}}$; (2) Algorithm 1 runs in $O(1 + \frac{N\sqrt{\mathsf{OUT}}}{\mathsf{OUT}_{\bowtie}})$ expected time. However, in the property-testing model, we seek a truly sublinear algorithm whose cost decreases as the output size OUT grows. This motivates a refined reduction from approximate counting to uniform sampling that explicitly leverages OUT.

Proposition 3.1 ([13]). *Suppose there is a uniform sampling algorithm $\mathcal{A}$ that can successfully return a query result with probability $\frac{\mathsf{OUT}}{W}$. If repeating Y independent invocations of $\mathcal{A}$ and ending up with X successes, then $\frac{X \cdot W}{Y}$ is an unbiased estimator of* OUT. *If $Y = \Omega(\frac{W}{\epsilon^2 \cdot \mathsf{OUT}})$, $\frac{X \cdot W}{Y}$ is an ϵ-approximation of* OUT *with constant probability.*

Our new reduction employs a hybrid strategy using two distinct instantiations of Algorithm 1. Specifically, we partition the values in attribute B into *heavy* and *light* sets based on their degrees in the input relations. This effectively divides the full join space into two parts. For query results witnessed by heavy B-values, we invoke SampleMatrix-H. Recall that SampleMatrix-H requires precomputed statistics for all B-values; this is feasible here because the number of heavy values is bounded due to the input size constraint. For query results witnessed by light B-values, we invoke SampleMatrix-L. Recall that SampleMatrix-L requires a prior upper bound on the degrees, which is naturally provided by the threshold defining the light set. Since the sets of projected query results from these two parts may overlap, our final estimator combines them using a specialized intersection estimation technique. Implementing this idea entails several key challenges:

- First, we do not know OUT unless we get an estimator for it, but our hybrid strategy depends on it. To overcome this barrier, we adopt the common strategy of geometric search for OUT from N^2, $\frac{N^2}{2}$, $\frac{N^2}{4}$ and so on, until we can get an estimator (almost) matching our guess.
- Second, precisely partitioning all B-values into heavy and light sets is infeasible, as computing the degree of every value requires $O(N)$ time. To overcome this bottleneck, we employ random sampling. Specifically, by drawing a subset of $\tilde{O}(N/\Delta)$ random tuples (with replacement) from R_2, we ensure that every B-value with degree at least Δ is detected with high probability. We then verify the true degree of each detected candidate to confirm if it is a heavy B-value.
- Third, by Proposition 3.1, we can obtain accurate estimators for the query results in these two parts separately. However, the cost of each estimator scales inversely with the true result size of its respective part. If one part is too small, a blind application would blow up the cost. We therefore seek an approximate counting protocol with a limited set of calibrated guesses: in the ideal case (neither part too small), it yields accurate estimates for both; otherwise, it returns a safe upper bound without significant loss in overall accuracy.
- Finally, we combine the two estimators. Since the heavy and light parts may produce overlapping query results, we must estimate the size of their intersection to avoid double counting. As neither result set is materialized, naive sampling and membership testing are too expensive. Therefore,

Algorithm 5: ApproxCountMatrix$_{\epsilon,\delta}(\mathcal{R})$

Input: An instance $\mathcal{R}$ of Q_{matrix}, approximation quality ϵ and error δ.
Output: An estimate of $|Q_{\text{matrix}}(\mathcal{R})|$.

$\Lambda \leftarrow N^2$, $k \leftarrow \log \frac{2}{\delta}$;
while $\Lambda \geqslant 1$ **do**
  for $i \in [k]$ **do** $s_i \leftarrow$ ApproxCountMatrixWithGuess$_{\epsilon,\delta/2}(\mathcal{R}, \Lambda)$; ▸ Algorithm 6;
  $s \leftarrow$ the median of $s_1, s_2, \cdots, s_k$;
  if $s \geqslant \Lambda$ **then return** s;
  $\Lambda \leftarrow \frac{\Lambda}{2}$;
return 0;

Algorithm 6: ApproxCountMatrixWithGuess$_{\epsilon,\delta}(\mathcal{R}, \Lambda)$

Input: An instance $\mathcal{R}$ of the matrix query $Q_{\text{matrix}} = \pi_{A,C} R_1(A,B) \bowtie R_2(B,C)$, parameter $\Lambda \in [1, N^2]$, approximation quality ϵ and error δ.
Output: An estimate of $|Q_{\text{matrix}}(\mathcal{R})|$.

$\epsilon' \leftarrow \epsilon/5$;
$B^{\text{heavy}} \leftarrow$ DetectHeavy$_{\delta/4}(\mathcal{R}, \Lambda)$; ▸ Algorithm 7;
$\mathcal{R}^{\text{heavy}} \leftarrow \{\sigma_{B \in B^{\text{heavy}}} R_1, \sigma_{B \in B^{\text{heavy}}} R_2\}$; `// conceptual; simulated by filtering`
$\mathcal{R}^{\text{light}} \leftarrow \{R_1 - \sigma_{B \in B^{\text{heavy}}} R_1, R_2 - \sigma_{B \in B^{\text{heavy}}} R_2\}$; `// conceptual; simulated by filtering`
$\widehat{\text{OUT}}_{\bowtie}^{\text{heavy}} \leftarrow \sum_{b \in B^{\text{heavy}}} |R_1 \ltimes b| \cdot |R_2 \ltimes b|$, $\widehat{\text{OUT}}_{\bowtie}^{\text{light}} \leftarrow N \cdot \sqrt{\Lambda}$;
$s^{\text{heavy}} \leftarrow$ ApproxCountWithThreshold$_{\epsilon',\delta/4}\left(\mathcal{R}^{\text{heavy}}, \widehat{\text{OUT}}_{\bowtie}^{\text{heavy}}, \frac{\epsilon'\Lambda}{16}\right)$; ▸ Algorithm 8;
$s^{\text{light}} \leftarrow$ ApproxCountWithThreshold$_{\epsilon',\delta/4}\left(\mathcal{R}^{\text{light}}, \widehat{\text{OUT}}_{\bowtie}^{\text{light}}, \frac{\epsilon'\Lambda}{16}\right)$; ▸ Algorithm 8;
if $s^{\text{heavy}} \neq \perp$ *and* $s^{\text{light}} \neq \perp$ **then**
  $\beta \leftarrow$ IntersectEstimate$_{\epsilon',\delta/4}(\mathcal{R}, \mathcal{R}^{\text{heavy}}, \mathcal{R}^{\text{light}})$; ▸ Algorithm 9;
  return $s^{\text{heavy}} + s^{\text{light}} - \min\{s^{\text{heavy}}, s^{\text{light}} \cdot \beta\}$;
else if $s^{\text{heavy}} \neq \perp$ *and* $s^{\text{light}} = \perp$ **then return** s^{heavy};
else if $s^{\text{heavy}} = \perp$ *and* $s^{\text{light}} \neq \perp$ **then return** s^{light};
else return 0;

we require a more efficient intersection estimation technique. Furthermore, we must account for the error inherent in this intersection estimate when constructing the final estimator.

3.2 Algorithm

ApproxCountMatrix. Our outermost procedure is presented in Algorithm 5. To address the first challenge, we try a sequence of guesses for the true output size, $\Lambda = N^2, \frac{N^2}{2}, \frac{N^2}{4}, \ldots, 1$. For each guess Λ, we invoke our core routine, ApproxCountMatrixWithGuess, to obtain an estimator conditioned on Λ. To amplify the success probability (as shown in our analysis), we repeat this estimation process $O(\log \frac{1}{\delta})$ times and take the median of these estimators. Given an estimator s based on the guess Λ, we compare it with Λ. If s exceeds Λ, we return it immediately; otherwise, we halve Λ and proceed to the next guess. This process continues until a valid estimate is found or all guesses have been exhausted.

ApproxCountMatrixWithGuess. As described in Algorithm 6, it starts with identifying the heavy values in B, using a primitive called DetectHeavy. A value $b \in \mathsf{adom}(B)$ is *heavy* if it appears in more than $\sqrt{\Lambda}$ tuples in R_2, and *light* otherwise. Since with high probability DetectHeavy correctly finds all heavy values, we simply assume it does so. We denote the set of heavy values as B^{heavy}. We *conceptually* use $B^{\mathsf{light}} = \mathsf{adom}(B) - B^{\mathsf{heavy}}$ to denote the light values, but we do not explicitly list B^{light}, because there might be too many of them. For the same reason, we *conceptually* partition $\mathcal{R}$ into two sub-instances (whose query results may overlap): $\mathcal{R}^{\mathsf{heavy}}$ contains all tuples with heavy B-values and $\mathcal{R}^{\mathsf{light}}$ contains the remaining input tuples. For $\mathcal{R}^{\mathsf{heavy}}$, we can exactly calculate the number of full join results. For $\mathcal{R}^{\mathsf{light}}$, we know that the number of full join results is at most $N\sqrt{\Lambda}$, since there are at most N tuples in R_1, and each of them joins with at most Λ tuples in R_2. That is essentially how we set up $\widehat{\mathsf{OUT}}_{\bowtie}^{\mathsf{heavy}}$ and $\widehat{\mathsf{OUT}}_{\bowtie}^{\mathsf{light}}$ as the upper bounds of full join results in each part.

Then, we rely on a separate procedure, ApproxCountWithThreshold, to estimate the number of query results in each part. This procedure works like this: if the actual count is larger than a predefined threshold τ, it successfully returns an ϵ-approximation estimator s. If the procedure reports a failure ($\perp$), it means the actual count must be less than the threshold τ. Using the feedback $(s^{\mathsf{heavy}}, s^{\mathsf{light}})$ on the two parts from ApproxCountWithThreshold, we combine them into a final estimator. Let $\mathsf{OUT}^{?}$ be the number of query results produced by $\mathcal{R}^{?}$, where ? is either heavy or light. We further distinguish the three cases.

- **Case 1: both estimates are successful ($s^{\mathsf{heavy}} \neq \perp$ and $s^{\mathsf{light}} \neq \perp$).** By the design of ApproxCountWithThreshold, s^{heavy} and s^{light} are accurate estimates of $\mathsf{OUT}^{\mathsf{heavy}}$ and $\mathsf{OUT}^{\mathsf{light}}$, respectively, and both counts exceed the threshold. Since the result sets $\mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{heavy}})$ and $\mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{light}})$ may overlap, their sum would double-count the intersection. We employ the procedure IntersectEstimate to estimate β, defined as the fraction of $\mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{light}})$ that is also present in $\mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{heavy}})$. Consequently, $s^{\mathsf{light}} \cdot \beta$ estimates the size of this intersection. However, stochastic errors in estimating β could yield an intersection estimate larger than $\mathsf{OUT}^{\mathsf{heavy}}$, which would destabilize the final result. To mitigate this, we leverage the fact that the intersection size cannot exceed $\mathsf{OUT}^{\mathsf{heavy}}$, yielding the robust combined estimator $s^{\mathsf{heavy}} + s^{\mathsf{light}} - \min\{s^{\mathsf{light}} \cdot \beta, s^{\mathsf{heavy}}\}$.
- **Case 2: Only one estimate fails.** Without loss of generality, assume $s^{\mathsf{light}} = \perp$. This failure implies that $\mathsf{OUT}^{\mathsf{light}}$ is negligible (below the threshold). Consequently, s^{heavy}, which accurately estimates $\mathsf{OUT}^{\mathsf{heavy}}$, serves as a sufficient approximation for the total output size OUT.
- **Case 3: Both estimates fail ($s^{\mathsf{heavy}} = \perp$ and $s^{\mathsf{light}} = \perp$).** These failures indicate that both $\mathsf{OUT}^{\mathsf{heavy}}$ and $\mathsf{OUT}^{\mathsf{light}}$ are below the threshold, implying that their union is at most $\frac{\epsilon'\Lambda}{8}$. As this is significantly smaller than the current guess Λ, we conclude that Λ is an overestimate and return 0 (signaling the algorithm to proceed to the next guess).

DetectHeavy. As described in Algorithm 7, it randomly samples $\tilde{O}(\frac{N}{\sqrt{\Lambda}})$ tuples from R_2, and collects B-values of all sampled tuples as the set B^{heavy}. Also, it checks the true degree of values in B^{heavy}, so that $B^{\mathsf{heavy}} \subseteq \{b \in \mathsf{adom}(B) : |R_2 \ltimes b| \geqslant \sqrt{\Lambda}\}$. Hence, we can bound the size of B^{heavy} as $O(\frac{N}{\sqrt{\Lambda}})$. As proved later, with high probability, every B-value with degree larger than $\sqrt{\Lambda}$ in R_2 is detected, hence $B^{\mathsf{heavy}} = \{b \in \mathsf{adom}(B) : |R_2 \ltimes b| \geqslant \sqrt{\Lambda}\}$.

Simulating Heavy and Light Sub-instances. As $\mathcal{R}^{\mathsf{heavy}}$ and $\mathcal{R}^{\mathsf{light}}$ are not materialized, we simulate access to them by reusing the original indices and storing B^{heavy} in a hash table to support membership checks. When executing SampleMatrix on $\mathcal{R}^{\mathsf{light}}$, we enforce the following:

Algorithm 7: DetectHeavy$_\delta(\mathcal{R}, \Lambda)$

Input: An instance $\mathcal{R}$ of $Q_{\text{matrix}} = \pi_{A,C} R_1(A,B) \bowtie R_2(B,C)$, a parameter Λ and error δ.
Output: A subset of values in $\text{adom}(B)$ that appears in more than $\sqrt{\Lambda}$ tuples in R_2.

$k \leftarrow \frac{N \cdot \log \frac{N}{\delta}}{\sqrt{\Lambda}}$, $B^{\text{heavy}} \leftarrow \emptyset$;
for $i \in [k]$ **do**
  $(b, c) \leftarrow$ a sample drawn from R_2 uniformly at random;
  if $|R_2 \ltimes b| > \sqrt{\Lambda}$ **then** $B^{\text{heavy}} \leftarrow B^{\text{heavy}} \cup \{b\}$;
return B^{heavy};

Algorithm 8: ApproxCountWithThreshold$_{\epsilon,\delta}(\mathcal{R}, \widehat{\text{OUT}}_\bowtie, \tau)$

Input: An instance $\mathcal{R}$ of $Q_{\text{matrix}} = \pi_{A,C} R_1(A,B) \bowtie R_2(B,C)$, an upper bound $\widehat{\text{OUT}}_\bowtie$ of the full join size $|R_1 \bowtie R_2|$ and threshold τ.
Output: An estimator of $|Q_{\text{matrix}}(\mathcal{R})|$ or $\perp$ indicating failure.

$\lambda \leftarrow N^2$, $k_\delta \leftarrow 3 \log \frac{2}{\delta}$;
while $\lambda \geqslant \tau$ **do**
  for $i \in [k_\delta]$ **do**
    $Z \leftarrow 0$, $k_\lambda \leftarrow \frac{6\widehat{\text{OUT}}_\bowtie}{\epsilon^2 \lambda}$;
    for $j \in [k_\lambda]$ **do**
      // SampleMatrix-H for $\mathcal{R}^{\text{heavy}}$ and SampleMatrix-L for $\mathcal{R}^{\text{light}}$;
      $s \leftarrow$ SampleMatrix $(\mathcal{R})$; ▸ Algorithm 1;
      if $s \neq \perp$ **then** $Z \leftarrow Z + \widehat{\text{OUT}}_\bowtie$;
    $s_i \leftarrow \frac{Z}{k_\lambda}$;
  $s \leftarrow$ the median of $s_1, s_2, \cdots, s_{k_\delta}$;
  if $s \geqslant \lambda$ **then return** s;
  $\lambda \leftarrow \frac{\lambda}{2}$;
return $\perp$;

- In **Step 1** (Algorithm 4), if a tuple (a, b) is sampled from R_1 such that $b \in B^{\text{heavy}}$, we treat it as a rejection (return $\perp$). This results in a sampling probability of $\frac{1}{N\Delta}$ for each full join result, but this still suffices for our analysis.
- In **Step 2** (Algorithm 3), when sampling a neighbor b' from the neighbor list $S = \pi_B(\sigma_{B \notin B^{\text{heavy}}}(R_1 \ltimes a))$, we simply sample values from $\pi_B(R_1 \ltimes a)$ and skip any b' if $b' \in B^{\text{heavy}}$. As analyzed later, this won't increase our complexity asymptotically.

When executing SampleMatrix on $\mathcal{R}^{\text{heavy}}$, we enforce the following:

- In **Step 1** (Algorithm 2) We simply focus on B^{heavy} instead of $\text{adom}(B)$. More specifically, we compute W_b for each $b \in B^{\text{heavy}}$ and $W = \sum_{b \in B^{\text{heavy}}} W_b$ in the preprocessing step.
- In **Step 2** (Algorithm 3), when sampling a neighbor b' from the neighbor list $S = \pi_B(\sigma_{B \in B^{\text{heavy}}}(R_1 \ltimes a))$, we simply sample values from $\pi_B(R_1 \ltimes a)$ and skip any b' if $b' \notin B^{\text{heavy}}$. As analyzed later, this won't increase our complexity asymptotically.

All these adaptations make the probabilities correct while still having nice time complexities.

Algorithm 9: IntersectEstimate$_{\epsilon,\delta}(\mathcal{R}, \mathcal{R}^{\text{heavy}}, \mathcal{R}^{\text{light}})$

Input: An instance $\mathcal{R}$ of the matrix query $Q_{\text{matrix}} = \pi_{A,C} R_1(A,B) \bowtie R_2(B,C)$, and a partition of $\mathcal{R} = (\mathcal{R}^{\text{heavy}}, \mathcal{R}^{\text{light}})$ based on a partition of attribute $B = (B^{\text{heavy}}, B^{\text{light}})$.

Output: An estimator of $\frac{|Q_{\text{matrix}}(\mathcal{R}^{\text{heavy}}) \cap Q_{\text{matrix}}(\mathcal{R}^{\text{light}})|}{|Q_{\text{matrix}}(\mathcal{R}^{\text{light}})|}$.

```
S, S' ← ∅;
while |S| ⩽ (1/(2ε²)) log(2/δ) do
    while true do
        s ← SampleMatrix-L(R^light);                ▸ Algorithm 1;
        if s ≠ ⊥ then S ← S ∪ {s} and break;
    for each b ∈ B^heavy do
        if (π_A s, b) ∈ R_1 and (b, π_C s) ∈ R_2 then S' ← S' ∪ {s};
return |S'|/|S|;
```

ApproxCountWithThreshold. As described in Algorithm 8, the high-level idea is very similar to Algorithm 5. We try a sequence of guesses for the true output size, $\lambda = N^2, \frac{N^2}{2}, \frac{N^2}{4}, \ldots, \tau$, where τ is the smallest guess we can try. This threshold is enforced to satisfy the time complexity constraints. Note that this guess λ is independent of the guess Λ in the outermost procedure Algorithm 5, since this is for estimating $\mathsf{OUT}^{\text{heavy}}, \mathsf{OUT}^{\text{light}}$ separately, which can be very different from OUT. For each guess λ, we simply apply the reduction in Proposition 3.1 as Lines 3-9. Recall that the MatrixSample procedure returns a uniform sample of the query result from $Q_{\text{matrix}}(\mathcal{R})$ with probability $\frac{\mathsf{OUT}}{\widehat{\mathsf{OUT}}_{\bowtie}}$. Repeating this procedure $O(\frac{\widehat{\mathsf{OUT}}_{\bowtie}}{\epsilon^2 \lambda})$ independent times should give us a constant-approximation with at least constant probability when $\lambda = O(\mathsf{OUT})$. To amplify the success probability (as shown in our analysis), we repeat this estimation process $O(\log \frac{1}{\delta})$ times and take the median of these estimators. If s exceeds λ, we return it immediately; otherwise, we halve λ and proceed to the next guess. This process continues until a valid estimate is found or all guesses have been exhausted.

IntersectEstimate. As described in Algorithm 9, it takes a set (S) of uniform samples from $Q_{\text{matrix}}(\mathcal{R}^{\text{light}})$ through Algorithm 1, and then checks the subset (S') of sampled query results that are also part of the query results $Q_{\text{matrix}}(\mathcal{R}^{\text{heavy}})$, by iterating through every value in B^{heavy}. When taking a uniform sample from $Q_{\text{matrix}}(\mathcal{R}^{\text{light}})$, it hits the overlap with probability $\frac{|Q_{\text{matrix}}(\mathcal{R}^{\text{heavy}}) \cap Q_{\text{matrix}}(\mathcal{R}^{\text{light}})|}{\mathsf{OUT}^{\text{light}}}$. Hence, $\frac{|S'|}{|S|} \cdot \mathsf{OUT}^{\text{light}}$ serves as an unbiased estimator of the overlap. As we prove later, as long as a sufficiently large number of query results are sampled from $Q_{\text{matrix}}(\mathcal{R}^{\text{light}})$, we can ensure an ϵ additive error on the approximation with probability at least $1 - \delta$.

3.3 Analysis

We begin by analyzing the approximation quality of our counting algorithm by establishing the key properties of its helper procedures. Specifically, we prove Lemmas 3.2 through 3.5, and finally derive Lemma 3.6 as a natural culmination.

Lemma 3.2. *In Algorithm 7, with probability at least* $1-\delta$, $B^{\text{heavy}} = \{b \in \text{adom}(B) : |R_2 \ltimes b| \geqslant \sqrt{\Lambda}\}$.

Proof of Lemma 3.2. As mentioned, $B^{\text{heavy}} \subseteq \{b \in \text{adom}(B) : |R_2 \ltimes b| \geqslant \sqrt{\Lambda}\}$ always holds. It suffices to show that $\{b \in \text{adom}(B) : |R_2 \ltimes b| \geqslant \sqrt{\Lambda}\} \subseteq B^{\text{heavy}}$. Consider an arbitrary value

$b \in \mathrm{adom}(B)$ with $|R_2 \ltimes b| \geqslant \sqrt{\Lambda}$. In each attempt, the probability that no tuple in $R_2 \ltimes b$ is sampled is $1 - \frac{|R_2 \ltimes b|}{|R_2|}$. Then, in $k = \frac{N \cdot \log \frac{N}{\delta}}{\sqrt{\Lambda}}$ independent attempts, the probability that no tuple in $R_2 \ltimes b$ is sampled is $\left(1 - \frac{|R_2 \ltimes b|}{|R_2|}\right)^{\frac{N \cdot \log \frac{N}{\delta}}{\sqrt{\Lambda}}} \leqslant \left(1 - \frac{\sqrt{\Lambda}}{N}\right)^{\frac{N \cdot \log \frac{N}{\delta}}{\sqrt{\Lambda}}} \leqslant \exp(-\log \frac{N}{\delta}) \leqslant \frac{\delta}{N}$. So, b is not detected with probability at most $\frac{\delta}{N}$. As there are at most N such values, by union bound, the probability that at least one of them is not detected is at most δ. □

Lemma 3.3. *In Algorithm 8, the following holds for an arbitrary input instance $\mathcal{R}$ of input size N and output size* OUT*:*

- *When $\tau \leqslant \frac{1}{2} \cdot \mathsf{OUT}$, Algorithm 8 returns $s \neq \perp$ with probability at least $1 - \delta$;*
- *When $s \neq \perp$ is returned, we always have $\mathbb{E}[s] = \mathsf{OUT}$.*
- *When $s \neq \perp$ is returned, s is an ϵ-approximation of* OUT *with probability at least $1 - \delta$.*

Proof of Lemma 3.3. Now, consider an arbitrary iteration of the while-loop with guess λ. We show that $\mathbb{E}[s_i] = \mathsf{OUT}$. Each invocation of SampleMatrix successfully returns any sample with probability $\frac{\mathsf{OUT}}{\widehat{\mathsf{OUT}}_{\bowtie}}$. Let X be the number of times when SampleMatrix succeeds out of k_λ attempts. Hence, $\mathbb{E}[X] = \frac{k_\lambda \cdot \mathsf{OUT}}{\widehat{\mathsf{OUT}}_{\bowtie}}$ and $\mathsf{Var}[X] \leqslant \frac{k_\lambda \cdot \mathsf{OUT}}{\widehat{\mathsf{OUT}}_{\bowtie}}$. Hence, $\mathbb{E}[s_i] = \frac{\widehat{\mathsf{OUT}}_{\bowtie}}{k_\lambda} \cdot \mathbb{E}[X] = \mathsf{OUT}$. Hence, if Algorithm 8 returns an estimator s, we must have $\mathbb{E}[s] = \mathsf{OUT}$. By the Markov inequality, $\Pr[s_i \geqslant 2\mathsf{OUT}] \leqslant \frac{1}{2}$. Taking the median of k_δ such estimators,

$$\Pr[s \leqslant 2\mathsf{OUT}] \geqslant 1 - (\frac{1}{2})^{3 \log \frac{2}{\delta}} = 1 - (\delta/2)^3 \geqslant 1 - \delta/2.$$

Furthermore, the variance of s_i is

$$\mathsf{Var}[s_i] = (\frac{1}{k_\lambda})^2 \cdot \mathsf{Var}[Z] = (\frac{\widehat{\mathsf{OUT}}_{\bowtie}}{k_\lambda})^2 \cdot \mathsf{Var}[X] \leqslant (\frac{\widehat{\mathsf{OUT}}_{\bowtie}}{k_\lambda})^2 \cdot \frac{k_\lambda \cdot \mathsf{OUT}}{\widehat{\mathsf{OUT}}_{\bowtie}} = \frac{\epsilon^2 \lambda}{6} \cdot \mathsf{OUT}.$$

By Chebyshev inequality, $\Pr[|s_i - \mathsf{OUT}| \geqslant \epsilon \mathsf{OUT}] \leqslant \frac{\mathsf{Var}[s_i]}{\epsilon^2 \cdot \mathsf{OUT}^2} \leqslant \frac{\lambda}{6\mathsf{OUT}}$. Taking the median of k_δ such estimators,

$$\Pr[|s - \mathsf{OUT}| \leqslant \epsilon \mathsf{OUT}] \geqslant 1 - (\frac{\lambda}{6\mathsf{OUT}})^{3 \log \frac{2}{\delta}}.$$

If $\lambda \leqslant 4\mathsf{OUT}$, then $\Pr[|s - \mathsf{OUT}| \leqslant \epsilon \mathsf{OUT}] \geqslant 1 - (1 - \frac{1}{3})^{3 \log \frac{2}{\delta}} \geqslant 1 - \exp(-\log \frac{2}{\delta}) \geqslant 1 - \delta/2$.

Assume $\tau \leqslant \frac{1}{2}\mathsf{OUT}$. If Algorithm 8 returns $\perp$, there must be a choice of $\lambda^* \in [\frac{\mathsf{OUT}}{2}, \mathsf{OUT}]$ tried. Let us focus on the s calculated by the iteration of the while-loop with λ^*. From the analysis above, $\Pr[|s - \mathsf{OUT}| \leqslant \epsilon \mathsf{OUT}] \geqslant 1 - \delta/2$. As $\lambda^* \leqslant \mathsf{OUT}$, we have

$$\Pr[s \geqslant \lambda^*] \geqslant \Pr[s \geqslant \mathsf{OUT}] \geqslant \Pr[s \geqslant (1 - \epsilon)\mathsf{OUT}] \geqslant 1 - \delta/2 \geqslant 1 - \delta,$$

hence s should be returned by triggering Line 10 with probability at least $1 - \delta$. If Algorithm 8 returns $\perp$ with probability at least δ, a contradiction would occur. Thus, the first bullet is proved.

Now, assume an estimator s^* is returned, together with the guess λ^* in this specific iteration of the while-loop. There must be $s^* \geqslant \lambda^*$ implied by Line 10. From the analysis above, with probability at least $1 - \delta/2$, $\lambda^* \leqslant 2\mathsf{OUT}$. If this holds, $\Pr[|s - \mathsf{OUT}| \leqslant \epsilon \mathsf{OUT}] \geqslant 1 - \delta/2$. By the union bound, s is an ϵ-approximation of OUT with probability at least $1 - \delta$. □

Lemma 3.4. *In Algorithm 9, with probability at least $1 - \delta$, $\left|\frac{|S'|}{|S|} - \frac{|\mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{heavy}}) \cap \mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{light}})|}{\mathsf{OUT}^{\mathsf{light}}}\right| \leqslant \epsilon$.*

Proof of Lemma 3.4. Let $J = \mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{light}})$ be the set of query results from the light part, and let $I = \mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{heavy}}) \cap \mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{light}})$ be the intersection. Let $p = \frac{|I|}{|J|}$ be the true fraction of

the intersection. Algorithm 9 draws a set S of independent uniform samples from J. For each sample $s_i \in S$, let X_i be an indicator random variable such that $X_i = 1$ if $s_i \in I$ (i.e., $s_i \in S'$), and $X_i = 0$ otherwise. Then X_i follows a Bernoulli distribution with parameter p, and $\mathbb{E}[X_i] = p$. The algorithm returns the estimator $\hat{p} = \frac{|S'|}{|S|} = \frac{1}{|S|}\sum_{s_i \in S} X_i$. Let $M = |S| = \frac{1}{2\epsilon^2}\log\frac{2}{\delta}$ be the sample size. By Hoeffding's inequality, for any $t > 0$: $\Pr[|\hat{p} - p| \geqslant t] \leqslant 2\exp(-2Mt^2)$. Setting $t = \epsilon$, $\Pr[|\hat{p} - p| \geqslant \epsilon] \leqslant 2\exp(-2\frac{1}{2\epsilon^2}\log\frac{2}{\delta} \cdot \epsilon^2) \leqslant \delta$. This implies that $|\hat{p} - p| \leqslant \epsilon$ with probability at least $1 - \delta$. □

Lemma 3.5. *Algorithm 6 returns an estimator s_Λ with $\mathbb{E}[s_\Lambda] \leqslant 2\mathsf{OUT}$. Furthermore, if $\Lambda \leqslant 4\mathsf{OUT}$, s_Λ is an ϵ-approximation estimator for OUT with probability at least $1 - \delta$.*

Proof of Lemma 3.5. Notice that Line 2 succeeds or not only affects the running time, not the accuracy. Implied by Lemma 3.3, if s^{heavy} (resp., s^{light}) is not assigned $\perp$ at Line 6 (resp., Line 7), then $\mathbb{E}[s^{\mathsf{heavy}}] = \mathsf{OUT}^{\mathsf{heavy}}$ (resp., $\mathbb{E}[s^{\mathsf{light}}] = \mathsf{OUT}^{\mathsf{light}}$). Even if both s^{heavy} and s^{light} are not $\perp$, we have $\mathbb{E}[s_\Lambda] \leqslant \mathbb{E}[s^{\mathsf{heavy}} + s^{\mathsf{light}}] = \mathsf{OUT}^{\mathsf{heavy}} + \mathsf{OUT}^{\mathsf{light}} \leqslant 2\mathsf{OUT}$. Analyze the other 3 cases of s^{heavy} and s^{light} analogously, we also have $\mathbb{E}[s_\Lambda] \leqslant 2 \cdot \mathsf{OUT}$.

From above, we have $\Pr[s_\Lambda > 4\mathsf{OUT}] < 1/2$. Now, suppose $\Lambda \leqslant 4\mathsf{OUT}$. We point out the fact that $\max\{\mathsf{OUT}^{\mathsf{heavy}}, \mathsf{OUT}^{\mathsf{light}}\} \geqslant \frac{\mathsf{OUT}}{2} \geqslant \frac{\Lambda}{8} \geqslant \frac{\epsilon'\Lambda}{8}$. Thus, at least one of $\mathsf{OUT}^{\mathsf{heavy}}$, $\mathsf{OUT}^{\mathsf{light}}$ is larger than $2 \cdot \frac{\epsilon'\Lambda}{16} = \frac{\epsilon'\Lambda}{8}$. Implied by Lemma 3.3, at least one of $s^{\mathsf{heavy}}, s^{\mathsf{light}}$ is not $\perp$ with probability at least $1 - \delta/4$. Conditioned on this, s_Λ is returned from the first three cases:

Case 1: $s^{\mathsf{heavy}} \neq \perp$ and $s^{\mathsf{light}} \neq \perp$.

Conditioned on Line 9 succeeding, we further analyze:

- **Case 1.1: $s^{\mathsf{heavy}} \geqslant \beta \cdot s^{\mathsf{light}}$.** In this subcase, $s_\Lambda = s^{\mathsf{heavy}} + s^{\mathsf{light}}(1 - \beta)$. We know that $s^{\mathsf{light}} \cdot \beta$ is an estimator for the intersection size $I = |\mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{heavy}}) \cap \mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{light}})|$. Specifically,

$$\begin{aligned}|s^{\mathsf{light}}\beta - I| &\leqslant |s^{\mathsf{light}}\beta - \mathsf{OUT}^{\mathsf{light}}\beta| + \left|\mathsf{OUT}^{\mathsf{light}}\beta - I\right| \\ &\leqslant \epsilon' \cdot \mathsf{OUT}^{\mathsf{light}} \cdot \beta + \mathsf{OUT}^{\mathsf{light}} \cdot \epsilon' \leqslant \epsilon'(1+\epsilon')\mathsf{OUT}^{\mathsf{light}} + \epsilon'\mathsf{OUT}^{\mathsf{light}} \leqslant 3\epsilon'\mathsf{OUT}^{\mathsf{light}}.\end{aligned}$$

  The total error of the estimator s_Λ is:

$$\begin{aligned}|s_\Lambda - \mathsf{OUT}| &= |(s^{\mathsf{heavy}} + s^{\mathsf{light}} - s^{\mathsf{light}}\beta) - (\mathsf{OUT}^{\mathsf{heavy}} + \mathsf{OUT}^{\mathsf{light}} - I)| \\ &\leqslant |s^{\mathsf{heavy}} - \mathsf{OUT}^{\mathsf{heavy}}| + |s^{\mathsf{light}} - \mathsf{OUT}^{\mathsf{light}}| + |I - s^{\mathsf{light}}\beta| \\ &\leqslant \epsilon'\mathsf{OUT}^{\mathsf{heavy}} + \epsilon'\mathsf{OUT}^{\mathsf{light}} + 3\epsilon'\mathsf{OUT}^{\mathsf{light}} \leqslant 5\epsilon'\mathsf{OUT} = \epsilon\mathsf{OUT}.\end{aligned}$$

  Thus, s_Λ is an ϵ-approximation of OUT.
- **Case 1.2: $s^{\mathsf{heavy}} < \beta \cdot s^{\mathsf{light}}$.** In this subcase, $s_\Lambda = s^{\mathsf{light}}$. The condition $s^{\mathsf{heavy}} < s^{\mathsf{light}}\beta$ implies that the estimated intersection is larger than the estimated heavy set. Since the true intersection I is a subset of $\mathcal{Q}_{\mathsf{matrix}}(\mathcal{R}^{\mathsf{heavy}})$, we have $I \leqslant \mathsf{OUT}^{\mathsf{heavy}}$. Using the bounds from Case 1.1, we have:

$$(1 - \epsilon')\mathsf{OUT}^{\mathsf{heavy}} \leqslant s^{\mathsf{heavy}} < s^{\mathsf{light}}\beta \leqslant I + 3\epsilon'\mathsf{OUT}^{\mathsf{light}}.$$

  This implies $\mathsf{OUT}^{\mathsf{heavy}} - I < \epsilon'\mathsf{OUT}^{\mathsf{heavy}} + 3\epsilon'\mathsf{OUT}^{\mathsf{light}}$. The true output size is $\mathsf{OUT} = \mathsf{OUT}^{\mathsf{light}} + (\mathsf{OUT}^{\mathsf{heavy}} - I)$. Substituting the bound above:

$$\mathsf{OUT}^{\mathsf{light}} \leqslant \mathsf{OUT} < \mathsf{OUT}^{\mathsf{light}} + \epsilon'\mathsf{OUT}^{\mathsf{heavy}} + 3\epsilon'\mathsf{OUT}^{\mathsf{light}} \leqslant \mathsf{OUT}^{\mathsf{light}} + 4\epsilon'\mathsf{OUT}.$$

  Since s^{light} is an ϵ'-approximation of $\mathsf{OUT}^{\mathsf{light}}$, we have:

$$s^{\mathsf{light}} \leqslant (1+\epsilon')\mathsf{OUT}^{\mathsf{light}} \leqslant (1+\epsilon')\mathsf{OUT} \leqslant (1+\epsilon)\mathsf{OUT}, \text{ and}$$

$$s^{\mathsf{light}} \geqslant (1-\epsilon')\mathsf{OUT}^{\mathsf{light}} \geqslant (1-\epsilon')(\mathsf{OUT} - 4\epsilon'\mathsf{OUT}) \geqslant (1-5\epsilon')\mathsf{OUT} = (1-\epsilon)\mathsf{OUT}.$$

  Thus, $s_\Lambda = s^{\mathsf{light}}$ is an ϵ-approximation of OUT.

Case 2: $s^{\text{heavy}} \neq \perp$ and $s^{\text{light}} = \perp$. In this case, $s_\Lambda = s^{\text{heavy}}$. Following Lemma 3.3, with probability at least $1-\delta/4$, $\mathsf{OUT}^{\text{light}} < \frac{\epsilon'\Lambda}{8}$, and s^{heavy} is an ϵ'-approximation of $\mathsf{OUT}^{\text{heavy}}$. Moreover, $\mathsf{OUT}^{\text{heavy}} \geqslant \frac{\mathsf{OUT}}{2} \geqslant \frac{\Lambda}{8}$. We work out an upper bound of OUT in terms of $\mathsf{OUT}^{\text{heavy}}$:

$$\mathsf{OUT} \leqslant \mathsf{OUT}^{\text{heavy}} + \mathsf{OUT}^{\text{light}} \leqslant \mathsf{OUT}^{\text{heavy}} + \frac{\epsilon'\Lambda}{8} \leqslant \mathsf{OUT}^{\text{heavy}} + \frac{\epsilon'}{2} \cdot \mathsf{OUT},$$

i.e., $\mathsf{OUT}(1-\frac{\epsilon'}{2}) \leqslant \mathsf{OUT}^{\text{heavy}}$. As s^{heavy} is an ϵ'-approximation of $\mathsf{OUT}^{\text{heavy}}$, $|s^{\text{heavy}} - \mathsf{OUT}^{\text{heavy}}| \leqslant \epsilon'\mathsf{OUT}^{\text{heavy}}$. Putting together with $\mathsf{OUT}^{\text{heavy}} \leqslant \mathsf{OUT}$ and $\mathsf{OUT}^{\text{heavy}} \geqslant \mathsf{OUT}(1-\frac{\epsilon'}{2})$, we get:

$$(1-\epsilon)\mathsf{OUT} \leqslant (1-\frac{3\epsilon'}{2})\mathsf{OUT} \leqslant (1-\epsilon')(1-\frac{\epsilon'}{2})\mathsf{OUT} \leqslant s_\Lambda \leqslant (1+\epsilon')\mathsf{OUT} \leqslant (1+\epsilon)\mathsf{OUT}.$$

Case 3: $s^{\text{heavy}} = \perp$ and $s^{\text{light}} \neq \perp$ Similar to Case 2.

Thus, if $\Lambda \leqslant 4\mathsf{OUT}$, with probability at least $1-\delta$, s_Λ is an ϵ-approximation for OUT. □

Lemma 3.6. *Algorithm 5 returns an ϵ-approximation of* OUT *with probability at least $1-\delta$.*

Proof of Lemma 3.6. Consider the invocation of Algorithm 6 at Line 4. From Lemma 3.5, $\mathbb{E}[s_i] \leqslant 2\mathsf{OUT}$. By the Markov inequality, $\Pr[s_i > 4\mathsf{OUT}] \leqslant \frac{1}{2}$. Taking the median, $\Pr[s \leqslant 4\mathsf{OUT}] \geqslant 1-(\frac{1}{2})^{\log\frac{2}{\delta}} \geqslant 1-\frac{\delta}{2}$. Let s^* be the final estimator returned with the guess Λ^*. We know that $s^* \geqslant \Lambda^*$. Hence, $\Lambda^* \leqslant 4\mathsf{OUT}$ holds with probability at least $1-\frac{\delta}{2}$. Following Lemma 3.5, when $\Lambda^* \leqslant 4\mathsf{OUT}$, each s_i is an ϵ-approximation of OUT with probability at least $1-\frac{\delta}{2}$. Thus, s^* is an ϵ-approximation of OUT with probability at least $1-\frac{\delta}{2}$. By the union bound, s^* is an ϵ-approximation of OUT with probability at least $1-\delta$. □

We next analyze the time complexity of our approximate counting algorithm:

Lemma 3.7. *Algorithm 5 runs in $O\left(\frac{N\cdot\log(N/\delta)\cdot\log(1/\delta)}{\epsilon^{2.5}\sqrt{\mathsf{OUT}}}\right)$ time with probability at least $1-\delta$.*

Proof of Lemma 3.7. We analyze the total time complexity by summing the cost of ApproxCountMatrixWithGuess over the sequence of guesses $\Lambda = N^2, N^2/2, \ldots, 1$. The algorithm terminates with high probability when $\Lambda \leqslant \mathsf{OUT}$. Thus, we only consider $\Lambda \gtrsim {}^8\mathsf{OUT}$.

For a fixed Λ, the procedure ApproxCountWithThreshold iterates guesses λ from N^2 down to a threshold $\tau = \frac{\epsilon'\Lambda}{16} = \Theta(\epsilon\Lambda)$. The cost for a specific λ is dominated by the sampling loop, which performs $k_\lambda = O(\frac{\widehat{\mathsf{OUT}_\bowtie}}{\epsilon^2\lambda})$ iterations. The expected cost per iteration is $O(1+\frac{N\cdot\sqrt{\mathsf{OUT}}}{\widehat{\mathsf{OUT}_\bowtie}})$. Thus, the total cost for a fixed λ is: $T(\lambda) \approx \frac{\widehat{\mathsf{OUT}_\bowtie}}{\epsilon^2\lambda}\left(1+\frac{N\cdot\sqrt{\mathsf{OUT}}}{\widehat{\mathsf{OUT}_\bowtie}}\right) = \frac{1}{\epsilon^2\lambda}(\widehat{\mathsf{OUT}_\bowtie} + N\sqrt{\mathsf{OUT}})$. For the light part, $\widehat{\mathsf{OUT}}_\bowtie^{\text{light}} = N\sqrt{\Lambda}$. For the heavy part, $\widehat{\mathsf{OUT}}_\bowtie^{\text{heavy}} \leqslant N\sqrt{\mathsf{OUT}^{\text{heavy}}} \leqslant N\sqrt{\Lambda}$ (since $\mathsf{OUT}^{\text{heavy}} \leqslant \mathsf{OUT} \leqslant \Lambda$ in the relevant range). Thus, $T(\lambda) \leqslant O(\frac{N\sqrt{\Lambda}}{\epsilon^2\lambda})$.

The inner loop stops when $\lambda \approx \mathsf{OUT}$ (if $\mathsf{OUT} \geqslant \tau$) or reaches τ (if $\mathsf{OUT} < \tau$). Let $\lambda_{stop} = \max(\Theta(\mathsf{OUT}), \tau)$. The total cost for a fixed Λ is dominated by the smallest λ, i.e., λ_{stop}: $\mathrm{Cost}(\Lambda) = \sum_{\lambda\geqslant\lambda_{stop}} T(\lambda) = O\left(\frac{N\sqrt{\Lambda}}{\epsilon^2\lambda_{stop}}\right)$.

We now sum $\mathrm{Cost}(\Lambda)$ over the geometric sequence of Λ from N^2 down to OUT. We split the sum into two ranges based on whether $\tau \leqslant \mathsf{OUT}$ (i.e., $\epsilon\Lambda \lesssim \mathsf{OUT}$) or $\tau > \mathsf{OUT}$.

- **Range 1: $\mathsf{OUT} \leqslant \Lambda \leqslant \frac{\mathsf{OUT}}{\epsilon}$.** Here, $\tau \approx \epsilon\Lambda \leqslant \mathsf{OUT}$, so the inner loop stops at $\lambda_{stop} \approx \mathsf{OUT}$. $\mathrm{Cost}(\Lambda) \approx \frac{N\sqrt{\Lambda}}{\epsilon^2\mathsf{OUT}}$. Summing over $\Lambda = 2^k\mathsf{OUT}$ for $k = 0\ldots\log(1/\epsilon)$: $\sum_\Lambda \mathrm{Cost}(\Lambda) \approx$

[8] greater than or approximately equal to.

$\frac{N}{\epsilon^2\mathsf{OUT}}\sum_k \sqrt{2^k\mathsf{OUT}} = \frac{N}{\epsilon^2\sqrt{\mathsf{OUT}}}\sum_k(\sqrt{2})^k$. The sum is dominated by the largest term ($\Lambda \approx \mathsf{OUT}/\epsilon$): $\mathrm{s}_1 \approx \frac{N}{\epsilon^2\sqrt{\mathsf{OUT}}}\cdot\sqrt{\frac{1}{\epsilon}} = \frac{N}{\epsilon^{2.5}\sqrt{\mathsf{OUT}}}$.

- **Range 2:** $\Lambda > \frac{\mathbf{OUT}}{\epsilon}$**.** Here, $\tau \approx \epsilon\Lambda > \mathsf{OUT}$, so the inner loop stops at $\lambda_{stop} = \tau \approx \epsilon\Lambda$. $\mathrm{Cost}(\Lambda) \approx \frac{N\sqrt{\Lambda}}{\epsilon^2(\epsilon\Lambda)} = \frac{N}{\epsilon^3\sqrt{\Lambda}}$. Summing over $\Lambda = 2^k(\mathsf{OUT}/\epsilon)$: $\sum_\Lambda \mathrm{Cost}(\Lambda) \approx \frac{N}{\epsilon^3}\sum_\Lambda \frac{1}{\sqrt{\Lambda}}$. The sum is dominated by the smallest term ($\Lambda \approx \mathsf{OUT}/\epsilon$): $\mathrm{s}_2 \approx \frac{N}{\epsilon^3\sqrt{\mathsf{OUT}/\epsilon}} = \frac{N}{\epsilon^{2.5}\sqrt{\mathsf{OUT}}}$.

Combining both ranges, the total time complexity is $O(\frac{N}{\epsilon^{2.5}\sqrt{\mathsf{OUT}}})$, multiplied by the logarithmic factors from probability amplification. □

Theorem 3.8. *For Q_{matrix}, given parameters $0 < \epsilon, \delta < 1$, there is an algorithm that can take as input an arbitrary input instance $\mathcal{R}$, and output an ϵ-approximation of $|Q_{\text{matrix}}(\mathcal{R})|$ in $O\left(\frac{N\cdot\log(N/\delta)\cdot\log(1/\delta)}{\epsilon^{2.5}\sqrt{\mathsf{OUT}}}\right)$ time with probability at least $1-\delta$.*

4 Lower Bounds for Matrix Query

In this section, we establish lower bounds for approximate counting and uniform sampling matrix query. Our lower bound for approximate counting is a reduction from the set disjointness problem in the two-party communication model. Given the universe $[m]$, Alice is given a set $S_A \subseteq [m]$ of $\frac{m}{4}$ elements and Bob is given a set $S_B \subseteq [m]$ of $\frac{m}{4}$ elements. The goal is to determine if $S_A \cap S_B = \emptyset$. It is known that any randomized communication protocol that solves this problem with constant success probability requires $\Omega(m)$ bits of communication [9, 27, 37].

Lemma 4.1. *For Q_{matrix}, given any nonnegative integers N, OUT, n, any algorithm that takes as input an arbitrary instance $\mathcal{R}$ of input size N, output size OUT, and active domain size n for B, and returns an $O(1)$ approximation of OUT with high probability, requires at least $\Omega(\min\{n, \frac{N}{\sqrt{\mathsf{OUT}}}\})$ time.*

Proof of Lemma 4.1. Given an input instance (S_A, S_B) of the set disjointness problem, Alice and Bob construct an instance $\mathcal{R}$ of Q_{matrix} based on their private sets S_A and S_B as follows. The domain of attribute B is the universe $[m]$. The active domain of attribute B is $S_A \cup S_B$. The active domains for attributes A and C are two disjoint sets, each of size $\sqrt{K}$ for some $K > 1$. Alice defines relation $R_1(A, B)$ as $R_1 = \mathrm{adom}(A) \times S_A$. Bob defines relation $R_2(B, C)$ as $R_2 = S_B \times \mathrm{adom}(C)$. The input size of $\mathcal{R}$ is $N = |R_1| + |R_2| = \sqrt{K}|S_A| + \sqrt{K}|S_B| = \sqrt{K}\cdot m/2$. The output size of $\mathcal{R}$ is $\mathsf{OUT} = |\pi_{A,C}(R_1 \bowtie R_2)|$ depends on the intersection of S_A and S_B. If $S_A \cap S_B = \emptyset$, then $\mathsf{OUT} = 0$. If $S_A \cap S_B \neq \emptyset$, then $\mathsf{OUT} = |\mathrm{adom}(A)| \cdot |\mathrm{adom}(C)| = K$.

Any algorithm $\mathcal{A}$ for approximate counting $\mathcal{R}$ can be used to solve the set disjointness instance (S_A, S_B) since it distinguishes between the $\mathsf{OUT} = 0$ case and the $\mathsf{OUT} = K$ case. Each query made by $\mathcal{A}$ will be answered by Alice and Bob communicating a small number of bits. For any query $\mathcal{A}$ makes (e.g., a tuple check of $(a, b) \in R_1$), Alice can answer it locally by checking if $b \in S_A$ and communicating the 1-bit answer to Bob. Other primitives are handled analogously. For queries on R_1 involving b (degree or neighbor access), Alice uses the fixed set $\mathrm{adom}(A)$ conditioned on $b \in S_A$. For queries involving a (degree or neighbor access), Alice uses the private set S_A (e.g., the degree of any $a \in \mathrm{adom}(A)$ is $|S_A|$, and the neighbors are the elements of S_A). Finally, for a sample query, Alice draws uniformly from $\mathrm{adom}(A) \times S_A$. Bob handles queries on R_2 symmetrically. If $\mathcal{A}$ provides an $O(1)$ approximation of OUT within $o(\min\{n, \frac{N}{\sqrt{\mathsf{OUT}}}\}) = o(n)$ queries, the set disjointness problem is solved within $o(n) = o(m)$ communication cost, leading to a contradiction. □

With this lemma, we can derive the following lower bound that is only parameterized by N and OUT, as it is always feasible to construct an instance with $n = \Theta(\frac{N}{\sqrt{\mathsf{OUT}}})$:

Theorem 4.2. *For Q_{matrix}, given any nonnegative integers N, OUT, any randomized algorithm that takes as input an arbitrary instance $\mathcal{R}$ of input size N and output size OUT, and outputs an $O(1)$ approximation of OUT with high probability, requires at least $\Omega(\frac{N}{\sqrt{\mathsf{OUT}}})$ time.*

Proof of Theorem 4.2. For any given N, OUT, we distinguish the following two cases:

If $\mathsf{OUT} \geqslant 1$, there is a hard instance for Q_{matrix} with $n = \Theta(\frac{N}{2\sqrt{\mathsf{OUT}}})$ as follows. The domain sizes of attribute A, B, C are $\sqrt{\mathsf{OUT}}, \frac{N}{2\sqrt{\mathsf{OUT}}}, \sqrt{\mathsf{OUT}}$ respectively. R_1 is the Cartesian product of $\mathrm{dom}(A)$ and $\mathrm{dom}(B)$. R_2 is the Cartesian product of $\mathrm{dom}(B)$ and $\mathrm{dom}(C)$. So, any algorithm needs $\Omega(\min\{n, \frac{N}{\sqrt{\mathsf{OUT}}}\}) = \Omega(\frac{N}{\sqrt{\mathsf{OUT}}})$ time.

If $\mathsf{OUT} = 0$, there is a hard instance for Q_{matrix} with $n = N$ as follows. The domain sizes of attribute A, B, C are $1, N, 1$ respectively. Relation R_1 is the Cartesian product of $\mathrm{dom}(A)$ and any $\frac{N}{2}$ values in $\mathrm{dom}(B)$. Relation R_2 is the Cartesian product of the remaining $\frac{N}{2}$ values in $\mathrm{dom}(B)$ and $\mathrm{dom}(C)$. So, any algorithm needs to spend $\Omega(\min\{n, \frac{N}{\sqrt{\mathsf{OUT}}}\}) = \Omega(N)$ time.

Combining these two cases yields the desired lower bound. □

This lower bound is derived from communication complexity, which imposes an information-theoretic limit independent of the specific computational operations used. Consequently, this applies to *any* algorithm, including those based on fast matrix multiplication, effectively ruling out the possibility of breaking the $\Omega(\frac{N}{\sqrt{\mathsf{OUT}}})$ barrier using algebraic techniques.

Following Proposition 3.1, Theorem 4.2 automatically implies a lower bound on uniform sampling over matrix query (for any algorithms), such that no index can be preprocessed in $O\left(\frac{N}{\mathsf{OUT}^{\frac{1}{2}+\gamma}}\right)$ time, while generating each (independent) uniform sample in $O\left(\frac{N}{\mathsf{OUT}^{\frac{1}{2}+\gamma}}\right)$ time, for any constant $\gamma > 0$. But this is not sufficient to show the optimality of our sampling algorithm, which uses $O(N)$ preprocessing time and generates samples in $O(\frac{N}{\sqrt{\mathsf{OUT}}})$ time. Below, we give a stronger lower bound for combinatorial algorithms following [19]:

Theorem 4.3. *For Q_{matrix}, assuming the combinatorial hardness of Boolean matrix multiplication[9], given any nonnegative integers N, OUT, there is no combinatorial algorithm that preprocesses an instance $\mathcal{R}$ of input size N and output size OUT in $O(N \cdot \mathsf{OUT}^{\frac{1}{2}-\gamma})$ time and generates a uniform sample from $Q_{\text{matrix}}(\mathcal{R})$ in $O\left(\frac{N}{\mathsf{OUT}^{\frac{1}{2}+\gamma}}\right)$ expected time, for any constant $\gamma > 0$.*

Proof of Theorem 4.3. We prove this by reduction from the problem of listing all query results. It is established that any combinatorial algorithm for evaluating the Q_{matrix} requires $\Omega(N\sqrt{\mathsf{OUT}})$ time [4]. Suppose, for the sake of contradiction, that there exists a sampling algorithm $\mathcal{A}$ with preprocessing time $O(N \cdot \mathsf{OUT}^{\frac{1}{2}-\gamma})$ and expected sampling time $T = O\left(\frac{N}{\mathsf{OUT}^{1/2+\gamma}}\right)$ for some constant $\gamma > 0$. We construct an algorithm to compute the full join result $Q_{\text{matrix}}(\mathcal{R})$ without prior knowledge of OUT, using the estimation technique from [19].

We repeatedly invoke $\mathcal{A}$ to generate samples and store them in a dictionary (e.g., a hash table) to track distinct tuples. We maintain a counter c for the number of *consecutive* samples that are already in the dictionary. We stop this process as soon as c reaches a threshold $\Delta = \Theta(\log N)$. Let k be the number of distinct tuples collected when we stop. We set an estimate $\Lambda = 2k$. As shown in [19], this stopping condition implies that with high probability, we have collected at least half of the total results, i.e., $k \geqslant \mathsf{OUT}/2$. Consequently, $\Lambda \in [\mathsf{OUT}, 2\mathsf{OUT}]$ with high probability. The number of samples drawn in this step is $\tilde{O}(\mathsf{OUT})$.

[9]Given two Boolean matrices of sizes $n \times n$ and $n \times n$, any combinatorial algorithm for computing their multiplication requires $\Omega(n^3)$ time.

Using the estimate Λ, we continue invoking $\mathcal{A}$ for a total of $\Theta(\Lambda \log N)$ trials. Since $\Lambda \geqslant \mathsf{OUT}$, this satisfies the requirement for the Coupon Collector's Problem ($\Theta(\mathsf{OUT} \log \mathsf{OUT}) = \Theta(\mathsf{OUT} \log N)$ samples) to ensure that all distinct tuples in $Q_{\mathsf{matrix}}(\mathcal{R})$ are collected with high probability.

The total expected number of samples drawn across both steps is $O(\mathsf{OUT} \log N)$. The total running time to enumerate the full query result is:

$$O(N \cdot \mathsf{OUT}^{\frac{1}{2}-\gamma}) + O\left(\mathsf{OUT} \log N \cdot \frac{N}{\mathsf{OUT}^{1/2+\gamma}}\right) = O\left(N \cdot \mathsf{OUT}^{1/2-\gamma} \log N\right).$$

For any $\mathsf{OUT} \geqslant \log N$, this runtime is strictly faster than $O(N\sqrt{\mathsf{OUT}})$, contradicting the combinatorial hardness of matrix multiplication. □

5 Star Query

In this section, we generalize our results from the matrix query to the star query. A star query with k relations is defined as $Q_{\mathsf{star}} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, where $\mathcal{V} = \{A_1, \ldots, A_k, B\}$, $\mathcal{E} = \{e_1, \ldots, e_k\}$ with $e_i = \{A_i, B\}$, and the output attributes are $\mathbf{y} = \{A_1, \ldots, A_k\}$. The query can be written as:

$$Q_{\mathsf{star}} = \pi_{A_1,\ldots,A_k}(R_1(A_1, B) \bowtie R_2(A_2, B) \bowtie \cdots \bowtie R_k(A_k, B)).$$

Note that the matrix query Q_{matrix} corresponds to the case where $k = 2$ (with $A_1 = A, A_2 = C$).

5.1 Uniform Sampling over Star Query

The sampling framework follows the same two-step structure as Algorithm 1:

- **Step 1:** Sample a full join result $\vec{t} = (a_1, \ldots, a_k, b)$ from $R_1 \bowtie \cdots \bowtie R_k$ uniformly at random.
- **Step 2:** Accept the projected tuple $\vec{a} = (a_1, \ldots, a_k)$ with probability at least $1/\deg(\vec{a})$, where $\deg(\vec{a})$ is the number of B-values connecting $a_1, \ldots, a_k$.

Step 1: sample full join result. We preprocess the instance to compute the weight of each $b \in \mathsf{adom}(B)$ as $W_b = \prod_{i=1}^{k} |R_i \ltimes b|$. Let $W = \sum_{b \in \mathsf{adom}(B)} W_b$. To sample a full join tuple:

- Sample $b \in \mathsf{adom}(B)$ with probability W_b/W.
- For each $i \in [k]$, sample a_i uniformly from $\pi_{A_i}(R_i \ltimes b)$.

The tuple $(a_1, \ldots, a_k, b)$ is returned with probability at least $1/W = 1/\mathsf{OUT}_{\bowtie}$.

Step 2: acceptance check. For a candidate tuple $\vec{a} = (a_1, \ldots, a_k)$, the degree is defined as:

$$\deg(\vec{a}) = \left| \bigcap_{i=1}^{k} \pi_B(R_i \ltimes a_i) \right|.$$

To accept with probability $1/\deg(\vec{a})$ without fully computing the intersection, we generalize Algorithm 3. We identify the relation with the smallest degree for the given tuple. Let $j = \arg\min_{i \in [k]} |R_i \ltimes a_i|$. We set $S = \pi_B(R_j \ltimes a_j)$. We then sample b' from S (excluding the witness b) and check if b' connects all other a_i (i.e., $\forall i \neq j, (a_i, b') \in R_i$). The number of failures F before finding a valid witness follows a negative hypergeometric distribution. We accept if $\texttt{RandInt}(1, |S|) \leqslant F + 1$.

Analysis. The correctness follows the same logic as Section 2.4. For the time complexity, the expected cost per sample is proportional to:

$$1 + \frac{1}{\mathsf{OUT}_{\bowtie}} \sum_{\vec{a} \in Q_{\mathsf{star}}(\mathcal{R})} \min_{i \in [k]} |R_i \ltimes a_i|.$$

Algorithm 10: SampleStar($\mathcal{R}, k$)

Input: An instance $\mathcal{R}$ for Q_{star}, with precomputed weights W_b.
Output: A query result of $Q_{\text{star}}(\mathcal{R})$.

$b \leftarrow$ sample from $\text{adom}(B)$ with prob. W_b/W;
for $i \in [k]$ **do** $a_i \leftarrow$ uniform sample from $\pi_{A_i}(R_i \ltimes b)$;
$j \leftarrow \arg\min_{i\in[k]} |R_i \ltimes a_i|$;
$S \leftarrow \pi_B(R_j \ltimes a_j)$;
$F \leftarrow 0$;
while *at least one element in S is not visited* **do**
  $b' \leftarrow$ random sample from non-visited in S;
  if $b' = b$ **then continue**;
  if $\exists i \neq j, (a_i, b') \notin R_i$ **then** $F \leftarrow F + 1$;
  else break;
if $\texttt{RandInt}(1, |S|) \leqslant F + 1$ **then return** $(a_1, \ldots, a_k)$;

Using the generalized AGM bound logic for star queries, it is known that $\sum_{\vec{a}} \min_i |R_i \ltimes a_i| \leqslant N \cdot \text{OUT}^{1-1/k}$. Thus, the expected runtime per successful sample is:

$$O\left(\frac{\text{OUT}_\bowtie}{\text{OUT}} \cdot \left(1 + \frac{N \cdot \text{OUT}^{1-1/k}}{\text{OUT}_\bowtie}\right)\right) = O\left(\frac{\text{OUT}_\bowtie}{\text{OUT}} + \frac{N}{\text{OUT}^{1/k}}\right).$$

Since $\text{OUT}_\bowtie \leqslant N \cdot \text{OUT}^{1-1/k}$ for star queries, the term simplifies to $O(N/\text{OUT}^{1/k})$.

Theorem 5.1. *For the star query Q_{star}, there is an algorithm that preprocesses an instance $\mathcal{R}$ of size N in $O(N)$ time and generates a uniform sample from $Q_{star}(\mathcal{R})$ in $O\left(\frac{N}{\text{OUT}^{1/k}}\right)$ expected time.*

5.2 Approximate Counting Star Query

The approximate counting algorithm for Q_{star} follows the hybrid strategy in Section 3. We partition the domain of B into heavy and light values based on a threshold τ. For the star query, the optimal threshold logic shifts from $\sqrt{\text{OUT}}$ to $\text{OUT}^{1/k}$.

- **Heavy Detect:** We identify $B^{\text{heavy}} = \{b \in \text{adom}(B) \mid \prod_{i=1}^{k} |R_i \ltimes b| \geqslant \tau\}$. This can be done by sampling or by iterating if we define heaviness based on the max degree in any single relation.
- **Hybrid Estimation:**
  - For the heavy part, we use the weighted sampling (Algorithm 10 adapted) which is efficient because the number of heavy B values is small.
  - For the light part, we use a rejection-based sampler (analogous to SampleMatrix-L) where we sample $a_1 \in R_1$, then $b \in R_1 \ltimes a_1$, then $a_2 \ldots a_k$. We accept based on the ratio of the tuple weight to the maximum possible light weight.

By setting the threshold parameters appropriately and applying the reduction from counting to sampling, we achieve the following bound:

Theorem 5.2. *For the star query Q_{star} and a constant $0 < \epsilon < 1$, there exists an algorithm that returns an ϵ-approximation of* OUT *with constant probability in $O\left(\frac{N}{\text{OUT}^{1/k}}\right)$ time.*

5.3 Lower Bounds for Approximate Counting Star Query

We establish the lower bound for the star query Q_{star} by a reduction from the k-party set disjointness problem in the communication complexity model. In this problem, there are k players, and each player i holds a private set $S_i \subseteq [m]$ of size $\frac{m}{2k}$. The goal is to determine if $\bigcap_{i=1}^{k} S_i = \emptyset$. It is known that any randomized communication protocol that solves this problem with constant success probability requires $\Omega(m)$ bits of communication.

Lemma 5.3. *For Q_{star}, given any $N, \mathsf{OUT}, m \in \mathbb{Z}^+$, any randomized algorithm that takes as input an arbitrary instance $\mathcal{R}$ of input size N, output size OUT, and active domain size m of attribute B, and outputs any multiplicative-factor approximation of OUT with high probability requires at least $\Omega(\min\{m, \frac{N}{\mathsf{OUT}^{1/k}}\})$ time.*

Proof. Given an input instance $(S_1, \ldots, S_k)$ of the k-party set disjointness problem where each $|S_i| = \frac{m}{2k}$, the k players construct an instance $\mathcal{R}$ of Q_{star} based on their private sets as follows.

The domain of attribute B is the universe $[m]$. The active domain of B is $\bigcup_{i=1}^{k} S_i$. For each $i \in [k]$, the active domain of attribute A_i, denoted $\mathsf{adom}(A_i)$, is a set of size $L = K^{1/k}$ for some parameter $K \geqslant 1$, and all $\mathsf{adom}(A_i)$ sets are mutually disjoint. Player i defines the relation $R_i(A_i, B)$ as the Cartesian product of $\mathsf{adom}(A_i)$ and their private set S_i: $R_i = \mathsf{adom}(A_i) \times S_i$. The input size of $\mathcal{R}$ is:

$$N = \sum_{i=1}^{k} |R_i| = \sum_{i=1}^{k} |\mathsf{adom}(A_i)| \cdot |S_i| = \sum_{i=1}^{k} K^{1/k} \cdot \frac{m}{2k} = K^{1/k} \cdot \frac{m}{2}.$$

The output size $\mathsf{OUT} = |Q_{\text{star}}(\mathcal{R})|$ depends on the intersection of the sets S_i:

- If $\bigcap_{i=1}^{k} S_i = \emptyset$, then the full join is empty, so $\mathsf{OUT} = 0$.
- If $\bigcap_{i=1}^{k} S_i \neq \emptyset$, then for every $b \in \bigcap_{i=1}^{k} S_i$, any combination of $(a_1, \ldots, a_k) \in \mathsf{adom}(A_1) \times \cdots \times \mathsf{adom}(A_k)$ is a valid result. Thus, $\mathsf{OUT} = \prod_{i=1}^{k} |\mathsf{adom}(A_i)| = (K^{1/k})^k = K$.

Any algorithm $\mathcal{A}$ for approximate counting $\mathcal{R}$ can be used to solve the set disjointness instance $(S_1, \ldots, S_k)$ since it distinguishes between the $\mathsf{OUT} = 0$ case and the $\mathsf{OUT} = K$ case. Each query made by $\mathcal{A}$ will be answered by the players communicating a small number of bits. For example, for a tuple check $(a, b) \in R_i$, player i checks if $b \in S_i$ locally (since $a \in \mathsf{adom}(A_i)$ is fixed and known).

If $\mathcal{A}$ provides a multiplicative-factor approximation of OUT within $o(\min\{m, \frac{N}{\mathsf{OUT}^{1/k}}\})$ queries, the players could solve the set disjointness problem with $o(m)$ communication, which contradicts the lower bound. Note that in the case $\mathsf{OUT} = K$, we have $m = \frac{2N}{K^{1/k}} = \frac{2N}{\mathsf{OUT}^{1/k}}$. □

With this lemma, we can derive the following lower bound that is only parameterized by N and OUT, as it is always feasible to construct an instance with $m = \Theta(\frac{N}{\mathsf{OUT}^{1/k}})$.

Theorem 5.4. *For Q_{star}, given any $N, \mathsf{OUT} \in \mathbb{Z}^+$, any randomized algorithm that takes as input an arbitrary instance $\mathcal{R}$ of input size N and output size OUT, and outputs any multiplicative-factor approximation of OUT with high probability requires at least $\Omega\left(\frac{N}{\mathsf{OUT}^{1/k}}\right)$ time.*

Proof. Consider any specific parameters $N, \mathsf{OUT} \in \mathbb{Z}^+$. We distinguish the following two cases:

- If $\mathsf{OUT} \geqslant 1$, there is a hard instance for Q_{star} with $m = \Theta(\frac{N}{\mathsf{OUT}^{1/k}})$ as follows. The domain size of attribute A_i is set to $L = \mathsf{OUT}^{1/k}$. The domain size of B is set to $m = \frac{2N}{L} = \frac{2N}{\mathsf{OUT}^{1/k}}$. Each relation R_i is constructed as in Lemma 5.3. Any algorithm needs $\Omega(\min\{m, \frac{N}{\mathsf{OUT}^{1/k}}\}) = \Omega(\frac{N}{\mathsf{OUT}^{1/k}})$ time.
- If $\mathsf{OUT} = 0$, there is a hard instance for Q_{star} with $m = \Theta(N)$ as follows. The domain size of attribute A_i is set to $L = 1$. The domain size of B is $m = N$. Each relation R_i consists of the

single value in $\mathrm{adom}(A_i)$ paired with a subset of B of size roughly N/k. Any algorithm needs $\Omega(\min\{m, \frac{N}{1}\}) = \Omega(N)$ time.

Combining these two cases yields the desired lower bound $\Omega\left(\frac{N}{\mathsf{OUT}^{1/k}}\right)$. □

5.4 Lower Bounds for Uniform Sampling Star Query

Similar to the matrix query, we can establish a lower bound for combinatorial algorithms by reducing from the problem of listing all query results.

Theorem 5.5. *For the star query Q_{star}, no combinatorial algorithm can preprocess an instance $\mathcal{R}$ of input size N and output size OUT in $O(N \cdot \mathsf{OUT}^{\frac{1}{k}-\gamma})$ time and generate a uniform sample from $Q_{star}(\mathcal{R})$ in $O\left(\frac{N}{\mathsf{OUT}^{\frac{1}{k}+\gamma}}\right)$ expected time, for any arbitrary small constant $\gamma > 0$, assuming the combinatorial hardness of listing star join results[10], under the property testing model.*

Proof. We prove this by reduction from the problem of listing all query results. Suppose, for the sake of contradiction, that there exists a sampling algorithm $\mathcal{A}$ with preprocessing time $O(N \cdot \mathsf{OUT}^{\frac{1}{k}-\gamma})$ and expected sampling time $T = O\left(\frac{N}{\mathsf{OUT}^{1/k+\gamma}}\right)$ for some constant $\gamma > 0$. We construct an algorithm to compute the full query result $Q_{\mathsf{star}}(\mathcal{R})$ without prior knowledge of OUT, following the strategy in Theorem 4.3.

Step 1: Estimate OUT. We repeatedly invoke $\mathcal{A}$ to generate samples and track collisions. As described in Theorem 4.3, by stopping when a sufficient number of collisions occur, we can obtain an estimate Λ such that $\Lambda \in [\mathsf{OUT}, 2\mathsf{OUT}]$ with high probability. This step requires $O(\mathsf{OUT})$ samples.

Step 2: Coupon Collection. Using the estimate Λ, we invoke $\mathcal{A}$ for $\Theta(\Lambda \log N)$ trials. Since $\Lambda \geqslant \mathsf{OUT}$ and $\log \mathsf{OUT} = O(\log N)$, this satisfies the requirement for the Coupon Collector's Problem to ensure that all distinct tuples in $Q_{\mathsf{star}}(\mathcal{R})$ are collected with high probability.

Complexity Analysis. The total number of samples drawn is $O(\mathsf{OUT} \log N)$. The total running time to enumerate the full query result is:

$$O(N \cdot \mathsf{OUT}^{\frac{1}{k}-\gamma}) + O\left(\mathsf{OUT} \log N \cdot \frac{N}{\mathsf{OUT}^{1/k+\gamma}}\right) = O\left(N \cdot \mathsf{OUT}^{1-\frac{1}{k}-\gamma} \log N\right).$$

For sufficiently large OUT, this runtime is strictly faster than the combinatorial lower bound of $\Omega(N \cdot \mathsf{OUT}^{1-1/k})$ for listing star query results, leading to a contradiction. □

6 Chain Query

6.1 Approximate Counting Chain Query

For $Q_{\mathsf{chain}} = \pi_{A_1,A_{k+1}}(R_1(A_1, A_2) \bowtie \cdots \bowtie R_k(A_k, A_{k+1}))$, we present an algorithm that runs in $\tilde{O}(N)$ time, based on the k-Minimum Values (KMV) summary [14, 15]. The core idea is to estimate the number of distinct reachable values in A_{k+1} for each value in A_1. Let $\deg(u) = |\{v \in \mathrm{adom}(A_{k+1}) \mid u \rightsquigarrow v\}|$ be the number of A_{k+1} values reachable from $u \in \mathrm{adom}(A_1)$. The total output size is $\mathsf{OUT} = \sum_{u \in \mathsf{adom}(A_1)} \deg(u)$. We can estimate each $\deg(u)$ efficiently using bottom-up dynamic programming combined with min-wise hashing.

Algorithm. We assign a random hash function $h : \mathrm{dom}(A_{k+1}) \to [0, 1]$. For any node u in the join graph, we wish to maintain the K-th smallest hash value among all $v \in \mathrm{adom}(A_{k+1})$ reachable from u. Let $H(u)$ denote the set of hash values of all A_{k+1} values reachable from u. We maintain a summary $S(u)$ containing the K smallest values in $H(u)$, where $K = O(1/\epsilon^2)$.

[10]It is known that any combinatorial algorithm for listing all results of a star query requires $\Omega(N \cdot \mathsf{OUT}^{1-1/k})$ time.

Algorithm 11: ApproxCountChain$_{\epsilon,\delta}(\mathcal{R})$

Input: An instance $\mathcal{R}$ of Q_{chain}, approximation quality ϵ and error δ.
Output: An estimate of $|Q_{\text{chain}}(\mathcal{R})|$.

$K \leftarrow \lceil 8/\epsilon^2 \rceil$, $m \leftarrow \lceil 12 \log(N/\delta) \rceil$;
Initialize map L where $L[u] \leftarrow []$ for all $u \in \text{adom}(A_1)$;
for $j \in [m]$ **do**
  Generate a pairwise independent hash function $h : \text{dom}(A_{k+1}) \to [0, 1]$;
  Initialize map S where $S[v] \leftarrow \{h(v)\}$ for all $v \in \text{adom}(A_{k+1})$;
  for $i \leftarrow k$ **to** 1 **do**
    Initialize map S';
    foreach $(u, v) \in R_i$ **do** $S'[u] \leftarrow \text{bottom-k}(S'[u] \cup S[v])$;
    $S \leftarrow S'$
  for $u \in \text{adom}(A_1)$ **do**
    if $|S[u]| < K$ **then** Append $|S[u]|$ to $L[u]$;
    else Append $\frac{K}{\max(S[u])}$ to $L[u]$;
$s \leftarrow 0$;
for $u \in \text{adom}(A_1)$ **do** $s \leftarrow s + \text{Median}(L[u])$;
return s;

The algorithm proceeds in a backward pass from R_k to R_1:

- **Initialization:** For each $v \in \text{adom}(A_{k+1})$, $S(v) = \{h(v)\}$.
- **Propagation:** For $i = k$ down to 1, for each $u \in \text{adom}(A_i)$, we compute $S(u)$ by merging the summaries of its neighbors in R_i: $S(u) = \text{bottom-k}\left(\bigcup_{(u,v)\in R_i} S(v)\right)$ where bottom-k retains the K smallest distinct values.
- **Estimation:** For each $u \in \text{adom}(A_1)$, we estimate $\deg(u)$. If $|S(u)| < K$, then $S(u)$ contains all reachable hash values, so $\widehat{\deg}(u) = |S(u)| = \deg(u)$. Otherwise, let v_K be the K-th smallest value in $S(u)$ (i.e., $\max(S(u))$), the estimator is $\widehat{\deg}(u) = \frac{K}{v_K}$.

To achieve the (ϵ, δ)-guarantee, we repeat this process $O(\log(N/\delta))$ times. For each u, we take the median of its estimates to boost the success probability, and finally sum these medians.

Analysis. The merging of KMV summaries at each step corresponds to the set union of reachable values. Since the size of each summary is bounded by $K = O(1/\epsilon^2)$, the merge operation for a tuple (u, v) takes $O(K)$ time. Across all relations, we process each tuple exactly once. Thus, one pass of the algorithm takes $O(N \cdot K)$ time.

Regarding the correctness, for each $u \in \text{adom}(A_1)$, the KMV estimator $\widehat{\deg}(u)$ is an ϵ-approximation of $\deg(u)$ with at least constant probability. By repeating the process $m = O(\log(N/\delta))$ times and taking the median, we boost the success probability for this specific u to $1 - \delta/N$. Since $|\text{adom}(A_1)| \leqslant N$, by applying the union bound over all $u \in \text{adom}(A_1)$, we guarantee that with probability at least $1 - \delta$, the estimates for all u are simultaneously ϵ-approximations. Consequently, their sum is an ϵ-approximation of OUT. The total time complexity is $O(N \cdot \frac{1}{\epsilon^2} \log \frac{N}{\delta})$.

Theorem 6.1. *For Q_{chain} and an arbitrary instance $\mathcal{R}$, Algorithm 11 returns an ϵ-approximation of $|Q_{chain}(\mathcal{R})|$ with probability at least $1 - \delta$ in $\tilde{O}(N)$ time.*

Algorithm 12: SampleChain($\mathcal{R}, \epsilon, \delta$)

Input: An instance $\mathcal{R}$ of Q_{chain}, parameters ϵ, δ.
Output: A uniform random sample from $Q_{\text{chain}}(\mathcal{R})$.
// Suppose $\texttt{ApproxCountChain}_{\epsilon,\delta}(\mathcal{R})$ returns $\widehat{\deg}(u)$ for all $u \in \text{adom}(A_1)$. We then compute $\widetilde{\deg}(u) = \frac{\widehat{\deg}(u)}{1-\epsilon}$ for all $u \in \text{adom}(A_1)$. Let $W = \sum_{u \in \text{adom}(A_1)} \widetilde{\deg}(u)$.
Sample $u^* \in \text{adom}(A_1)$ with probability $\frac{\widetilde{\deg}(u^*)}{W}$;
Compute $V_{u^*} \leftarrow \{v \in \text{adom}(A_{k+1}) \mid u^* \rightsquigarrow v\}$ via graph traversal;
$\deg(u^*) \leftarrow |V_{u^*}|$;
if $\deg(u^*) > \widetilde{\deg}(u^*)$ **then return** $\perp$;
Sample v^* uniformly from V_{u^*};
if $\texttt{Uniform}(0,1) \leqslant \frac{\deg(u^*)}{\widetilde{\deg}(u^*)}$ **then return** (u^*, v^*);

6.2 Uniform Sampling over Chain Query

We next show how to use the results from approximate counting to perform uniform sampling for Q_{chain}. Recall that for each $u \in \text{adom}(A_1)$, we obtain an estimator $\widehat{\deg}(u)$ such that $(1-\epsilon)\deg(u) \leqslant \widehat{\deg}(u) \leqslant (1+\epsilon)\deg(u)$ holds with high probability. To facilitate rejection sampling, we define an upper bound proxy for the degree: $\widetilde{\deg}(u) = \frac{\widehat{\deg}(u)}{1-\epsilon}$. It follows that $\deg(u) \leqslant \widetilde{\deg}(u) \leqslant \frac{1+\epsilon}{1-\epsilon}\deg(u)$. We use these proxy degrees to guide the sampling process towards "heavy" starting nodes, and then correct the bias using rejection sampling.

Algorithm. The sampling procedure consists of a preprocessing phase and a sampling phase. In the preprocessing phase, we run the approximate counting algorithm to compute $\widetilde{\deg}(u)$ for all $u \in \text{adom}(A_1)$. We then construct a data structure (e.g., an alias table or a prefix sum array) to support weighted sampling of $u \in \text{adom}(A_1)$ proportional to $\widetilde{\deg}(u)$.

In the sampling phase, we first sample a start node u^* according to the weights $\widetilde{\deg}$. Then, we compute the exact set of reachable values $V_{u^*} = \{v \in \text{adom}(A_{k+1}) \mid u^* \rightsquigarrow v\}$ by traversing the join graph starting from u^*. Note that $|V_{u^*}| = \deg(u^*)$. We draw a value v^* uniformly at random from V_{u^*}. Finally, we accept the pair (u^*, v^*) as a valid sample with probability $\frac{\deg(u^*)}{\widetilde{\deg}(u^*)}$. If rejected, we restart the sampling phase.

Analysis. The probability of returning the pair $(u, v) \in Q_{\text{chain}}(\mathcal{R})$ in one iteration is the product of the probability of choosing u, the probability of choosing v given u, and the acceptance probability:

$$\Pr[\text{return } (u,v)] = \underbrace{\frac{\widetilde{\deg}(u)}{W}}_{\text{choose } u} \cdot \underbrace{\frac{1}{\deg(u)}}_{\text{choose } v} \cdot \underbrace{\frac{\deg(u)}{\widetilde{\deg}(u)}}_{\text{accept}} = \frac{1}{W}.$$

Since this probability is independent of the specific pair (u, v), the returned sample is uniform.

Regarding the running time, the preprocessing takes $\tilde{O}(N)$ time as established in Section 6.1. In the sampling loop, computing V_{u^*} takes $O(N)$ time in the worst case (traversing the join path). The expected number of iterations is determined by the acceptance probability. Since $\widetilde{\deg}(u) \leqslant \frac{1+\epsilon}{1-\epsilon}\deg(u)$, the acceptance probability is at least $\frac{1-\epsilon}{1+\epsilon} \approx 1 - 2\epsilon$. Thus, the expected number of trials is $O(1)$. The total expected sampling time is $\tilde{O}(N)$.

Theorem 6.2. *For Q_{chain}, there is an algorithm that can take as input an arbitrary instance $\mathcal{R}$ of input size N and builds an index in $\tilde{O}(N)$ time such that with probability at least $1 - 1/N^{O(1)}$, a uniform sample from the query result $Q(\mathcal{R})$ can be returned in $\tilde{O}(N)$ expected time.*

Remark. It should be noted that this sampling algorithm only guarantees the uniformity with high probability, while our sampling algorithms for matrix and star queries can guarantee the uniformity for sure.

6.3 Lower Bounds for Approximate Counting Chain Query

So far, we have established matching upper and lower bounds for approximately counting and uniform sampling matrix queries, which correspond to the 2-chain query. However, the problem becomes significantly harder when its length increases. In this section, we establish a lower bound $\Omega(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ for the 3-chain query and $\Omega(N)$ for the 4-chain query, which can also be generalized to other queries that contain this small join as a core.

Theorem 6.3. *For $Q_{chain} = \pi_{A,D} R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D)$, given any parameter $N, \mathsf{OUT} \in \mathbb{Z}^+$, any randomized algorithm that takes as input an arbitrary instance $\mathcal{R}$ of input size N and output size OUT, and returns a constant-approximation of $|Q_{chain}(\mathcal{R})|$ with constant probability, requires at least $\Omega(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ time, under the property testing model.*

Proof of Theorem 6.3. Consider an arbitrary constant $\theta \geqslant 2$ and two arbitrary integers $\Delta, L \in \mathbb{Z}^+$ such that $\Delta \cdot L \leqslant \frac{N^2}{\theta}$ and $(\theta+1)L \leqslant N$. We define two distributions of instances, $\mathcal{D}_0$ and $\mathcal{D}_1$. Let $\mathrm{dom}(A) = \{a_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$, $\mathrm{dom}(B) = \{b_i : i \in [3N]\}$, $\mathrm{dom}(C) = \{c_i : i \in [3N]\}$, and $\mathrm{dom}(D) = \{d_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$.

First, we choose a subset $B_\alpha \subset \mathrm{dom}(B)$ of size $\frac{N}{\sqrt{\Delta}}$ and a subset $C_\alpha \subset \mathrm{dom}(C)$ of size $\frac{N}{\sqrt{\Delta}}$. Let $B_\beta = \mathrm{dom}(B) - B_\alpha$ and $C_\beta = \mathrm{dom}(C) - C_\alpha$. We fix $R_1 = \{(a_{ij}, b_i) : b_i \in B_\alpha, j \in [\sqrt{\Delta}]\}$ and $R_3 = \{(c_i, d_{ij}) : c_i \in C_\alpha, j \in [\sqrt{\Delta}]\}$. Note that $|R_1| = |R_3| = |B_\alpha|\sqrt{\Delta} = N$.

The two distributions differ only in the construction of R_2:

- **Distribution $\mathcal{D}_0$:** We construct R_2 by adding L random tuples from $B_\alpha \times C_\alpha$, $N - L$ random tuples from $B_\alpha \times C_\beta$, $N - L$ random tuples from $B_\beta \times C_\alpha$, and L random tuples from $B_\beta \times C_\beta$. The output size is $\mathsf{OUT} = L \cdot (\sqrt{\Delta})^2 = L \cdot \Delta$.
- **Distribution $\mathcal{D}_1$:** We construct R_2 by adding θL random tuples from $B_\alpha \times C_\alpha$, $N - \theta L$ random tuples from $B_\alpha \times C_\beta$, $N - \theta L$ random tuples from $B_\beta \times C_\alpha$, and θL random tuples from $B_\beta \times C_\beta$. The output size is $\mathsf{OUT}_1 = \theta L \cdot \Delta$.

In both cases, $|R_2| = 2N$. The constraints on Δ and L ensure that the subsets $B_\alpha \times C_\alpha$, etc., are large enough to support these selections without replacement.

Let $\mathcal{A}$ be an algorithm that distinguishes $\mathcal{D}_0$ from $\mathcal{D}_1$. Since $\mathsf{OUT}_1 = \theta \cdot \mathsf{OUT}$ and $\theta \geqslant 2$, any constant-factor approximation algorithm must distinguish these two cases. The difficulty lies in identifying tuples in the "critical region" $B_\alpha \times C_\alpha$. Although the distributions also differ in regions involving B_β or C_β, distinguishing based on those regions is statistically harder. The regions $B_\alpha \times C_\beta$ and $B_\beta \times C_\alpha$ contain $\Theta(N)$ tuples with a count difference of only $\Theta(L)$, requiring $\Omega((N/L)^2)$ queries to distinguish the bias. The region $B_\beta \times C_\beta$ has a domain size of $\Theta(N^2)$ with only $\Theta(L)$ tuples, requiring $\Omega(N^2/L)$ queries to find a witness. Thus, the most efficient strategy for $\mathcal{A}$ is to focus on $B_\alpha \times C_\alpha$. Consider the queries made to R_2:

- **test tuple:** $\mathcal{A}$ can check if $(b, c) \in R_2$. To distinguish the distributions, $\mathcal{A}$ must find tuples in $B_\alpha \times C_\alpha$. The size of this domain is $|B_\alpha||C_\alpha| = \frac{N^2}{\Delta}$. In $\mathcal{D}_0$, the density of tuples is $\rho_0 = \frac{L}{N^2/\Delta}$, and in $\mathcal{D}_1$, it is $\rho_1 = \frac{\theta L}{N^2/\Delta}$. To distinguish these densities, $\mathcal{A}$ requires $\Omega(\frac{1}{\rho_1}) = \Omega(\frac{N^2}{\theta L \Delta})$ queries.

- **sample tuple:** $\mathcal{A}$ can sample uniformly from R_2. The probability of sampling a tuple from $B_\alpha \times C_\alpha$ is $\frac{L}{2N}$ in $\mathcal{D}_0$ and $\frac{\theta L}{2N}$ in $\mathcal{D}_1$. To distinguish these biases, $\mathcal{A}$ requires $\Omega(\frac{2N}{\theta L}) = \Omega(\frac{N}{L})$ samples.
- **compute degree:** In both distributions, the total number of edges incident to B_α in R_2 is exactly N. Since the endpoints are chosen uniformly at random, the marginal distribution of the degree of any specific node $b \in B_\alpha$ is identical in both $\mathcal{D}_0$ and $\mathcal{D}_1$. Thus, the degree oracle provides no information to distinguish the distributions.
- **access neighbor:** $\mathcal{A}$ can query the j-th neighbor of a value $b \in B_\alpha$. Across all values in B_α, there are exactly N incident edges, creating a total of N valid query slots (pairs of (b, j)). In our construction, the L (in $\mathcal{D}_0$) or θL (in $\mathcal{D}_1$) critical edges connecting B_α to C_α are assigned to these slots uniformly at random (effectively by randomizing the neighbor orderings). Consequently, querying a specific slot (b, j) is statistically equivalent to sampling an edge uniformly at random from the N edges incident to B_α. The probability that such a query reveals a critical edge is $\frac{L}{N}$ in $\mathcal{D}_0$ and $\frac{\theta L}{N}$ in $\mathcal{D}_1$. As with the **sample tuple** oracle, distinguishing these cases requires $\Omega(\frac{N}{L})$ queries.

Combining these above, any algorithm requires $\Omega\left(\min\left\{\frac{N}{L}, \frac{N^2}{L\Delta}\right\}\right)$ time to distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$. The lower bound $\Omega\left(\min\{N, \frac{N^2}{\mathsf{OUT}}\}\right)$ follows by choosing $\Delta = \Theta(\mathsf{OUT})$ and $L = O(1)$. □

Theorem 6.4. *For $Q_{chain} = \pi_{A,E} R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D) \bowtie R_4(D, E)$, given any parameter $N, \mathsf{OUT} \in \mathbb{Z}^+$, any randomized algorithm that takes as input an arbitrary instance $\mathcal{R}$ of input size N and output size OUT, and returns a constant-approximation of $|Q_{chain}(\mathcal{R})|$ with constant probability, requires at least $\Omega(N)$ time, under the property testing model.*

Proof of Theorem 6.4. If $\mathsf{OUT} \leqslant N$, the lower bound follows from the 3-chain case (Theorem 6.3) by setting R_1 as a one-to-one mapping. We focus on the case where $\mathsf{OUT} > N$.

Consider any constant $\theta \geqslant 2$. We choose parameters $L = \Theta(1)$ and $\Delta = \Theta(\mathsf{OUT})$ such that $\Delta \cdot L \leqslant \frac{N^2}{\theta}$ and $(\theta + 1)L \leqslant N$ hold. We define two distributions of instances, $\mathcal{D}_0$ and $\mathcal{D}_1$. Let $\mathrm{dom}(A) = \{a_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$, $\mathrm{dom}(B) = \{b_i : i \in [3N]\}$, $\mathrm{dom}(C) = \{c_i : i \in [3N]\}$, $\mathrm{dom}(D) = \{d_i : i \in [3N]\}$, and $\mathrm{dom}(E) = \{e_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$. We fix subsets $B_\alpha \subset \mathrm{dom}(B)$ and $D_\alpha \subset \mathrm{dom}(D)$, each of size $\frac{N}{\sqrt{\Delta}}$. Let $B_\beta = \mathrm{dom}(B) - B_\alpha$ and $D_\beta = \mathrm{dom}(D) - D_\alpha$. We also choose a subset $C_\alpha \subset \mathrm{dom}(C)$ of size N and partition it into $\ell = \frac{N^2}{\mathsf{OUT}}$ groups $C_1, \ldots, C_\ell$, each of size $\frac{\mathsf{OUT}}{N}$. We fix $R_1 = \{(a_{ij}, b_i) : b_i \in B_\alpha, j \in [\sqrt{\Delta}]\}$ and $R_4 = \{(d_i, e_{ij}) : d_i \in D_\alpha, j \in [\sqrt{\Delta}]\}$. Note that $|R_1| = |R_4| = N$. The distributions differ in R_2 and R_3:

- **Distribution $\mathcal{D}_0$:** We select L distinct pairs from $B_\alpha \times D_\alpha$ uniformly at random. For each selected pair (b_i, d_j), we exclusively assign a distinct group C_h from the partition of C_α, pick a value c_m uniformly at random from C_h, and add (b_i, c_m) to R_2 and (c_m, d_j) to R_3. To mask these critical tuples, we add random noise tuples (e.g., from $B_\alpha \times C_\beta$, etc.) until $|R_2| = |R_3| = 2N$, similar to the construction in Theorem 6.3. The output size is $\mathsf{OUT}_0 = L \cdot (\sqrt{\Delta})^2 = L \cdot \Delta$.
- **Distribution $\mathcal{D}_1$:** The construction is identical, except we select θL pairs instead of L. The output size is $\mathsf{OUT}_1 = \theta L \cdot \Delta$.

Since $\mathsf{OUT}_1 = \theta \cdot \mathsf{OUT}_0$, any constant-factor approximation must distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$. Consider the queries made:

- **test tuple:** To distinguish the distributions, the algorithm must identify the "critical" paths $b \to c \to d$. There are $|B_\alpha||D_\alpha| = \frac{N^2}{\Delta}$ candidate pairs in $B_\alpha \times D_\alpha$. The probability that a random pair is critical is $\rho = \frac{L}{N^2/\Delta}$ (in $\mathcal{D}_0$). Even if the algorithm successfully guesses a critical pair

(b, d) (which requires $\Omega(1/\rho)$ trials), and even if it knows the specific group C_h assigned to this pair, it must still find the specific witness $c_m \in C_h$. Since $|C_h| = \frac{\mathsf{OUT}}{N}$, finding the witness requires $\Omega(\frac{\mathsf{OUT}}{N})$ queries. The total complexity is: $\Omega\left(\frac{1}{\rho} \cdot \frac{\mathsf{OUT}}{N}\right) = \Omega\left(\frac{N^2}{L\Delta} \cdot \frac{\mathsf{OUT}}{N}\right)$. Substituting $\mathsf{OUT} = \Theta(L\Delta)$, the cost is $\Omega(N)$.

- **sample tuple:** The critical tuples in R_2 (those in $B_\alpha \times C_\alpha$) constitute a fraction of $\frac{L}{2N}$ of the relation size. Distinguishing the bias between L and θL requires $\Omega(\frac{N}{L})$ samples. Since we chose $L = \Theta(1)$, this cost is $\Omega(N)$.
- **compute degree or access neighbor:** As in Theorem 6.3, the marginal degree distributions of B_α and D_α are masked by the noise tuples. Accessing a neighbor is statistically equivalent to sampling, yielding the same $\Omega(N)$ lower bound. □

6.4 Lower Bounds for Uniform Sampling over Chain Query

We establish the lower bound for uniform sampling chain queries via the reduction from approximate counting. The reduction in Proposition 3.1 requires that the sampling algorithm returns a query result with probability $\frac{\mathsf{OUT}}{W}$ for some parameter W, so the lower bounds derived from this reduction apply to this class of sampling algorithms. We call such algorithms *W-uniform sampling* algorithms: given an index built during preprocessing, each invocation returns a uniform sample from $Q(\mathcal{R})$ with probability $\frac{\mathsf{OUT}}{W}$, and always returns the value of W. Note that all sampling algorithms presented in this paper fall into this class, as do all prior sampling algorithms for join-project queries that we are aware of.

Theorem 6.5. *For Q_{chain} with $k = 3$, given parameters N, OUT with $\mathsf{OUT} \leqslant N$, no W-uniform sampling algorithm can preprocess an instance $\mathcal{R}$ of input size N and output size OUT in $O(\min\{N^{1-\gamma}, (\frac{N^2}{\mathsf{OUT}})^{1-\gamma}\})$ time and generate a sample in $O(\min\{N^{1-\gamma}, (\frac{N^2}{\mathsf{OUT}})^{1-\gamma}\})$ time, for any arbitrary small constant $\gamma > 0$.*

Proof of Theorem 6.5. Following Proposition 3.1, Theorem 6.3 automatically implies a lower bound on W-uniform sampling over chain queries. Specifically, if there exists a W-uniform sampling algorithm with preprocessing time $O(\min\{N^{1-\gamma}, (\frac{N^2}{\mathsf{OUT}})^{1-\gamma}\})$ that can generate a sample in $O(\min\{N^{1-\gamma}, (\frac{N^2}{\mathsf{OUT}})^{1-\gamma}\})$, we can construct an approximate counting algorithm by invoking the sampling procedure $\tilde{O}(1)$ times. The total running time would be $O(\min\{N^{1-\gamma}, (\frac{N^2}{\mathsf{OUT}})^{1-\gamma}\})$, which contradicts the lower bound $\Omega(\min\{N, \frac{N^2}{\mathsf{OUT}}\})$ established in Theorem 6.3. □

Corollary 6.6. *For Q_{chain} with $k = 3$, given parameters N, OUT with $\mathsf{OUT} \leqslant N$, no W-uniform sampling algorithm can preprocess an instance $\mathcal{R}$ of input size N and output size OUT in $O(N^{1-\gamma})$ time, and produce an index that can support generating a sample within $O(N^{1-\gamma})$ time.*

Theorem 6.7. *For Q_{chain} with $k \geqslant 4$, no W-uniform sampling algorithm can preprocess an instance $\mathcal{R}$ of input size N in $O(N^{1-\gamma})$ time and generate a sample from $Q_{chain}(\mathcal{R})$ in $O(N^{1-\gamma})$ time, for any arbitrary small constant $\gamma > 0$.*

Proof of Theorem 6.7. Following Proposition 3.1, Theorem 6.4 automatically implies a lower bound on W-uniform sampling over chain queries. Specifically, if there exists a W-uniform sampling algorithm with preprocessing time $O(N^{1-\gamma})$ that can generate a sample in $O(N^{1-\gamma})$ time, we can construct an approximate counting algorithm by invoking the sampling procedure $\tilde{O}(1)$ times. The total running time would be $\tilde{O}(N^{1-\gamma})$, which contradicts the $\Omega(N)$ lower bound for approximate counting established in Theorem 6.4. □

Note that, unlike the combinatorial lower bounds for matrix and star queries, this result relies on the information-theoretic hardness of distinguishing distributions (as used in Theorem 6.3 and Theorem 6.4) and thus applies to any W-uniform sampling algorithm, not just combinatorial ones.

7 Conclusion

This work initiates a rigorous study into the fine-grained complexity of uniform sampling and approximate counting for join-project queries. We established the first asymptotically optimal algorithms for fundamental query classes – specifically matrix, star, and chain queries – marking a pivotal step towards output-optimality for general join-project queries. Our findings open several avenues for future research:

- *A Unified Parameterized Framework:* A primary open problem is to synthesize the techniques developed for star and chain queries into a unified algorithm capable of handling arbitrary join-project queries. Note that the techniques introduced for star queries can be easily generalized to all join-project queries (see Appendix A), but they are already not optimal for even for the length-4 chain query. Furthermore, a rigorous parameterized complexity analysis is required to characterize algorithmic performance relative to both input and output sizes.
- *Support Selection Predicate:* So far, our algorithms have investigated only join and projection operators. A natural extension is to incorporate selection predicates to fully support general conjunctive queries and to determine the impact of selectivity on sampling complexity. Very recently, [25] investigated the sampling indices for acyclic joins when the point selection predicates exist. But it is still open for more complicated predicates.
- *Beyond Combinatorial Limits:* While our results define the boundaries of combinatorial approaches in some scenarios, the application of algebraic techniques remains largely unexplored. A compelling frontier is to determine whether methods such as fast matrix multiplication can circumvent existing combinatorial barriers. Recent efforts have shown that fast matrix multiplication can be used to accelerate counting subgraph patterns [3, 10, 11, 17, 18, 21, 29, 31, 38], , such as triangles, cycles, and cliques. But it's still unknown how much these techniques can be extended to general join queries or even join-project queries.

Acknowledgement

Xiao Hu was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant. Jinchao Huang was supported by the Overseas Research Attachment Program of The Chinese University of Hong Kong during his research visit to the University of Waterloo. We thank Sepehr Assadi for invaluable discussions and feedback on this problem, as well as the anonymous reviewers for their constructive comments. Jinchao Huang also expresses gratitude to Sibo Wang for supporting his overseas research at the University of Waterloo.

References

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, pages 275–286, 1999.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[3] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[4] R. R. Amossen and R. Pagh. Faster join-projects and sparse matrix multiplications. In *ICDT*, pages 121–126. ACM, 2009.

[5] M. Arenas, L. A. Croquevielle, R. Jayaram, and C. Riveros. When is approximate counting for conjunctive queries tractable? In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1015–1027, 2021.

[6] S. Assadi, M. Kapralov, and S. Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In *Proc. Innovations in Theoretical Computer Science*, pages 6:1–6:20, 2019.

[7] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42 (4):1737–1767, 2013.

[8] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.

[9] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. Information theory methods in communication complexity. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 93–102. IEEE, 2002.

[10] K. Censor-Hillel, T. Even, and V. Vassilevska Williams. Fast approximate counting of cycles. In *ICALP 2024*, 2024.

[11] K. Censor-Hillel, T. Even, and V. Vassilevska Williams. Output-sensitive approximate counting via a measure-bounded hyperedge oracle, or: How asymmetry helps estimate k-clique counts faster. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 1985–1996, 2025.

[12] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. *ACM SIGMOD Record*, 28(2):263–274, 1999.

[13] Y. Chen and K. Yi. Random sampling and size estimation over cyclic joins. In *ICDT*, 2020.

[14] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In I. Gupta and R. Wattenhofer, editors, *Proc. ACM Symposium on Principles of Distributed Computing*, pages 225–234. ACM, 2007.

[15] E. Cohen and H. Kaplan. Tighter estimation using bottom k sketches. *Proceedings of the VLDB Endowment*, 1(1): 213–224, 2008.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.

[17] R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223, 2017.

[18] M. Dalirrooyfard, S. Mathialagan, V. Vassilevska Williams, and Y. Xu. Towards optimal output-sensitive clique listing or: Listing cliques from smaller cliques. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 923–934, 2024.

[19] S. Deng, S. Lu, and Y. Tao. On join sampling and hardness of combinatorial output-sensitive join algorithms. In *PODS*, 2023.

[20] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

[21] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004.

[22] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*. Hafner Publishing Company, 1953.

[23] X. Hu. Cover or pack: New upper and lower bounds for massively parallel joins. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 181–198, 2021.

[24] X. Hu and K. Yi. Parallel algorithms for sparse matrix multiplication and join-aggregate queries. In *Proc. ACM Symposium on Principles of Database Systems*, pages 411–425, 2020.

[25] J. Huang, Y. Tao, and S. Wang. Acyclic join sampling under selections: Dichotomy, union sampling, and enumeration. In *Proc. International Conference on Database Theory*, 2026.

[26] N. L. Johnson, A. W. Kemp, and S. Kotz. *Univariate discrete distributions*. John Wiley & Sons, 2005.

[27] B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

[28] K. Kim, J. Ha, G. Fletcher, and W.-S. Han. Guaranteeing the $\tilde{O}$(AGM/OUT) runtime for uniform sampling and size estimation over joins. In *PODS*, page 113–125, 2023.

[29] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74 (3-4):115–121, 2000.

[30] D. E. Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley Professional, 1998.

[31] M. Kowaluk, A. Lingas, and E. Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discret. Math.*, 27(2):892–909, 2013.

[32] H. Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proc. ACM Symposium on Principles of Database Systems*, pages 111–124. ACM, 2018.

[33] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In *Proc. ACM Symposium on Principles of Database Systems*, pages 37–48, 2012.

[34] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: new developments in the theory of join algorithms. *Acm Sigmod Record*, 42(4):5–16, 2014.

Algorithm 13: SAMPLEJOINPROJECT($\mathcal{R}$)

Input: An instance $\mathcal{R}$ for the join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, with some auxiliary indices built for $\mathcal{R}$ if needed.

Output: A uniform sample of the query result $Q(\mathcal{R})$.

```
while true do
    s ← JOINSAMPLE(Q, R) [19, 28, 40];
    if s ≠ ⊥ then break;
if ACCEPTJOINPROJECT(Q, R, s) = true then return π_y s;        ▸ Algorithm 14;
else return ⊥;
```


[35] F. Olken and D. Rotem. Simple random sampling from relational databases. In *Proceedings of the VLDB Endowment*, pages 160–169, 1986.

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

[37] A. A. Razborov. On the distributional complexity of disjointness. In *International Colloquium on Automata, Languages, and Programming*, pages 249–253. Springer, 1990.

[38] J. Tětek. Approximate Triangle Counting via Sampling and Fast Matrix Multiplication. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, pages 107:1–107:20, 2022.

[39] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. International Conference on Very Large Data Bases*, pages 82–94, 1981.

[40] Z. Zhao, R. Christensen, F. Li, X. Hu, and K. Yi. Random sampling over joins revisited. In *SIGMOD*, pages 1525–1539, 2018.

A Extension to General Join-Project Queries

Our framework introduced for matrix and star queries can be easily extended to arbitrary join-project queries. We first recall the AGM bound [7]. For a join query $q = (\mathcal{V}, \mathcal{E})$, let $\mathbf{N} : \mathcal{E} \to \mathbb{Z}^+$ be the size function, and w be a fractional edge covering such that $\sum_{e:x\in e} w(e) \geqslant 1$ for each attribute $A \in \mathcal{V}$. The AGM bound [7] states

$$\max_{\mathcal{R}:|R_e|\leqslant \mathbf{N}_e, \forall e\in\mathcal{E}} |Q_{\bowtie}(\mathcal{R})| \leqslant \prod_{e\in\mathcal{E}} \mathbf{N}_e^{w(e)} \tag{1}$$

We also use $\mathsf{AGM}(q, \mathbf{N})$ to denote the smallest upper bound for q under the input function $\mathbf{N}$, i.e., the smallest right-hand-side of E.q. (1) over all possible fractional edge coverings. As a special case, when $\mathbf{N}_e = N$ for each $e \in \mathcal{E}$, we have $\mathsf{AGM}(q, \mathbf{N}) = N^{\sum_{e\in\mathcal{E}} w(e)}$. The tightest upper bound would be the one minimizing $\sum_{e\in\mathcal{E}} w(e)$, i.e., $\rho^*(q)$. For a join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, let $Q_{\bowtie} = (\mathcal{V}, \mathcal{E})$ be the underlying full join query of Q. Given an instance $\mathcal{R}$ for Q, let $\mathsf{OUT}_{\bowtie} = |Q_{\bowtie}(\mathcal{R})|$ be the number of full join results. From the AGM bound, $\mathsf{OUT}_{\bowtie} \leqslant N^{\rho^*(Q_{\bowtie})}$.

Algorithm. As described in Algorithm 13, we first get a sample from the full join result using the existing sampling algorithm [19, 28], say s and then accept it with probability $\frac{1}{\deg(\pi_{\mathbf{y}} s)}$, where $\deg(\pi_{\mathbf{y}} s)$ is the number of full join results that $\pi_{\mathbf{y}} s$ appears in. Following exactly the same analysis, we can show that each query result $t \in Q(\mathcal{R})$ is sampled with probability

$$\sum_{s\in Q_{\bowtie}(\mathcal{R}):\pi_{\mathbf{y}} s = t} \frac{1}{|Q_{\bowtie}(\mathcal{R})|} \cdot \frac{1}{\deg(t)} = \frac{1}{|Q_{\bowtie}(\mathcal{R})|},$$

it is therefore a uniform sample of the query result.

To implement this algorithm, we will need a primitive JOINSAMPLE proposed by prior work [19, 28, 40], which takes as input a full join query $Q_{\bowtie}$ and an instance $\mathcal{R}$, and always returns a uniform

Algorithm 14: AcceptJoinProject-I($Q, \mathcal{R}, s$)

Input: A join-project query Q, an instance $\mathcal{R}$ for the join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, and a full join result $s \in Q_{\bowtie}(\mathcal{R})$ with $t = \pi_{\mathbf{y}} s$.

Output: A Boolean value indicating acceptance.

1 $Q_{\bar{\mathbf{y}}} \leftarrow (\mathcal{V} - \mathbf{y}, \{e - \mathbf{y} : e \in \mathcal{E}\}, \mathcal{V} - \mathbf{y})$;

2 $\mathcal{R}_s \leftarrow \{\pi_{e-\mathbf{y}} \left(R_e \ltimes (\pi_{\mathbf{y}} s)\right) : e \in \mathcal{E}\}$;

3 $F \leftarrow 0$;

// For acyclic queries, line 4 can be skipped, and line 5 is the Cartesian product of a subset of relations such that every attribute of $\mathcal{V} - \mathbf{y}$ appears in at least one selected relation

4 Initialize the auxiliary indices for JoinSample if needed;

5 $S \leftarrow$ the sampling universe defined for $Q_{\bar{\mathbf{y}}}$ on $\mathcal{R}_s$ [19];

// We sample from S using uniform sampling without replacement

6 **while** *at least one element in S is not visited yet* **do**

7 $s' \leftarrow$ a random sample from non-visited elements in S using JoinSample($Q_{\bar{\mathbf{y}}}, \mathcal{R}_s$);

8 **if** $s' = \pi_{\mathcal{V}-\mathbf{y}} s$ **then continue**;

9 **if** $s' \notin Q_{\bar{\mathbf{y}}}(\mathcal{R}_s)$ **then** $F \leftarrow F + 1$;

10 **else break**;

11 **if** RandInt$(1, \text{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes (\pi_{\mathbf{y}} s)| : e \in \mathcal{E}\})) \leqslant F + 1$ **then return true**;

12 **return false**;

sample from the full join result $Q_{\bowtie}(\mathcal{R})$ with probability 1 for acyclic joins and $\frac{|Q_{\bowtie}(\mathcal{R})|}{\text{AGM}(Q_{\bowtie}, \{|R_e| : e \in \mathcal{E}\})}$ for cyclic joins. We will repeat this primitive until a valid join result is returned.

Below, we focus on the AcceptJoinProject primitive. Suppose s is a full join result returned by the sampling procedure JoinSample. Let $Q_{\bar{\mathbf{y}}}$ be the full join query induced by non-output attributes. Let $\mathcal{R}_s$ be the induced sub-instance of $\mathcal{R}$ that can be joined with s on output attributes. Note that any join result of $Q_{\bar{\mathbf{y}}}(\mathcal{R}_s)$ will form with t as a full join result of $Q_{\bowtie}(\mathcal{R})$. The following steps follow Section 2. As computing $\deg(t)$ is too expensive, we simulate this probability using Proposition 2.1.

To apply this proposition, we first identify S as the sampling universe of $Q_{\bar{\mathbf{y}}}$ on $\mathcal{R}_s$ defined by the JoinSample primitive, where $|S| = \text{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e| : e \in \mathcal{E}\})$. Note that $\pi_{\mathcal{V}-\mathbf{y}} s$ is a valid join result of $Q_{\bar{\mathbf{y}}}(\mathcal{R}_s)$ (a "success") because s was sampled from the full join result $Q_{\bowtie}(\mathcal{R})$. We construct the random variable Y by sampling from the *other* elements in S. Specifically, we sample without replacement from $S - \{\pi_{\mathcal{V}-\mathbf{y}} s\}$ and count the number of "failures" (F) encountered before finding another valid join result in $Q_{\bar{\mathbf{y}}}(\mathcal{R}_s)$. If no other valid join value exists (i.e., $\pi_{\mathcal{V}-\mathbf{y}} s$ is unique), we continue until $S - \{\pi_{\mathcal{V}-\mathbf{y}} s\}$ is exhausted. We define $Y = F + 1$. It iterates through random samples from S by invoking the JoinSample primitive. The expectation of this specific construction is $\mathbb{E}[Y] = \frac{|S|}{\deg(t)}$. Finally, we generate a random integer X uniformly from $[|S|]$. If $X \leqslant F + 1$, we return **true**; otherwise, we return **false**. By Proposition 2.1, the acceptance probability is exactly $\frac{1}{|S|} \cdot \frac{|S|}{\deg(t)} = \frac{1}{\deg(t)}$. Additionally, if the residual query $Q_{\bar{\mathbf{y}}}$ is acyclic, the sampling universe can be easily computed since every acyclic join has an optimal fractional edge covering that is also integral [23]. Hence, the Cartesian product of those selected relations (i.e., assigned weight 1 in the optimal fractional edge covering) forms the sampling universe. Moreover, no additional auxiliary indices are required, as relations in $\mathcal{R}_s$ are well indexed.

Algorithm 15: AcceptJoinProject-II($Q, \mathcal{R}, s$)

Input: A join-project query Q, an instance $\mathcal{R}$ for the join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, and a full join result $s \in Q_{\bowtie}(\mathcal{R})$ with $t = \pi_{\mathbf{y}} s$.

Output: A Boolean value indicating acceptance.

1 $Q_{\bar{\mathbf{y}}} \leftarrow (\mathcal{V} - \mathbf{y}, \{e - \mathbf{y} : e \in \mathcal{E}\}, \mathcal{V} - \mathbf{y})$;
2 $\mathcal{R}_s \leftarrow \{\pi_{e-\mathbf{y}} \left(R_e \ltimes (\pi_{\mathbf{y}} s)\right) : e \in \mathcal{E}\}$;
3 Compute $|Q_{\bar{\mathbf{y}}}(\mathcal{R}_s)|$;
4 **return true**with probability $\frac{1}{|Q_{\bar{\mathbf{y}}}(\mathcal{R}_s)|}$ and **false**with probability $1 - \frac{1}{|Q_{\bar{\mathbf{y}}}(\mathcal{R}_s)|}$;

Analysis. We now analyze the time complexity of Algorithm 13. We also distinguish our analysis for acyclic and cyclic queries separately, since there could be much more efficient implementation of primitives for acyclic queries.

Analysis for acyclic queries. The preprocessing step takes $O(N)$ time. After preprocessing, each invocation of JoinSample takes $\tilde{O}(1)$ time and always returns a successful sample. Hence the while-loop at Lines 1-3 takes $O(1)$ time. The invocation of Algorithm 14 at Line 4 takes $O(N + F + 1)$ time. From our analysis above, the expectation $\mathbb{E}[F + 1] = \frac{|S|}{\deg(s)} = \frac{\mathrm{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes (\pi_{\mathbf{y}} s)| : e \in \mathcal{E}\})}{\deg(s)}$. Let $Q_{\mathbf{y}} = (\mathcal{V}, \mathcal{E} \cup \{\mathbf{y}\})$ be the full join by adding an additional relation that contains all output attributes of $\mathbf{y}$. So, following the decomposition lemma in [34]:

$$\sum_{t \in Q(\mathcal{R})} \mathrm{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes t| : e \in \mathcal{E}\}) \leqslant N^{\sum_{e \in \mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})}$$

for any fractional edge covering w of $Q_{\mathbf{y}}$. Hence, Algorithm 13 takes (omitting the big-O):

$$\begin{aligned}
\mathbb{E}[\text{time per invocation}] &= 1 + \sum_{t \in Q(\mathcal{R})} \frac{\deg(t)}{\mathsf{OUT}_{\bowtie}} \cdot \frac{\mathrm{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes t| : e \in \mathcal{E}\})}{\deg(t)} \\
&\leqslant 1 + \frac{\sum_{t \in Q(\mathcal{R})} \mathrm{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes t| : e \in \mathcal{E}\})}{\mathsf{OUT}_{\bowtie}} \\
&\leqslant 1 + \frac{N^{\sum_{e \in \mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})}}{\mathsf{OUT}_{\bowtie}}
\end{aligned}$$

where w is an arbitrary fractional edge covering of $Q_{\mathbf{y}}$.

As an alternative, Algorithm 15 only takes $O(N)$ time to compute the join size for acyclic queries. Algorithm 13 succeeds with probability $\frac{\mathsf{OUT}}{\mathsf{OUT}_{\bowtie}}$, so the expected number of invocations before getting one successful sample is $\frac{\mathsf{OUT}_{\bowtie}}{\mathsf{OUT}}$. The total expected cost of exploiting the power of two strategies is $O\left(\min\left\{\frac{\mathsf{OUT}_{\bowtie}}{\mathsf{OUT}} + N^{\sum_{e \in \mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})-1}, \frac{N \cdot \mathsf{OUT}_{\bowtie}}{\mathsf{OUT}}\right\}\right)$.

Analysis for cyclic queries. The preprocessing step takes $O(N)$ time. After preprocessing, each invocation of JoinSample takes $\tilde{O}(1)$ time. As JoinSample returns a successful sample with probability $\frac{\mathsf{OUT}_{\bowtie}}{N^{\rho^*(Q_{\bowtie})}}$ for cyclic queries, hence the while-loop at Lines 1-3 takes $O(\frac{N^{\rho^*(Q_{\bowtie})}}{\mathsf{OUT}_{\bowtie}})$ time for cyclic queries. The invocation of Algorithm 14 at Line 4 takes $O(F + 1)$ time. From our analysis above, the expectation $\mathbb{E}[F + 1] = \frac{|S|}{\deg(s)} = \frac{\mathrm{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes (\pi_{\mathbf{y}} s)| : e \in \mathcal{E}\})}{\deg(s)}$. Hence, Algorithm 13 takes (omitting

the big-O):

$$\begin{aligned}\mathbb{E}[\text{time per invocation}] &= \frac{N^{\rho^*(Q_\bowtie)}}{\mathsf{OUT}_\bowtie} + N + \sum_{t\in Q(\mathcal{R})} \frac{\deg(t)}{\mathsf{OUT}_\bowtie} \cdot \frac{\mathrm{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes t| : e \in \mathcal{E}\})}{\deg(t)} \\ &\leqslant \frac{N^{\rho^*(Q_\bowtie)}}{\mathsf{OUT}_\bowtie} + N + \frac{\sum_{t\in Q(\mathcal{R})} \mathrm{AGM}(Q_{\bar{\mathbf{y}}}, \{|R_e \ltimes t| : e \in \mathcal{E}\})}{\mathsf{OUT}_\bowtie} \\ &\leqslant \frac{N^{\rho^*(Q_\bowtie)}}{\mathsf{OUT}_\bowtie} + N + \frac{N^{\sum_{e\in\mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})}}{\mathsf{OUT}_\bowtie}\end{aligned}$$

expected time for cyclic queries. Algorithm 13 succeeds with probability $\frac{\mathsf{OUT}}{\mathsf{OUT}_\bowtie}$, so the expected number of invocations before getting one successful sample is $\frac{\mathsf{OUT}_\bowtie}{\mathsf{OUT}}$. The total expected cost is:

$$O\left(\frac{N^{\rho^*(Q_\bowtie)}}{\mathsf{OUT}} + \frac{N \cdot \mathsf{OUT}_\bowtie}{\mathsf{OUT}} + N^{\sum_{e\in\mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})-1}\right)$$

Putting everything together, we obtain:

Theorem A.1. *For an acyclic join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, there is an algorithm that can take as input an arbitrary instance $\mathcal{R}$ of input size N, output size OUT and full join size $\mathsf{OUT}_\bowtie$, and builds an index in $O(N)$ time such that a uniform sample from the query result $Q(\mathcal{R})$ can be returned in*

$$O\left(\min\left\{\frac{\mathsf{OUT}_\bowtie}{\mathsf{OUT}} + N^{\sum_{e\in\mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})-1}, \ \frac{N \cdot \mathsf{OUT}_\bowtie}{\mathsf{OUT}}\right\}\right)$$

expected time, where w is the fractional edge covering of $Q_{\mathbf{y}} = (\mathcal{V}, \mathcal{E} \cup \{\mathbf{y}\})$.

Remark 1. For an acyclic join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, let $q = (\mathcal{V}, \mathcal{E})$ be the underlying full join query. Theorem A.2 can always improve the state-of-the-art method [13, 19, 28] since $\mathsf{OUT}_\bowtie \leqslant N^{\rho^*(Q_\bowtie)}$ and $\min_w N^{\sum_{e\in\mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})} \leqslant N^{\rho^*(Q_\bowtie)}$, since w degenerates to any fractional edge covering for q together with assigning $w(\mathbf{y}) = 0$. Consider an example query Q with $\mathcal{V} = \{A_1, A_2, B, C, D\}$, $\mathcal{E} = \{\{A_1, B\}, \{A_2, B\}, \{B, C\}, \{C, D\}\}$ and $\mathbf{y} = \{A_1, A_2, D\}$. On Q, we observe that $\mathsf{OUT}_\bowtie \leqslant N \cdot \mathsf{OUT}$ and $\min_w N^{\sum_{e\in\mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})} \leqslant N \cdot \mathsf{OUT}$. By our loose analysis, the state-of-the-art method would take $O(\frac{N^3}{\mathsf{OUT}})$ expected time to obtain a uniform sample while Algorithm 13 takes at most $O(N)$ expected time to obtain a uniform sample.

Remark 2. Theorem A.2 can be extended to support approximate counting following the reduction in [13]. More specifically, each trial of Algorithm 13 succeeds in returning a valid query result with probability $\frac{\mathsf{OUT}}{\mathsf{OUT}_\bowtie}$ and $\mathsf{OUT}_\bowtie$ can be computed exactly in $O(N)$ for acyclic queries. Putting everything together, we obtain:

Theorem A.2. *For an acyclic join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, there is an algorithm that can take as input an arbitrary instance $\mathcal{R}$ of input size N, output size OUT and full join size $\mathsf{OUT}_\bowtie$, and returns a constant-approximation of OUT with constant probability in*

$$O\left(N + \frac{\mathsf{OUT}_\bowtie}{\mathsf{OUT}} + \min_w N^{\sum_{e\in\mathcal{E}} w(e)} \cdot \mathsf{OUT}^{w(\mathbf{y})-1}\right)$$

time, where w is the fractional edge covering of $Q_{\mathbf{y}} = (\mathcal{V}, \mathcal{E} \cup \{\mathbf{y}\})$.

Theorem A.3. *For a cyclic join-project query $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, there is an algorithm that can take as input an arbitrary instance $\mathcal{R}$ of input size N, output size OUT and full join size $\mathsf{OUT}_\bowtie$, and builds*

an index in $O(N)$ time such that a uniform sample from the query result $Q(\mathcal{R})$ can be returned in

$$O\left(\frac{N^{\rho^*(Q_\bowtie)}}{\mathsf{OUT}} + \frac{N \cdot \mathsf{OUT}_\bowtie}{\mathsf{OUT}} + \min_w \frac{N^{\sum_{e\in\mathcal{E}} w(e)}}{\mathsf{OUT}^{1-w(\mathrm{y})}}\right)$$

expected time, where $\rho^*(Q_\bowtie)$ is the fractional edge covering number of $Q_\bowtie = (\mathcal{V}, \mathcal{E})$, and w is the fractional edge cover of $Q_{\mathrm{y}} = (\mathcal{V}, \mathcal{E} \cup \{\mathrm{y}\})$.

Remark 3. For cyclic join-project queries, even if we could bypass this initialization step, the leading term $\frac{N^{\rho^*(Q_\bowtie)}}{\mathsf{OUT}}$ in our total expected time is already as costly as the state-of-the-art methods [13, 19, 28], hence Theorem A.3 does not gain any improvement for cyclic queries.

Received December 2025; accepted February 2026