

# 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 [12] 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** → **Sketching and sampling**.

Additional Key Words and Phrases: Join-project query, Approximate Counting, Uniform Sampling, Sublinear Algorithms

## ACM Reference Format:

Xiao Hu and Jinchao Huang. 2026. Towards Output-Optimal Uniform Sampling and Approximate Counting for Join-Project Queries. *Proc. ACM Manag. Data* 4, 2 (PODS), Article 120 (May 2026), 25 pages. <https://doi.org/10.1145/3801916>

## 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 [31, 32] have settled the worst-case complexity up to the AGM bound [6], 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  $N^{\rho^*}$ , where  $\rho^*$  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.

---

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



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/5-ART120

<https://doi.org/10.1145/3801916>

For full join queries (where all attributes are retained without projection), these two problems have been extensively studied by the database community [1, 11, 18, 27, 33, 34, 38] and the algorithm community [5, 19]. Acharya et al. [1] and Chaudhuri et al. [11] initiated the investigation of uniform sampling over joins in 1999, identifying the primary barrier: the sampling operator cannot be pushed down, i.e.,  $\text{sample}(R \bowtie S) \neq \text{sample}(R) \bowtie \text{sample}(S)$ . To overcome this, research focused on precomputing indices to guide sampling. Zhao et al. [38] 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. [27] and Deng et al. [18] tackled general joins (including cyclic joins) by showing that an index can be built in  $O(N)$  time to support  $\tilde{O}\left(\frac{N\rho^*}{\text{OUT}_{\bowtie}}\right)$  uniform sampling, where  $\text{OUT}_{\bowtie}$  is the size of the full join result. Parallel to uniform sampling, Kim et al. [27] showed that the same algorithmic principles apply to approximate counting. The size of the full join,  $\text{OUT}_{\bowtie}$ , can be estimated with the 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  $\tilde{O}\left(\min\left\{N, \frac{N\rho^*}{\text{OUT}_{\bowtie}}\right\}\right)$  by leveraging the exact counting algorithm [37].

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 [12] 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  $\text{OUT}$ ; 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  $\tilde{O}\left(\frac{N\rho^*}{\text{OUT}}\right)$ . This also remains the state-of-the-art for approximate counting via a reduction to uniform sampling [12].

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,  $\mathcal{E} = \{e_1, e_2, \dots, e_k\} \subseteq 2^{\mathcal{V}}$  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  $\text{dom}(A)$  be the domain of an attribute  $A \in \mathcal{V}$ . Let  $\text{dom}(X) = \prod_{A \in X} \text{dom}(A)$  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  $\text{dom}(A)$  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:

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

We use  $N = \sum_{e \in \mathcal{E}} |R_e|$  to denote the *input size*, i.e., the total number of tuples in the input relations, and  $\text{OUT} = |\mathcal{Q}(\mathcal{R})|$  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  $\mathcal{Q}$ , which consists of 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  $\mathcal{Q}_{\text{matrix}}$ ); if we take  $\mathcal{V} = \{A_1, A_2, \dots, A_k, B\}$  with  $\mathbf{y} = \{A_1, A_2, \dots, A_k\}$ , and  $\mathcal{E} = \{\{A_1, B\}, \{A_2, B\}, \dots, \{A_k, B\}\}$ , it becomes the  $k$ -star query (denoted as  $\mathcal{Q}_{\text{star}}$ ); and if we take  $\mathcal{V} = \{A_1, A_2, \dots, A_{k+1}\}$  with  $\mathbf{y} = \{A_1, A_{k+1}\}$ , and  $\mathcal{E} = \{\{A_1, A_2\}, \{A_2, A_3\}, \dots, \{A_k, A_{k+1}\}\}$ , it becomes the  $k$ -chain query (denoted as  $\mathcal{Q}_{\text{chain}}$ ).

An important operator used together with join is semi-join. For two relations  $R_e$  and  $R_{e'}$ , the semi-join  $R_e \times R_{e'}$  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  $R_e \times t$  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  $\mathcal{Q}$  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  $\mathcal{Q}(\mathcal{R})$ . 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  $0 < \epsilon < 1$ ,  $\hat{V}$  is an  $\epsilon$ -approximation of  $V$  if  $(1 - \epsilon)V \leq \hat{V} \leq (1 + \epsilon)V$ .

**Problem 1.2** (Approximate Counting for Join-Project Query). Given parameters  $0 < \epsilon, \delta < 1$ , a join-project query  $\mathcal{Q}$  and an instance  $\mathcal{R}$ , the goal is to return an  $\epsilon$ -approximation  $\widehat{\text{OUT}}$  of the output size  $\text{OUT}$  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 a constant and measuring the complexity of algorithms by data-dependent quantities, such as input size  $N$  and output size  $\text{OUT}$ , and additional parameters, such as approximation quality  $\epsilon$  and the failure probability  $\delta$ .

**Model of Computation.** We use the standard word RAM model with uniform cost measures [15]. 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  $\text{RandInt}(a, b)$ .

In addition, for both problems on graphs (as a special case of self-joins), researchers have adopted the *property testing model* [5], 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 [27], 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  $R_e$ .

- **(access degree)** For a relation  $R_e \in \mathcal{R}$ , a tuple  $t$  defined on attributes  $e' \subseteq e$ , it returns  $|R_e \times t|$ , 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  $\pi_{e-e'}(R_e \times t)$  under a fixed ordering.<sup>1</sup>
- **(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  $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ . 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 schemas containing  $A$  is at least 1, i.e.,  $\sum_{e \in \mathcal{E}: A \in e} w(e) \geq 1$ . The *fractional edge covering number*  $\rho^*$  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  $\text{OUT}_{\bowtie}$ . The *active domain* of an attribute  $A \in \mathcal{V}$ , denoted as  $\text{adom}_{\mathcal{R}}(A)$ , is the set of values from  $\text{dom}(A)$  appearing in some tuple of  $\mathcal{R}$ , i.e.,  $\text{adom}_{\mathcal{R}}(A) = \{a \in \text{dom}(A) \mid \exists e \in \mathcal{E}, \exists t \in R_e, \pi_A t = a\}$ . When the context is clear, we use  $\text{adom}(A)$  to denote  $\text{adom}_{\mathcal{R}}(A)$ . For any  $n \in \mathbb{Z}^+$ , we use  $[n]$  to denote  $\{1, 2, \dots, n\}$ . 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  $\tilde{O}(\cdot)$  to suppress any poly-logarithmic factors.

**Handling Empty Results.** For clarity of presentation, we focus on the case when  $\text{OUT} \geq 1$ . The edge case  $\text{OUT} = 0$  is handled using a standard timeout strategy [12, 27]. Since the complexities of our algorithms are monotonically non-increasing with respect to  $\text{OUT}$  (e.g.,  $O(\frac{N}{\sqrt{\text{OUT}}})$ ), the worst-case runtime upper bound occurs when  $\text{OUT} = 1$ . Consequently, the case of  $\text{OUT} = 0$  can be detected by running the algorithm with a time budget corresponding to a constant times the complexity for  $\text{OUT} = 1$ . 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  $\delta$ ), we terminate and report  $\text{OUT} = 0$  with high probability. This timeout cost satisfies our general complexity bounds. With a slight abuse of notation, we use terms like  $O(\frac{N}{\sqrt{\text{OUT}}})$  to denote  $O(\frac{N}{\sqrt{\max\{1, \text{OUT}\}}})$  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<sup>2</sup> [38] and in  $\tilde{O}(\frac{N\rho^*}{\text{OUT}})$  time for cyclic joins [18, 27]<sup>3</sup>. The previous result for acyclic joins can be extended to the class of free-connex queries<sup>4</sup>, 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 [12] 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 [18, 27] for sampling a full join result, an index can be built in  $O(N)$  time, such that one uniform sample can be generated in  $\tilde{O}(\frac{N\rho^*}{\text{OUT}})$  time, where

<sup>1</sup>The primitives in [27] 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  $\pi_I(R_e \times t)$  under a fixed ordering.

<sup>2</sup>A join query  $q = (\mathcal{V}, \mathcal{E})$  is  $\alpha$ -acyclic [37] 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.

<sup>3</sup>[27] 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.

<sup>4</sup>A join-project query  $Q = (\mathcal{V}, \mathcal{E}, \mathbf{y})$  is free-connex [7] 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.

Problem		Matrix	Star	Chain	
				$k = 3$	$k \geq 4$
Uniform Sampling	Prior Work [12, 27]	$O(\frac{N^2}{\text{OUT}})$	$O(\frac{N^k}{\text{OUT}})$	$O(\frac{N^2}{\text{OUT}})$	$O(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\text{OUT}})$
	Our UB	$O(\frac{N}{\sqrt{\text{OUT}}})$	$O(\frac{N}{\text{OUT}^{1/k}})$	$O(\min\{N, \frac{N^2}{\text{OUT}}\})$	$O(N)$
	Our LB	$\Omega(\frac{N}{\sqrt{\text{OUT}}})$	$\Omega(\frac{N}{\text{OUT}^{1/k}})$	$\Omega(N)$ for $\text{OUT} \leq N$	$\Omega(N)$
Approximate Counting	Prior Work [12, 27]	$O(\frac{N^2}{\text{OUT}})$	$O(\frac{N^k}{\text{OUT}})$	$O(\frac{N^2}{\text{OUT}})$	$O(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\text{OUT}})$
	Our UB	$O(\frac{N}{\sqrt{\text{OUT}}})$	$O(\frac{N}{\text{OUT}^{1/k}})$	$O(\min\{N, \frac{N^2}{\text{OUT}}\})$	$O(N)$
	Our LB	$\Omega(\frac{N}{\sqrt{\text{OUT}}})$	$\Omega(\frac{N}{\text{OUT}^{1/k}})$	$\Omega(\min\{N, \frac{N^2}{\text{OUT}}\})$	$\Omega(N)$

Fig. 1. Comparison between prior results and our new results (in red) in the property testing model.  $N$  is the input size, and  $\text{OUT}$  is the output size.  $k$  is the number of relations. For uniform sampling, we assume  $O(N)$  preprocessing time. The lower bounds for uniform sampling over chain queries only apply to  $W$ -uniform sampling algorithms.<sup>5</sup>For approximate counting, we assume  $\epsilon$  as a small constant and  $\delta = 1/N^{O(1)}$  here.

$\rho^*$  is the fractional edge covering number of the underlying full join  $q$ . Hence, the best sampling time is  $\tilde{O}(\frac{N^2}{\text{OUT}})$  for matrix queries,  $\tilde{O}(\frac{N^k}{\text{OUT}})$  for  $k$ -star queries, and  $\tilde{O}(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\text{OUT}})$  for  $k$ -chain queries.

Chen and Yi [12] 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  $\frac{\text{OUT}}{N\rho^*}$ , then a constant-approximation of the output size can be returned with constant probability in  $\tilde{O}(N + \frac{N\rho^*}{\text{OUT}})$  time.

### 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  $O(\frac{N}{\sqrt{\text{OUT}}})$  expected time.

This strictly outperforms the prior best bound of  $O(\frac{N^2}{\text{OUT}})$ , yielding an improvement factor of  $O(\frac{N}{\sqrt{\text{OUT}}})$ . Since  $\text{OUT}$  ranges from 1 to  $N^2$ , this implies a polynomial speedup when  $\text{OUT} = o(N^2)$ .

**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  $\tilde{O}(\frac{N}{\sqrt{\text{OUT}}})$  time. In contrast, the previous result by the standard reduction to uniform sampling [12] requires  $\tilde{O}(\frac{N^2}{\text{OUT}})$  time. Consequently, our algorithm improves the existing result by a factor of  $\tilde{O}(\frac{N}{\sqrt{\text{OUT}}})$ .

<sup>5</sup>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  $\frac{\text{OUT}}{W}$  for some known value  $W > \text{OUT}$ . 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.

**Section 4 (lower bounds for matrix query).** We establish a matching lower bound of  $\tilde{\Omega}\left(\frac{N}{\sqrt{\text{OUT}}}\right)$  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 [12], it translates to the same lower bound for uniform sampling, assuming  $\tilde{O}\left(\frac{N}{\sqrt{\text{OUT}}}\right)$  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 [18]. The intuition is that repeatedly drawing samples for  $\tilde{O}(\text{OUT})$  iterations would retrieve the full query result with high probability. Since any combinatorial algorithm for evaluating matrix query requires  $\Omega(N\sqrt{\text{OUT}})$  time [4], a sampling algorithm (assuming  $O(N)$  preprocessing time) that improves to  $O\left(\frac{N}{\text{OUT}^{\frac{1}{2}+\gamma}}\right)$  for any small constant  $\gamma > 0$ , would imply a combinatorial evaluation algorithm running in  $\tilde{O}\left(N \cdot \text{OUT}^{\frac{1}{2}-\gamma}\right)$  time, which is impossible.

The upper and lower bounds derived for matrix query extend naturally to **star queries**. We achieve a tight time complexity of  $\Theta\left(\frac{N}{\text{OUT}^{1/k}}\right)$  for the  $k$ -star query. Due to the page limit, all missing details are provided in the full version [22].

**Section 5 (lower bounds for chain query).** We further extend the hardness from matrix query to general chain queries. For the approximate counting problem, we prove a matching lower bound  $\Omega(\min\{N, \frac{N^2}{\text{OUT}}\})$  for the 3-chain query and  $\Omega(N)$  for general chain queries with  $k \geq 4$ .

Following the reduction from approximate counting to uniform sampling [12], it translates to a lower bound of  $\Omega(N)$  for general chain queries with  $k \geq 4$  and for the 3-chain query when  $\text{OUT} \leq N$ , 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. Additionally, for both the uniform sampling and approximate counting problems, we show that these lower bounds are optimal by providing tight upper bounds. More specifically, we show algorithms with time complexity  $O(N)$  by resorting to the *k*-minimum value summary [13, 14, 23]. Together with the existing upper bound  $O\left(\frac{N^{\lceil \frac{k+1}{2} \rceil}}{\text{OUT}}\right)$  [12], we obtain a hybrid upper bound  $O(\min\{N, \frac{N^2}{\text{OUT}}\})$  for the 3-chain query and  $O(N)$  for general chain query with  $k \geq 4$ . Due to the page limit, all missing details are provided in the full version [22].

## 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  $\frac{1}{\text{deg}(a, c)}$ , where  $\text{deg}(a, c) = |\pi_B(R_1 \bowtie a) \cap \pi_B(R_2 \bowtie c)|$ , 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  $\mathcal{Q}_{\text{matrix}}$  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  $\frac{\text{deg}(a, c)}{\text{OUT}_{\bowtie}}$ , since  $(a, c)$  will

**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  $Q_{\text{matrix}}(\mathcal{R})$ .

```

1 while true do
2    $s \leftarrow \text{JOINSAMPLEMATRIX}(\mathcal{R});$  ▶ Algorithm 2 or Algorithm 4;
3   if  $s \neq \perp$  then break;
4 if MATRIXACCEPT( $\mathcal{R}, s$ ) = true then return  $(\pi_{AS}, \pi_{CS})$  ▶ Algorithm 3;
5 else return  $\perp$ ;
```

**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  $b \in \text{adom}(B)$ ,  $W_b = |R_1 \times b| \cdot |R_2 \times b|$ ; and  $W = \sum_{b \in \text{adom}(B)} W_b$ .

```

1  $b \leftarrow$  a random sample from  $\text{adom}(B)$  with probability  $\frac{W_b}{W}$ ;
2  $a \leftarrow$  the RandInt( $1, |R_1 \times b|$ )-th value in  $\pi_A(R_1 \times b)$ ;
3  $c \leftarrow$  the RandInt( $1, |R_2 \times b|$ )-th value in  $\pi_C(R_2 \times b)$ ;
4 return  $(a, b, c)$ ;
```

participate in  $\text{deg}(a, c)$  full join results, each with a distinct value in  $(R_1 \times a) \cap (R_2 \times c)$ . As  $\text{deg}(a, c)$  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  $\frac{\text{deg}(a, c)}{\text{OUT}_{\bowtie}} \cdot \frac{1}{\text{deg}(a, c)} = \frac{1}{\text{OUT}_{\bowtie}}$ , 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”  $W_b = |R_1 \times b| \cdot |R_2 \times b|$ .
- 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  $\frac{W_b}{W} \cdot \frac{1}{|R_1 \times b|} \cdot \frac{1}{|R_2 \times b|} = \frac{1}{W}$ . Consequently,  $(a, b, c)$  is a uniform sample from  $R_1 \bowtie R_2$ .

<sup>6</sup>We can implement sampling without replacement efficiently using an online version of Fisher-Yates shuffle [21, 29], which has a cost linear only to the number of samples drawn instead of the total number of elements in the universe, which is

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

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

**Output:** A Boolean value indicating acceptance.

```

1  $F \leftarrow 0$ ;
2 if  $|R_1 \times a| < |R_2 \times c|$  then  $S \leftarrow \pi_B(R_1 \times a)$ ;
3 else  $S \leftarrow \pi_B(R_2 \times c)$ ;
  // We sample from  $S$  using uniform sampling without replacement6.
4 while at least one element in  $S$  is not visited yet do
5    $b' \leftarrow$  a random sample from non-visited elements in  $S$ ;
6   if  $b' = b$  then continue;
7   if  $(a, b') \notin R_1$  or  $(b', c) \notin R_2$  then  $F \leftarrow F + 1$ ;
8   else break;
9 if  $\text{RandInt}(1, |S|) \leq F + 1$  then return true;
10 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  $\deg(a, c) = |\pi_B(R_1 \times a) \cap \pi_B(R_2 \times c)|$ . 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  $\Pr[U \leq Y] = \frac{1}{M} \cdot \mathbb{E}[Y]$ .*

**PROOF OF PROPOSITION 2.1.** By the Law of Total Probability and the independence of  $U$  and  $Y$ :  $\Pr[U \leq Y] = \sum_{y=1}^M \Pr[U \leq y] \cdot \Pr[Y = y]$ . Since  $U$  is uniform on  $[M]$ ,  $\Pr[U \leq y] = \frac{y}{M}$ . Substituting this yields:  $\Pr[U \leq Y] = \frac{1}{M} \sum_{y=1}^M y \cdot \Pr[Y = y] = \frac{\mathbb{E}[Y]}{M}$ .  $\square$

To apply this proposition, we first identify the smaller set of candidate neighbors,  $S$ . Wlog, assume  $|\pi_B(R_1 \times a)| \leq |\pi_B(R_2 \times c)|$ , so we set  $S = \pi_B(R_1 \times a)$  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  $b' \in \pi_B(R_1 \times a) \cap \pi_B(R_2 \times c)$ . 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.

<sup>6</sup> $|S|$  in the context of Algorithm 3. The procedure uses an initially empty hash map  $H$  to store index swaps. Conceptually,  $H[i] = i$  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  $\{i, i + 1, \dots, |S|\}$ , 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  $X \leq F + 1$ , 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 [25]). Consider an experiment of drawing balls without replacement from a box containing  $P$  balls, of which  $K$  are successes. Let a random variable  $Z \sim \text{NHG}(r, P, K)$  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,  $F \sim \text{NHG}(1, |S| - 1, \deg(a, c) - 1)$ . 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  $U \leq Y$ . 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  $(a, c) \in \mathcal{Q}_{\text{matrix}}(\mathcal{R})$ . In **one iteration** of the loop, the probability that the sampled full join result projects to  $(a, c)$  is  $\frac{\deg(a,c)}{\text{OUT}_{\bowtie}}$ . 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  $\frac{\deg(a,c)}{\text{OUT}_{\bowtie}} \cdot \frac{1}{\deg(a,c)} = \frac{1}{\text{OUT}_{\bowtie}}$ . Since this probability is the same for any  $(a, c) \in \mathcal{Q}_{\text{matrix}}(\mathcal{R})$ , the final output (conditioned on acceptance) is a uniform sample of  $\mathcal{Q}_{\text{matrix}}(\mathcal{R})$ .

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  $\mathbb{E}[F + 1] = \frac{|S|}{\deg(a,c)} = \frac{\min\{|R_1 \times a|, |R_2 \times c|\}}{\deg(a,c)}$ . Hence, Algorithm 1 takes:

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

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

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

<sup>7</sup> $\sum_{(a,c) \in \mathcal{Q}_{\text{matrix}}(\mathcal{R})} \min\{|R_1 \times a|, |R_2 \times c|\} \leq \sum_{(a,c) \in \mathcal{Q}_{\text{matrix}}(\mathcal{R})} \sqrt{|R_1 \times a| \cdot |R_2 \times c|} \leq N\sqrt{\text{OUT}}$ .

**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:

$$\Delta \geq \max_{b \in \text{adom}(B)} |R_2 \times b|.$$

- 1  $(a, b) \leftarrow$  a uniform sample from  $R_1$ ;
- 2 **if**  $|R_2 \times b| = 0$  **then return**  $\perp$ ;
- 3  $j \leftarrow \text{RandInt}(1, |R_2 \times b|)$ ;
- 4  $c \leftarrow$  the  $j$ -th value in  $\pi_C(R_2 \times b)$ ;
- 5 **if**  $\text{RandInt}(1, \Delta) > |R_2 \times b|$  **then return**  $\perp$ ;
- 6 **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  $\Delta$  on the maximum degree of  $B$ -values in one of the relations. Wlog, we assume  $\Delta \geq \max_{b \in \text{adom}(B)} |R_2 \times b|$ .

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  $\pi_C(R_2 \times b)$ ). A specific candidate join result  $(a, b, c)$  is initially proposed with probability  $\frac{1}{|R_1|} \cdot \frac{1}{|R_2 \times b|}$ . However, this distribution is not uniform, as the degree  $|R_2 \times b|$  varies across different  $B$ -values. To correct for this bias, the algorithm accepts the candidate  $(a, b, c)$  with probability  $\frac{|R_2 \times b|}{\Delta}$ . This is a valid probability since  $\Delta \geq |R_2 \times b|$ . Consequently, the probability of returning a specific tuple  $(a, b, c)$  becomes  $\frac{1}{|R_1|} \cdot \frac{1}{|R_2 \times b|} \cdot \frac{|R_2 \times b|}{\Delta} = \frac{1}{|R_1| \cdot \Delta}$ , 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  $R_2 \times b$ , and passing the rejection check:  $\Pr[\text{return } t] = \frac{1}{|R_1|} \cdot \frac{1}{|R_2 \times b|} \cdot \frac{|R_2 \times b|}{\Delta} = \frac{1}{|R_1| \cdot \Delta}$ . 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 [12] 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  $\epsilon$ -approximation of  $\text{OUT}$  in expected time  $O\left(N + \frac{N}{\epsilon^2 \cdot \sqrt{\text{OUT}}}\right) = O(N)$ , by the following facts: (1) Algorithm 1 returns a query result with probability  $\frac{\text{OUT}}{\text{OUT}_{\text{est}}}$ ; (2) Algorithm 1 runs in  $O\left(1 + \frac{N\sqrt{\text{OUT}}}{\text{OUT}_{\text{est}}}\right)$  expected time. However, in the property-testing model, we seek a truly sublinear algorithm whose cost decreases as the output size  $\text{OUT}$  grows. This motivates a refined reduction from approximate counting to uniform sampling that explicitly leverages  $\text{OUT}$ .

**Proposition 3.1** ([12]). *Suppose there is a uniform sampling algorithm  $\mathcal{A}$  that can successfully return a query result with probability  $\frac{\text{OUT}}{W}$ . 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  $\text{OUT}$ . If  $Y = \Omega\left(\frac{W}{\epsilon^2 \cdot \text{OUT}}\right)$ ,  $\frac{X \cdot W}{Y}$  is an  $\epsilon$ -approximation of  $\text{OUT}$  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  $\text{OUT}$  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  $\text{OUT}$  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  $\Delta$  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  $\mathcal{Q}_{\text{matrix}}$ , approximation quality  $\epsilon$  and error  $\delta$ .**Output:** An estimate of  $|\mathcal{Q}_{\text{matrix}}(\mathcal{R})|$ .

---

```

1  $\Lambda \leftarrow N^2, k \leftarrow \log \frac{2}{\delta}$ ;
2 while  $\Lambda \geq 1$  do
3   for  $i \in [k]$  do  $s_i \leftarrow \text{APPROXCOUNTMATRIXWITHGUESS}_{\epsilon, \delta/2}(\mathcal{R}, \Lambda)$ ;           ▶ Algorithm 6;
4    $s \leftarrow$  the median of  $s_1, s_2, \dots, s_k$ ;
5   if  $s \geq \Lambda$  then return  $s$ ;
6    $\Lambda \leftarrow \frac{\Lambda}{2}$ ;
7 return 0;

```

---

**Algorithm 6:** APPROXCOUNTMATRIXWITHGUESS $_{\epsilon, \delta}(\mathcal{R}, \Lambda)$ **Input:** An instance  $\mathcal{R}$  of the matrix query  $\mathcal{Q}_{\text{matrix}} = \pi_{A,C}R_1(A, B) \bowtie R_2(B, C)$ , parameter  $\Lambda \in [1, N^2]$ , approximation quality  $\epsilon$  and error  $\delta$ .**Output:** An estimate of  $|\mathcal{Q}_{\text{matrix}}(\mathcal{R})|$ .

---

```

1  $\epsilon' \leftarrow \epsilon/5$ ;
2  $B^{\text{heavy}} \leftarrow \text{DETECTHEAVY}_{\delta/4}(\mathcal{R}, \Lambda)$ ;           ▶ Algorithm 7;
3  $\mathcal{R}^{\text{heavy}} \leftarrow \{\sigma_{B \in B^{\text{heavy}}}R_1, \sigma_{B \in B^{\text{heavy}}}R_2\}$ ;           // conceptual; simulated by filtering
4  $\mathcal{R}^{\text{light}} \leftarrow \{R_1 - \sigma_{B \in B^{\text{heavy}}}R_1, R_2 - \sigma_{B \in B^{\text{heavy}}}R_2\}$ ; // conceptual; simulated by filtering
5  $\widehat{\text{OUT}}_{\bowtie}^{\text{heavy}} \leftarrow \sum_{b \in B^{\text{heavy}}} |R_1 \bowtie b| \cdot |R_2 \bowtie b|, \widehat{\text{OUT}}_{\bowtie}^{\text{light}} \leftarrow N \cdot \sqrt{\Lambda}$ ;
6  $s^{\text{heavy}} \leftarrow \text{APPROXCOUNTWITHTHRESHOLD}_{\epsilon', \delta/4}(\mathcal{R}^{\text{heavy}}, \widehat{\text{OUT}}_{\bowtie}^{\text{heavy}}, \frac{\epsilon' \Lambda}{16})$ ;           ▶ Algorithm 8;
7  $s^{\text{light}} \leftarrow \text{APPROXCOUNTWITHTHRESHOLD}_{\epsilon', \delta/4}(\mathcal{R}^{\text{light}}, \widehat{\text{OUT}}_{\bowtie}^{\text{light}}, \frac{\epsilon' \Lambda}{16})$ ;           ▶ Algorithm 8;
8 if  $s^{\text{heavy}} \neq \perp$  and  $s^{\text{light}} \neq \perp$  then
9    $\beta \leftarrow \text{INTERSECTESTIMATE}_{\epsilon', \delta/4}(\mathcal{R}, \mathcal{R}^{\text{heavy}}, \mathcal{R}^{\text{light}})$ ;           ▶ Algorithm 9;
10  return  $s^{\text{heavy}} + s^{\text{light}} - \min\{s^{\text{heavy}}, s^{\text{light}} \cdot \beta\}$ ;
11 else if  $s^{\text{heavy}} \neq \perp$  and  $s^{\text{light}} = \perp$  then return  $s^{\text{heavy}}$ ;
12 else if  $s^{\text{heavy}} = \perp$  and  $s^{\text{light}} \neq \perp$  then return  $s^{\text{light}}$ ;
13 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,  $\Lambda = N^2, \frac{N^2}{2}, \frac{N^2}{4}, \dots, 1$ . For each guess  $\Lambda$ , we invoke our core routine, APPROXCOUNTMATRIXWITHGUESS, to obtain an estimator conditioned on  $\Lambda$ . 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  $\Lambda$ , we compare it with  $\Lambda$ . If  $s$  exceeds  $\Lambda$ , we return it immediately; otherwise, we halve  $\Lambda$  and proceed to the next guess. This process continues until a valid estimate is found or all guesses have been exhausted.

**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  $\Lambda$  and error  $\delta$ .  
**Output:** A subset of values in  $\text{adom}(B)$  that appears in more than  $\sqrt{\Lambda}$  tuples in  $R_2$ .

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

---

*APPROXCOUNTMATRIXWITHGUESS.* As described in Algorithm 6, it starts with identifying the heavy values in  $B$ , using a primitive called DETECTHEAVY. A value  $b \in \text{adom}(B)$  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  $B^{\text{heavy}}$ . We *conceptually* use  $B^{\text{light}} = \text{adom}(B) - B^{\text{heavy}}$  to denote the light values, but we do not explicitly list  $B^{\text{light}}$ , 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):  $\mathcal{R}^{\text{heavy}}$  contains all tuples with heavy  $B$ -values and  $\mathcal{R}^{\text{light}}$  contains the remaining input tuples. For  $\mathcal{R}^{\text{heavy}}$ , we can exactly calculate the number of full join results. For  $\mathcal{R}^{\text{light}}$ , 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  $\Lambda$  tuples in  $R_2$ . That is essentially how we set up  $\widehat{\text{OUT}}_{\bowtie}^{\text{heavy}}$  and  $\widehat{\text{OUT}}_{\bowtie}^{\text{light}}$  as upper bounds on the number 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  $\tau$ , it successfully returns an  $\epsilon$ -approximation estimator  $s$ . If the procedure reports a failure ( $\perp$ ), it means the actual count must be less than the threshold  $\tau$ . Using the feedback ( $s^{\text{heavy}}, s^{\text{light}}$ ) on the two parts from APPROXCOUNTWITHTHRESHOLD, we combine them into a final estimator. Let  $\text{OUT}^?$  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** ( $s^{\text{heavy}} \neq \perp$  and  $s^{\text{light}} \neq \perp$ ). By the design of APPROXCOUNTWITHTHRESHOLD,  $s^{\text{heavy}}$  and  $s^{\text{light}}$  are accurate estimates of  $\text{OUT}^{\text{heavy}}$  and  $\text{OUT}^{\text{light}}$ , respectively, and both counts exceed the threshold. Since the result sets  $Q_{\text{matrix}}(\mathcal{R}^{\text{heavy}})$  and  $Q_{\text{matrix}}(\mathcal{R}^{\text{light}})$  may overlap, their sum would double-count the intersection. We employ the procedure INTERSECTESTIMATE to estimate  $\beta$ , defined as the fraction of  $Q_{\text{matrix}}(\mathcal{R}^{\text{light}})$  that is also present in  $Q_{\text{matrix}}(\mathcal{R}^{\text{heavy}})$ . Consequently,  $s^{\text{light}} \cdot \beta$  estimates the size of this intersection. However, stochastic errors in estimating  $\beta$  could yield an intersection estimate larger than  $\text{OUT}^{\text{heavy}}$ , which would destabilize the final result. To mitigate this, we leverage the fact that the intersection size cannot exceed  $\text{OUT}^{\text{heavy}}$ , yielding the robust combined estimator  $s^{\text{heavy}} + s^{\text{light}} - \min\{s^{\text{light}} \cdot \beta, s^{\text{heavy}}\}$ .
- **Case 2: Only one estimate fails.** Without loss of generality, assume  $s^{\text{light}} = \perp$ . This failure implies that  $\text{OUT}^{\text{light}}$  is negligible (below the threshold). Consequently,  $s^{\text{heavy}}$ , which accurately estimates  $\text{OUT}^{\text{heavy}}$ , serves as a sufficient approximation for the total output size  $\text{OUT}$ .
- **Case 3: Both estimates fail** ( $s^{\text{heavy}} = \perp$  and  $s^{\text{light}} = \perp$ ). These failures indicate that both  $\text{OUT}^{\text{heavy}}$  and  $\text{OUT}^{\text{light}}$  are below the threshold, implying that their union is at most  $\frac{\epsilon \cdot \Lambda}{8}$ . As this is significantly smaller than the current guess  $\Lambda$ , we conclude that  $\Lambda$  is an overestimate and return 0 (signaling the algorithm to proceed to the next guess).

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

**Input:** An instance  $\mathcal{R}$  of  $\mathcal{Q}_{\text{matrix}} = \pi_{A,C}R_1(A, B) \bowtie R_2(B, C)$ , an upper bound  $\widehat{\text{OUT}}_{\bowtie}$  of the full join size  $|R_1 \bowtie R_2|$  and threshold  $\tau$ .

**Output:** An estimator of  $|\mathcal{Q}_{\text{matrix}}(\mathcal{R})|$  or  $\perp$  indicating failure.

```

1  $\lambda \leftarrow N^2, k_\delta \leftarrow 3 \log \frac{2}{\delta};$ 
2 while  $\lambda \geq \tau$  do
3   for  $i \in [k_\delta]$  do
4      $Z \leftarrow 0, k_\lambda \leftarrow \frac{6\widehat{\text{OUT}}_{\bowtie}}{\epsilon^2 \lambda};$ 
5     for  $j \in [k_\lambda]$  do
6       // SampleMatrix-H for  $\mathcal{R}^{\text{heavy}}$  and SampleMatrix-L for  $\mathcal{R}^{\text{light}}$ ;
7        $s \leftarrow \text{SAMPLEMATRIX}(\mathcal{R});$  ► Algorithm 1;
8       if  $s \neq \perp$  then  $Z \leftarrow Z + \widehat{\text{OUT}}_{\bowtie};$ 
9      $s_i \leftarrow \frac{Z}{k_\lambda};$ 
10     $s \leftarrow$  the median of  $s_1, s_2, \dots, s_{k_\delta};$ 
11    if  $s \geq \lambda$  then return  $s;$ 
12     $\lambda \leftarrow \frac{\lambda}{2};$ 
13 return  $\perp;$ 

```

*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  $B^{\text{heavy}}$ . Also, it checks the true degree of values in  $B^{\text{heavy}}$ , so that  $B^{\text{heavy}} \subseteq \{b \in \text{adom}(B) : |R_2 \bowtie b| \geq \sqrt{\Lambda}\}$ . Hence, we can bound the size of  $B^{\text{heavy}}$  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  $B^{\text{heavy}} = \{b \in \text{adom}(B) : |R_2 \bowtie b| \geq \sqrt{\Lambda}\}$ .

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

- 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  $\frac{1}{N\Lambda}$  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  $S = \pi_B(\sigma_{B \notin B^{\text{heavy}}}(R_1 \times a))$ , we simply sample values from  $\pi_B(R_1 \times a)$  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  $B^{\text{heavy}}$  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  $S = \pi_B(\sigma_{B \in B^{\text{heavy}}}(R_1 \times a))$ , we simply sample values from  $\pi_B(R_1 \times a)$  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.

*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,  $\lambda = N^2, \frac{N^2}{2}, \frac{N^2}{4}, \dots, \tau$ , where  $\tau$

**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  $\frac{|Q_{\text{matrix}}(\mathcal{R}^{\text{heavy}}) \cap Q_{\text{matrix}}(\mathcal{R}^{\text{light}})|}{|Q_{\text{matrix}}(\mathcal{R}^{\text{light}})|}$ .

```

1  $S, S' \leftarrow \emptyset$ ;
2 while  $|S| \leq \frac{1}{2\epsilon^2} \log \frac{2}{\delta}$  do
3   while true do
4      $s \leftarrow \text{SAMPLEMATRIX-L}(\mathcal{R}^{\text{light}})$ ; ► Algorithm 1;
5     if  $s \neq \perp$  then  $S \leftarrow S \cup \{s\}$  and break;
6   for each  $b \in B^{\text{heavy}}$  do
7     if  $(\pi_{AS}, b) \in R_1$  and  $(b, \pi_{CS}) \in R_2$  then  $S' \leftarrow S' \cup \{s\}$ ;
8 return  $\frac{|S'|}{|S|}$ ;
```

is the smallest guess we can try. This threshold is enforced to satisfy the time complexity constraints. Note that this guess  $\lambda$  is independent of the guess  $\Lambda$  in the outermost procedure Algorithm 5, since this is for estimating  $\text{OUT}^{\text{heavy}}, \text{OUT}^{\text{light}}$  separately, which can be very different from  $\text{OUT}$ . For each guess  $\lambda$ , we simply apply the reduction in Proposition 3.1 as Lines 3-9. Recall that the  $\text{SAMPLEMATRIX}$  procedure returns a uniform sample of the query result from  $Q_{\text{matrix}}(\mathcal{R})$  with probability  $\frac{\text{OUT}}{\text{OUT}_{\bowtie}}$ .

Repeating this procedure  $O(\frac{\text{OUT}_{\bowtie}}{\epsilon^2 \lambda})$  independent times should give us a constant-approximation with at least constant probability when  $\lambda = O(\text{OUT})$ . 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  $\lambda$ , we return it immediately; otherwise, we halve  $\lambda$  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  $B^{\text{heavy}}$ . When taking a uniform sample from  $Q_{\text{matrix}}(\mathcal{R}^{\text{light}})$ , it hits the overlap with probability  $\frac{|Q_{\text{matrix}}(\mathcal{R}^{\text{heavy}}) \cap Q_{\text{matrix}}(\mathcal{R}^{\text{light}})|}{\text{OUT}^{\text{light}}}$ . Hence,  $\frac{|S'|}{|S|} \cdot \text{OUT}^{\text{light}}$  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  $\epsilon$  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  $1 - \delta$ ,  $B^{\text{heavy}} = \{b \in \text{adom}(B) : |R_2 \times b| \geq \sqrt{\Lambda}\}$ .*

**PROOF OF LEMMA 3.2.** As mentioned,  $B^{\text{heavy}} \subseteq \{b \in \text{adom}(B) : |R_2 \times b| \geq \sqrt{\Lambda}\}$  always holds. It suffices to show that  $\{b \in \text{adom}(B) : |R_2 \times b| \geq \sqrt{\Lambda}\} \subseteq B^{\text{heavy}}$ . Consider an arbitrary value  $b \in \text{adom}(B)$  with  $|R_2 \times b| \geq \sqrt{\Lambda}$ . In each attempt, the probability that no tuple in  $R_2 \times b$  is sampled is  $1 - \frac{|R_2 \times b|}{|R_2|}$ . Then, in  $k = \frac{N \cdot \log \frac{N}{\delta}}{\sqrt{\Lambda}}$  independent attempts, the probability that no tuple in  $R_2 \times b$

is sampled is  $\left(1 - \frac{|R_2 \times b|}{|R_2|}\right)^{\frac{N \cdot \log \frac{N}{\delta}}{\sqrt{\lambda}}} \leq \left(1 - \frac{\sqrt{\lambda}}{N}\right)^{\frac{N \cdot \log \frac{N}{\delta}}{\sqrt{\lambda}}} \leq \exp(-\log \frac{N}{\delta}) \leq \frac{\delta}{N}$ . 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  $\delta$ .  $\square$

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

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

**PROOF OF LEMMA 3.3.** Now, consider an arbitrary iteration of the while-loop with guess  $\lambda$ . We show that  $\mathbb{E}[s_i] = \text{OUT}$ . Each invocation of `SAMPLEMATRIX` successfully returns any sample with probability  $\frac{\text{OUT}}{\text{OUT}_{\text{pt}}}$ . Let  $X$  be the number of times when `SAMPLEMATRIX` succeeds out of  $k_\lambda$  attempts. Hence,  $\mathbb{E}[X] = \frac{k_\lambda \cdot \text{OUT}}{\text{OUT}_{\text{pt}}}$  and  $\text{Var}[X] \leq \frac{k_\lambda \cdot \text{OUT}}{\text{OUT}_{\text{pt}}}$ . Hence,  $\mathbb{E}[s_i] = \frac{\widehat{\text{OUT}}_{\text{pt}}}{k_\lambda} \cdot \mathbb{E}[X] = \text{OUT}$ . Hence, if Algorithm 8 returns an estimator  $s$ , we must have  $\mathbb{E}[s] = \text{OUT}$ . By the Markov inequality,  $\Pr[s_i \geq 2\text{OUT}] \leq \frac{1}{2}$ . Taking the median of  $k_\delta$  such estimators,

$$\Pr[s \leq 2\text{OUT}] \geq 1 - \left(\frac{1}{2}\right)^{3 \log \frac{2}{\delta}} = 1 - (\delta/2)^3 \geq 1 - \delta/2.$$

Furthermore, the variance of  $s_i$  is

$$\text{Var}[s_i] = \left(\frac{1}{k_\lambda}\right)^2 \cdot \text{Var}[Z] = \left(\frac{\widehat{\text{OUT}}_{\text{pt}}}{k_\lambda}\right)^2 \cdot \text{Var}[X] \leq \left(\frac{\widehat{\text{OUT}}_{\text{pt}}}{k_\lambda}\right)^2 \cdot \frac{k_\lambda \cdot \text{OUT}}{\widehat{\text{OUT}}_{\text{pt}}} = \frac{\epsilon^2 \lambda}{6} \cdot \text{OUT}.$$

By Chebyshev inequality,  $\Pr[|s_i - \text{OUT}| \geq \epsilon \text{OUT}] \leq \frac{\text{Var}[s_i]}{\epsilon^2 \cdot \text{OUT}^2} \leq \frac{\lambda}{6\text{OUT}}$ . Taking the median of  $k_\delta$  such estimators,

$$\Pr[|s - \text{OUT}| \leq \epsilon \text{OUT}] \geq 1 - \left(\frac{\lambda}{6\text{OUT}}\right)^{3 \log \frac{2}{\delta}}.$$

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

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

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

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

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

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

**PROOF OF LEMMA 3.4.** Let  $J = \mathcal{Q}_{\text{matrix}}(\mathcal{R}^{\text{light}})$  be the set of query results from the light part, and let  $I = \mathcal{Q}_{\text{matrix}}(\mathcal{R}^{\text{heavy}}) \cap \mathcal{Q}_{\text{matrix}}(\mathcal{R}^{\text{light}})$  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  $t > 0$ :  $\Pr[|\hat{p} - p| \geq t] \leq 2\exp(-2Mt^2)$ . Setting  $t = \epsilon$ ,  $\Pr[|\hat{p} - p| \geq \epsilon] \leq 2\exp(-2\frac{1}{2\epsilon^2} \log \frac{2}{\delta} \cdot \epsilon^2) \leq \delta$ . This implies that  $|\hat{p} - p| \leq \epsilon$  with probability at least  $1 - \delta$ .  $\square$

**Lemma 3.5.** *Algorithm 6 returns an estimator  $s_\Lambda$  with  $\mathbb{E}[s_\Lambda] \leq 2\text{OUT}$ . Furthermore, if  $\Lambda \leq 4\text{OUT}$ ,  $s_\Lambda$  is an  $\epsilon$ -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  $s^{\text{heavy}}$  (resp.,  $s^{\text{light}}$ ) is not assigned  $\perp$  at Line 6 (resp., Line 7), then  $\mathbb{E}[s^{\text{heavy}}] = \text{OUT}^{\text{heavy}}$  (resp.,  $\mathbb{E}[s^{\text{light}}] = \text{OUT}^{\text{light}}$ ). Even if both  $s^{\text{heavy}}$  and  $s^{\text{light}}$  are not  $\perp$ , we have  $\mathbb{E}[s_\Lambda] \leq \mathbb{E}[s^{\text{heavy}} + s^{\text{light}}] = \text{OUT}^{\text{heavy}} + \text{OUT}^{\text{light}} \leq 2\text{OUT}$ . Analyze the other 3 cases of  $s^{\text{heavy}}$  and  $s^{\text{light}}$  analogously, we also have  $\mathbb{E}[s_\Lambda] \leq 2 \cdot \text{OUT}$ .

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

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

Conditioned on Line 9 succeeding, we further analyze:

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

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

The total error of the estimator  $s_\Lambda$  is:

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

Thus,  $s_\Lambda$  is an  $\epsilon$ -approximation of OUT.

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

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

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

$$\text{OUT}^{\text{light}} \leq \text{OUT} < \text{OUT}^{\text{light}} + \epsilon'\text{OUT}^{\text{heavy}} + 3\epsilon'\text{OUT}^{\text{light}} \leq \text{OUT}^{\text{light}} + 4\epsilon'\text{OUT}.$$

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

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

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

Thus,  $s_\Lambda = s^{\text{light}}$  is an  $\epsilon$ -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  $1 - \delta/4$ ,  $\text{OUT}^{\text{light}} < \frac{\epsilon' \Lambda}{8}$ , and  $s^{\text{heavy}}$  is an  $\epsilon'$ -approximation of  $\text{OUT}^{\text{heavy}}$ . Moreover,  $\text{OUT}^{\text{heavy}} \geq \frac{\text{OUT}}{2} \geq \frac{\Lambda}{8}$ . We work out an upper bound of  $\text{OUT}$  in terms of  $\text{OUT}^{\text{heavy}}$ :

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

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

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

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

Thus, if  $\Lambda \leq 4\text{OUT}$ , with probability at least  $1 - \delta$ ,  $s_\Lambda$  is an  $\epsilon$ -approximation for  $\text{OUT}$ .  $\square$

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

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

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

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

**PROOF OF LEMMA 3.7.** We analyze the total time complexity by summing the cost of `APPROX-COUNTMATRIXWITHGUESS` over the sequence of guesses  $\Lambda = N^2, N^2/2, \dots, 1$ . The algorithm terminates with high probability when  $\Lambda \leq \text{OUT}$ . Thus, we only consider  $\Lambda \geq 8\text{OUT}$ .

For a fixed  $\Lambda$ , the procedure `APPROXCOUNTWITHTHRESHOLD` iterates guesses  $\lambda$  from  $N^2$  down to a threshold  $\tau = \frac{\epsilon' \Lambda}{16} = \Theta(\epsilon \Lambda)$ . The cost for a specific  $\lambda$  is dominated by the sampling loop, which performs  $k_\lambda = O\left(\frac{\widehat{\text{OUT}}_{\text{light}}}{\epsilon^2 \lambda}\right)$  iterations. The expected cost per iteration is  $O\left(1 + \frac{N \cdot \sqrt{\text{OUT}}}{\text{OUT}_{\text{light}}}\right)$ .

Thus, the total cost for a fixed  $\lambda$  is:  $T(\lambda) \approx \frac{\widehat{\text{OUT}}_{\text{light}}}{\epsilon^2 \lambda} \left(1 + \frac{N \cdot \sqrt{\text{OUT}}}{\text{OUT}_{\text{light}}}\right) = \frac{1}{\epsilon^2 \lambda} (\widehat{\text{OUT}}_{\text{light}} + N \sqrt{\text{OUT}})$ . For the light part,  $\widehat{\text{OUT}}_{\text{light}} = N\sqrt{\Lambda}$ . For the heavy part,  $\widehat{\text{OUT}}_{\text{heavy}} \leq N\sqrt{\text{OUT}^{\text{heavy}}} \leq N\sqrt{\Lambda}$  (since  $\text{OUT}^{\text{heavy}} \leq \text{OUT} \leq \Lambda$  in the relevant range). Thus,  $T(\lambda) \leq O\left(\frac{N\sqrt{\Lambda}}{\epsilon^2 \lambda}\right)$ .

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

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

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

<sup>8</sup>greater than or approximately equal to

$\frac{N}{\epsilon^2 \text{OUT}} \sum_k \sqrt{2^k \text{OUT}} = \frac{N}{\epsilon^2 \sqrt{\text{OUT}}} \sum_k (\sqrt{2})^k$ . The sum is dominated by the largest term ( $\Lambda \approx \text{OUT}/\epsilon$ ):

$$s_1 \approx \frac{N}{\epsilon^2 \sqrt{\text{OUT}}} \cdot \sqrt{\frac{1}{\epsilon}} = \frac{N}{\epsilon^{2.5} \sqrt{\text{OUT}}}.$$

- **Range 2:**  $\Lambda > \frac{\text{OUT}}{\epsilon}$ . Here,  $\tau \approx \epsilon\Lambda > \text{OUT}$ , so the inner loop stops at  $\lambda_{\text{stop}} = \tau \approx \epsilon\Lambda$ .  $\text{Cost}(\Lambda) \approx \frac{N\sqrt{\Lambda}}{\epsilon^2(\epsilon\Lambda)} = \frac{N}{\epsilon^3\sqrt{\Lambda}}$ . Summing over  $\Lambda = 2^k(\text{OUT}/\epsilon)$ :  $\sum_{\Lambda} \text{Cost}(\Lambda) \approx \frac{N}{\epsilon^3} \sum_{\Lambda} \frac{1}{\sqrt{\Lambda}}$ . The sum is dominated by the smallest term ( $\Lambda \approx \text{OUT}/\epsilon$ ):  $s_2 \approx \frac{N}{\epsilon^3 \sqrt{\text{OUT}/\epsilon}} = \frac{N}{\epsilon^{2.5} \sqrt{\text{OUT}}}$ .

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

**Theorem 3.8.** For  $Q_{\text{matrix}}$ , given parameters  $0 < \epsilon, \delta < 1$ , there is an algorithm that can take as input an arbitrary input instance  $\mathcal{R}$ , and output an  $\epsilon$ -approximation of  $|Q_{\text{matrix}}(\mathcal{R})|$  in  $O\left(\frac{N \cdot \log(N/\delta) \cdot \log(1/\delta)}{\epsilon^{2.5} \sqrt{\text{OUT}}}\right)$  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 [8, 26, 35].

**Lemma 4.1.** For  $Q_{\text{matrix}}$ , given any nonnegative integers  $N, \text{OUT}, n$ , any algorithm that takes as input an arbitrary instance  $\mathcal{R}$  of input size  $N$ , output size  $\text{OUT}$ , and active domain size  $n$  for  $B$ , and returns an  $O(1)$  approximation of  $\text{OUT}$  with high probability, requires at least  $\Omega(\min\{n, \frac{N}{\sqrt{\text{OUT}}}\})$  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  $Q_{\text{matrix}}$  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  $R_1 = \text{adom}(A) \times S_A$ . Bob defines relation  $R_2(B, C)$  as  $R_2 = S_B \times \text{adom}(C)$ . 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  $\text{OUT} = |\pi_{A,C}(R_1 \bowtie R_2)|$  depends on the intersection of  $S_A$  and  $S_B$ . If  $S_A \cap S_B = \emptyset$ , then  $\text{OUT} = 0$ . If  $S_A \cap S_B \neq \emptyset$ , then  $\text{OUT} = |\text{adom}(A)| \cdot |\text{adom}(C)| = K$ .

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  $\text{OUT} = 0$  case and the  $\text{OUT} = K$  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  $\text{adom}(A)$  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  $a \in \text{adom}(A)$  is  $|S_A|$ , and the neighbors are the elements of  $S_A$ ). Finally, for a sample query, Alice draws uniformly from  $\text{adom}(A) \times S_A$ . Bob handles queries on  $R_2$  symmetrically. If  $\mathcal{A}$  provides an  $O(1)$  approximation of  $\text{OUT}$  within  $o(\min\{n, \frac{N}{\sqrt{\text{OUT}}}\}) = o(n)$  queries, the set disjointness problem is solved within  $o(n) = o(m)$  communication cost, leading to a contradiction.  $\square$

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

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

PROOF OF THEOREM 4.2. For any given  $N, \text{OUT}$ , we distinguish the following two cases:

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

If  $\text{OUT} = 0$ , there is a hard instance for  $\mathcal{Q}_{\text{matrix}}$  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  $\text{dom}(A)$  and any  $\frac{N}{2}$  values in  $\text{dom}(B)$ . Relation  $R_2$  is the Cartesian product of the remaining  $\frac{N}{2}$  values in  $\text{dom}(B)$  and  $\text{dom}(C)$ . So, any algorithm needs to spend  $\Omega(\min\{n, \frac{N}{\sqrt{\text{OUT}}}\}) = \Omega(N)$  time.

Combining these two cases yields the desired lower bound.  $\square$

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  $\Omega(\frac{N}{\sqrt{\text{OUT}}})$  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  $O(\frac{N}{\text{OUT}^{\frac{1}{2}+\gamma}})$  time, while generating each (independent) uniform sample in  $O(\frac{N}{\text{OUT}^{\frac{1}{2}+\gamma}})$  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  $O(\frac{N}{\sqrt{\text{OUT}}})$  time. Below, we give a stronger lower bound for combinatorial algorithms following [18]:

**Theorem 4.3.** For  $\mathcal{Q}_{\text{matrix}}$ , assuming the combinatorial hardness of Boolean matrix multiplication<sup>9</sup>, given any nonnegative integers  $N, \text{OUT}$ , there is no combinatorial algorithm that preprocesses an instance  $\mathcal{R}$  of input size  $N$  and output size  $\text{OUT}$  in  $O(N \cdot \text{OUT}^{\frac{1}{2}-\gamma})$  time and generates a uniform sample from  $\mathcal{Q}_{\text{matrix}}(\mathcal{R})$  in  $O(\frac{N}{\text{OUT}^{\frac{1}{2}+\gamma}})$  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  $\mathcal{Q}_{\text{matrix}}$  requires  $\Omega(N\sqrt{\text{OUT}})$  time [4]. Suppose, for the sake of contradiction, that there exists a sampling algorithm  $\mathcal{A}$  with preprocessing time  $O(N \cdot \text{OUT}^{\frac{1}{2}-\gamma})$  and expected sampling time  $T = O(\frac{N}{\text{OUT}^{1/2+\gamma}})$  for some constant  $\gamma > 0$ . We construct an algorithm to compute the full join result  $\mathcal{Q}_{\text{matrix}}(\mathcal{R})$  without prior knowledge of  $\text{OUT}$ , using the estimation technique from [18].

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 [18], this stopping condition implies that with high probability, we have collected at least half of the total results, i.e.,  $k \geq \text{OUT}/2$ . Consequently,  $\Lambda \in [\text{OUT}, 2\text{OUT}]$  with high probability. The number of samples drawn in this step is  $\tilde{O}(\text{OUT})$ .

<sup>9</sup>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  $\Lambda$ , we continue invoking  $\mathcal{A}$  for a total of  $\Theta(\Lambda \log N)$  trials. Since  $\Lambda \geq \text{OUT}$ , this satisfies the requirement for the Coupon Collector's Problem ( $\Theta(\text{OUT} \log \text{OUT}) = \Theta(\text{OUT} \log N)$  samples) to ensure that all distinct tuples in  $\mathcal{Q}_{\text{matrix}}(\mathcal{R})$  are collected with high probability.

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

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

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

## 5 Lower Bounds for 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  $\Omega(\min\{N, \frac{N^2}{\text{OUT}}\})$  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 5.1.** *For  $\mathcal{Q}_{\text{chain}} = \pi_{A,D}R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$ , given any parameters  $N, \text{OUT} \in \mathbb{Z}^+$ , any randomized algorithm that takes as input an arbitrary instance  $\mathcal{R}$  of input size  $N$  and output size  $\text{OUT}$ , and returns a constant-approximation of  $|\mathcal{Q}_{\text{chain}}(\mathcal{R})|$  with constant probability, requires at least  $\Omega(\min\{N, \frac{N^2}{\text{OUT}}\})$  time, under the property testing model.*

**PROOF OF THEOREM 5.1.** Consider an arbitrary constant  $\theta \geq 2$  and two arbitrary integers  $\Delta, L \in \mathbb{Z}^+$  such that  $\Delta \cdot L \leq \frac{N^2}{\theta}$  and  $(\theta + 1)L \leq N$ . We define two distributions of instances,  $\mathcal{D}_0$  and  $\mathcal{D}_1$ . Let  $\text{dom}(A) = \{a_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$ ,  $\text{dom}(B) = \{b_i : i \in [3N]\}$ ,  $\text{dom}(C) = \{c_i : i \in [3N]\}$ , and  $\text{dom}(D) = \{d_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$ .

First, we choose a subset  $B_\alpha \subset \text{dom}(B)$  of size  $\frac{N}{\sqrt{\Delta}}$  and a subset  $C_\alpha \subset \text{dom}(C)$  of size  $\frac{N}{\sqrt{\Delta}}$ . Let  $B_\beta = \text{dom}(B) - B_\alpha$  and  $C_\beta = \text{dom}(C) - C_\alpha$ . 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  $\text{OUT} = L \cdot (\sqrt{\Delta})^2 = L \cdot \Delta$ .
- **Distribution  $\mathcal{D}_1$ :** We construct  $R_2$  by adding  $\theta L$  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  $\theta L$  random tuples from  $B_\beta \times C_\beta$ . The output size is  $\text{OUT}_1 = \theta L \cdot \Delta$ .

In both cases,  $|R_2| = 2N$ . The constraints on  $\Delta$  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  $\text{OUT}_1 = \theta \cdot \text{OUT}$  and  $\theta \geq 2$ , 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  $B_\beta$  or  $C_\beta$ , 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 that a sampled tuple falls into  $B_\alpha \times C_\alpha$  is  $\frac{L}{2N}$  in  $\mathcal{D}_0$  and  $\frac{\theta L}{2N}$  in  $\mathcal{D}_1$ . To distinguish these cases,  $\mathcal{A}$  requires  $\Omega(\frac{2N}{\theta L}) = \Omega(\frac{N}{L})$  samples.
- **access degree** In both distributions, the total number of edges incident to  $B_\alpha$  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  $B_\alpha$ , there are exactly  $N$  incident edges, creating a total of  $N$  valid query slots (i.e.,  $(b, j)$  pairs). In our construction, the  $L$  (in  $\mathcal{D}_0$ ) or  $\theta L$  (in  $\mathcal{D}_1$ ) critical edges connecting  $B_\alpha$  to  $C_\alpha$  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  $B_\alpha$ . The probability that such a query reveals a critical edge is  $\frac{L}{N}$  in  $\mathcal{D}_0$  and  $\frac{\theta L}{N}$  in  $\mathcal{D}_1$ . With the **sample tuple** oracle, distinguishing these cases needs  $\Omega(\frac{N}{L})$  queries.

Combining the 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  $\Omega\left(\min\{N, \frac{N^2}{\text{OUT}}\}\right)$  follows by choosing  $\Delta = \Theta(\text{OUT})$  and  $L = O(1)$ .  $\square$

**Theorem 5.2.** For  $\mathcal{Q}_{\text{chain}} = \pi_{A,E}R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D) \bowtie R_4(D, E)$ , given any parameters  $N, \text{OUT} \in \mathbb{Z}^+$ , any randomized algorithm that takes as input an arbitrary instance  $\mathcal{R}$  of input size  $N$  and output size  $\text{OUT}$ , and returns a constant-approximation of  $|\mathcal{Q}_{\text{chain}}(\mathcal{R})|$  with constant probability, requires at least  $\Omega(N)$  time, under the property testing model.

**PROOF OF THEOREM 5.2.** If  $\text{OUT} \leq N$ , the lower bound follows from the 3-chain case (Theorem 5.1) by setting  $R_1$  as a one-to-one mapping. We focus on the case where  $\text{OUT} > N$ .

Consider any constant  $\theta \geq 2$ . We choose parameters  $L = \Theta(1)$  and  $\Delta = \Theta(\text{OUT})$  such that  $\Delta \cdot L \leq \frac{N^2}{\theta}$  and  $(\theta + 1)L \leq N$  hold. We define two distributions of instances,  $\mathcal{D}_0$  and  $\mathcal{D}_1$ . Let  $\text{dom}(A) = \{a_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$ ,  $\text{dom}(B) = \{b_i : i \in [3N]\}$ ,  $\text{dom}(C) = \{c_i : i \in [3N]\}$ ,  $\text{dom}(D) = \{d_i : i \in [3N]\}$ , and  $\text{dom}(E) = \{e_{ij} : i \in [3N], j \in [\sqrt{\Delta}]\}$ . We fix subsets  $B_\alpha \subset \text{dom}(B)$  and  $D_\alpha \subset \text{dom}(D)$ , each of size  $\frac{N}{\sqrt{\Delta}}$ . Let  $B_\beta = \text{dom}(B) - B_\alpha$  and  $D_\beta = \text{dom}(D) - D_\alpha$ . We also choose a subset  $C_\alpha \subset \text{dom}(C)$  of size  $N$  and partition it into  $\ell = \frac{N^2}{\text{OUT}}$  groups  $C_1, \dots, C_\ell$ , each of size  $\frac{\text{OUT}}{N}$ . 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  $C_\alpha$ , 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 5.1. The output size is  $\text{OUT}_0 = L \cdot (\sqrt{\Delta})^2 = L \cdot \Delta$ .
- **Distribution  $\mathcal{D}_1$ :** The construction is identical, except we select  $\theta L$  pairs instead of  $L$ . The output size is  $\text{OUT}_1 = \theta L \cdot \Delta$ .

Since  $\text{OUT}_1 = \theta \cdot \text{OUT}_0$ , 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  $b \rightarrow c \rightarrow d$ . 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  $|C_h| = \frac{\text{OUT}}{N}$ , finding the witness requires  $\Omega(\frac{\text{OUT}}{N})$  queries. The total complexity is:  $\Omega\left(\frac{1}{\rho} \cdot \frac{\text{OUT}}{N}\right) = \Omega\left(\frac{N^2}{L\Delta} \cdot \frac{\text{OUT}}{N}\right)$ . Substituting  $\text{OUT} = \Theta(L\Delta)$ , 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  $\theta L$  requires  $\Omega(\frac{N}{L})$  samples. Since we chose  $L = \Theta(1)$ , this cost is  $\Omega(N)$ .
- **access degree or neighbor:** As in Theorem 5.1, the marginal degree distributions of  $B_\alpha$  and  $D_\alpha$  are masked by the noise tuples. Accessing a neighbor is statistically equivalent to sampling, yielding the same  $\Omega(N)$  lower bound.  $\square$

## 6 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. Furthermore, a rigorous parameterized complexity analysis is required to characterize algorithmic performance relative to both input and output sizes.
- *Support for 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 select-project-join queries and to determine the impact of selectivity on the complexity. Very recently, [24] investigated the sampling indices for acyclic joins when point-selection predicates are present. But the problem remains open for more complex 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, 9, 10, 16, 17, 20, 28, 30, 36], 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 Sibor 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] 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.
- [6] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.
- [7] 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.
- [8] 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.
- [9] K. Censor-Hillel, T. Even, and V. Vassilevska Williams. Fast approximate counting of cycles. In *ICALP 2024*, 2024.
- [10] 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.
- [11] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. *ACM SIGMOD Record*, 28(2):263–274, 1999.
- [12] Y. Chen and K. Yi. Random sampling and size estimation over cyclic joins. In *ICDT*, 2020.
- [13] 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.
- [14] E. Cohen and H. Kaplan. Tighter estimation using bottom k sketches. *Proceedings of the VLDB Endowment*, 1(1): 213–224, 2008.
- [15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.
- [16] 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.
- [17] 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.
- [18] S. Deng, S. Lu, and Y. Tao. On join sampling and hardness of combinatorial output-sensitive join algorithms. In *PODS*, 2023.
- [19] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.
- [20] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004.
- [21] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*. Hafner Publishing Company, 1953.
- [22] X. Hu and J. Huang. Towards output-optimal uniform sampling and approximate counting for join-project queries. In <http://arxiv.org/abs/2603.12560>, 2026.
- [23] 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.
- [24] 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.
- [25] N. L. Johnson, A. W. Kemp, and S. Kotz. *Univariate discrete distributions*. John Wiley & Sons, 2005.
- [26] B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.
- [27] K. Kim, J. Ha, G. Fletcher, and W.-S. Han. Guaranteeing the  $\tilde{O}(\text{AGM}/\text{OUT})$  runtime for uniform sampling and size estimation over joins. In *PODS*, page 113–125, 2023.
- [28] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000.
- [29] D. E. Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley Professional, 1998.

- [30] M. Kowaluk, A. Lingas, and E. Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discret. Math.*, 27(2):892–909, 2013.
- [31] 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.
- [32] 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.
- [33] F. Olken and D. Rotem. Simple random sampling from relational databases. In *Proceedings of the VLDB Endowment*, pages 160–169, 1986.
- [34] F. Olken and D. Rotem. Random sampling from databases: a survey. *Statistics and Computing*, 5(1):25–42, 1995.
- [35] A. A. Razborov. On the distributional complexity of disjointness. In *International Colloquium on Automata, Languages, and Programming*, pages 249–253. Springer, 1990.
- [36] 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.
- [37] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. International Conference on Very Large Data Bases*, pages 82–94, 1981.
- [38] Z. Zhao, R. Christensen, F. Li, X. Hu, and K. Yi. Random sampling over joins revisited. In *SIGMOD*, pages 1525–1539, 2018.

Received December 2025; accepted February 2026