On Join Sampling and the Hardness of Combinatorial Output-Sensitive Join Algorithms

Shiyuan Deng, Shangqi Lu, and Yufei Tao
{sydeng,sqlu,taoyf}@cse.cuhk.edu.hk
Chinese University of Hong Kong
Hong Kong, China

ABSTRACT

We present a dynamic index structure for *join sampling*. Built for an (equi-) join Q — let IN be the total number of tuples in the input relations of Q — the structure uses $\tilde{O}(\mathrm{IN})$ space, supports a tuple update of any relation in $\tilde{O}(1)$ time, and returns a uniform sample from the join result in $\tilde{O}(\mathrm{IN}^{\rho^*}/\max\{1, \mathrm{OUT}\})$ time with high probability (w.h.p.), where OUT and ρ^* are the join's output size and fractional edge covering number, respectively; notation $\tilde{O}(.)$ hides a factor polylogarithmic to IN. We further show how our result justifies the $O(\mathrm{IN}^{\rho^*})$ running time of existing worst-case optimal join algorithms (for full result reporting) even when $\mathrm{OUT} \ll \mathrm{IN}^{\rho^*}$. Specifically, unless the *combinatorial k-clique hypothesis* is false, no combinatorial algorithms (i.e., algorithms not relying on fast matrix multiplication) can compute the join result in $O(\mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p. even if $\mathrm{OUT} \le \mathrm{IN}^{\epsilon}$, regardless of how small the constant $\epsilon > 0$ is.

CCS CONCEPTS

• **Theory of computation** → **Database query processing and optimization (theory)**;

KEYWORDS

Join Algorithms, Sampling, Conjunctive Queries, Lower Bounds

ACM Reference Format:
Shiyuan Deng, Shangqi Lu, and Yufei Tao. 2023. On Join Sampling and the Hardness of Combinatorial Output-Sensitive Join Algorithms. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '23), June 18–23, 2023, Seattle, WA, USA.* ACM, 13 pages. https://doi.org/10.1145/3584372.3588666

1 INTRODUCTION

Joins, which combine the tuples across multiple tables based on equality conditions[1], are a fundamental operation in relational algebra and a main performance bottleneck in database systems. Research on joins has been a core field of database theory. Recent years have witnessed significant advances in this field. Particularly, in the realistic scenario where a join involves a constant number of attributes, the community has discovered join algorithms [6, 36, 42, 44–47, 54] that can achieve the asymptotically optimal performance (sometimes up to a polylogarithmic factor) in the worst case.

Unfortunately, joins remain expensive even in the presence of worst-case optimal algorithms. The culprit is the output size: a join can produce $\Theta(\mathrm{IN}^{\rho^*})$ tuples [8] where IN denotes the total number of tuples in the input relations, and ρ^* represents the join's *fractional edge covering number*. We will defer the formal definition of ρ^* to Section 2, whereas, for our introductory discussion, it suffices to understand ρ^* as a constant at least 1 that is decided by the participating relations' schemas. For instance, ρ^* equals 2 for a join between a relation with attributes A, B and another relation with attributes B, C, suggesting that the join may output up to $\Theta(\mathrm{IN}^2)$ tuples. Even just listing the result necessitates $\Omega(\mathrm{IN}^2)$ time in the worst case (regardless of which join algorithm is deployed). The phenomenon has a severe impact on database systems because the value IN is gigantic in today's big-data era, and the value ρ^* can be considerably higher for other joins.

Fortunately, many downstream tasks of the join operation do not require a complete result, but can benefit significantly from random samples. A classical example is "approximate aggregation" (which arises in OLAP frequently), whose objective is to estimate the result tuples' total value on a selected attribute (e.g., `sales`). It is well-known that an accurate estimate can be derived from a small number of tuples drawn uniformly at random from the join result. Another, more modern, example is "fair representative reporting" [50], whose objective is to return a few tuples diverse enough to adequately illustrate the join output's overall distribution. Random samples again serve the purpose very well.

Due to its profound importance, "join sampling" — the problem of extracting a join result tuple uniformly at random — has attracted considerable attention since its introduction by Chaudhuri et al. [18] in 1999 (a survey will appear in Section 2). The state of the art is an algorithm due to Chen and Yi [21] which, after an initial $\tilde{O}(\mathrm{IN})$-time preprocessing, draws a sample tuple in time

$$\tilde{O}(\mathrm{IN}^{\rho^*+1}/\max\{1, \mathrm{OUT}\}) \tag{1}$$

with high probability (or w.h.p. for short) — namely, with a probability at least $1 - 1/\mathrm{IN}^c$ for an arbitrarily large constant c — where OUT is the join's output size (i.e., how many tuples in the join result), and the notation $\tilde{O}(.)$ hides a factor polylogarithmic to IN. Considering that computing the join result takes $\Omega(\mathrm{IN}^{\rho^*})$ time in the worst case, Chen and Yi's method produces a sample in shorter time when $\mathrm{OUT} \gg \mathrm{IN}$. They called it an "intriguing open problem" to design an index structure that, after $\tilde{O}(\mathrm{IN})$-time preprocessing, can be used

[1]The joins discussed in this paper are more precisely known as *equi-joins*, as opposed to *theta-joins* that use non-equality conditions.

to extract a sample in

$$\tilde{O}(\mathrm{IN}^{\rho^*}/\max\{1, \mathrm{OUT}\}) \tag{2}$$

time w.h.p., i.e., reducing the bound in (1) by a factor of $O(\mathrm{IN})$. In [21], Chen and Yi managed to achieve the purpose for a special class of joins, but not for arbitrary joins.

Overview of Our Results and Techniques. Given an arbitrary join involving a constant number of attributes, we present an index structure fulfilling all the requirements below:

- It occupies $\tilde{O}(\mathrm{IN})$ space and can be built in $\tilde{O}(\mathrm{IN})$ time.
- Its sampling time is bounded by (2) w.h.p. (without the value of OUT given). The random samples obtained by repeatedly applying our sampling algorithm are mutually independent.
- It is fully dynamic: inserting and deleting a tuple in any underlying relation takes $\tilde{O}(1)$ time.

Our structure is different from all the previous solutions to join sampling. In particular, we take a perspective that can be viewed as the opposite of how Chen and Yi [21] approached the problem. They construct a random tuple by growing one attribute at a time, in a manner similar to *Generic Join* [47], which is a worst-case optimal join algorithm. Crucial to their strategy is the following subproblem: assuming that we have fixed the values $x_1, ..., x_i$ on attributes $X_1, X_2, ..., X_i$ for some integer i, generate a (random) value x_{i+1} for the next attribute X_{i+1} according to a carefully crafted distribution[2]. The subproblem, however, is a major technical barrier, to which Chen and Yi's solution incurs $\tilde{O}(\mathrm{IN})$ time, which is the main reason behind the gap between their complexity (1) and the desired complexity in (2).

Rather than attribute *values* (the finest granularity), our approach works on the attribute *space* (the coarsest granularity), which is the cartesian product of the domains of all the attributes participating in the join. We prove, in what we call *the AGM split theorem*, that it is always possible to divide the space into a constant number of subspaces such that

- the maximum number of join result tuples in each subspace — characterized by the so-called "AGM bound" [8] (to be introduced in Section 2) — is at most half of the AGM bound of the original space, and
- the sum of the AGM bounds of the subspaces does not exceed the AGM bound of the original space

unless the AGM bound of the original space is already small enough to allow the join to be evaluated in $\tilde{O}(1)$ time. Recursively performing the partitioning leads to a sampling algorithm faster (and simpler) than the state of the art [21]. As a high-level idea, space partitioning has been leveraged to design join algorithms before [6, 27, 36, 42, 44]. However, the idea's deployment in those scenarios was for purposes drastically different from ours. The relevant algorithms, as well as their analysis, in the aforementioned works also differ from ours considerably. Our technical development is new and, we believe, clean enough for teaching at a graduate level.

We also study how much further improvement over (2) is still possible. In fact, perhaps the most challenging step is to pose the right question in this regard. At first glance, (2) appears obviously optimal because OUT, as mentioned, can reach $\Omega(\mathrm{IN}^{\rho^*})$, in which case (2) becomes $\tilde{O}(1)$, clearly the best achievable (we are not concerned with polylogarithmic factors in this work). Although correct, this argument tells us nothing in the realistic scenario where $\mathrm{OUT} \ll \mathrm{IN}^{\rho^*}$. We instead ask a more meaningful question:

> The join sampling question. Is there a constant ϵ satisfying $0 < \epsilon < 1/2$, under which we can find a structure that, after an initial $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$-time preprocessing, extracts a uniformly random tuple from the join result in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon}/\mathrm{OUT})$ time w.h.p. when $1 \le \mathrm{OUT} \le \mathrm{IN}^{\epsilon}$?

Note that the structure does not need to guarantee anything for $\mathrm{OUT} = 0$ or $\mathrm{OUT} > \mathrm{IN}^{\epsilon}$ and has sampling time polynomially better than (2) for $\mathrm{OUT} \in [1, \mathrm{IN}^{\epsilon}]$.

We study the question within the class of *combinatorial structures*. These are structures whose preprocessing and sampling algorithms are *combinatorial*, namely, the algorithms do not rely on fast matrix multiplication (à la Strassen's). We prove that the answer to the question is "no" subject to the hypothesis below.

> Combinatorial k-clique hypothesis. There does not exist any fixed constant $\epsilon > 0$ under which a combinatorial algorithm can achieve the following for every constant $k \ge 3$: it can detect with probability at least 1/3 in $O(n^{k-\epsilon})$ time whether an undirected graph of n vertices has a k-clique.

The background behind the above hypothesis deserves some explanation. For k being a (constant) multiple of 3, k-clique existence in an n-vertex graph can be detected in $O(n^{\omega k/3})$ time where ω is the matrix multiplication exponent [43] (see [28] for a more complex bound for arbitrary $k = O(1)$). However, if only combinatorial algorithms are permitted, even beating the naive $O(n^k)$-time approach is difficult: the fastest combinatorial algorithm [53] takes $O(n^k/\log^{k-1} n)$ time for constant $k \ge 3$. It is widely conjectured that no combinatorial algorithms can detect k-cliques in $O(n^{k-\epsilon})$ time for every constant $k \ge 3$; such a hypothesis has been applied to argue for computational hardness on a great variety of problems [1, 2, 10–13, 16, 17, 34, 35, 40, 41]. Our hypothesis (which requires success probability 1/3) has been explicitly stated in [10, 34].

In fact, if answering the join sampling question was the sole purpose, our argument (in Section 5) could be shortened. However, the argument discloses an inherent connection between join sampling and *output-sensitive* join computation. For a precise explanation, let us introduce the notion of "ϵ-output sensitivity":

> Let ϵ be a constant with $0 < \epsilon < 1/2$. An algorithm is *ϵ-output sensitive* if it can output all the tuples of the join result in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p. whenever $\mathrm{OUT} \le \mathrm{IN}^{\epsilon}$.

Note that the above is different from demanding an algorithm to run in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon} + \mathrm{OUT})$ time for *all* OUT values because the notion does not require the algorithm to guarantee anything when $\mathrm{OUT} > \mathrm{IN}^{\epsilon}$. The significance of ϵ-output sensitive algorithms is that they *convincingly* enhance worst-case join algorithms. Although there has been research [6, 36, 44, 49] on how to compute joins in time sensitive to OUT, none of the known algorithms is ϵ-output sensitive no matter how ϵ is chosen: those algorithms' running time can still degenerate into $O(\mathrm{IN}^{\rho^*})$ even when $\mathrm{OUT} = 0$.

[2] More specifically, the probability of generating x_{i+1} is decided based on how many join result tuples would satisfy $X_j = x_j$ for all $j \in [1, i+1]$.

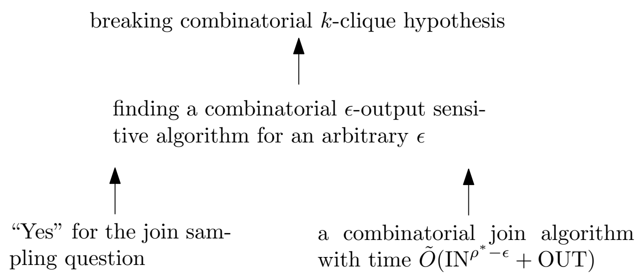
breaking combinatorial k-clique hypothesis
finding a combinatorial ϵ-output sensitive algorithm for an arbitrary ϵ
"Yes" for the join sampling question
a combinatorial join algorithm with time $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon} + \mathrm{OUT})$

Figure 1: Reduction relationships (arrow means "implies")

We show that any combinatorial ϵ-output sensitive algorithm can be combined with our join sampling solution to *detect* whether $\mathcal{Join}(Q)$ is empty in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p. regardless of the value OUT. However, such a detection algorithm (which is combinatorial) can determine — for all constants $k \geq 3$ — in $\tilde{O}(n^{k-2\epsilon})$ time w.h.p. k-clique existence in an n-vertex graph, thus breaking the combinatorial k-clique hypothesis. We further show that any combinatorial structure answering "yes" to the join sampling question with a constant ϵ satisfying $0 < \epsilon < 1/2$ implies a combinatorial ϵ-output sensitive join algorithm. The above discussion yields the relationships in Figure 1, where an arrow from problem A to problem B means "A implies B" (i.e., B can be reduced to A).

The findings in Figure 1 have another notable implication. The known worst-case optimal join algorithms (for result reporting) are all combinatorial. Previously, justification on the optimality of an $O(\mathrm{IN}^{\rho^*})$-time join algorithm relied heavily on $\mathrm{OUT} = \Omega(\mathrm{IN}^{\rho^*})$. Our result suggests that term IN^{ρ^*} is necessary (up to a sub-polynomial factor) even if $\mathrm{OUT} \leq \mathrm{IN}^{\epsilon}$ for any constant $\epsilon > 0$, subject to the combinatorial k-clique hypothesis. For example, if a combinatorial algorithm could compute $\mathcal{Join}(Q)$ in $O(\mathrm{IN}^{\rho^*-0.001})$ time when $\mathrm{OUT} \leq \mathrm{IN}^{0.001}$, it would make a 0.001-output sensitive algorithm and thus break the hypothesis.

Our sampling structure also yields new algorithms on several related problems (e.g., join size estimation, subgraph sampling, randomly permuting the join result with a small delay, join union sampling, etc.). We will elaborate on the details in later sections.

2 PRELIMINARIES

Section 2.1 will formally define the join sampling problem. Then, Section 2.2 will introduce the AGM bound, which plays a crucial role in our techniques. Finally, Section 2.3 will present a survey of the existing join algorithms.

2.1 The Problem of Join Sampling

Denote by **att** a finite set whose elements are called *attributes*. Given a set $U \subseteq \mathbf{att}$, we define a *tuple* over U as a function $\boldsymbol{u} : U \to \mathbb{N}$, where $\mathbb{N}$ is the set of integers. If V is a subset of U, we define the *projection* of u on V — denoted as $\boldsymbol{u}[V]$ — to be the tuple $\boldsymbol{v}$ over V satisfying $\boldsymbol{u}(X) = \boldsymbol{v}(X)$ for every attribute $X \in V$. A *relation* is a set R of tuples over an identical set U of attributes. We refer to U as the *schema* of R and represent this fact with $var(R) := U$.

We define a *join* as a set Q of relations with distinct schemas. Let $var(Q) := \bigcup_{R \in Q} var(R)$. The result of the join, denoted as $\mathcal{Join}(Q)$, is a relation with schema $var(Q)$ given by

$$\mathcal{Join}(Q) := \{\text{tuple } \boldsymbol{u} \text{ over } var(Q) \mid \forall R \in Q : \boldsymbol{u}[var(R)] \in R\}.$$

A *join sample* of Q is a uniformly random tuple from $\mathcal{Join}(Q)$.

The main problem we aim to solve is to design an index structure for Q to support the two operations below:

- Extract a join sample of Q. It is required that repeated extraction should produce mutually independent samples (i.e., every sample must be uniformly random even conditioned on all the samples already taken). In the case where $\mathcal{Join}(Q)$ is empty, this operation should declare so with a special output.
- Insert or delete a tuple in any relation of Q, collectively called an "update".

Focusing on data complexities, we assume that Q has a constant number of relations, and the schema of every relation in Q has a constant number of attributes. We introduce

$$\mathrm{IN} := \sum_{R \in Q} |R|, \text{ and } \mathrm{OUT} := |\mathcal{Join}(Q)|$$

and refer to them as the *input* and *output size* of Q, respectively.

2.2 The AGM Bound

A fundamental question in join processing is how many tuples there can be in the join result. The AGM bound, derived by Atserias et al. [8], answers the question from the graph theory perspective.

To start with, given a join Q, we define a hypergraph $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ where $\mathcal{X} := var(Q)$ and $\mathcal{E} := \{var(R) \mid R \in Q\}$. In other words, each vertex in $\mathcal{G}$ corresponds to an attribute involved in the join, and each (hyper) edge in $\mathcal{G}$ corresponds to the schema of an input relation of Q. We will refer to $\mathcal{G}$ as the *schema graph* of Q. For each edge $e \in \mathcal{E}$, let R_e be the (only) relation in Q whose schema is e.

A *fractional edge covering* of $\mathcal{G}$ is a function $W : \mathcal{E} \to \mathbb{R}$, where $\mathbb{R}$ is the set of real values such that

- for each $e \in \mathcal{E}$, $W(e) \geq 0$;
- for each $X \in \mathcal{X}$, $\sum_{e \in \mathcal{E}: X \in e} W(e) \geq 1$, i.e., the total weight of the edges covering vertex (a.k.a., attribute) X is at least 1.

The AGM bound shows that any fractional edge covering of $\mathcal{G}$ yields an upper bound on the join result size OUT, as stated below:

Lemma 1 (AGM bound [8]). *Given any fractional edge covering W of $\mathcal{G}$, we have $|\mathcal{Join}(Q)| \leq \mathrm{AGM}_W(Q)$ where*

$$\mathrm{AGM}_W(Q) := \prod_{e \in \mathcal{E}} |R_e|^{W(e)}. \tag{3}$$

We will refer to $\mathrm{AGM}_W(Q)$ as the "AGM bound of Q" when W is understood from the context.

Equation (3) expresses an upper bound of OUT using the concrete sizes of the relations in Q. We often want to describe the upper bound more directly using the input size IN. For this purpose, we can apply the trivial fact $|R_e| \leq \mathrm{IN}$ (for all $e \in \mathcal{E}$) to simplify Lemma 1 into $\mathrm{OUT} \leq \mathrm{IN}^{\sum_{e \in \mathcal{E}} W(e)}$. Note that the exponent $\sum_{e \in \mathcal{E}} W(e)$ is exactly the total weight of all the edges in $\mathcal{E}$ assigned by W. This motivates the concept of *fractional edge covering number* of $\mathcal{G}$ — denoted as ρ^* — which is the smallest $\sum_{e \in \mathcal{E}} W(e)$ among all the fractional edge coverings W. Hence, it always holds that $\mathrm{OUT} \leq \mathrm{IN}^{\rho^*}$.

The AGM bound is tight: given any integer $\mathrm{IN} \geq 1$ and hypergraph $\mathcal{G}$, we can always find a join Q, which has input size IN and schema graph $\mathcal{G}$, such that the output size of Q reaches $\Omega(\mathrm{IN}^{\rho^*})$ [8].

2.3 Algorithms on Join Processing

Full Join Computation. The tightness of the AGM bound establishes $\Omega(\text{IN}^{\rho^*})$ as a lower bound for the running time of any join algorithm because this amount of time is needed even just to output $\mathcal{J}oin(Q)$ in the worst case. Ngo et al. [45] were the first to discover a worst-case optimal algorithm that can evaluate any join in $O(\text{IN}^{\rho^*})$ time. Since their invention, quite a number of algorithms with time complexity $\tilde{O}(\text{IN}^{\rho^*})$ have been subsequently developed [6, 36, 42, 44, 46, 47, 54].

In practice, a join's output size OUT may be far less than IN^{ρ^*}. This motivates the study of *output-sensitive algorithms* whose running time is sensitive to both IN and OUT. As a notable success, Yannakakis [56] presented an algorithm to process any "acyclic join" in $\tilde{O}(\text{IN} + \text{OUT})$ time. However, acyclic joins are a special type of joins on which stringent restrictions are imposed. Evaluating a join in $\tilde{O}(\text{IN}^{\rho^*-\epsilon} + \text{OUT})$ time in the absence of those restrictions, even for an arbitrarily small constant $\epsilon > 0$, is still open.

Several non-trivial attempts have been made to tackle the open challenge. When executed on a general join, all the known output-sensitive algorithms [6, 36, 44, 49] have running time of the form

$$\tilde{O}(\mathtt{Cer}^{\mathtt{width}} + \text{OUT})$$

where $\mathtt{width}$ quantifies how much the schema graph $\mathcal{G}$ deviates from a tree (it can be defined using, e.g., tree width [24], query width [19], fractional hypertree width [31], etc.), and $\mathtt{Cer}$ is a value, called the "certificate size", measuring how difficult the input relations are for join processing in the instance-oriented sense (e.g., IN is a common value for $\mathtt{Cer}$; see [36, 44] for other $\mathtt{Cer}$ definitions based on specialized "certificates"). Unfortunately, for all the algorithms in [6, 36, 44, 49], the term $\mathtt{Cer}^{\mathtt{width}}$ always ends up being $\Omega(\text{IN}^{\rho^*})$ at some "unfriendly" joins. Therefore, as far as general joins are concerned, no algorithms with $\tilde{O}(\text{IN}^{\rho^*-\epsilon} + \text{OUT})$ time have been found, no matter how small the constant ϵ is.

All the above algorithms are combinatorial. We are not aware of any non-combinatorial approaches for computing $\mathcal{J}oin(Q)$. There exist, however, algorithms [7, 25, 26] that use matrix multiplication to evaluate conjunctive queries with *projections*. A join in our definition can be understood as a conjunctive query without projections, in which case the algorithms of [7, 25, 26] do not promise faster running time than the combinatorial methods discussed earlier.

Join Sampling. In 1999, Chaudhuri et al. [18] initialized the study on join sampling. They focused on joins with two relations, i.e., $Q = \{R_1, R_2\}$, and described a structure of $O(\text{IN})$ space that allows extracting a sample from $\mathcal{J}oin(Q)$ in constant time. They also proved a lower bound that, if no preprocessing is allowed, taking a sample demands $\Omega(\text{IN})$ time. Acharya et al. [3] considered joins with more than two relations, but their formal results apply only when the relations of Q obey the so-called "star schema", namely, there is a "center relation" that has a foreign key to every other relation. Sampling is trivial on star-schema joins because it boils down to drawing a random tuple from a single relation (i.e., the center one).

No theoretical progress had been documented on join sampling in the next 18 years following the work of [3, 18] (in the meantime, the problem had received a huge amount of attention from the system community, as discussed later). Theory advancement resumed in 2018. For any acyclic join Q, Zhao et al. [58] presented an $O(\text{IN})$-space structure that permits drawing a sample from $\mathcal{J}oin(Q)$ in constant time. By combining their structure with hypertree decompositions[3], one can obtain a structure for an arbitrary join Q that has $O(\text{IN})$ space and $\tilde{O}(\text{IN}^{\text{fhtw}})$ sampling time, where fhtw is the fractional hypertree width of Q; in the worst case, however, $\text{fhtw} = \rho^*$, causing the sampling time to degenerate into $\tilde{O}(\text{IN}^{\rho^*})$. In 2020, Chen and Yi [21] identified a class of joins under the name *sequenceable joins*, for which they obtained a (static) structure of $O(\text{IN})$ space that can sample from the join result in $\tilde{O}(\text{IN}^{\rho^*}/\max\{1, \text{OUT}\})$ time w.h.p.[4]. For general (non-sequenceable) joins, their structure still works, but the sampling time deteriorates by a factor of $O(\text{IN})$ to $\tilde{O}(\text{IN}^{\rho^*+1}/\max\{1, \text{OUT}\})$.

Closely relevant to join sampling is the *direct access* (DA) problem on join computation. In that problem, there is a pre-agreed ordering on the tuples of $\mathcal{J}oin(Q)$ such that, given an integer $k \in [1, \text{OUT}]$, a DA query returns the k-th tuple in $\mathcal{J}oin(Q)$. If a structure can answer a DA query in T_{DA} time, we can use the query to draw a sample from $\mathcal{J}oin(Q)$ in $O(T_{\text{DA}})$ time with a random value $k \in [1, \text{OUT}]$, where the value OUT is available from preprocessing. When Q is "free-connex", there is a structure of $\tilde{O}(\text{IN})$ space answering a DA query in $\tilde{O}(1)$ time [14, 15], which means that the structure also guarantees $\tilde{O}(1)$ sample time. However, because a free-connex Q is necessarily acyclic, the sampling result is subsumed by that of [58].

Finally, we note that join sampling has been studied extensively in system research (see [5, 20, 33, 38, 39, 48, 51, 55, 57, 59] and the references therein), which has produced numerous empirically efficient solutions. In the worst case, however, those solutions all require $\Omega(\text{IN}^{\rho^*})$ time to draw one sample, regardless of the value of OUT. All the above join sampling solutions, theoretical or empirical, are combinatorial.

3 THE AGM SPLIT THEOREM

This section will establish *the AGM split theorem*, which serves as the technical core of our sampling algorithm. The theorem provides a simple and intuitive way to split the "attribute space", with the guarantee that the upper bound on the join result size given by the AGM bound gets (at least) halved in each subspace after the split.

Recall that $var(Q)$ is the set of attributes involved in the join. Set $d := |var(Q)|$. Let us impose an arbitrary ordering on the attributes of $var(Q)$, which can then be denoted as $X_1, X_2, ..., X_d$. This way, every tuple $\boldsymbol{u}$ in the join result $\mathcal{J}oin(Q)$ can be interpreted as a point in $\mathbb{N}^d$, where $\boldsymbol{u}(X_i)$ is the point's i-th coordinate for each $i \in [1, d]$. The *attribute space* is now formally defined to be $\mathbb{N}^d$.

Next, we introduce *box-induced sub-join*, a notion imperative in our subsequent discussion. Let B be a *box* in the attribute space, namely, B has the form $[x_1, y_1] \times [x_2, y_2] \times ... \times [x_d, y_d]$. For each $i \in [1, d]$, we use $B(X_i)$ to denote $[x_i, y_i]$, i.e., the projection of B on the i-th attribute. On every relation $R \in Q$, the box B induces a "sub-relation" $R(B)$, which includes all the tuples of R "falling" into B. Care must be taken here because R may not include all the attributes in $var(Q)$. We say that a tuple $\boldsymbol{u} \in R$ *falls* in B if $B(X)$ covers $\boldsymbol{u}(X)$

[3]See Appendix A of [36] for an introduction to such decompositions.

[4]In [21], Chen and Yi claimed the sampling time as $O(\text{IN}^{\rho^*}/\max\{1, \text{OUT}\})$ in expectation, but under the assumption that an expression of the form x^y (for a fractional y) can be evaluated in constant time. Removing the assumption incurs an $O(\log \text{IN})$ multiplicative factor. Moreover, it is standard to make their time complexity hold w.h.p. by paying yet another $O(\log \text{IN})$ multiplicative factor.

for every attribute $X \in var(R)$. $R(B)$ can then be formalized as

$$R(B) := \{\boldsymbol{u} \in R \mid \boldsymbol{u} \text{ falls in } B\}. \tag{4}$$

By putting together the $R(B)$ of all $R \in Q$, we have the "sub-join induced by B", formally defined as

$$Q(B) := \{R(B) \mid R \in Q\}. \tag{5}$$

Given a sub-join $Q(B)$ and an attribute $X \in var(Q)$, we will need to be concerned with the set of X-values appearing in at least one relation of $Q(B)$. This can be formalized as

$$\mathbf{actdom}(X, B) := \{x \in \mathbb{N} \mid \exists R(B) \in Q(B), \boldsymbol{u} \in R(B) : X \in var(R), \boldsymbol{u}(X) = x\} \tag{6}$$

which will be referred to as the "active X-domain induced by B".

We will reason about the AGM bounds on box-induced sub-joins under a *fixed* fractional edge covering W. For that purpose, we "overload" the function AGM_W, defined in (3), in a manner that will prove handy in our analysis. Let $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ be the schema graph of Q (defined in Section 2.2) and W be an arbitrary fractional edge covering of $\mathcal{G}$. Given a box B, we define

$$\text{AGM}_W(B) := \text{AGM}_W(Q(B)) = \prod_{e \in \mathcal{E}} |R_e(B)|^{W(e)}. \tag{7}$$

With a slight abuse of notation, the "overloading" allows AGM_W to take a box as the parameter directly. No ambiguity can arise because, as we have seen, every box B defines a sub-join $Q(B)$. By Lemma 1, $\text{AGM}_W(B)$ is an upper bound of the result size of $Q(B)$. It is worth mentioning that $\text{AGM}_W(\mathbb{N}^d)$ — the parameter is set to the attribute space (the largest box) — equals $\text{AGM}(Q)$, the AGM bound of the original join Q. We will refer to $\text{AGM}_W(B)$ as the "AGM bound of B", when W is clear from the context.

Before unveiling the AGM split theorem, we need to clarify some "oracles" that provide efficient implementations of certain primitive operations. Specifically, two oracles will be useful:

- *Count oracle*: Given a relation $R \in Q$ and a box B, the oracle returns the number of tuples of $R(B)$ in $\tilde{O}(1)$ time, where $R(B)$ is defined in (4).
- *Median oracle*: Given an attribute $X \in var(Q)$ and a box B, the oracle returns the median value[5] of $\mathbf{actdom}(X, B)$ in $\tilde{O}(1)$ time, where $\mathbf{actdom}(X, B)$ is defined in (6).

Both oracles can be implemented with rudimentary data structures, as will be discussed in later sections. Here, we will proceed by assuming that they have been made available at our disposal.

Theorem 2 (AGM Split Theorem). *Fix an arbitrary fractional edge covering W of $\mathcal{G}$, and assume the availability of count and median oracles. Given any box B with $\text{AGM}_W(B) \geq 2$, we can find in $\tilde{O}(1)$ time a set C of at most $2d + 1$ (where $d := |var(Q)|$) boxes such that*

(1) *the boxes in C are disjoint and have B as their union;*
(2) *for each box $B' \in C$, $\text{AGM}_W(B') \leq \frac{1}{2}\text{AGM}_W(B)$;*
(3) $\sum_{B' \in C} \text{AGM}_W(B') \leq \text{AGM}_W(B)$.

The rest of the section serves as a proof of the theorem, which consists of two main parts. First, we will present a technical lemma to reveal an underlying mathematical relationship in AGM bounds that concerns splitting a box along one of the attributes. Then, we will utilize the lemma to design an efficient algorithm to produce the desired set C of boxes.

[5]If a set S has n values, the median of S is the $\lceil n/2 \rceil$-th smallest value in S.

Let us start with the technical lemma. To facilitate explanation, it will be helpful to introduce a function `replace`(B, i, I), where B is a box in the attribute space, i is an integer between 1 and d, and I is an interval of integers. The function yields the box

$$\texttt{replace}(B, i, I) := B(X_1) \times ... \times B(X_{i-1}) \times I \times B(X_{i+1}) \times ... \times B(X_d)$$

that is, replacing the projection of B on attribute X_i with I, while retaining the projections on the other attributes.

To state our lemma, let us fix a fractional edge covering W of $\mathcal{G}$ and a box B, as we do in Theorem 2. In addition, fix an arbitrary attribute X_i, for some $i \in [1, d]$. Suppose that we partition the (integer) interval $B(X_i)$ arbitrarily into s disjoint integer intervals $I_1, I_2, ..., I_s$ where s can be any value at least 2 (s does not need to be a constant). For each $j \in [1, s]$, the interval I_j defines a box

$$B_j \quad := \quad \texttt{replace}(B, i, I_j). \tag{8}$$

It is clear that $B_1, B_2, ..., B_s$ are mutually disjoint and have B as their union. Our technical lemma can now be presented as:

Lemma 3. $\sum_{j=1}^{s} \text{AGM}_W(B_j) \leq \text{AGM}_W(B)$.

The above lemma is, in fact, Lemma 6 of [27] in disguise. Unfortunately, Lemma 6 of [27] was presented in a sophisticated context, because of which the reader would find it difficult to recognize the two lemmas' resemblance. We present a standalone proof in Appendix A for the sake of self-containment.

Equipped with the lemma, next we explain how to obtain the set C of boxes in Theorem 2 using $\tilde{O}(1)$ time. The following simple proposition will be useful throughout the paper.

Proposition 1. *Given any box B, we can compute $\text{AGM}_W(B)$ in $\tilde{O}(1)$ time.*

Proof. We first use the count oracle to obtain $|R_e(B)|$ in $\tilde{O}(1)$ time for each edge $e \in \mathcal{E}$ (remember that $\mathcal{E}$ is the set of edges in the schema graph $\mathcal{G}$), and then feed these values into (7) to compute $\text{AGM}_W(B)$.[6] The proposition holds because $\mathcal{E}$ has a constant number of edges. □

Figure 2 presents our algorithm for splitting a box B into a set C of smaller boxes meeting the requirements of Theorem 2. The algorithm, `split`(i, B), admits two parameters: an integer $i \in [1, d]$ and a box $B = [x_1, y_1] \times ... \times [x_d, y_d]$. The box should satisfy the constraint that its projections on the first $i - 1$ attributes must be *singleton* intervals (a singleton interval $[x, y]$ contains only one

[6]Computing a quantity like $|R_e(B)|^{W(e)}$ requires a power operator — one that evaluates an expression like x^y for a fractional y — which is commonly assumed to take constant time in the join literature; e.g., see previous work [21, 27]. Strictly speaking the standard RAM model does not provide such an operator. One simple way to circumvent the issue is to use the observation that $W(e)$ has only $|\mathcal{E}| = O(1)$ different choices, i.e., one for each $e \in |\mathcal{E}|$, and $|R_e(B)|$ must be an integer from 0 to IN. Therefore, we can prepare, for each $e \in \mathcal{E}$, the values $1^{W(e)}, 2^{W(e)}, ..., \text{IN}^{W(e)}$ in preprocessing and store them all in $O(\text{IN})$ extra space. As a less straightforward, but more powerful, remedy, we can calculate $|R_e(B)|^{W(e)}$ up to an additive error of $1/\text{IN}^c$ for some sufficiently large constant c, which is easy to achieve in $\tilde{O}(1)$ time. Such a precision level is sufficient for our algorithms in this paper to work. Henceforth, we will no longer dwell on the issue but will simply assume that the power operator takes $\tilde{O}(1)$ time. The reader may also refer to Section 6 of [32] for a relevant discussion.

```
algorithm split($i, B$)
/* assume $B = [x_1, y_1] \times ... \times [x_d, y_d]$; it is required
that $x_1 = y_1$, $x_2 = y_2$, ..., and $x_{i-1} = y_{i-1}$ */
1. $C \leftarrow \emptyset$
2. $z \leftarrow$ the largest value in $[x_i, y_i]$ s.t. $\text{AGM}_W(B_{left}) \le \frac{1}{2}\text{AGM}_W(B)$
   where $B_{left} \leftarrow$ replace($B, i, [x_i, z-1]$)
3. if $z - 1 \ge x_i$ then $C \leftarrow C \cup \{B_{left}\}$
4. $B_{mid} \leftarrow$ replace($B, i, [z, z]$)
5. if $i = d$ then add $B_{mid}$ to $C$
6.     else $C \leftarrow C \cup$ split($i + 1, B_{mid}$)
7. if $z + 1 \le y_i$ then $C \leftarrow C \cup \{B_{right}\}$ where
       $B_{right} \leftarrow$ replace($B, i, [z+1, y_i]$)
8. return $C$
```

Figure 2: The split algorithm for Theorem 2

value, i.e., $x = y$). Note that the constraint does not prevent us from supplying an arbitrary box as B, as long as we set $i = 1$ in that case. The integer i stands for the "split attribute". Specifically, given any value $z \in [x_i, y_i]$, we can split B into (at most) three boxes:

$$\begin{aligned} B_{left} &:= \texttt{replace}(B, i, [x_i, z-1]) \\ B_{mid} &:= \texttt{replace}(B, i, [z, z]) \\ B_{right} &:= \texttt{replace}(B, i, [z+1, y_i]). \end{aligned}$$

As a special case, B_{left} or B_{right} does not exist if $z = x_i$ or y_i, respectively. We want to find the largest z satisfying the condition $\text{AGM}_W(B_{left}) \le \frac{1}{2}\text{AGM}_W(B)$ (Line 2); z definitely exists because the condition is fulfilled by $z = x_i$ (in which case B_{left} is empty and the condition is vacuously met). The boxes B_{left} and B_{right}, if they exist, are added directly to C (Line 3 and 7). Regarding B_{mid}, notice that it now has singleton projections on the first i attributes. If $i = d$, then B_{mid} has degenerated into a point in the attribute space and is also added to C (Line 5). Otherwise, we recursively invoke split$(i + 1, B_{mid})$ to split B_{mid} into a set C' of boxes and union C' into C (Line 6).

Next, we show that the set C produced by split$(1, B)$ has the three properties in Theorem 2. Property 1 is obvious and omitted. Let us turn to Property 2. For every B_{left} created by a recursive call split(i, B'), where B' is some box inside B generated in the recursion, we have $\text{AGM}_W(B_{left}) \le \frac{1}{2}\text{AGM}_W(B') \le \frac{1}{2}\text{AGM}_W(B)$. Hence, every B_{left} added to C must satisfy Property 2.

Let us switch attention to the B_{right} in an (arbitrary) recursive call split(i, B'). By definition of the value z at Line 2, it must hold that $\text{AGM}_W(B_{left} \cup B_{mid}) \ge \frac{1}{2}\text{AGM}_W(B')$. Lemma 3, on the other hand, tells us that $\text{AGM}_W(B_{left} \cup B_{mid}) + \text{AGM}_W(B_{right}) \le \text{AGM}_W(B')$ (apply the lemma with $s = 2$, $B_1 = B_{left} \cup B_{mid}$, and $B_2 = B_{right}$). This gives $\text{AGM}_W(B_{right}) \le \frac{1}{2}\text{AGM}_W(B') \le \frac{1}{2}\text{AGM}_W(B)$. Hence, every B_{right} added to C also satisfies Property 2.

It remains to discuss B_{mid}. In all the recursive calls to split, the only B_{mid} added to C is the final one that has degenerated into a point (Line 5). For such a B_{mid}, we have $\text{AGM}_W(B_{mid}) \le 1$, which is at most $\frac{1}{2}\text{AGM}_W(B)$ because $\text{AGM}_W(B) \ge 2$.

Lastly, Property 3 is a corollary of Lemma 3. At any recursive call split(i, B'), the lemma indicates $\text{AGM}_W(B_{left}) + \text{AGM}_W(B_{mid}) + \text{AGM}_W(B_{right}) \le \text{AGM}_W(B')$. Now, Property 3 follows from a simple inductive argument on i (for breaking $\text{AGM}_W(B_{mid})$ into the sum of the AGM bounds of smaller boxes).

The set C returned by split$(1, B)$ has a size no more than $2d+1$ because we add at most two boxes to C for each $i \in [1, d-1]$ and at most three for $i = d$. Each call to split, excluding the recursive invocation at Line 6, runs in $\tilde{O}(1)$ time. In particular, Line 2 can be implemented in $\tilde{O}(1)$ time due to Proposition 1 and the fact that z can be found with binary search in the active X_i-domain induced by B, which necessitates only $O(\log \text{IN})$ calls to the median oracle. As the overall recursion depth is $d = O(1)$, the total running time of split$(1, B)$ is $\tilde{O}(1)$. This completes the proof of Theorem 2.

Remark. In Proposition 8 of [27], Deep and Koutris presented a splitting result in the so-called "lexicographical order". Under the lexicographical order, each tuple $\boldsymbol{u}$ in the attribute space is viewed as a string of d characters $\boldsymbol{u}(X_1)\boldsymbol{u}(X_2)...\boldsymbol{u}(X_d)$, and two tuples are compared by their string representations alphabetically. An interval $[\boldsymbol{u}_1, \boldsymbol{u}_2]$ under the order includes all the tuples $\boldsymbol{v}$ alphabetically between $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$. The goal in [27] is to divide $[\boldsymbol{u}_1, \boldsymbol{u}_2]$ into (i) two intervals whose "AGM bounds" (see [27] for what this means) are at most half of that of $[\boldsymbol{u}_1, \boldsymbol{u}_2]$ and (ii) a tuple. Their split algorithm, which differs from ours in Figure 2, also uses "boxes" somehow, but the "boxes" there are specially constrained, as opposed to being arbitrary boxes. Although related, the statements in our Theorem 2 and Proposition 8 of [27] present distinct findings, neither of which subsumes the other.

4 JOIN SAMPLING

We are ready to solve the join sampling problem. To that end, Section 4.1 first introduces "the join box-tree", a conceptual hierarchy that paves the foundation of the proposed sampling algorithm, which is presented in Section 4.2. Our discussion in Sections 4.1 and 4.2 will assume the count and median oracles (defined in Section 3), whose implementation will be explained in Section 4.3.

4.1 The Join Box-Tree

Our AGM split theorem shows how to split an arbitrary box in the attribute space. Next, we will repeatedly utilize the theorem to partition the space into sufficiently small boxes with "trivial AGM bounds". This will produce a tree $\mathcal{T}$ that we name the *join box-tree*.

Fix an arbitrary fractional edge covering W of the schema graph $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ of the input join Q. Define $\rho := \sum_{e \in \mathcal{E}} W(e)$, i.e., the total weight of the edges in $\mathcal{E}$ under W. The value ρ is a constant that does not depend on IN.

The join box-tree $\mathcal{T}$ is a tree dependent on W. An internal node in $\mathcal{T}$ has at most $2d + 1$ child nodes, where $d := |var(Q)|$. Every node, no matter leaf or internal, is associated with a box, and no two nodes are associated with the same box. For this reason, henceforth, if a node is associated with a box B, we will use B to denote that node as well. Given a node B, we refer to $\text{AGM}_W(B)$, defined in (7), as the node's AGM bound. Every internal node has an AGM bound at least 2, whereas every leaf node has an AGM bound less than 2.

Next, we formally define $\mathcal{T}$ in a top-down manner. The root of $\mathcal{T}$ is associated with the box $\mathbb{N}^d$, i.e., the entire attribute space. Consider, in general, a node associated with box B. If node B has an AGM bound less than 2, we make it a leaf of $\mathcal{T}$. Otherwise, B is an internal node with child nodes created in two steps.

(1) Apply Theorem 2 to split B into a set C of boxes.
(2) For each box $B' \in C$, create a child node, associated with box B', of node B. The number of child nodes of B is $|C|$.

We now proceed to explore the properties of the join box-tree, starting with:

PROPOSITION 2. *$\mathcal{T}$ has height $O(\log \text{IN})$.*

PROOF. Every time we descend from a parent node to a child, the AGM bound decreases by at least a factor of two (Property 2 of Theorem 2). The root B of $\mathcal{T}$ has an AGM bound that equals the AGM bound of Q (given by Lemma 1), which is no more than IN^{ρ}, where $\rho = \sum_{e \in \mathcal{E}} W(e)$ as mentioned earlier. As a node becomes a leaf as soon as its AGM bound drops below 2, we can descend only $O(\log \text{IN}^{\rho}) = O(\log \text{IN})$ levels. □

The following property focuses on the leaf nodes of $\mathcal{T}$.

PROPOSITION 3. *The boxes of all the leaves of $\mathcal{T}$ are disjoint and have the attribute space $\mathbb{N}^d$ as the union.*

PROOF. The root of $\mathcal{T}$ has the entire attribute space as its associated box. In general, the box of a parent node is partitioned by the boxes of the child nodes, and the boxes of the child nodes are always disjoint (Property 1 of Theorem 2). This proves the proposition. □

The lemma below echoes our statement in Section 1 that a box with a "small-enough" AGM bound corresponds to a join that can be evaluated in near-constant time.

LEMMA 4. *Consider an arbitrary leaf of $\mathcal{T}$. Let B be the box associated with the leaf. The result of the join $Q(B)$, which contains at most one tuple, can be computed in $\tilde{O}(1)$ time, assuming the count and median oracles.*

PROOF. First, compute $\text{AGM}_W(B)$ in $\tilde{O}(1)$ time (Proposition 1). If $\text{AGM}_W(B) = 0$, we declare the join result $\mathit{Join}(Q(B))$ empty. Otherwise, $\text{AGM}_W(B)$ is at least 1 (because Equation (7), if not equal to 0, must be at least 1) but less than 2 (because B is a leaf). It follows immediately that $\mathit{Join}(Q(B))$ can have at most one tuple.

We can utilize algorithm `split` in Figure 2 to compute the result of $Q(B)$. For this purpose, run $\texttt{split}(1, B)$ and collect the set C of boxes returned. We claim that every box $B' \in C$ must satisfy $\text{AGM}_W(B') = 0$, except possibly only one box B''; furthermore, if B'' exists, it must have degenerated into a point! We thus report the point if it is a join result tuple (this is equivalent to checking if $\text{AGM}_W(B'') = 1$). The overall running time is $\tilde{O}(1)$.

It remains to prove our claim. Recall that whenever `split` adds B_{left} to C, it ensures $\text{AGM}_W(B_{left}) \le \frac{1}{2}\text{AGM}_W(B)$, which is less than 1. This means $\text{AGM}_W(B_{left}) = 0$ (as mentioned, Equation (7) is either 0 or at least 1). The same applies to every B_{right} added to C. Hence, if the aforementioned box B'' indeed exists, it must have been added to C as B_{mid}. However, during all the recursive calls to `split`, only one B_{mid} is added to C (at Line 5 of Figure 2), and this B_{mid} must have degenerated into a point. This explains why B'' is unique and must be a point. □

```
algorithm sample($W$)
/* $W$ is a fractional edge covering of $Q$ */
1.  $B \leftarrow \mathbb{N}^d$ /* the attribute space */
2.  while $\text{AGM}_W(B) \ge 2$ do
3.      apply Theorem 2 to split $B$ into a set $C$ of boxes
4.      take a random box $B_{child}$ from $C$ such that
        $\mathbf{Pr}[B_{child} = B'] = \frac{\text{AGM}_W(B')}{\text{AGM}_W(B)}$ for each $B' \in C$, and
        $\mathbf{Pr}[B_{child} = nil] = 1 - \sum_{B' \in C} \text{AGM}_W(B')/\text{AGM}_W(B)$
5.      if $B_{child} = nil$ then return "failure"
6.      $B \leftarrow B_{child}$
7.  apply Lemma 4 to compute $\mathit{Join}(Q(B))$
8.  if $\mathit{Join}(Q(B)) = \emptyset$ then return "failure"
9.  toss a coin with heads probability $1/\text{AGM}_W(B)$
10. if the coin comes up heads then
        return the (only) tuple in $\mathit{Join}(Q(B))$
11. return "failure"
```

Figure 3: The proposed sampling algorithm

4.2 The Sampling Algorithm

We emphasize that the join box-tree $\mathcal{T}$ is conceptual: its size is too large[7] such that we cannot afford to materialize it. To extract a sample from the join result $\mathit{Join}(Q)$, our algorithm will generate — on the fly — a single root-to-leaf path of $\mathcal{T}$ in $\tilde{O}(1)$ time and then wipe off the path from memory immediately. The path generation may not always produce a sample, but it does so with probability $\text{OUT}/\text{AGM}_W(Q)$, where $\text{OUT} = |\mathit{Join}(Q)|$. Thus, $\text{AGM}_W(Q)/\text{OUT}$ repeats will get us a sample in expectation.

Figure 3 presents the details of our sampling algorithm. We start from the root of $\mathcal{T}$. In general, suppose that we are currently standing at a node with box B. Let us first consider B to be an internal node. We obtain the set C of its child nodes using the AGM split theorem in $\tilde{O}(1)$ time, and then descend into a child B_{child} randomly selected from C with weighted sampling. Specifically, each $B' \in C$ is chosen with probability $\text{AGM}_W(B')/\text{AGM}_W(B)$. By Property 3 of Theorem 2, we have $\sum_{B' \in C} \text{AGM}_W(B') \le \text{AGM}_W(B)$. Thus, with probability $1 - \sum_{B' \in C} \text{AGM}_W(B')/\text{AGM}_W(B)$, no child is selected, in which case we declare "failure" and the algorithm terminates. Because C has size at most $2d + 1 = O(1)$, the weighted sampling takes $\tilde{O}(1)$ time, which is the cost to compute the AGM bounds of all the boxes in C (Proposition 1).

Next, let us look at the scenario where B is a leaf. We use Lemma 4 to compute the result of the sub-join $Q(B)$ in $\tilde{O}(1)$ time. If $\mathit{Join}(Q(B)) = \emptyset$, the algorithm terminates with "failure". Otherwise, $\mathit{Join}(Q(B))$ has only a single tuple $\boldsymbol{u}$ (Lemma 4). We return $\boldsymbol{u}$ (as the join sample of the original join Q) with probability $1/\text{AGM}_W(B)$, but still declare "failure" with probability $1 - 1/\text{AGM}_W(B)$.

It is clear that `sample` runs in $\tilde{O}(1)$ time (remember that $\mathcal{T}$ has height $\tilde{O}(1)$; see Proposition 2). Next, we prove that if it returns a tuple, then the tuple must have been taken from $\mathit{Join}(Q)$ uniformly at random. Proposition 3 guarantees that every tuple $\boldsymbol{u} \in \mathit{Join}(Q)$, which can be regarded as a point in the attribute space, is covered by the box B of exactly one leaf in $\mathcal{T}$. Consider any $\boldsymbol{u} \in \mathit{Join}(Q)$ and its covering leaf B. Let Π be the path from the root of $\mathcal{T}$ to the node

[7]The number of nodes in $\mathcal{T}$ is at least $|\mathit{Join}(Q)|$.

B. Algorithm `sample` outputs $\boldsymbol{u}$ with probability

$$\left(\prod_{\text{non-root } B' \in \Pi} \frac{\mathrm{AGM}_W(B')}{\mathrm{AGM}_W(parent(B'))}\right) \cdot \frac{1}{\mathrm{AGM}_W(B)} \tag{9}$$

where $parent(B')$ represents the parent of node B'. It is easy to see that (9) evaluates to $1/\mathrm{AGM}_W(\mathbb{N}^d)$, where $\mathrm{AGM}_W(\mathbb{N}^d)$ is the AGM bound of the root of $\mathcal{T}$ and equals $\mathrm{AGM}_W(Q)$. In other words, $\boldsymbol{u}$ is sampled with probability $1/\mathrm{AGM}_W(Q)$. As the probability is identical for all $\boldsymbol{u} \in \mathcal{J}oin(Q)$, our algorithm returns a uniformly random tuple in $\mathcal{J}oin(Q)$, provided that it does not declare failure.

We can now calculate the probability that algorithm "succeeds" (i.e., returning a tuple) as

$$\sum_{\boldsymbol{u} \in \mathcal{J}oin(Q)} \mathbf{Pr}[\boldsymbol{u} \text{ is returned}] = \frac{\mathrm{OUT}}{\mathrm{AGM}_W(Q)}.$$

A standard application of Chernoff bounds shows that we can get a sample w.h.p. by repeating the algorithm $\tilde{O}(\mathrm{AGM}_W(Q)/\mathrm{OUT})$ times with a total cost of $\tilde{O}(\mathrm{AGM}_W(Q)/\mathrm{OUT})$.

A special case occurs when $\mathrm{OUT} = 0$, which would force us into infinite repeats. The issue can be easily dealt with by stopping after $\tilde{O}(\mathrm{AGM}_W(Q))$ repeats and reverting to a worst-case optimal join algorithm (e.g., Generic Join [47]) to evaluate Q in full (which will confirm $\mathrm{OUT} = 0$). The overall cost is $\tilde{O}(\mathrm{AGM}_W(Q))$.

4.3 Oracles

It is time to clarify the oracles. In fact, the count and median oracles only require solving problems that are nowadays considered rudimentary in the data-structure area of computer science. We can, for example, maintain a set of range trees [9, 23] to implement the count oracle and a set of binary search trees to implement the median oracle. All these trees occupy $\tilde{O}(\mathrm{IN})$ space, can be built in $\tilde{O}(\mathrm{IN})$ time, and can be modified in $\tilde{O}(1)$ time along with each update in the relations of Q. Further details are available in Appendix B. We thus have established the following theorem.

Theorem 5. *Consider an arbitrary (natural) join Q involving a constant number of attributes. Let W be an arbitrary fractional edge covering of the schema graph of Q. There is an index structure of $\tilde{O}(\mathrm{IN})$ space that can be used to extract, with high probability, a sample from the join result of Q in $\tilde{O}(\mathrm{AGM}_W(Q)/\max\{1, \mathrm{OUT}\})$ time, and supports an update in the relations of Q in $\tilde{O}(1)$ time, where IN and OUT, respectively, are the input and output sizes of Q, and $\mathrm{AGM}_W(Q)$ is the AGM bound of Q under W given by Lemma 1. The structure's update and sampling algorithms are combinatorial.*

By choosing an optimal fractional edge covering W, the sample time in the theorem is bounded by the complexity in (2).

5 HARDNESS OF JOIN SAMPLING AND OUTPUT-SENSITIVE JOIN ALGORITHMS

This section will provide evidence that Theorem 5 can no longer be improved significantly for all joins. For this purpose, we will argue that no combinatorial structure can give a "yes" answer to the join sampling question unless the combinatorial k-clique hypothesis is wrong. To do so, we will take a de-tour to bridge join sampling with output-sensitive join evaluation.

Let $\mathcal{A}$ be a combinatorial algorithm for computing $\mathcal{J}oin(Q)$. Recall from Section 1 that $\mathcal{A}$ is ϵ-output sensitive — where $0 < \epsilon < 1/2$ — if it runs in time $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ as long as $\mathrm{OUT} \le \mathrm{IN}^{\epsilon}$. The lemma below is proved in Appendix C.

Lemma 6. *If a combinatorial structure can answer "yes" to the join sampling question for a constant $\epsilon \in (0, 1/2)$, there is a combinatorial ϵ-output sensitive algorithm for join computation.*

An ϵ-output sensitive algorithm $\mathcal{A}$ does not need to be fast when $\mathrm{OUT} > \mathrm{IN}^{\epsilon}$. However, when $\mathrm{OUT} > \mathrm{IN}^{\epsilon}$, the sampling algorithm in Theorem 5 runs in $\tilde{O}(\frac{\mathrm{IN}^{\rho^*}}{\mathrm{OUT}}) = \tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ time. In Appendix D, we combine the two algorithms to prove:

Lemma 7. *Given a combinatorial ϵ-output sensitive algorithm, we can design a combinatorial algorithm to detect whether $\mathcal{J}oin(Q)$ is empty in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p., regardless of the value* OUT.

As shown in Appendix F, however, the emptiness-detection algorithm in Lemma 7 breaks the combinatorial k-clique hypothesis. In summary, finding a combinatorial ϵ-output sensitive algorithm is at least as hard as breaking the hypothesis. Lemma 6 indicates that it can be only harder to improve our result in Theorem 5 by a polynomial factor even when $\mathrm{OUT} \ll \mathrm{IN}^{\rho^*}$.

6 APPLICATIONS

Our join sampling structure can be utilized to tackle many problems, a partial list of which is presented below.

- (Join size estimation) It is standard [21] to use join sampling to estimate $|\mathcal{J}oin(Q)|$ up to a relative error λ. Our structure in Theorem 5 can be applied to produce an estimate in $\tilde{O}(\frac{1}{\lambda^2}\frac{\mathrm{IN}^{\rho^*}}{\max\{1,\mathrm{OUT}\}})$ time w.h.p., improving the state of the art in [21] by an $O(\mathrm{IN})$ factor. We can also support each update in $\tilde{O}(1)$ time.
- (Subgraph sampling) Let $G := (V, E)$ be a simple undirected graph. Given a pattern graph Q of a constant size, a query samples an occurrence[8] of Q in G uniformly at random. We obtain a structure of $\tilde{O}(|E|)$ space that answers a query in $\tilde{O}(|E|^{\rho^*}/\max\{1, \mathrm{OCC}\})$ time w.h.p., where ρ^* is the fractional edge covering number of Q and OCC is the number of occurrences of Q in G. The structure supports an edge insertion and deletion in $\tilde{O}(1)$ time. Previously, a structure matching our guarantees was given in [29]. Our solution is drastically different and actually settles a problem we call "join sampling with predicates", which captures subgraph sampling as a (simple) special case. Details are available in Appendix E.
- (Joins with random enumeration) Carmeli et al. [15] proposed a variant of the join computation problem, where the objective is to design an algorithm that, after an initial pre-processing, can (i) produce a random permutation of the tuples in $\mathcal{J}oin(Q)$, and (ii) do so with a small delay Δ (the maximum time gap between the reporting of two consecutive tuples in the permutation). We obtain the first algorithm that, after an initial

[8]An occurrence is a subgraph of G isomorphic to Q.

$\tilde{O}(\text{IN})$-time preprocessing, produces the whole permutation in $\tilde{O}(\text{IN}^{\rho^*})$ time (i.e., worst-case optimal up to a polylogarithmic factor) with a delay $\tilde{O}(\text{IN}^{\rho^*}/\max\{1, \text{OUT}\})$ w.h.p.. Details are available in Appendix G.

- (Join Union Sampling) Let Q_1, Q_2, ..., Q_k be joins with $var(Q_1) = var(Q_2) = ... = var(Q_k)$, where $k \geq 2$ is a constant. We want to sample from $\bigcup_{i=1}^{k} \mathcal{J}oin(Q_i)$ uniformly at random. Let IN be the total number of tuples in the input relations of Q_1, ..., Q_k, OUT $:= |\bigcup_{i=1}^{k} \mathcal{J}oin(Q_i)|$, and ρ^* be the maximum fractional edge covering number of the schema graphs of Q_1, ..., Q_k. We obtain a structure of $\tilde{O}(\text{IN})$ space that can extract a uniform sample in $\tilde{O}(\text{IN}^{\rho^*}/\max\{1, \text{OUT}\})$ time w.h.p. and support an update in any input relation in $\tilde{O}(1)$ time. Details are available in Appendix H.

7 POST-ACCEPTANCE REMARKS

In another paper [37] accepted to PODS'23, Kim et al. also developed a join sampling algorithm achieving performance guarantees similar to ours. Their algorithm is elegant and approaches the problem from a perspective different from our work. In [37], Kim et al. further discussed (i) some scenarios where better running time was possible and (ii) how to estimate the join result size.

ACKNOWLEDGEMENTS

This work was supported in part by GRF projects 14207820, 14203421, and 14222822 from HKRGC.

REFERENCES

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 98–117, 2015.

[2] Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 7:1–7:15, 2019.

[3] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 275–286, 1999.

[4] Pankaj K. Agarwal. Range searching. In *Handbook of Discrete and Computational Geometry, 2nd Ed*, pages 809–837. Chapman and Hall/CRC, 2004.

[5] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Eurosys*, pages 29–42, 2013.

[6] Kaleb Alway, Eric Blais, and Semih Salihoglu. Box covers and domain orderings for beyond worst-case join processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 3:1–3:23, 2021.

[7] Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In Ronald Fagin, editor, *Proceedings of International Conference on Database Theory (ICDT)*, volume 361, pages 121–126, 2009.

[8] Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

[9] Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters (IPL)*, 8(5):244–251, 1979.

[10] Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1836–1855, 2021.

[11] Karl Bringmann, Nick Fischer, and Marvin Kunnemann. A fine-grained analogue of schaefer's theorem in P: dichotomy of exists^k-forall-quantified first-order graph properties. In *Computational Complexity Conference*, pages 31:1–31:27, 2019.

[12] Karl Bringmann, Allan Gronlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–318, 2017.

[13] Karl Bringmann and Philip Wellnitz. Clique-based lower bounds for parsing tree-adjoining grammars. In *Proceedings of Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 12:1–12:14, 2017.

[14] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 325–341, 2021.

[15] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 393–409, 2020.

[16] Timothy M. Chan. A (slightly) faster algorithm for klee's measure problem. *Comput. Geom.*, 43(3):243–250, 2010.

[17] Yi-Jun Chang. Hardness of RNA folding problem with four symbols. *Theor. Comput. Sci.*, 757:11–26, 2019.

[18] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. On random sampling over joins. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 263–274, 1999.

[19] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

[20] Yu Chen and Ke Yi. Two-level sampling for join size estimation. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 759–774, 2017.

[21] Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 7:1–7:18, 2020.

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

[23] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[24] Rina Dechter and Judea Pearl. Tree-clustering schemes for constraint-processing. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, pages 150–154, 1988.

[25] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Enumeration algorithms for conjunctive queries with projection. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 186, pages 14:1–14:17.

[26] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 1213–1223, 2020.

[27] Shaleen Deep and Paraschos Koutris. Compressed representations of conjunctive query results. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 307–322, 2018.

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

[29] Hendrik Fichtenberger, Mingze Gao, and Pan Peng. Sampling arbitrary subgraphs exactly uniformly in sublinear time. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 45:1–45:13, 2020.

[30] Ehud Friedgut. Hypergraphs, entropy, and inequalities. *Am. Math. Mon.*, 111(9):749–760, 2004.

[31] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences (JCSS)*, 66(4):775–808, 2003.

[32] Etienne Grandjean and Louis Jachiet. Which arithmetic operations can be performed in constant time in the RAM model with addition? *CoRR*, abs/2206.13851, 2022.

[33] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 287–298, 1999.

[34] Kathrin Hanauer, Monika Henzinger, and Qi Cheng Hua. Fully dynamic four-vertex subgraph counting. In *Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, volume 221, pages 18:1–18:17, 2022.

[35] Ce Jin and Yinzhan Xu. Tight dynamic problem lower bounds from generalized BMM and omv. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, 2022.

[36] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Re, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems (TODS)*, 41(4):22:1–22:45, 2016.

[37] Kyoungmin Kim, Jaehyun Ha, George Fletcher, and Wook-Shin Han. Guaranteeing the $\tilde{O}(\text{AGM/OUT})$ runtime for uniform sampling and size estimation over joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2023.

[38] Kyoungmin Kim, Hyeonji Kim, George Fletcher, and Wook-Shin Han. Combining sampling and synopses with worst-case optimal runtime and quality guarantees for graph pattern cardinality estimation. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 964–976, 2021.

[39] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 615–629, 2016.

[40] Jason Li. Faster minimum k-cut of a simple graph. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1056–1077, 2019.

[41] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1236–1252, 2018.

[42] Gonzalo Navarro, Juan L. Reutter, and Javiel Rojas-Ledesma. Optimal joins using compact data structures. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 155, pages 21:1–21:21, 2020.

[43] Jaroslav Nesetril and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[44] Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 234–245, 2014.

[45] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-Case Optimal Join Algorithms: [Extended Abstract]. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2012.

[46] Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018.

[47] Hung Q. Ngo, Christopher Re, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013.

[48] Supriya Nirkhiwale, Alin Dobra, and Christopher M. Jermaine. A sampling algebra for aggregate estimation. *Proceedings of the VLDB Endowment (PVLDB)*, 6(14):1798–1809, 2013.

[49] Dan Olteanu and Jakub Zavodny. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):2:1–2:44, 2015.

[50] Rodrygo L. T. Santos, Craig MacDonald, and Iadh Ounis. Search result diversification. *Found. Trends Inf. Retr.*, 9(1):1–90, 2015.

[51] Ali Mohammadi Shanghooshabad, Meghdad Kurmanji, Qingzhi Ma, Michael Shekelyan, Mehrdad Almasi, and Peter Triantafillou. Pgmjoins: Random join sampling with graphical models. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 1610–1622, 2021.

[52] Yufei Tao and Ke Yi. Intersection joins under updates. *Journal of Computer and System Sciences (JCSS)*, 124:41–64, 2022.

[53] Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters (IPL)*, 109(4):254–257, 2009.

[54] Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

[55] David Vengerov, Andre Cavalheiro Menck, Mohamed Zaït, and Sunil Chakkappen. Join size estimation subject to filter conditions. *Proceedings of the VLDB Endowment (PVLDB)*, 8(12):1530–1541, 2015.

[56] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94, 1981.

[57] Feng Yu, Wen-Chi Hou, Cheng Luo, Dunren Che, and Mengxia Zhu. CS2: a new database synopsis for query estimation. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 469–480, 2013.

[58] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 1525–1539, 2018.

[59] Zhuoyue Zhao, Feifei Li, and Yuxi Liu. Efficient join synopsis maintenance for data warehouse. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 2027–2042, 2020.

APPENDIX

A PROOF OF LEMMA 3

We first review Friedgut's inequality (sometimes called the generalized Höder's inequality). Fix some integers p and q at least 1. Let $\{a_{i,j} \mid i \in [1,p], j \in [1,q]\}$ be a set of non-negative real values. Also, let $\{b_i \mid i \in [1,q]\}$ be another set of non-negative real values satisfying $\sum_{k=1}^{q} b_k \geq 1$. Assuming $0^0 = 0$, Friedgut's inequality [30] states

$$\sum_{i=1}^{p} \prod_{j=1}^{q} a_{i,j}^{b_j} \leq \prod_{j=1}^{q} \left(\sum_{i=1}^{p} a_{i,j} \right)^{b_j}. \tag{10}$$

Returning to the context of Lemma 3, recall that we have already fixed an integer $i \in [1,d]$: this is the "i" used to create $B_1, B_2, ..., B_s$ in (8). Define $\mathcal{E}_i = \{e \in \mathcal{E} \mid X_i \in e\}$, the set of edges in the schema graph of Q that cover attribute X_i. For each $j \in [1,s]$, we have

$$\begin{aligned}
\text{AGM}_W(B_j) &= \prod_{e \in \mathcal{E}} |R_e(B_j)|^{W(e)} \qquad \text{(see (7))} \\
&= \prod_{e \in \mathcal{E} \setminus \mathcal{E}_i} |R_e(B_j)|^{W(e)} \cdot \prod_{e \in \mathcal{E}_i} |R_e(B_j)|^{W(e)} \\
&= \prod_{e \in \mathcal{E} \setminus \mathcal{E}_i} |R_e(B)|^{W(e)} \cdot \prod_{e \in \mathcal{E}_i} |R_e(B_j)|^{W(e)}
\end{aligned}$$

where the last equality used the fact $R_e(B_j) = R_e(B)$ for $e \in \mathcal{E} \setminus \mathcal{E}_i$, which holds because B and B_j have the same projections on all attributes other than X_i, but $X_i \notin e$. We can now derive

$$\begin{aligned}
&\sum_{j=1}^{s} \text{AGM}_W(B_j) \\
=\ & \sum_{j=1}^{s} \left(\prod_{e \in \mathcal{E} \setminus \mathcal{E}_i} |R_e(B)|^{W(e)} \cdot \prod_{e \in \mathcal{E}_i} |R_e(B_j)|^{W(e)} \right) \\
=\ & \prod_{e \in \mathcal{E} \setminus \mathcal{E}_i} |R_e(B)|^{W(e)} \cdot \sum_{j=1}^{s} \left(\prod_{e \in \mathcal{E}_i} |R_e(B_j)|^{W(e)} \right) \\
\leq\ & \prod_{e \in \mathcal{E} \setminus \mathcal{E}_i} |R_e(B)|^{W(e)} \cdot \prod_{e \in \mathcal{E}_i} \left(\sum_{j=1}^{s} |R_e(B_j)| \right)^{W(e)} \\
& \text{(applying (10), noticing that } \sum_{e \in \mathcal{E}_j} W(e) \geq 1) \\
=\ & \prod_{e \in \mathcal{E} \setminus \mathcal{E}_i} |R_e(B)|^{W(e)} \cdot \prod_{e \in \mathcal{E}_i} |R_e(B)|^{W(e)} \\
& \text{(because } \bigcup_{j=1}^{s} B_j = B \text{ and } B_1, ..., B_s \text{ are disjoint)} \\
=\ & \text{AGM}_W(B).
\end{aligned}$$

B ORACLE IMPLEMENTATION

The count oracle essentially deals with a problem known as *orthogonal range counting*. In that problem, the input is a set P of n points in d-dimensional space $\mathbb{R}^d$ for some constant d. Given an axis-parallel rectangle q in $\mathbb{R}^d$, a query returns $|P \cap q|$, namely, the number of points in P that are covered by q. The goal is to store P in a data structure to answer queries efficiently. We refer the reader to [4] for a survey on the known data structures solving this problem. Among them is the range tree [9, 23], which consumes $O(n \log^{d-1} n)$ space, answers a query in $O(\log^d n)$ time, and supports a point insertion and deletion in P using $O(\log^d n)$ time. It serves as a count oracle meeting our requirements.

Before implementing the median oracle, let us look at an alternative problem first. The input is a set S of n real values. Given an interval q in $\mathbb{R}$ and an integer $k \geq 1$, a query returns the k-th smallest integer in $S \cap q$ (or returns nothing in the special case where $k > |S \cap q|$). The goal is to store S in a data structure to answer queries efficiently. We can create a binary search tree (BST) on S and keep, at each node v in the tree, the number of descendant nodes of v. The tree occupies $O(n)$ space, answers a query in $O(\log n)$ time (see Chapter 14 "Augmenting Data Structures" of [22]), and supports an insertion or deletion in S using $O(\log n)$ time. The same structure can also find the size of $S \cap q$ in $O(\log n)$ time (see the above chapter of [22] again). It thus follows that we can report the median value in $S \cap q$ in $O(\log n)$ time.

To implement a median oracle, for every attribute $X \in var(Q)$, we maintain the aforementioned (slightly-augmented) BST on the set S of X-values that appear in at least one relation in $\boldsymbol{Q}$. The structure occupies $\tilde{O}(\mathrm{IN})$ space and, given a box B, finds the median of $\mathbf{actdom}(X, B)$ — which is the median of the values of S covered by the interval $B(X)$ — in $O(\log \mathrm{IN})$ time. It is straightforward to maintain the tree in $O(\log \mathrm{IN})$ time per update.

C PROOF OF LEMMA 6

Suppose that, given a join $\boldsymbol{Q}$, we can build a combinatorial structure Υ in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ time that can extract a uniform sample from $\mathcal{J}oin(\boldsymbol{Q})$ in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon}/\mathrm{OUT})$ time when $\mathrm{OUT} \in [1, \mathrm{IN}^{\epsilon}]$. Next, we will show how to use Υ to compute $\mathcal{J}oin(\boldsymbol{Q})$ in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ time when $\mathrm{OUT} \le \mathrm{IN}^{\epsilon}$, thereby establishing the lemma.

Let us first assume that we know, by magic, the value OUT. If $\mathrm{OUT} = 0$, $\mathcal{J}oin(\boldsymbol{Q})$ is empty and there is nothing to do. Otherwise, we deploy Υ to extract $s := c \cdot \mathrm{OUT} \cdot \ln \mathrm{IN}$ samples from $\mathcal{J}oin(\boldsymbol{Q})$ where c is a sufficiently large constant. W.h.p., every tuple $\boldsymbol{u} \in \mathcal{J}oin(\boldsymbol{Q})$ must have been sampled at least once. Indeed, the probability for $\boldsymbol{u}$ to have been missed by all those s samples is $(1 - \frac{1}{\mathrm{OUT}})^s$, which is at most $e^{-s/\mathrm{OUT}} = 1/\mathrm{IN}^c$. It thus holds with probability at least $1 - \mathrm{OUT}/\mathrm{IN}^c \ge 1 - \mathrm{IN}^{\rho^*}/\mathrm{IN}^c = 1 - 1/\mathrm{IN}^{c-\rho^*}$ that all the tuples in $\mathcal{J}oin(\boldsymbol{Q})$ are sampled. The total running time is $O(s \cdot \mathrm{IN}^{\rho^*-\epsilon}/\mathrm{OUT}) = \tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$.

The rest of the proof explains how to remove the magic assumption. Instead of the exact OUT, we aim to obtain an over-estimate $\widehat{\mathrm{OUT}}$ satisfying $\mathrm{OUT} \le \widehat{\mathrm{OUT}} \le 2\mathrm{OUT}$. By replacing OUT with $\widehat{\mathrm{OUT}}$ in the above, we can only decrease the algorithm's failure probability, while keeping the execution time at the same order. In [21], Chen and Yi described a method for estimating OUT, but their method requires special knowledge of the sampling algorithm of Υ.[9] Our method, presented below, works for any sampling algorithm.

We start by using the sample algorithm of Υ to find out if $\mathrm{OUT} = 0$. Recall that, if $\mathrm{OUT} \ge 1$, the algorithm is required to return a sample in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p. (we can assume $\mathrm{OUT} \le \mathrm{IN}^{\epsilon}$ because otherwise our ϵ-output sensitive algorithm does not need to guarantee anything). Motivated by this, we allow the algorithm to execute for $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ time and then manually terminate it if it has not finished yet. If a sample has been returned, then obviously $\mathrm{OUT} > 0$; otherwise, we declare $\mathcal{J}oin(\boldsymbol{Q}) = \emptyset$.

The subsequent discussion concentrates on the scenario of $\mathrm{OUT} > 0$. We use Υ to extract samples continuously and, for each sample, check if it has been seen before (this can be done in $\tilde{O}(1)$ time by maintaining a dictionary-search structure, e.g., the BST, on the seen samples). The extraction stops as soon as Υ churns out $\Delta := c' \log \mathrm{IN}$ seen samples *in a row*, where c' is a sufficiently large constant. At this moment, count the number t of distinct samples already obtained and finalize our estimate $\widehat{\mathrm{OUT}} := 2t$.

[9] Specifically, Chen and Yi's method assumes that the sampling algorithm of Υ works by repeatedly making trials, each of which either declares "failure" or produces a sample. The failure probability must be available for their method to work.

It is easy to analyze the running time. Until termination, the algorithm finds a new tuple in $\mathcal{J}oin(\boldsymbol{Q})$ after drawing at most Δ samples in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon}/\mathrm{OUT})$ time. As $\mathcal{J}oin(\boldsymbol{Q})$ has OUT tuples, the total execution time is no more than $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$.

To complete the whole proof, we argue that $t \ge \mathrm{OUT}/2$ w.h.p., which means $\widehat{\mathrm{OUT}} \in [\mathrm{OUT}, 2\mathrm{OUT}]$ w.h.p., as desired. Fix an arbitrary integer $\tau \in [0, \mathrm{OUT}/2)$. For the algorithm to terminate with $t = \tau$, Υ needs to output Δ seen samples in a row when $\mathcal{J}oin(\boldsymbol{Q})$ still has at least $\mathrm{OUT} - \tau \ge \mathrm{OUT}/2$ tuples never sampled. As each sample is uniformly random, it has at least $1/2$ probability to hit an unseen tuple. The probability of fetching Δ seen samples continuously is at most $(1/2)^{\Delta} = 1/\mathrm{IN}^{c'}$, which is thus an upper bound for the algorithm to terminate with $t = \tau$. Accounting for all the possible τ values, we can conclude that the algorithm finishes with a $t \in [0, \mathrm{OUT}/2)$ with probability $O(\mathrm{OUT}/\mathrm{IN}^{c'}) = O(1/\mathrm{IN}^{c'-\rho^*})$. Therefore, t has probability $1 - O(1/\mathrm{IN}^{c'-\rho^*})$ to be at least $\mathrm{OUT}/2$.

D PROOF OF LEMMA 7

As before, let $\mathcal{A}$ be the given combinatorial ϵ-output sensitive algorithm. Denote by $\mathcal{A}'$ the sampling algorithm of our structure in Theorem 5. To detect whether $\mathcal{J}oin(\boldsymbol{Q})$ is empty, we first build our structure on $\boldsymbol{Q}$ in $\tilde{O}(\mathrm{IN})$ time by inserting every tuple of the input relations one by one. Then, run $\mathcal{A}$ and $\mathcal{A}'$ in an interleaving manner, that is, run a step (of constant time) of $\mathcal{A}$, followed by a step of $\mathcal{A}'$, another step of $\mathcal{A}$, then a step of $\mathcal{A}'$, and so on. The interleaving process stops as soon as *either* algorithm finishes. At that moment, check whether $\mathcal{A}$ and $\mathcal{A}'$ have found any tuple of $\mathcal{J}oin(\boldsymbol{Q})$. If so, obviously $\mathcal{J}oin(\boldsymbol{Q})$ is not empty; otherwise, declare $\mathcal{J}oin(\boldsymbol{Q})$ empty.

Let us represent the above emptiness-detection algorithm as $\mathcal{A}_{emp}$. Next, we will prove that $\mathcal{A}_{emp}$, w.h.p., correctly decides if $\mathcal{J}oin(\boldsymbol{Q}) = \emptyset$ and runs in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ time. Our analysis is through a case-by-case discussion on the value of OUT.

- If $\mathrm{OUT} = 0$, $\mathcal{A}_{emp}$ always returns "$\mathcal{J}oin(\boldsymbol{Q})$ empty". Because $\mathcal{A}$ is ϵ-output sensitive, it terminates in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p.. The cost of $\mathcal{A}_{emp}$ is thus bounded by $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ w.h.p..
- Consider now $0 < \mathrm{OUT} \le \mathrm{IN}^{\epsilon}$. Being ϵ-output sensitive, $\mathcal{A}$ must report the full $\mathcal{J}oin(\boldsymbol{Q})$ in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p.. Hence, $\mathcal{A}_{emp}$ finds a tuple of $\mathcal{J}oin(\boldsymbol{Q})$ in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p..
- The final case is $\mathrm{OUT} > \mathrm{IN}^{\epsilon}$. By Theorem 5, $\mathcal{A}'$ must return a join sample of $\boldsymbol{Q}$ in $\tilde{O}(\mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p.. $\mathcal{A}_{emp}$ thus finds a tuple of $\mathcal{J}oin(\boldsymbol{Q})$ in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^*-\epsilon})$ time w.h.p..

E SUBGRAPH SAMPLING

Before discussing subgraph sampling, we first extend Theorem 5 to a scenario we call *join sampling with predicates*, where sampling is performed on only a subset of the join result. Let $\boldsymbol{Q}$ be a join defined in Section 2.1. Given a boolean predicate σ, define $\mathcal{J}oin(\sigma, \boldsymbol{Q}) := \{\boldsymbol{u} \in \mathcal{J}oin(\boldsymbol{Q}) \mid \boldsymbol{u} \text{ satisfies } \sigma\}$, i.e., the subset of tuples in $\mathcal{J}oin(\boldsymbol{Q})$ passing the filtering condition σ. A *σ-join sample* of $\boldsymbol{Q}$ is a tuple taken uniformly at random from $\mathcal{J}oin(\sigma, \boldsymbol{Q})$. We want to create an index structure on $\boldsymbol{Q}$ to allow fast extraction of σ-join samples.

Interestingly, our structure in Theorem 5 can be deployed *directly* to draw a σ-join sample, even if the predicate σ is supplied at run time. For this purpose, simply apply the `sample` algorithm in Figure 3. If `sample` declares "failure", we declare the same. Otherwise, suppose that `sample` returns a sample $\boldsymbol{u} \in \mathcal{J}oin(\mathcal{Q})$, which we output only if $\boldsymbol{u}$ satisfies σ. Otherwise ($\boldsymbol{u}$ violates σ), we again declare "failure". The above algorithm (for drawing a σ-join sample) will be referred to as σ-`sample` henceforth.

Recall that `sample` draws $\boldsymbol{u}$ from $\mathcal{J}oin(\mathcal{Q})$ uniformly at random. Thus, every tuple in $\mathcal{J}oin(\sigma, \mathcal{Q})$, which is a subset of $\mathcal{J}oin(\mathcal{Q})$, has the same chance to be taken as $\boldsymbol{u}$. Hence, σ-`sample`, if it succeeds (i.e., outputting a tuple), returns a σ-join sample of $\mathcal{Q}$. To analyze its success probability, let $\mathrm{OUT}_\sigma := |\mathcal{J}oin(\sigma, \mathcal{Q})|$. As shown in Section 4.2, `sample` outputs a tuple of $\mathcal{J}oin(\mathcal{Q})$ with probability $\frac{\mathrm{OUT}}{\mathrm{AGM}_W(\mathcal{Q})}$, which tells us that σ-`sample` succeeds with probability $\frac{\mathrm{OUT}}{\mathrm{AGM}_W(\mathcal{Q})} \cdot \frac{\mathrm{OUT}_\sigma}{\mathrm{OUT}} = \frac{\mathrm{OUT}_\sigma}{\mathrm{AGM}_W(\mathcal{Q})}$. It thus follows that we can, by repeating the algorithm until a success, draw a σ-join sample of $\mathcal{Q}$ in $\tilde{O}(\mathrm{AGM}_W(\mathcal{Q})/\max\{1, \mathrm{OUT}_\sigma\})$ time, which can be made $\tilde{O}(\mathrm{IN}^{\rho^*}/\max\{1, \mathrm{OUT}_\sigma\})$ by choosing the fractional edge covering W optimally, where ρ^* is the fractional edge covering number of the schema graph of $\mathcal{Q}$.

We are ready to solve the subgraph sampling problem. Recall that the goal is to preprocess an undirected graph $G := (V, E)$ such that, given a constant-size pattern graph Q, we can uniformly sample an occurrence of Q from G. Let $\mathcal{X}$ be the set of vertices in Q, and $\mathcal{E}$ be the set of edges in Q. It is worth reminding the reader that every edge in G and Q contains exactly two vertices.

We create a join $\mathcal{Q}$ with $|\mathcal{E}|$ relations as follows. For each edge $e = \{X, Y\}$ in Q (where X and Y are vertices in Q), create a relation R_e with schema $var(R_e) = \{X, Y\}$. The size of R_e is $2|E|$, i.e., twice the number of edges in G. Specifically, for every edge $\{x, y\}$ in G (where x and y are vertices in G), we create two tuples in R_e: tuple $\boldsymbol{u}_1$ satisfying $\boldsymbol{u}_1(X) = x$ and $\boldsymbol{u}_1(Y) = y$, and tuple $\boldsymbol{u}_2$ satisfying $\boldsymbol{u}_2(X) = y$ and $\boldsymbol{u}_2(Y) = x$. This completes the construction of $\mathcal{Q}$, which has input size $\mathrm{IN} = |\mathcal{E}| \cdot (2|E|) = O(|E|)$.

Every tuple $\boldsymbol{u}$ in the result $\mathcal{J}oin(\mathcal{Q})$ of $\mathcal{Q}$ can be thought of as mapping each edge $\{X, Y\}$ in Q to an edge $\{\boldsymbol{u}(X), \boldsymbol{u}(Y)\}$ in G. If the set of (mapped) edges $\{\{\boldsymbol{u}(X), \boldsymbol{u}(Y)\} \mid \{X, Y\} \in \mathcal{E}\}$ induces an occurrence of Q, we say that $\boldsymbol{u}$ *describes* the occurrence.

The following are two (folklore) facts about relationships between the tuples in $\mathcal{J}oin(\mathcal{Q})$ and the occurrences of Q.

- Fact 1: Every occurrence of Q is described by the same number c of tuples in $\mathcal{J}oin(\mathcal{Q})$, where $c \geq 1$ is a constant. For example, consider Q to be a triangle with vertices X, Y, and Z. An occurrence of Q, which is a triangle with vertices x, y, and z in G, is described by the tuple $\boldsymbol{u} \in \mathcal{J}oin(\mathcal{Q})$ where $\boldsymbol{u}(X) = x$, $\boldsymbol{u}(Y) = y$, and $\boldsymbol{u}(Z) = z$. It is easy to see that the triangle is described by six tuples of $\mathcal{J}oin(\mathcal{Q})$ in total, and six is exactly the number of automorhisms of Q.
- Fact 2: It is possible for $\mathcal{J}oin(\mathcal{Q})$ to contain tuples that do not describe any occurrence of Q. For example, consider Q to be a 4-cycle with vertices X_1, X_2, X_3, and X_4. The tuple $\boldsymbol{u}$ with $(\boldsymbol{u}(X_1), \boldsymbol{u}(X_2), \boldsymbol{u}(X_3), \boldsymbol{u}(X_4)) = (x, y, x, y)$, where $\{x, y\}$ is an edge in G, belongs to $\mathcal{J}oin(\mathcal{Q})$ but does not describe any occurrence of Q.

To sample occurrences, we create a structure of Theorem 5 on $\mathcal{Q}$, which occupies $\tilde{O}(|E|)$ space and can be easily maintained in $\tilde{O}(1)$ time per edge insertion and deletion. To draw a sample, we apply the σ-`sample` algorithm by setting the predicate σ to "tuple $\boldsymbol{u}$ should describe an occurrence of Q". The predicate can be evaluated in constant time. By Fact 1, the value of OUT_σ equals $c \cdot \mathrm{OCC}$, where OCC is the number of occurrences of Q in G. Our earlier discussion indicates that the sample time is $\tilde{O}(|E|^{\rho^*}/\max\{1, \mathrm{OCC}\})$.

F ϵ-OUTPUT SENSITIVITY BREAKS COMBINATORIAL k-CLIQUE HYPOTHESIS

By Lemma 7, the existence of an ϵ-output sensitive algorithm implies a combinatorial algorithm $\mathcal{A}_{emp}$ that, given any join $\mathcal{Q}$, can decide whether $\mathcal{J}oin(\mathcal{Q}) = \emptyset$ in $\tilde{O}(\mathrm{IN} + \mathrm{IN}^{\rho^* - \epsilon})$ time w.h.p.. Next, we show how to break the combinatorial k-clique hypothesis with $\mathcal{A}_{emp}$ for any constant $k \geq 3$.

Let $G := (V, E)$ be a simple undirected graph. In Appendix E, we presented a strategy to convert subgraph sampling to join sampling for any pattern graph Q. Here, we apply the same ideas to construct a join $\mathcal{Q}$ from G, setting Q to k-clique. As before, for each edge $e = \{X, Y\}$ in Q, create a relation R_e with schema $var(R_e) = \{X, Y\}$. For every edge $\{x, y\}$ in G, relation R_e contains two tuples: tuple $\boldsymbol{u}_1$ with $\boldsymbol{u}_1(X) = x$ and $\boldsymbol{u}_1(Y) = y$, and tuple $\boldsymbol{u}_2$ with $\boldsymbol{u}_2(X) = y$ and $\boldsymbol{u}_2(Y) = x$. Recall that the conversion has two properties, presented as Facts 1 and 2 in Appendix E. Crucially, Fact 2 can now be strengthened into the claim below (for $Q = k$-clique):

Every tuple in $\mathcal{J}oin(\mathcal{Q})$ describes an occurrence of Q in G.

To explain why, consider any tuple $\boldsymbol{u} \in \mathcal{J}oin(\mathcal{Q})$. For each edge $e = \{X, Y\}$ in Q, we know that $\{\boldsymbol{u}(X), \boldsymbol{u}(Y)\}$ is in R_e and, hence, is an edge in G. It thus follows that G has an edge between vertices $\boldsymbol{u}(X)$ and $\boldsymbol{u}(Y)$ for any two distinct vertices X and Y in Q. Therefore, $\boldsymbol{u}$ describes a k-clique occurrence in G.

Combined with Fact 1, the above claim indicates that G has a k-clique if and only if $\mathcal{J}oin(\mathcal{Q})$ is non-empty. Therefore, we can apply $\mathcal{A}_{emp}$ to detect the emptiness of $\mathcal{J}oin(\mathcal{Q})$ and then decide on the k-clique presence in G. To analyze running time, we note that the fractional edge covering number of k-clique is $k/2$, as is also the fractional edge covering number ρ^* of the schema graph Q of $\mathcal{Q}$. Therefore, the algorithm described earlier runs in $\tilde{O}(|E| + |E|^{\frac{k}{2} - \epsilon})$ time. Applying the trivial fact $|E| \leq |V|^2$, we can relax the time complexity to $\tilde{O}(|V|^2 + |V|^{k-2\epsilon})$, which is $\tilde{O}(|V|^{k-2\epsilon})$ because $k \geq 3$ and $\epsilon < 1/2$, thus breaking the combinatorial k-clique hypothesis.

G JOINS WITH RANDOM ENUMERATION

Our solution to this problem combines ideas in Appendix C and a technique in [52] that adapts a delay-oblivious reporting algorithm for small-delay enumeration.

First, create a structure of Theorem 5 on the given join $\mathcal{Q}$ in $\tilde{O}(\mathrm{IN})$ time. Then, we find out whether $\mathrm{OUT} = 0$. For this purpose, run the sampling algorithm of Theorem 5 — denoted as $\mathcal{A}$ henceforth — to see if it returns a sample. If so, $\mathcal{A}$ must have done so in $\tilde{O}(\mathrm{IN}^{\rho^*}/\mathrm{OUT})$ time w.h.p.; obviously, $\mathrm{OUT} > 0$ in this scenario. Otherwise, $\mathrm{OUT} = 0$ w.h.p., in which case the execution time of $\mathcal{A}$ is $\tilde{O}(\mathrm{IN}^{\rho^*})$, and we have already solved the join.

The subsequent discussion focuses on $\mathrm{OUT} > 0$. We carry out two steps in the same fashion as in Appendix C.

- The first step obtains an estimate $\widehat{\text{OUT}} \in [\text{OUT}, 2\text{OUT}]$ and, in the meantime, reports at least $\text{OUT}/2$ tuples in $\mathcal{J}oin(Q)$. The algorithm is the same as described before: keep sampling with $\mathcal{A}$ until seeing $\Delta := O(\log \text{IN})$ seen tuples in a row. The cost of the step is $\tilde{O}(\frac{\text{IN}^{\rho^*}}{\text{OUT}} \cdot \text{OUT} \cdot \Delta) = \tilde{O}(\text{IN}^{\rho^*})$.
- The second step reports the remaining tuples of $\mathcal{J}oin(Q)$ that have not been found yet. As in Appendix C, we use $\mathcal{A}$ to extract $s := O(\widehat{\text{OUT}} \cdot \ln \text{IN})$ samples and output a sample only if it has never been reported before. The cost of the step is $\tilde{O}(\frac{\text{IN}^{\rho^*}}{\text{OUT}} \cdot s) = \tilde{O}(\text{IN}^{\rho^*})$.

It is immediate from the analysis in Appendix C that the above two-step algorithm w.h.p. manages to output the entire $\mathcal{J}oin(Q)$ in a random permutation. However, it is not designed to achieve a small delay. In fact, Step 1 is fine: it enumerates at least $\text{OUT}/2$ result tuples with a delay $\tilde{O}(\frac{\text{IN}^{\rho^*}}{\text{OUT}} \cdot \Delta) = \tilde{O}(\text{IN}^{\rho^*}/\text{OUT})$. The trouble lies in Step 2 where we may fetch a large number of samples before hitting an unseen tuple.

To eliminate the issue, we resort to a technique of Tao and Yi [52]. They defined a join reporting algorithm $\mathcal{A}'$ to be *α-aggressive* if, after t running time[10] for any integer $t \geq 1$, $\mathcal{A}'$ must have discovered at least $\lfloor t/\alpha \rfloor$ distinct result tuples. They showed that any such $\mathcal{A}'$ can be converted into an algorithm that reports all the result tuples with a delay $O(\alpha)$. Furthermore, the converted algorithm outputs the result tuples in the same order as $\mathcal{A}'$ does.

We claim that the two-step algorithm explained earlier is α-aggressive with

$$\alpha \quad = \quad \beta \cdot \Delta \cdot \frac{\text{IN}^{\rho^*}}{\text{OUT}} \qquad (11)$$

where β is a sufficiently large $\tilde{O}(1)$ factor. Thus, the method of [52] turns our algorithm into one that, w.h.p., outputs a random permutation of $\mathcal{J}oin(Q)$ with delay $\tilde{O}(\text{IN}^{\rho^*}/\text{OUT})$.

To understand the claim, first note that Step 1 is $\tilde{O}(\text{IN}^{\rho^*}/\text{OUT})$-aggressive because, as mentioned, it ensures a delay $\tilde{O}(\text{IN}^{\rho^*}/\text{OUT})$. Consider any moment during Step 2; let t be the running time from the start of Step 1 till that moment. As $t = \tilde{O}(\text{IN}^{\rho^*})$, we can raise β to some $\tilde{O}(1)$ factor to make sure

$$\lfloor t/\alpha \rfloor \leq \lfloor \tilde{O}(\text{IN}^{\rho^*})/\alpha \rfloor = \lfloor \text{OUT} \cdot \tilde{O}(1)/\beta \rfloor \leq \text{OUT}/2$$

with the value α calculated in (11). To argue that our algorithm is α-aggressive, it suffices to show that at least $\text{OUT}/2$ distinct result tuples must have been found at any moment during Step 2. This is true because Step 1 has output at least $\text{OUT}/2$ tuples.

H JOIN UNION SAMPLING

As before, let $Q_1, Q_2, ..., Q_k$ be the joins given (where constant $k \geq 2$), IN be the total input size of all these joins, and $\text{OUT} := |\bigcup_{i=1}^k \mathcal{J}oin(Q_i)|$. For each $i \in [1, k]$, define

- $\mathcal{G}_i := (\mathcal{X}_i, \mathcal{E}_i)$ as the schema graph of Q_i;
- ρ_i^* as the fractional edge covering number of $\mathcal{G}_i$;
- W_i as an optimal fractional edge covering of $\mathcal{G}_i$, namely, $\rho_i^* = \sum_{e \in \mathcal{E}_i} W_i(e)$.

[10]Recall that "running time" in the RAM model is defined as the number of atomic operations, each of which performs constant-time work such as comparison, register assignment, arithmetic computation, memory access, etc.

Therefore, $\rho^* = \max_{i=1}^k \rho_i^*$. Introduce

$$\text{AGMSUM} \quad := \quad \sum_{i=1}^k \text{AGM}_{W_i}(Q_i).$$

It is easy to see that $\text{AGMSUM} = O(\text{IN}^{\rho^*})$.

A tuple $\boldsymbol{u} \in \bigcup_{i=1}^k \mathcal{J}oin(Q_i)$ can be in the result of more than one join among $Q_1, ..., Q_k$. For each $\boldsymbol{u}$, we define its *owner* as the Q_i with the smallest $i \in [1, k]$ satisfying $\boldsymbol{u} \in \mathcal{J}oin(Q_i)$. Once $\boldsymbol{u}$ is given, its owner can be easily determined in $O(k) = O(1)$ time.

For each $i \in [1, k]$, we build a structure Υ_i of Theorem 5 on Q_i, under the fractional edge covering W_i. Since k is a constant, the space consumption of all the structures is $\tilde{O}(\text{IN})$, and it is straightforward to handle an update in any input relation using $\tilde{O}(1)$ time.

The rest of the section will explain how to extract a sample from $\bigcup_{i=1}^k \mathcal{J}oin(Q_i)$. Our algorithm, named `union-sample`, combines our `sample` algorithm in Figure 3 with ideas from [15]. Algorithm `union-sample` has the properties below:

- It finishes in $\tilde{O}(1)$ time;
- It declares "failure" with probability $1 - \text{OUT}/\text{AGMSUM}$.
- If not declaring failure, it outputs a tuple $\boldsymbol{u}$ from $\bigcup_{i=1}^k \mathcal{J}oin(Q_i)$ uniformly at random.

The above properties allow us to extract a sample from $\bigcup_{i=1}^k \mathcal{J}oin(Q_i)$ w.h.p. in time $\tilde{O}(\text{AGMSUM}/\max\{1, \text{OUT}\}) = \tilde{O}(\text{IN}^{\rho^*}/\max\{1, \text{OUT}\})$.

Next, we present the details of `union-sample`. It starts by generating a random integer $i \in [1, k]$ such that

$$\mathbf{Pr}[i = j] = \text{AGM}_{W_j}(Q_j)/\text{AGMSUM}$$

holds for each $j \in [1, k]$. The generation takes $\tilde{O}(1)$ time, thanks to Proposition 1. Then, the algorithm instructs Υ_i to execute the `sample` algorithm (Figure 3) *only once* in $\tilde{O}(1)$ time. If `sample` declares "failure", `union-sample` does the same. Otherwise, `sample` has obtained a tuple $\boldsymbol{u} \in \mathcal{J}oin(Q_i)$. Now, check in constant time whether $\mathcal{J}oin(Q_i)$ is the owner of $\boldsymbol{u}$. If so, `union-sample` outputs $\boldsymbol{u}$ as a sample of $\bigcup_{i=1}^k \mathcal{J}oin(Q_i)$; otherwise, it declares "failure".

To analyze the algorithm, consider any tuple $\boldsymbol{u} \in \bigcup_{i=1}^k \mathcal{J}oin(Q_i)$. Assume, w.l.o.g., that Q_{i^*} is the owner of $\boldsymbol{u}$ for some $i^* \in [1, k]$. As `union-sample` can return $\boldsymbol{u}$ only when the random variable i selected in the beginning equals i^*, the probability of outputting $\boldsymbol{u}$ equals

$$\frac{\text{AGM}_{W_{i^*}}(Q_{i^*})}{\text{AGMSUM}} \cdot \mathbf{Pr}[\text{the } \texttt{sample} \text{ algorithm of } \Upsilon_{i^*} \text{ samples } \boldsymbol{u}]. \quad (12)$$

As explained in Section 4.2, Υ_{i^*} samples $\boldsymbol{u}$ with probability $\frac{1}{\text{AGM}_{W_{i^*}}(Q_{i^*})}$. Therefore, (12) can be simplified into

$$\frac{\text{AGM}_{W_{i^*}}(Q_{i^*})}{\text{AGMSUM}} \cdot \frac{1}{\text{AGM}_{W_{i^*}}(Q_{i^*})} = \frac{1}{\text{AGMSUM}}.$$

We thus conclude that `union-sample` returns a uniformly random tuple of $\bigcup_{i=1}^k \mathcal{J}oin(Q_i)$ with probability OUT/AGMSUM, and declares "failure" with probability $1 - \text{OUT}/\text{AGMSUM}$.