Split Theorems for Join Reporting and Sampling

Yufei Tao
Chinese University of Hong Kong
Hong Kong, China
taoyf@cse.cuhk.edu.hk

Ru Wang
Chinese University of Hong Kong
Hong Kong, China
rwang21@cse.cuhk.edu.hk

ABSTRACT

The *box-tree technique* is an elegant and powerful tool for natural join processing. Recently developed by the database theory community, it yields simple algorithms and data structures for solving three fundamental problems — *join reporting*, *small-delay enumeration*, and *join sampling* — with performance matching the best known bounds, up to polylogarithmic factors. This article presents the technique's theoretical foundation in a form accessible to the broader database community. At the core of this foundation are the so-called *split theorems*, which uncover a combinatorial and intrinsically geometric structure underlying natural joins. Two versions of these theorems are proved using elementary, self-contained arguments.

1. INTRODUCTION

This article studies *natural joins*. Consider $t \geq 2$ relations $R_1, R_2, ..., R_t$. Conceptually, a natural join first computes the Cartesian product $R_1 \times R_2 \times ... \times R_t$ and then selects from this product only the tuples satisfying the SQL-style condition "$R_i.A = R_j.A$" for every $1 \leq i < j \leq t$ and every common attribute A of R_i and R_j. This intuitive yet informal definition will suffice for the discussion until Section 2, where a formal definition of the natural join will be introduced.

Natural joins are fundamental to database systems for two main reasons. First, they are indispensable for answering multi-relation queries and serve as the backbone of most query execution plans. Second, they are computationally expensive operations that often dominate the cost of query processing. The design and analysis of join algorithms have long been a central topic in database theory. A solid understanding of their theoretical foundation provides valuable insights that guide the development of high-performance systems.

Join sampling is the operation that draws uniformly at random a tuple from the result of a natural join. The sample must be independent from those returned by all previous join sampling operations — this requirement prevents the "cheating" solution that keeps only one sample element from the join result and returns it for every single operation.

Recent years have witnessed considerable research interest in join sampling, a trend that can be attributed to at least three key factors. First, with the rapid growth in database volumes, the result size of a natural join increases combinatorially. This fundamentally changes the way users interact with databases: it is no longer practical to materialize or display the entire join result. Instead, users are increasingly turning to sampling-based methods to explore large query outputs. Second, many analytical tasks — especially those of a statistical nature — can actually be carried out effectively on samples rather than full join results. Third, join sampling can usually be performed significantly faster than producing the complete join result.

A main objective of this article is to present an elegant *box-tree technique* for join sampling that has been developed by the database theory community in recent years. The technique is simple enough to be taught in a graduate-level course, yet powerful enough to yield sampling algorithms and data structures with theoretical guarantees that match the state of the art up to polylogarithmic factors. The technique finds its root in two *split theorems*, which reveal a beautiful mathematical structure — specifically, a *geometric* structure — underlying natural joins. We will provide self-contained proofs of both theorems using elementary arguments.

The remainder of this article is organized as follows. Section 2 lays the foundation for our discussion by formalizing natural joins, degree constraints, and the problems of join reporting and sampling. Section 3 introduces the polymatroid bound, a key parameter for characterizing the performance of join reporting and sampling algorithms. Section 4 presents the article's main protagonist — the split theorems — and elaborates on their

derivation. Section 5 demonstrates how these theorems can be applied to derive algorithms that perform join sampling and reporting with strong efficiency guarantees. Section 6 surveys related work and discusses alternative approaches to join reporting and sampling. Finally, Section 7 concludes with a summary and some remarks for future research.

2. FORMALIZATION

Math Conventions. The symbol $\mathbb{N}$ represents the set of integers. For an integer $x \geq 1$, the notation $[x]$ denotes the set $\{1, 2, ..., x\}$; as a special case, $[0]$ gives the empty set. Every logarithm $\log(\cdot)$ has base 2.

Tuples and Relations. Let $\mathbf{att}$ be a finite set, where each element is called an *attribute*. For a non-empty set $\mathcal{X} \subseteq \mathbf{att}$ of attributes, a *tuple* over $\mathcal{X}$ is a function $\boldsymbol{u} : \mathcal{X} \to \mathbb{N}$. For any non-empty subset $\mathcal{Y} \subseteq \mathcal{X}$, we define $\boldsymbol{u}[\mathcal{Y}]$ — called the *projection* of $\boldsymbol{u}$ on $\mathcal{Y}$ — as the tuple $\boldsymbol{v}$ over $\mathcal{Y}$ satisfying $\boldsymbol{v}(Y) = \boldsymbol{u}(Y)$ for every attribute $Y \in \mathcal{Y}$. A *relation* R is a set of tuples over the same set $\mathcal{Z}$ of attributes; we refer to $\mathcal{Z}$ as the *schema* of R and represent it as $schema(R)$.

Joins. We define a *join* as a set $\mathcal{Q}$ of relations with distinct schemas. Let $schema(\mathcal{Q})$ be the union of the attributes of the relations in $\mathcal{Q}$, i.e., $schema(\mathcal{Q}) = \bigcup_{R \in \mathcal{Q}} schema(R)$. The result of $\mathcal{Q}$ is a relation over $schema(\mathcal{Q})$ formalized as:

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

Degrees. Let R be a relation. For subsets $\mathcal{X}$ and $\mathcal{Y}$ of $schema(R)$ satisfying $\mathcal{X} \subset \mathcal{Y}$ — note: $\mathcal{X}$ is a *proper* subset of $\mathcal{Y}$ — define the $(\mathcal{Y}|\mathcal{X})$-*degree of* R as

$$\deg_{\mathcal{Y}|\mathcal{X}}(R) = \max_{\text{tuple } \boldsymbol{u} \text{ over } \mathcal{X}} \left| \left\{ \boldsymbol{v}[\mathcal{Y}] \mid \boldsymbol{v} \in R, \boldsymbol{v}[\mathcal{X}] = \boldsymbol{u} \right\} \right| \quad (2)$$

For an intuitive explanation, imagine grouping the tuples of R by $\mathcal{X}$ and counting, for each group, how many *distinct* $\mathcal{Y}$-projections therein. Then, the $(\mathcal{Y}|\mathcal{X})$-degree of R equals the maximum count of all groups. Note that, when $\mathcal{X} = \emptyset$, then $\deg_{\mathcal{Y}|\mathcal{X}}(R)$ is simply $|\Pi_{\mathcal{Y}}(R)|$ where Π is the "projection" operator in relational algebra. If in addition $\mathcal{Y} = schema(R)$, then $\deg_{\mathcal{Y}|\mathcal{X}}(R) = |R|$.

Degree Constraints. We define a *degree constraint* as a triple $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$ where $\mathcal{X}$ and $\mathcal{Y}$ are subsets of $\mathbf{att}$ satisfying $\mathcal{X} \subset \mathcal{Y}$, and $N_{\mathcal{Y}|\mathcal{X}} \geq 1$ is an integer. A relation R is said to *guard* the constraint if

$$\mathcal{Y} \subseteq schema(R), \text{ and } \deg_{\mathcal{Y}|\mathcal{X}}(R) \leq N_{\mathcal{Y}|\mathcal{X}}.$$

Let DegC be a set of degree constraints. It defines a *constraint dependency graph* $\mathcal{G}$ as follows:

- The vertex set of $\mathcal{G}$ consists of every attribute $Y \in \mathbf{att}$ such that $Y \in \mathcal{Y}$ for at least one constraint $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in$ DegC.
- For each degree constraint $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in$ DegC with $\mathcal{X} \neq \emptyset$, the graph $\mathcal{G}$ has a directed edge from X to Y for every pair $(X, Y) \in \mathcal{X} \times (\mathcal{Y} \setminus \mathcal{X})$.

If $\mathcal{G}$ is an acyclic graph, DegC is said to be *acyclic*; otherwise, DegC is *cyclic*.

The set DegC is said to be *clean* if it does not have two constraints $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$ and $(\mathcal{X}', \mathcal{Y}', N_{\mathcal{Y}'|\mathcal{X}'})$ with $\mathcal{X} = \mathcal{X}'$ and $\mathcal{Y} = \mathcal{Y}'$. Intuitively, when two such constraints exist — w.l.o.g., assume $N_{\mathcal{Y}|\mathcal{X}} \leq N_{\mathcal{Y}'|\mathcal{X}'}$ — the constraint $(\mathcal{X}', \mathcal{Y}', N_{\mathcal{Y}'|\mathcal{X}'})$ is redundant and can be removed from DegC.

A join $\mathcal{Q}$ is said to *conform to* DegC — written as $\mathcal{Q} \models$ DegC — if the two following conditions hold:

- For each relation $R \in \mathcal{Q}$, the set DegC contains a constraint $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$ where $\mathcal{X} = \emptyset$, $\mathcal{Y} = schema(R)$, and $N_{\mathcal{Y}|\mathcal{X}} = |R|$. This constraint is referred to as a *cardinality constraint*.
- Every constraint in DegC is guarded by at least one relation in $\mathcal{Q}$.

As an important special case, when DegC consists purely of cardinality constraints, $\mathcal{G}$ has no edges and, therefore, is trivially acyclic. In general, one should avoid the misconception that "an acyclic $\mathcal{G}$ implies $\mathcal{Q}$ being an acyclic join". These two notions of acyclicity are unrelated. Readers unfamiliar with the concept of an acyclic join may refer to [1, Chapter 6.4].

Join Reporting and Sampling. Given (i) a join $\mathcal{Q}$ whose $schema(\mathcal{Q})$ has a constant size (this article focuses on "data complexity") and (ii) a clean, acyclic set DegC of degree constraints such that $\mathcal{Q} \models$ DegC, we will discuss three representative problems:

- *Join reporting.* Compute the join result $join(\mathcal{Q})$, as defined in (1).
- *Small-delay enumeration.* Given a positive integer Δ, preprocess $\mathcal{Q}$ into a data structure that enumerates $join(\mathcal{Q})$ with a *delay* of Δ. Specifically, within $O(\Delta)$ time, the enumeration algorithm must report a tuple in $join(\mathcal{Q})$ or declare $join(\mathcal{Q}) = \emptyset$. Thereafter, within each subsequent $O(\Delta)$ time period, the algorithm must either report a new tuple in $join(\mathcal{Q})$ or declare the end of enumeration.
- *Join sampling.* Preprocess $\mathcal{Q}$ into a data structure that supports sampling a tuple uniformly at random from $join(\mathcal{Q})$. Repeated invocations of the

sampling operation should produce mutually independent samples.

These problems have been studied extensively and often in isolation (see Section 6). As shown in Section 5, they actually share a common geometric structure.

We reserve the symbols IN and OUT for $\sum_{R\in\mathcal{Q}}|R|$ and $|join(\mathcal{Q})|$, respectively, which we refer to as the *input size* and *output size* of $\mathcal{Q}$.

3. POLYMATROID BOUNDS

Fix an arbitrary clean, acyclic set DegC of degree constraints. Define

$$\mathcal{V} = \bigcup_{(\mathcal{X},\mathcal{Y},N_{\mathcal{Y}|\mathcal{X}})\in\mathsf{DegC}} \mathcal{Y}; \tag{3}$$

namely, $\mathcal{V}$ is the vertex set in the constraint dependency graph defined by DegC.

We formulate the *XY-space* of DegC as

$$\text{space}(\mathsf{DegC}) = \{(\mathcal{X},\mathcal{Y}) \mid (\mathcal{X},\mathcal{Y},N_{\mathcal{Y}|\mathcal{X}}) \in \mathsf{DegC}\}.$$

Henceforth, the symbol π will be used to reference a pair $(\mathcal{X},\mathcal{Y})$ in $\text{space}(\mathsf{DegC})$. Accordingly, we use

- N_π to represent the value of $N_{\mathcal{Y}|\mathcal{X}}$ in the corresponding constraint $(\mathcal{X},\mathcal{Y},N_{\mathcal{Y}|\mathcal{X}}) \in \mathsf{DegC}$;
- (π, N_π) to represent the constraint $(\mathcal{X},\mathcal{Y},N_{\mathcal{Y}|\mathcal{X}})$;
- $diffatt(\pi)$ to represent the set $\mathcal{Y}\setminus\mathcal{X}$.

The introduction of XY-space permits us to define vectors over it. Specifically, every such vector W has a component W_π, which is a real value, for each pair $\pi \in \text{space}(\mathsf{DegC})$. The vector W is said to be a *fractional XY-pair cover* of DegC if it satisfies two conditions:

- $0 \le W_\pi \le 1$ for every $\pi \in \text{space}(\mathsf{DegC})$.
- $\sum_{\substack{\pi\in\text{space}(\mathsf{DegC})\\ \text{s.t. } A\in diffatt(\pi)}} W_\pi \ge 1$ for every $A \in \mathcal{V}$.

Every fractional XY-pair cover W is associated with a *polymatroid value* defined as

$$polymat_W = \prod_{\pi\in\text{space}(\mathsf{DegC})} N_\pi^{W_\pi}. \tag{4}$$

We now introduce a crucial lemma that relates polymatroid values to joins.

LEMMA 1 ([18, 21]). *Let $\mathcal{Q}$ be an arbitrary join conforming to DegC. Given any fractional XY-pair cover W of DegC, it holds that $|join(\mathcal{Q})| \le polymat_W$.*

An *optimal fractional XY-pair cover* of DegC is a fractional XY-pair cover W^* minimizing (4). Lemma 1 assures us that if $\mathcal{Q} \models \mathsf{DegC}$, then the output size of $\mathcal{Q}$ is at most the polymatroid value determined by W^*, which we refer to as the *polymatroid bound* of $\mathcal{Q}$.

Polymatroid bounds are asymptotically tight in the worst case. To explain, let us call a clean, acyclic set of degree constraints *realizable* if there exists at least one join that conforms to it. For any realizable set DegC of degree constraints, it is possible [27] to construct a join $\mathcal{Q}$ satisfying the conditions that $\mathcal{Q} \models \mathsf{DegC}$ and $|join(\mathcal{Q})| = \Omega(polymat_{W^*})$, where W^* is an optimal fractional XY-pair cover of DegC.

Relevance to the AGM Bound. Before proceeding, we draw relevance from the polymatroid bound to the "AGM bound" [4], which is well known in the database community nowadays. As will be clear shortly, the concepts introduced earlier in this section generalize those underlying the AGM bound.

Given a join $\mathcal{Q}$, we use $\mathcal{H}$ to denote the hypergraph $(\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H})$ where

$$\begin{aligned} \mathcal{V}_\mathcal{H} &= schema(\mathcal{Q}) \\ \mathcal{E}_\mathcal{H} &= \{schema(R) \mid R \in \mathcal{Q}\}. \end{aligned}$$

For each hyperedge $e \in \mathcal{E}_\mathcal{H}$, let R_e represent the (only) relation $R \in \mathcal{Q}$ with $e = schema(R)$.

Next, we regard $E_\mathcal{H}$ as a "space" and define vectors over it. Every such vector $\mathscr{W}$ has a component $\mathscr{W}_e$, which is a real value, for each hyperedge $e \in \mathcal{E}_\mathcal{H}$. The vector $\mathscr{W}$ is said to be a *fractional edge cover* of $\mathcal{H}$ if it satisfies two conditions:

- $0 \le \mathscr{W}_e \le 1$ for every $e \in \mathcal{E}_\mathcal{H}$.
- $\sum_{e\in\mathcal{E}_\mathcal{H}: A\in e} \mathscr{W}_e \ge 1$ for every $A \in \mathcal{V}_\mathcal{H}$.

Every fractional edge cover $\mathscr{W}$ is associated with an *AGM value* defined as

$$AGM_\mathscr{W} = \prod_{e\in\mathcal{E}_\mathcal{H}} |R_e|^{\mathscr{W}_e}. \tag{5}$$

As proved in [4], the inequality $|join(\mathcal{Q})| \le AGM_\mathscr{W}$ holds for any fractional edge cover $\mathscr{W}$ of $\mathcal{H}$. An *optimal fractional edge cover* of $\mathcal{H}$ is a fractional edge cover $\mathscr{W}^*$ minimizing (5). The *AGM bound* of $\mathcal{Q}$ is defined as the AGM value of $\mathscr{W}^*$.

The polymatroid bound of $\mathcal{Q}$ specializes to its AGM bound when DegC comprises *only* the cardinality constraints from $\mathcal{Q}$, that is, $\mathsf{DegC} = \{(\emptyset, schema(R), |R|) \mid R \in \mathcal{Q}\}$. In this case, the XY-space $\text{space}(\mathsf{DegC})$ — which is $\{(\emptyset, schema(R)) \mid R \in \mathcal{Q}\}$ — has a one-one correspondence to $\mathcal{E}_\mathcal{H}$. Accordingly, every vector

W over $\text{space}(\mathsf{DegC})$ corresponds to a distinct vector $\mathscr{W}$ over $\mathcal{E}_{\mathcal{H}}$ and vice versa: for each $R \in \mathcal{Q}$, the two vectors satisfy $W_\pi = \mathscr{W}_e$ where $\pi = (\emptyset, schema(R))$ and $e = schema(R)$. Hence, W is a fractional XY-pair cover of DegC if and only if the corresponding $\mathscr{W}$ is a fractional edge cover of $\mathcal{E}_{\mathcal{H}}$. It thus follows from a comparison of (4) and (5) that the polymatroid bound of $\mathcal{Q}$ is simply its AGM bound.

In general, when DegC contains more than just cardinality constraints, the polymatroid bound of a join $\mathcal{Q} \models \mathsf{DegC}$ is never higher than its AGM bound.[1] On the other hand, it is possible [18, 27] to construct joins where the polymatroid bound is significantly less than the AGM bound.

4. SPLIT THEOREMS

In this section, we fix a join $\mathcal{Q}$, and a clean, acyclic set DegC of degree constraints such that $\mathcal{Q} \models \mathsf{DegC}$. Denote by $\mathcal{G}$ the constraint dependency graph of DegC, and by $\mathcal{V}$ the vertex set of $\mathcal{G}$ (see (3)); note that $\mathcal{V} = schema(\mathcal{Q})$. Set $d = |\mathcal{V}|$, and let the attribute sequence

$$A_1, A_2, ..., A_d \tag{6}$$

be an (arbitrary) topological order of $\mathcal{G}$. This is an important order that we will rely on in our discussion. Such an order definitely exists because $\mathcal{G}$ is acyclic. It is worth emphasizing that the order includes every attribute in $schema(\mathcal{Q})$.

Recall that, for each $\pi = (\mathcal{X}, \mathcal{Y})$ in $\text{space}(\mathsf{DegC})$, the notation (π, N_π) represents the constraint $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$. We denote by R_π the relation in $\mathcal{Q}$ guarding this constraint. Such a relation must exist because $\mathcal{Q} \models \mathsf{DegC}$. In case π is guarded by multiple relations in $\mathcal{Q}$, any one of them may serve as R_π.

4.1 "Boxing" Everything

We use the term *box* to refer to a rectangle B in $\mathbb{N}^d$ of the form $[x_1, y_1] \times [x_2, y_2] \times ... \times [x_d, y_d]$. For any $i \in [d]$, let $B(A_i)$ represent the interval $[x_i, y_i]$, i.e., the projection of B onto the i-th dimension. Particularly, we say that $B(A_i)$ is a *singleton interval* if $x_i = y_i$. If $B(A_i)$ is a singleton interval on all $i \in [d]$, we call B a *singleton box*.

For each relation $R \in \mathcal{Q}$, we define

$$R(B) = \{\text{tuple } \boldsymbol{u} \in R \mid \forall A_i \in schema(R) : \boldsymbol{u}(A_i) \in B(A_i)\}. \tag{7}$$

[1] Given an optimal fractional edge cover $\mathscr{W}^*$ of $\mathcal{H}$, one can always construct a fractional XY-pair cover W of DegC with $polymat_W = AGM_{\mathscr{W}^*}$. Hence, $polymat_{W^*} \leq polymat_W = AGM_{\mathscr{W}^*}$, where W^* is an optimal fractional XY-pair cover of DegC.

Intuitively, $R(B)$ is the subset of tuples in R that are "selected" by B. Accordingly, define

$$\mathcal{Q}(B) = \{R(B) \mid R \in \mathcal{Q}\}. \tag{8}$$

Note that $\mathcal{Q}(B)$ is a join.

Next, we build a set of degree constraints — denoted as $\mathsf{DegC}(B)$ — dedicated to $\mathcal{Q}(B)$. Consider an arbitrary constraint $(\pi, N_\pi) \in \mathsf{DegC}$ where $\pi = (\mathcal{X}, \mathcal{Y})$. Define

$$N_\pi(B) = \deg_{\mathcal{Y}|\mathcal{X}}(R_\pi(B)) \tag{9}$$

where — let us recall — R_π is the relation in $\mathcal{Q}$ guarding (π, N_π). We can now express $\mathsf{DegC}(B)$ formally as:

$$\mathsf{DegC}(B) = \{(\pi, N_\pi(B)) \mid (\pi, N_\pi) \in \mathsf{DegC}\}.$$

The facts below are easy to verify:

- $\mathsf{DegC}(B)$ and DegC share the same XY-space.
- Any fractional XY-pair cover of DegC is also a fractional XY-pair cover of $\mathsf{DegC}(B)$.
- $\mathcal{Q}(B)$ conforms to $\mathsf{DegC}(B)$.

Take any fractional XY-pair cover W of DegC (which must also be a fractional XY-pair cover of $\mathsf{DegC}(B)$). The *polymatroid value of W on B* is defined as

$$polymat_W(B) = \prod_{\pi \in \text{space}(\mathsf{DegC}(B))} (N_\pi(B))^{W_\pi}$$

which can be rewritten as

$$polymat_W(B) = \prod_{\pi \in \text{space}(\mathsf{DegC})} (N_\pi(B))^{W_\pi} \tag{10}$$

because $\text{space}(\mathsf{DegC}) = \text{space}(\mathsf{DegC}(B))$. By Lemma 1, we must have $|join(\mathcal{Q}(B))| \leq polymat_W(B)$.

4.2 The Main Theorems

Given a non-singleton box B, we define its *split dimension* as the smallest $\sigma \in [d]$ such that $B(A_\sigma)$ is not a singleton interval. A set of boxes $B_1, B_2, ..., B_s$ (where $s \geq 2$) is said to *refine* B if

- $B_1, B_2, ..., B_s$ are mutually disjoint, and their union is B;
- for every $i \in [d] \setminus \{\sigma\}$, it holds that $B_1(A_i) = B_2(A_i) = ...B_s(A_i) = B(A_i)$.

We now present the first split theorem:

THEOREM 2. (SIMPLE POLYMATROID SPLIT THEOREM) *Consider a clean, acyclic* DegC *of degree constraints and a join $\mathcal{Q}$ conforming to* DegC. *Denote by W an arbitrary fractional XY-pair cover of* DegC. *Let B be a non-singleton box; and let B_1, B_2*

be any two boxes refining B. We must have

$polymat_W(B_1) + polymat_W(B_2) \leq polymat_W(B)$.

As shown in Section 5, the above theorem is already sufficient to obtain join reporting and sampling algorithms with excellent guarantees. Nevertheless, it falls short of revealing the combinatorial structure of the join problem. In particular, the theorem does not ensure that $polymat_W(B_1)$ and $polymat_W(B_2)$ are both substantially lower than $polymat_W(B)$. The next theorem remedies this limitation.

THEOREM 3. (POLYMATROID SPLIT THEOREM) *Consider a clean, acyclic* DegC *of degree constraints and a join $\mathcal{Q}$ conforming to* DegC*. Denote by W an arbitrary fractional XY-pair cover of* DegC*. Let B be a box with $polymat_W(B) \geq 2$. There is a set S of at most $2d+1$ boxes — where $d = |schema(\mathcal{Q})|$ — such that*

(1) the boxes in S are mutually disjoint, and their union is B;

(2) $polymat_W(B') \leq \frac{1}{2} \cdot polymat_W(B)$ holds for every box $B' \in S$;

(3) $\sum_{B' \in S} polymat_W(B') \leq polymat_W(B)$.

When DegC contains only cardinality constraints (i.e., constraints $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$ with $\mathcal{X} = \emptyset$), Theorem 3 reduces to the *AGM split theorem* proved by Deng, Lu, and Tao [14].

4.3 Proving the Theorems

This subsection serves as a proof of Theorem 3; the argument also establishes Theorem 2 as a by-product. The material presented here is not required to follow the subsequent sections.

Let us start with an arbitrary non-singleton box B with split dimension σ. Let $\{B_1, B_2, ..., B_s\}$ be a set of $s \geq 2$ boxes refining B. We will prove:

LEMMA 4. *For any fractional XY-pair cover W of* DegC*, it holds that*

$$\sum_{t=1}^{s} polymat_W(B_t) \leq polymat_W(B).$$

PROOF. Let us first review Friedgut's inequality [15]. Fix arbitrary integers p and q at least 1. Take a set of non-negative real values $\{a_{i,j} \mid i \in [p], j \in [q]\}$. In addition, take another set of non-negative real values $\{b_i \mid i \in [q]\}$ satisfying $\sum_{i=1}^{q} b_i \geq 1$. Assuming $0^0 = 0$, Friedgut's inequality states that

$$\sum_{i=1}^{p} \prod_{j=1}^{q} a_{i,j}^{b_j} \leq \prod_{j=1}^{q} \left(\sum_{i=1}^{p} a_{i,j} \right)^{b_j}. \quad (11)$$

Denote by σ the split dimension of B. Define

$$P = \{\pi \in \text{space}(\mathsf{DegC}) \mid A_\sigma \in \mathit{diffatt}(\pi)\}. \quad (12)$$

Recall that each π denotes a pair $(\mathcal{X}, \mathcal{Y})$ and that $\mathit{diffatt}(\pi) = \mathcal{Y} \setminus \mathcal{X}$. Thus, $A_\sigma \in \mathit{diffatt}(\pi)$ tells us that every attribute — say A_i — in $\mathcal{X}$ must have an outedge to A_σ in $\mathcal{G}$ (the constraint dependency graph of DegC). This further implies $i < \sigma$ because the sequence $A_1, A_2, ..., A_d$ is a topological order of $\mathcal{G}$.

For each $t \in [s]$, we have from (10)

$$\begin{aligned} & polymat_W(B_t) \\ = & \prod_{\pi \in \text{space}(\mathsf{DegC})} (N_\pi(B_t))^{W_\pi} \\ = & \prod_{\pi \in \text{space}(\mathsf{DegC}) \setminus P} (N_\pi(B_t))^{W_\pi} \cdot \prod_{\pi \in P} (N_\pi(B_t))^{W_\pi} \\ \leq & \prod_{\pi \in \text{space}(\mathsf{DegC}) \setminus P} (N_\pi(B))^{W_\pi} \cdot \prod_{\pi \in P} (N_\pi(B_t))^{W_\pi} \quad (13) \end{aligned}$$

where the last step used the fact $N_\pi(B_t) \leq N_\pi(B)$ for all $\pi \in \text{space}(\mathsf{DegC})$, which holds true because $B_t \subset B$ (the reader may want to revisit the definition in (9)).

Before proceeding, we need two observations. First, as W is a fractional XY-pair cover of DegC, the definition of P in (12) tells us

$$\sum_{\pi \in P} W_\pi \geq 1. \quad (14)$$

Second, for every $\pi \in P$, we must have

$$\sum_{t=1}^{s} N_\pi(B_t) = N_\pi(B). \quad (15)$$

To explain why, let us assume $\pi = (\mathcal{X}, \mathcal{Y})$. As argued earlier, every attribute in $\mathcal{X}$ precedes A_σ in the topological order $A_1, A_2, ..., A_d$ of $\mathcal{G}$. This, together with σ being the split dimension of B, indicates that $B(A)$ must be a singleton interval on *every* attribute $A \in \mathcal{X}$. As the set $\{B_1, B_2, ..., B_s\}$ refines B, we can assert that $B_t(A)$ is a singleton interval for all $A \in \mathcal{X}$ and $t \in [s]$. Consider now the relation $R_\pi \in \mathcal{Q}$ that guards π. All the tuples in $R_\pi(B)$ must have exactly the same projection on $\mathcal{X}$. Hence, the $(\mathcal{Y}|\mathcal{X})$-degree of $R_\pi(B)$ — which is simply $N_\pi(B)$ — is the number of distinct projections on $\mathcal{Y}$ of the tuples in $R_\pi(B)$. Similarly, for each $t \in [s]$, the $(\mathcal{Y}|\mathcal{X})$-degree of $R_\pi(B_t)$ — which is simply $N_\pi(B_t)$ — is the number of distinct projections on $\mathcal{Y}$ of the tuples in $R_\pi(B_t)$. As the attribute A_σ is in $\mathcal{Y}$, the

set of projections on $\mathcal{Y}$ made by the tuples in $R_\pi(B_t)$ is disjoint from the set made by the tuples in $R_\pi(B_{t'})$, for any $1 \leq t < t' \leq s$. This proves the correctness of (15).

Equipped with these observations, we can derive

$$\begin{aligned}
&\sum_{t=1}^{s} polymat_W(B_t) \\
\leq\ & \prod_{\pi \in \text{space}(\mathsf{DegC}) \setminus P} (N_\pi(B))^{W_\pi} \cdot \sum_{t=1}^{s} \prod_{\pi \in P} (N_\pi(B_t))^{W_\pi} \\
\leq\ & \prod_{\pi \in \text{space}(\mathsf{DegC}) \setminus P} (N_\pi(B))^{W_\pi} \cdot \prod_{\pi \in P} \sum_{t=1}^{s} (N_\pi(B_t))^{W_\pi} \\
=\ & \prod_{\pi \in \text{space}(\mathsf{DegC}) \setminus P} (N_\pi(B))^{W_\pi} \cdot \prod_{\pi \in P} (N_\pi(B))^{W_\pi} \\
=\ & polymat_W(B)
\end{aligned}$$

where the first inequality applied (13), the second applied Friedgut's inequality (12) — the application is made possible by (14) — and the third applied (15). □

Theorem 2 is an immediate corollary of Lemma 4. More effort is needed to prove Theorem 3. The rest of the argument is built on ideas used in [14] to establish the AGM split theorem.

Let us introduce a function $\texttt{replace}(B, i, I)$, where B is a box, i is an integer in $[d]$, and I is an interval of integers. The function yields the box

$$B(A_1) \times ... \times B(A_{i-1}) \times I \times B(A_{i+1}) \times ... \times B(A_d)$$

that is, replacing the projection of B on dimension i with I, while retaining its projections on the other dimensions.

Figure 1 presents an algorithm for splitting a box B into a set S of smaller boxes meeting the requirements of Theorem 3. The procedure, $\texttt{split}(i, B)$, admits two parameters: an integer $i \in [d]$ and a box $B = [x_1, y_1] \times ... \times [x_d, y_d]$. The box must satisfy the condition that its projections on the first $i-1$ attributes should be singleton intervals. Note that the condition is always met when i is set to 1, regardless of B. Indeed, to obtain the aforementioned set S, one should invoke the procedure with $\texttt{split}(1, B)$.

The procedure $\texttt{split}(i, B)$ examines a type of three-way splits along dimension i. Specifically, given an integer $z \in [x_i, y_i]$, we can split B into three boxes:

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

Note that B_{left} (resp., B_{right}) does not exist if $z = x_i$ (resp., y_i). The procedure identifies the largest z satisfying $polymat_W(B_{left}) \leq \frac{1}{2} \cdot polymat_W(B)$ (Line 2). Such a z must exist since the condition holds for $z = x_i$; in that case, B_{left} is empty and hence $polymat_W(B_{left}) = 0$. The boxes B_{left} and B_{right}, if they exist, are added to S directly (Line 3 and 7). Let us now turn our attention to B_{mid}. Its projections on the first i dimensions are singleton intervals. If $i = d$, then B_{mid} has shrunk into a singleton box, in which case it is also added to S (Line 5). Otherwise, we recursively invoke $\texttt{split}(i+1, B_{mid})$ to split B_{mid} into a set S' of boxes and then merge S' into S (Line 6).

algorithm $\texttt{split}(i, B)$
/* 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. $S \leftarrow \emptyset$
2. $z \leftarrow$ the largest value in $[x_i, y_i]$ s.t. $polymat_W(B_{left}) \leq \frac{1}{2} \cdot polymat_W(B)$ where $B_{left} = \texttt{replace}(B, i, [x_i, z-1])$
3. **if** $z - 1 \geq x_i$ **then** $S \leftarrow S \cup \{B_{left}\}$
4. $B_{mid} \leftarrow \texttt{replace}(B, i, [z, z])$
5. **if** $i = d$ **then** add B_{mid} to S
6. **else** $S \leftarrow S \cup \texttt{split}(i+1, B_{mid})$
7. **if** $z + 1 \leq y_i$ **then** $S \leftarrow S \cup \{B_{right}\}$ where $B_{right} \leftarrow \texttt{replace}(B, i, [z+1, y_i])$
8. **return** S

Figure 1: A split algorithm for Theorem 3

Next, we show that the set S produced by $\texttt{split}(1, B)$ has the three properties in Theorem 3. Property (1) is obvious and omitted.

To prove Property (2), we analyze each type of boxes added to S:

- For every B_{left} created by a recursive call $\texttt{split}(i, B')$ — here, B' is some box inside B generated in the recursion — we have $polymat_W(B_{left}) \leq \frac{1}{2} \cdot polymat_W(B') \leq \frac{1}{2} \cdot polymat_W(B)$. Hence, every B_{left} satisfies Property (2).

- Consider the B_{right} created by $\texttt{split}(i, B')$. By definition of the value z at Line 2, it must hold that $polymat_W(B_{left} \cup B_{mid}) \geq \frac{1}{2} \cdot polymat_W(B')$. By Lemma 4, $polymat_W(B_{left} \cup B_{mid}) + polymat_W(B_{right}) \leq polymat_W(B')$. This implies $polymat_W(B_{right}) \leq \frac{1}{2} \cdot polymat_W(B') \leq \frac{1}{2} \cdot polymat_W(B)$. Hence, every B_{right} satisfies Property (2).

- In all the recursive calls to $\texttt{split}$, the only B_{mid} added to S is the one that has shrunk into a singleton box (Line 5). For that B_{mid}, we

have $polymat_W(B_{mid}) \leq 1$, which is at most $\frac{1}{2} \cdot polymat_W(B)$ because $polymat_W(B) \geq 2$.

Property (3) is a corollary of Lemma 4. At any recursive call $\texttt{split}(i, B')$, the lemma guarantees $polymat_W(B_{left}) + polymat_W(B_{mid}) + polymat_W(B_{right}) \leq polymat_W(B')$. Now, the property follows from a simple inductive argument on i, essentially breaking $polymat_W(B_{mid})$ into the sum of the polymatroid bounds of W on smaller boxes.

The set S returned by $\texttt{split}(1, B)$ has a size no more than $2d + 1$ because we add at most two boxes to S for each $i \in [d-1]$ and at most three for $i = d$. This completes the proof of Theorem 3.

5. THE BOX-TREE TECHNIQUE

This section will explain how to use the split theorems to perform join reporting and sampling. We will strive to use only Theorem 2, which is surprisingly powerful — especially on static data (i.e., no updates) — despite its primitive appearance. In Section 5.4, we include a discussion of when the additional power offered by Theorem 3 becomes useful.

Throughout the section, we fix

- a join $\mathcal{Q}$ and a clean, acyclic set DegC of degree constraints such that $\mathcal{Q} \models$ DegC;
- $A_1, A_2, ..., A_d$ defined in (6) — recall that $d = |schema(\mathcal{Q})|$;
- an optimal fractional XY-pair cover W^* of DegC.

Note that $polymat_{W^*}$ is the polymatroid bound of $\mathcal{Q}$.

We will make two assumptions until Section 5.4.

- **A1:** The input size IN of $\mathcal{Q}$ is a power of 2.
- **A2:** Attribute values lie in the so-called *rank space*. Specifically, for any relation $R \in \mathcal{Q}$, any attribute $A \in schema(R)$, and any tuple $\boldsymbol{u} \in R$, we consider that $\boldsymbol{u}(A)$ is an integer in $[\text{IN}]$.

These assumptions allow us to illustrate the core ideas while abstracting away certain implementation details. In fact, they are reasonable when the relations in $\mathcal{Q}$ are static, as explained in Section 5.4.

From now, it is essential to view the join result from a geometric perspective. Every tuple of $join(\mathcal{Q})$ can be seen as a point in the box $[\text{IN}]^d$ — subject to assumption **A2** — whose i-th coordinate ($i \in [d]$) is the tuple's value on A_i. Note, however, that the converse does not hold: not every point in $[\text{IN}]^d$ necessarily corresponds to a tuple of $join(\mathcal{Q})$. We refer to $[\text{IN}]^d$ as the *join space*.

5.1 The Box-Tree

Parameterized by an integer $\Delta \geq 1$, the *box tree* $\mathcal{T}$ is built by splitting the join space recursively. This is a binary tree whose nodes are each associated with a distinct box. Henceforth, we will denote a node using its associated box. For a node B, we define its *polymatroid value* as $polymat_{W^*}(B)$; see (10) for its calculation. The polymatroid value of a node is at least the sum of the polymatroid values of its children.

Specifically, the root of $\mathcal{T}$ is associated with the join space. In general, consider a node associated with box B. If node B has a polymatroid value at most Δ, it becomes a leaf of $\mathcal{T}$. Otherwise, B must be a non-singleton box, which we split *evenly* along its split dimension into two boxes B_1 and B_2 (the reader may want to revisit the definition of the split dimension in Section 4.2). By Theorem 2, $polymat_{W^*}(B_1) + polymat_{W^*}(B_2) \leq polymat_{W^*}(B)$. We then create two child nodes of B, associated with B_1 and B_2, respectively. This completes the definition of the box tree. It is worth mentioning that even splits are always possible due to assumption **A1**.

The tree $\mathcal{T}$ has four properties:

- **P1**: Every internal node has a polymatroid value over Δ, while every leaf node has a polymatroid value at most Δ.
- **P2**: The leaf boxes are mutually disjoint and their union is $[\text{IN}]^d$.
- **P3**: $\mathcal{T}$ has height $O(\log \text{IN})$.
- **P4**: $\mathcal{T}$ has $O((polymat_{W^*}/\Delta) \cdot \log \text{IN})$ nodes.

Properties **P1** and **P2** follow immediately from the definition of $\mathcal{T}$. To understand **P3**, notice that the box of a child node covers half as many points in the join space as that of its parent node. Since the join space has IN^d points (assumption **A2**), it can be split at most $d \log \text{IN} = O(\log \text{IN})$ times before reducing to a singleton box (which must become a leaf because its polymatroid value is at most 1). Hence, the height of $\mathcal{T}$ is $O(\log \text{IN})$.

To prove **P4**, imagine removing all the leaf nodes of $\mathcal{T}$, and let $\mathcal{T}'$ denote the resulting tree (note that $\mathcal{T}'$ need not be binary). Every leaf of $\mathcal{T}'$ is an internal node in $\mathcal{T}$ and hence has a polymatroid value over Δ. The sum of the polymatroid values of all leaves of $\mathcal{T}'$ cannot exceed the polymatroid value of the root, which is precisely $polymat_{W^*}$. Hence, $\mathcal{T}'$ has less than $polymat_{W^*}/\Delta$ leaves, and thus at most $O((polymat_{W^*}/\Delta) \cdot \log \text{IN})$ nodes in total. This proves **P4**, since each node of $\mathcal{T}'$ can parent at most 2 leaves of $\mathcal{T}$.

Henceforth, we assume a *polymatroid oracle* that can calculate $polymat_{W^*}(B)$ for any box B. The implementation of such an oracle will be discussed in Section 5.4. The reader can safely assume that each call takes $O(\log \text{IN})$ time by resorting to a separate data structure of $O(\text{IN})$ space.

5.2 Join Reporting

Property **P2** of the box tree $\mathcal{T}$ implies:

$$\bigcup_{\text{leaf } B} join(\mathcal{Q}(B)) \quad = \quad join(\mathcal{Q}).$$

where $\mathcal{Q}(B)$ is defined in (8). This offers a simple way to compute $join(\mathcal{Q})$. Set $\Delta = 1$ and generate the entire $\mathcal{T}$, which (by property **P4**) requires $O(polymat_{W^*} \log \text{IN})$ CPU times plus $O(polymat_{W^*} \log \text{IN})$ calls to the polymatroid oracle. By property **P1**, each leaf B of $\mathcal{T}$ has a polymatroid value at most 1. This implies that, for each $R \in \mathcal{Q}$, the relation $R(B)$ as defined in (7) has at most a single tuple. Thus, $join(\mathcal{Q}(B))$ can be obtained in $O(1)$ time. This yields a join reporting algorithm with running time $O(polymat_{W^*} \log^2 \text{IN})$, which is worst-case optimal up to an $O(\log^2 \text{IN})$ factor.

Let us now attend to the small-delay enumeration problem (see Section 2), where the goal is to minimize the space of the data structure. To build the structure, first create a box tree $\mathcal{T}$ by setting the parameter Δ to the target delay. Collect the set $\mathcal{L}$ of leaves B whose $\mathcal{Q}(B)$ has a non-empty result. The data structure simply stores $\mathcal{L}$. The space consumption is $O((\text{polylog}_{W^*}/\Delta) \log \text{IN})$ by property **P4**. To enumerate $join(\mathcal{Q})$, for each box $B \in \mathcal{L}$ we compute $join(\mathcal{Q}(B))$ using a worst-case optimal join algorithm [21], which runs in $O(polymat_{W^*}(B)) = O(\Delta)$ time (property **P1**). This fulfills the delay requirement as $join(\mathcal{Q}(B))$ cannot be empty. In comparison, the state of the art [13, Proposition 3] demands $O((\text{AGM}/\Delta) \log \text{IN})$ space, where AGM is the AGM bound of $\mathcal{Q}$. This is never better than $O((\text{polylog}_{W^*}/\Delta) \log \text{IN})$ and can, in fact, be considerably worse.

5.3 Join Sampling

We now tackle the join sampling problem defined in Section 2. Let $\mathcal{T}$ be the box tree defined by $\Delta = 1$. However, rather than materializing the entire $\mathcal{T}$ (as in Section 5.2), this time we store nothing of $\mathcal{T}$, but sample its root-to-leaf paths on an "as-needed" basis.

The `sample` algorithm, shown in Figure 2, begins at the root of $\mathcal{T}$. In general, if the algorithm is currently at an internal node (with box) B, it first obtains the child nodes B_1 and B_2 of B in $\mathcal{T}$ (this requires only splitting B evenly on its split dimension; see Section 5.1). Then,

algorithm `sample`
/* $\mathcal{T}$ is the box tree defined by $\Delta = 1$ */
1. $B \leftarrow [\text{IN}]^d$ /* the join space */
2. **while** $polymat_W(B) > 1$ **do**
3. obtain the child nodes B_1, B_2 of B in $\mathcal{T}$
4. $B^+ \leftarrow$ a rand. value drawn according to (16)-(18)
5. **if** $B^+ = \perp$ **then return** "failure"
6. $B \leftarrow B^+$

/* $polymat_W(B) \leq 1$ here */
7. **if** $join(\mathcal{Q}(B)) \neq \emptyset$ **then**
8. **return** the only tuple in $join(\mathcal{Q}(B))$
9. **else return** "failure"

Figure 2: An algorithm for join sampling

it draws a random value $B^+ \in \{B_1, B_2, \perp\}$ from the following distribution:

$$\mathbf{Pr}[B^+ = B_1] \quad = \quad \frac{polymat_{W^*}(B_1)}{polymat_{W^*}(B)} \tag{16}$$

$$\mathbf{Pr}[B^+ = B_2] \quad = \quad \frac{polymat_{W^*}(B_2)}{polymat_{W^*}(B)} \tag{17}$$

$$\mathbf{Pr}[B^+ = \perp] \quad = \quad 1 - \mathbf{Pr}[B^+ = B_1] - \mathbf{Pr}[B^+ = B_2]. \tag{18}$$

If $B^+ = \perp$, then the algorithm declares "failure" and terminates. Otherwise, it descends to the child B^+.

Now consider that the algorithm has reached a leaf B, whose polymatroid value is at most 1 (property **P1**). As explained in Section 5.2, $join(\mathcal{Q}(B))$ has at most one tuple and can be computed in $O(1)$ time. If $join(\mathcal{Q}(B))$ is not empty, the algorithm returns the only tuple therein; otherwise, it declares "failure".

By Property **P3**, `sample` runs in $O(\log \text{IN})$ time plus $O(\log \text{IN})$ calls to the polymatroid oracle. Whenever `sample` does not declare failure, we say that it *succeeds*. Next, we argue that if the algorithm succeeds, it returns a uniformly random tuple from $join(\mathcal{Q})$.

Fix an arbitrary tuple $\boldsymbol{u} \in join(\mathcal{Q})$. As mentioned at the beginning of Section 5, $\boldsymbol{u}$ can be regarded as a point in the join space. Let B be the leaf of $\mathcal{T}$ whose box covers that point — such a leaf must exist due to Property **P2**. It must hold that

$$polymat_{W^*}(B) = 1. \tag{19}$$

To see why, note that $polymat_{W^*}(B) \leq 1$ by property **P1**, while, on the other hand, $polymat_{W^*}(B) \geq |join(\mathcal{Q}(B))| \geq 1$. Let $B_0, B_1, ..., B_h$ be the nodes on the path of $\mathcal{T}$ from the root to B (i.e., B_0 is the root and

$B_h = B$). The probability that `sample` outputs $\boldsymbol{u}$ is

$$\prod_{i=1}^{h} \frac{polymat_{W^*}(B_i)}{polymat_{W^*}(B_{i-1})} = \frac{polymat_{W^*}(B)}{polymat_{W^*}([\mathrm{IN}]^d)}$$
$$\text{(by (19))} = \frac{1}{polymat_{W^*}}. \quad (20)$$

As the above is identical for all $\boldsymbol{u}$, we conclude that `sample` returns a uniformly random tuple of $join(\mathcal{Q})$ whenever it succeeds.

The success probability of `sample` is

$$\sum_{\boldsymbol{u} \in join(\mathcal{Q})} \mathbf{Pr}[\boldsymbol{u} \text{ is output}] = \frac{\mathrm{OUT}}{polymat_{W^*}}.$$

In expectation, $\frac{polymat_{W^*}}{\mathrm{OUT}}$ repeats of the algorithm are needed to acquire a sample.

A special case occurs when $\mathrm{OUT} = 0$ — this will force us to repeat the algorithm forever. To remedy the issue, we stop using `sample` after $polymat_{W^*}$ repeats and revert to a worst-case optimal join algorithm (e.g., [21]) to compute $join(\mathcal{Q})$ in $O(polymat_{W^*})$ time, which will either confirm $\mathrm{OUT} = 0$ or allow us to obtain a random tuple of $join(\mathcal{Q})$ with no extra cost.

We have not mentioned any data structure. Indeed, the `sample` algorithm itself requires no data structures. Nevertheless, to implement a call to the polymatroid oracle in $O(\log \mathrm{IN})$ time, one still needs a data structure of $O(\mathrm{IN})$ space, as will be discussed in Section 5.4. This makes the expected join sampling time $O(\frac{polymat_{W^*}}{\max\{1,\mathrm{OUT}\}} \log^2 \mathrm{IN})$, which is worse than the state of the art [29] by only a factor of $O(\log^2 \mathrm{IN})$.

5.4 Discussion

We have made two assumptions, **A1** and **A2**, at the beginning of Section 5. These assumptions are quite "harmless" when the relations of $\mathcal{Q}$ are static. First, if IN is not a power of 2, one can add dummy tuples to increase IN to the nearest power of 2. Second, **A2** can be satisfied through *value renaming*: map the tuple values to distinct elements in $[\mathrm{IN}]$, which can be achieved with sorting in $O(\mathrm{IN} \log \mathrm{IN})$ time.

If value renaming is difficult or inconvenient (e.g., when insertions are allowed), however, we would have to work directly with the original tuple values. Suppose, in general, that every value is in a domain $[U]$ for some integer U, which can be arbitrarily larger than IN. The box tree $\mathcal{T}$ described in Section 5.1 still applies, but its height becomes $O(\log U)$; accordingly, the number of nodes in T becomes $O((polymat_{W^*}/\Delta) \log U)$.

This is a place where Theorem 3 is useful. Specifically, it allows us build an alternative box tree $\mathcal{T}$ satisfying properties **P1**-**P4** without assumptions **A1** and **A2**. Parameterized by an integer $\Delta \geq 2$, this $\mathcal{T}$ is a $(2d+1)$-ary tree whose nodes are each associated with a distinct box. As before, the polymatroid value of a node is at least the sum of the polymatroid values of its children. The root of $\mathcal{T}$ is associated with $\mathbb{N}^d$. In general, consider a node associated with box B. If B has a polymatroid value at most Δ, it becomes a leaf of $\mathcal{T}$. Otherwise, we split B into a set S of boxes promised by Theorem 3 and, for each $B' \in S$, create a child of B associated with B'. This completes the definition of the new box tree.

Properties **P1** and **P2** follow immediately from the definition. Regarding property **P3**, the argument in Section 5.1 no longer works. Nevertheless, we can "rescue" the property as follows. First, the polymatroid value of the root is $O(\mathrm{IN}^d)$ (this is a trivial upper bound for the polymatroid bound of $\mathcal{Q}$). Every time we descend a level, the polymatroid value drops by at least half, as guaranteed by Theorem 3. Therefore, the number of levels cannot exceed $O(\log \mathrm{IN})$. Finally, property **P4** can be established using an argument similar to the one in Section 5.1.

All the algorithms in Sections 5.2 and 5.3 can be adapted to work with the new box tree. Splitting a box using Theorem 3 would require $O(\log \mathrm{IN})$ calls [14] to the polymatroid oracle (rather than $O(1)$ calls in Section 5.1). This may cause a slow down in CPU time by a factor of $O(\log \mathrm{IN})$.

Finally, we briefly discuss polymatroid oracles. The essence of implementing such an oracle is to support a *range count* operation on each relation $R \in \mathcal{Q}$. Assume, w.l.o.g., that $schema(R) = \{A_{i_1}, A_{i_2}, ..., A_{i_t}\}$, where $t = |schema(R)|$ and $i_1 < i_2 < ... < i_t$. Define a lexicographic order over R by regarding each tuple $\boldsymbol{u} \in R$ as a string of t "characters": $\boldsymbol{u}(A_{i_1})\boldsymbol{u}(A_{i_2})...\boldsymbol{u}(A_{i_t})$. Given an interval I over the lexicographic order, a range count operation finds the number of tuples whose encodings fall within I. It is elementary to design a structure of $O(\mathrm{IN})$ space that performs the operation in $O(\log \mathrm{IN})$ time and supports an update in $O(\log \mathrm{IN})$ time.

6. A LITERATURE SURVEY

Modern research on natural joins was revitalized by the discovery of the AGM bound — named after Atserias, Grohe, and Marx [4] — which is an upper bound on the join result size that is asymptotically tight in the worst case under data complexity. Ngo et al. [23] designed the first algorithm that computes a join $\mathcal{Q}$ in $O(AGM)$ time, where AGM is the AGM bound of $\mathcal{Q}$. This is

worst-case optimal because even just reporting the result $join(\mathcal{Q})$ may necessitate $\Omega(AGM)$ time, as is implied by the tightness of the AGM bound. Since then, the community has developed quite a large number of join algorithms [3,7,17,20,22,24–26,28] achieving the worst-case optimal time, up to polylogarithmic factors.

The AGM bound, however, tends to be loose in practice. One reason is that it accounts only for cardinality constraints (as are defined in Section 2), while ignoring degree constraints $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$ with $\mathcal{X} \neq \emptyset$. In [18], Khamis, Ngo, and Suciu introduced the polymatroid bound, which generalizes the AGM bound by incorporating all degree constraints. In Section 3, we described polymatroid bounds for *acyclic* sets of degree constraints, as they are the focus of this article. However, it should be noted that polymatroid bounds also apply to *cyclic* sets of degree constraints. Khamis, Ngo, and Suciu [18] presented an algorithm named PANDA that can process any join $\mathcal{Q}$ in $O(polymat \operatorname{polylog} \mathrm{IN})$ time, where $polymat$ is the polymatroid bound of $\mathcal{Q}$. In [21], Ngo showed that the time can be improved to $O(polymat)$ when the set of degree constraints is acyclic. The running time is worst-case optimal in the sense explained in Section 3.

Deep and Koutris [13] studied the small-delay enumeration problem in a setup more general than the one defined in Section 2: their formulation allows a selection condition to be provided at runtime. Of particular relevance here is the following variant of their setting: a user supplies a box B and demands $join(\mathcal{Q}(B))$ — as defined in (8) — to be enumerated with a delay of $O(\Delta)$. The goal is to process $\mathcal{Q}$ into a data structure to support this operation for any B. The ideas presented in Section 5.2 can be integrated with the standard *dyadic interval* technique to achieve the purpose, although the space consumption would increase by an $O(\operatorname{polylog} \mathrm{IN})$ factor. Readers interested in this line of work may refer to [31] for alternative space-delay tradeoffs.

For the problem of join sampling, improving over previous results [2,11,12], Deng, Lu, and Tao [14] presented a structure of $O(\mathrm{IN} \operatorname{polylog} \mathrm{IN})$ space that achieves expected sample time $O(\frac{\mathrm{AGM}}{\max\{1,\mathrm{OUT}\}} \operatorname{polylog} \mathrm{IN})$, where $\mathrm{OUT} = |join(\mathcal{Q})|$. They introduced the AGM split theorem, which as mentioned earlier is the "AGM-counterpart" of Theorem 3, and the box tree described in Section 5.4. Kim et al. [19] improved the result of [14] by shaving off the $\operatorname{polylog} \mathrm{IN}$ factor in both the space usage and sample time. Later, Wang and Tao [29] managed to reduce the expected sample time to $O(polymat / \max\{1, \mathrm{OUT}\})$, while keeping the space at $O(\mathrm{IN})$, for joins with acyclic sets of degree constraints. The methods of [19, 29] are rather sophisticated and do not fall into the box-tree framework. Recently, targeting the same type of joins as [29], Capelli, Irwin, and Salvati [7] gave a structure of $O(\mathrm{IN} \log \mathrm{IN})$ space and $O(\frac{polymat}{\max\{1,\mathrm{OUT}\}} \log \mathrm{IN})$ expected sample time. Their method can be understood as a way to implement what was described in Section 5.3.[2]

Deng, Lu, and Tao [14] observed a connection between join sampling and small-delay enumeration. Specifically, they showed that the latter problem can be reduced, with high probability (i.e., at least $1 - 1/\mathrm{IN}^c$ for any arbitrarily large constant), to the former. By applying their reduction to the result of [29], one obtains with high probability a structure of $O(\mathrm{IN})$ space that can enumerate $join(\mathcal{Q})$ with a delay of $O(\frac{polymat}{\max\{1,\mathrm{OUT}\}} \operatorname{polylog} \mathrm{IN})$. Even better, the enumeration actually produces a random permutation of the elements in $join(\mathcal{Q})$.

The three problems defined in Section 2 admit better solutions when the join $\mathcal{Q}$ is "acyclic" as per the definition in [1, Chapter 6.4] (this should not be confused with the notion of acyclicity on degree constraint sets). Specifically, Yannakakis' algorithm [30] settles the join reporting problem in $O(\mathrm{IN} + \mathrm{OUT})$ time. Moreover, the algorithm implies a structure of $O(\mathrm{IN})$ space that can enumerate $join(\mathcal{Q})$ with a constant-time delay. For join sampling, one can preprocess an acyclic $\mathcal{Q}$ into a structure of $O(\mathrm{IN})$ space that guarantees $O(1)$ sample time [32].

In general, given a non-acyclic join $\mathcal{Q}$, it is possible to convert it to an acyclic join $\mathcal{Q}'$ with exactly the same result, i.e., $join(\mathcal{Q}) = join(\mathcal{Q}')$. The conversion takes $O(\mathrm{IN}^{\mathrm{fhtw}(\mathcal{Q})})$ time, where $\mathrm{fhtw}(Q)$ is the *fractional hyper-tree width* of $\mathcal{Q}$. We refer the reader to [16] for the precise calculation of $\mathrm{fhtw}(Q)$ and remark that $\mathrm{fhtw}(Q) > 1$ for every non-acyclic $\mathcal{Q}$. The new join $\mathcal{Q}'$ has an input size of $O(\mathrm{IN}^{\mathrm{fhtw}(\mathcal{Q})})$. By combining this idea with our earlier discussion on acyclic joins, one obtains a structure of $O(\mathrm{IN}^{\mathrm{fhtw}(\mathcal{Q})})$ space that can be used to enumerate $join(\mathcal{Q})$ with a constant-time delay or to sample from it in constant time.

Both the "small-delay enumeration" and "join sampling" problems are closely related to the *direct access* problem. Let $\prec$ be a total order imposed on a join result $join(\mathcal{Q})$. Given an integer $i \in [\mathrm{OUT}]$ (where $\mathrm{OUT} = |join(\mathcal{Q})|$), a direct-access query returns the i-th tuple of $join(\mathcal{Q})$ under the ordering of $\prec$. The goal of the direct access problem is to preprocess $\mathcal{Q}$ into a

[2]The method of [7] treats a relation of t attributes as an "expanded" relation of $t \log \mathrm{IN}$ attributes, each taking values 0 or 1. We find that treatment unnecessary when adopting the box-tree perspective.

data structure that can answer such a query in Δ time, where Δ is a target bound. Given a direct-access structure and the value of OUT, we can enumerate $join(\mathcal{Q})$ with a delay of Δ by issuing direct-access queries with $i = 1, 2, ..., \text{OUT}$, respectively, or sample from $join(\mathcal{Q})$ in Δ time by issuing a query with a random $i \in [\text{OUT}]$. Readers interested in this topic may refer to representative works [5, 6, 8, 9] and the references therein. For small-delay enumeration and join sampling, however, the performance guarantees provided by the existing direct-access methods are subsumed by the results stated earlier.

7. CONCLUSIONS

The box-tree technique is a method for join reporting and sampling developed by the database theory community in recent years. It is intuitive, conceptually clean, and can be deployed easily to design algorithms for join processing. Its theoretical foundation lies in the AGM split theorem introduced by [14]. In this article, we proved two new split theorems that extend the result of [14] by replacing the AGM bound with a polymatroid bound. These new theorems further strengthen the box-tree technique, enabling it to yield algorithms and data structures that rival the current state of the art.

One limitation of the technique is that the box tree $\mathcal{T}$ has $O(\log \text{IN})$ levels — the logarithmic factor prevents the technique from fully matching the best time bounds for join reporting and sampling. This limitation could be eliminated at the risk of losing the technique's elegance: given a box B, we could split it, along its split dimension (see its meaning in Section 4.1), into a maximum number of boxes B' whose $\mathcal{Q}(B')$ — as defined in (8) — has a non-empty size. Such an approach allows us to "reinterpret" the reporting algorithm of [21] and the sampling algorithm of [29] from the box-tree perspective. Nevertheless, we do not yet have an interpretation clean enough for graduate-level classrooms at the moment.

Another, more severe limitation of the technique is that it cannot deal with cyclic sets of degree constraints. The root cause lies in our split theorems, which have been proved only for acyclic sets of degree constraints. We invite the community to address this open problem.

8. REFERENCES

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

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

[3] 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.

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

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

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

[7] Florent Capelli, Oliver Irwin, and Sylvain Salvati. A simple algorithm for worst case optimal join and sampling. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 23:1–23:19, 2025.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] Shiyuan Deng, Shangqi Lu, and Yufei Tao. On join sampling and the hardness of combinatorial output-sensitive join algorithms. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 99–111, 2023.

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

[16] Martin Grohe and Daniel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):4:1–4:20, 2014.

[17] 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.

[18] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 429–444, 2017.

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

[20] 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.

[21] Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 111–124, 2018.

[22] 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.

[23] Hung Q. Ngo, Ely Porat, Christopher Re, 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.

[24] 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.

[25] 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.

[26] 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.

[27] Dan Suciu. Applications of information inequalities to database theory problems. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–30, 2023.

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

[29] Ru Wang and Yufei Tao. Join and subgraph sampling under degree constraints. *Journal of Computer and System Sciences (JCSS)*, 155, 2026.

[30] 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.

[31] Hangdong Zhao, Shaleen Deep, and Paraschos Koutris. Space-time tradeoffs for conjunctive queries with access patterns. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 59–68, 2023.

[32] 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.