A Simple Algorithm for Worst Case Optimal Join and Sampling

Florent Capelli
Université d'Artois, CNRS, UMR 8188 - CRIL, F-62300 Lens, France

Oliver Irwin
Université de Lille, CNRS, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France

Sylvain Salvati
Université de Lille, CNRS, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France

Abstract

We present an elementary branch and bound algorithm with a simple analysis of why it achieves worstcase optimality for join queries on classes of databases defined respectively by cardinality or acyclic degree constraints. We then show that if one is given a reasonable way for recursively estimating upper bounds on the number of answers of the join queries, our algorithm can be turned into algorithm for uniformly sampling answers with expected running time $\tilde{O}(\mathsf{UP}/\mathsf{OUT})$ where UP is the upper bound, OUT is the actual number of answers and $\tilde{O}(\cdot)$ ignores polylogarithmic factors. Our approach recovers recent results on worstcase optimal join algorithm and sampling in a modular, clean and elementary way.

2012 ACM Subject Classification Information systems → Relational database model; Theory of computation → Branch-and-bound

Keywords and phrases join queries, worst-case optimality, uniform sampling

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.23

Related Version *Full version with proofs*: `https://arxiv.org/abs/2409.14094`

Funding This work was supported by project ANR KCODA, ANR-20-CE48-0004.

1 Introduction

Join queries are expressions of the form $Q \coloneqq R_1(\mathbf{x_1}), \dots, R_m(\mathbf{x_m})$, where every R_i is a relation symbol and the $\mathbf{x_i}$ is a tuple of variables over a set X. Evaluating join queries is a central task when answering database queries. Since the combined complexity of deciding whether a given join query has at least one answer on a given database is NP-complete [2], it is unlikely that one can list all its answer in time linear in the number of answers. An interesting line of research has been the design of so called worst case optimal join (*WCOJ*) algorithm. In this setting, for a given query Q, we consider the worst possible database among a class of instances, that is, the one where the number of answers of Q is maximal. Now, even if we cannot find the answers of Q in time linear in the number of answers of Q, we can still aim at finding every answer in time linear in the number of answers of the worst possible database in the class. Such an algorithm will be said to be a WCOJ algorithm.

Consider for example the *triangle query* $Q_\triangle \coloneqq R(x_1, x_2), S(x_2, x_3), T(x_1, x_3)$.

We assume that R, S, T are relations of size respectively N_R, N_S and N_T. It is not hard to see that $Q_\triangle$ will have never more than $N_R \times N_S \times N_T$ answers. Even better, one can notice that the variables of R and S already cover all variables of $Q_\triangle$. Therefore, $Q_\triangle$ can not have more than $N_R \times N_S$, and by symmetry, no more than $\min(N_R N_S, N_S N_T, N_R N_T)$. The work of Atserias, Grohe and Marx [1] extends this notion of covering all the variables to the idea of a *fractional cover*, leading to an even better bound on the number of answers which is $(N_R N_S N_T)^{1/2}$ and this bound is actually *optimal* in the sense that there exists an instance

28th International Conference on Database Theory (ICDT 2025).
Editors: Sudeepa Roy and Ahmet Kara; Article No. 23; pp. 23:1–23:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of Q_Δ where R, S and T have respectively size at most N_R, N_S, N_T and $\tilde{\mathcal{O}}((N_R N_S N_T)^{1/2})$ answers where $\tilde{\mathcal{O}}(\cdot)$ hides polylogarithmic factors in the relation sizes and polynomial factor in the query size, considered constant. Therefore, an algorithm able to compute the answers of Q_Δ in time $\tilde{\mathcal{O}}((N_R N_S N_T)^{1/2})$ is a WCOJ algorithm for the class of instances of Q_Δ where R, S and T have size at most N_R, N_S, N_T respectively. It is optimal in the sense that it is linear in the size of the worst possible instance of the class.

In this simplified example, the class is defined via *cardinality constraints*: we consider instances where each relation has a size (or cardinality) that is bounded by a given integer. Building on the understanding of the worst case for such classes given in [1], Ngo, Porat, Ré and Rudra proposed the first WCOJ algorithm for instances defined by cardinality constraints in [11]. A simplified branch and bound algorithm, Triejoin, has been proposed by Veldhuizen in [16] and a more general version, known as GenericJoin has been introduced by Ngo in [10], which is also worst case optimal on classes defined by so-called *acyclic degree constraints*, which is a strict generalisation. Since then, a fruitful line of research has focused in understanding worst case bounds for classes of instances defined via more complex constraints (e.g., functional dependencies or non acyclic degree constraints). A deep connection with information theory has been made in [8] by Khamis, Ngo and Suciu, allowing the design of PANDA, which can perform join queries in time that is not far from worst case optimality, see [15] for an enlighting survey by Suciu on this connection.

Another related line of research has focused on designing algorithms to uniformly sample answers of join queries. One naive way of doing so is to first list $\mathsf{ans}(Q)$ explicitly and then uniformly sample an element of the list. Using a WCOJ algorithm, this gives a method allowing constant time sampling after a preprocessing linear in the worst case. This complexity however does not match the intuition one could have of the hardness of the problem. Indeed, it is reasonable to expect a query to be easier to sample if it has many solutions, because, intuitively, they are easier to find. It turns out that this intuition can be turned into a formally proven algorithm which achieves the following: for a class $\mathcal{C}$ of queries defined via cardinality constraints, Deng, Lu and Tao [5] simultaneously with Kim, Ha, Flechter and Han [9] proved that one can achieve uniform sampling for a join query $Q \in \mathcal{C}$ in time of $\tilde{\mathcal{O}}(\frac{|\mathsf{wc}(\mathcal{C})|}{\max(1,|\mathsf{ans}(Q)|)})$ where $\mathsf{wc}(\mathcal{C})$ is the worst case instance of class $\mathcal{C}$ and $\mathsf{ans}(Q)$ is the set of answers of Q. This result has recently been generalised to the case of acyclic degree constraints by Wang and Tao [17].

Our contributions. In this paper, we propose a very simple join algorithm with a very simple analysis which achieves worst case optimality (up to logarithmic factor) on classes of instances defined by cardinality constraints and degree constraints. Our algorithm is a simple branch and bound algorithm which assigns one variable to every possible value in the domain and backtracks whenever an inconsistency is detected. As such, this can be seen as an extremely simplified version of GenericJoin [10] or TrieJoin [16]. However, for these algorithms, a clever data structure is needed to branch only on relevant values. This is actually necessary since this naive branch and bound algorithm is not really worst case optimal. Indeed, on a query on domain D, an extra factor of $|D|$ appears in the complexity. We turn it into a WCOJ algorithm with a simple trick: instead of branching directly on domain values, we branch on the values bit by bit. An illustration of our algorithm is given in Figure 1 for the triangle query Q_Δ with tables given in Table 1. On the left, we show the branch and bound algorithm where values of $x_1, \dots, x_3$ are iteratively set to values in the domain $\{0, 1, 2, 3\}$. Whenever a relation is inconsistent with the current partial assignment, we backtrack, which is represented with $\perp$ in the tree. Observe on the example that after setting x_1 to 0, we explore many "useless" values for x_2 that directly give an inconsistency. In this simple example, we can directly read from R that only value $x_2 \mapsto 0$ is relevant, but in some more complex queries, we may need to compute more complex intersections efficiently, which is exactly how GenericJoin and TrieJoin address the problem.

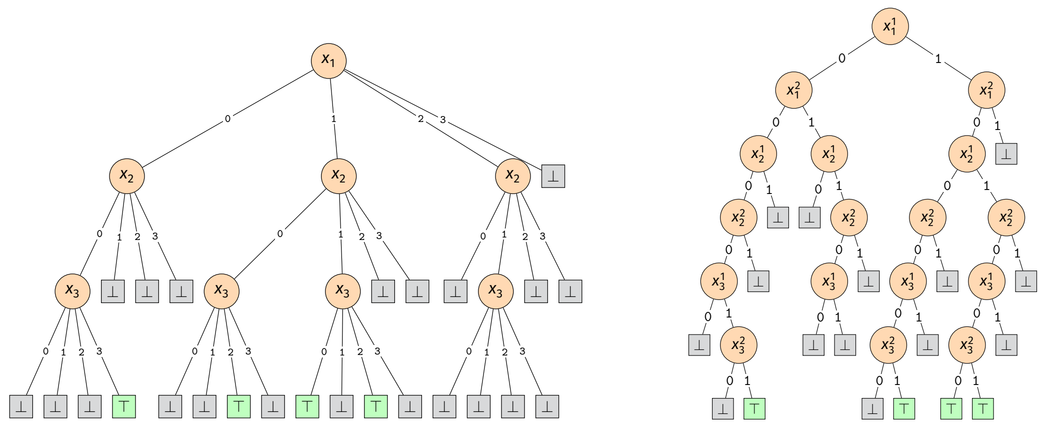
To avoid the need for such a data structure, we slightly modify the algorithm as shown on the right part of Figure 1. We encode the domain $\{0, 1, 2, 3\}$ with two bits on $\{0, 1\}^2$ and now branch on the first bit x_1^1 of x_1 and then on the second bit x_1^2 of x_1 and so on. We can directly see on the example that when x_1 is set to 0, that is, when $x_1^1 \mapsto 0, x_1^2 \mapsto 0$, then we do not explore the values $1, 3$ for x_2 as we directly detect inconsistency when setting the first bit of x_2 to 1. This simple trick is enough to guarantee worst case optimality of a simple branch and bound algorithm.

Table 1 An instance of $Q_\triangle$ on domain $\{0, 1, 2, 3\}$.

R	x1	x2
	0	0
	1	0
	1	1
	2	1

S	x2	x3
	0	2
	0	3
	1	0
	1	2

T	x1	x3
	0	3
	1	0
	1	2
	2	3



(a) Over 4-valued domain. **(b)** Binarised version.

Figure 1 Trace of an execution of our algorithm over the triangle query $Q_\triangle$ defined in Table 1. Satisfying assignments are labelled $\top$ and shown with a greenish node. On the right side, the same algorithm branching on the bits of the values instead of the values themselves.

One strength of our result is that worst case optimality is proven without any knowledge of the actual worst case of the class. This is in contrast with most existing WCOJ algorithms, whose analysis often relies on how the worst case value is computed (with the exception of leapfrog triejoin [16]). This knowledge is infused into the proof and, sometimes, even in the algorithm itself, for example in the first NPPR algorithm [11]. Our analysis only exploits one property that we call *prefix closed*. A class of instances $\mathcal{C}$ is intuitively prefix closed if for every $Q \in \mathcal{C}$, the number of answers of Q where we have removed some variables never exceeds the worst case $\mathsf{wc}(\mathcal{C})$. This property is straightforward to establish for classes defined with cardinality constraints and with acyclic degree constraints resulting in an elementary proof of worst case optimality.

The second contribution of this paper is to show how uniformly sampling answers can be achieved in expected runtime of $\tilde{\mathcal{O}}(\frac{\mathsf{wc}(\mathcal{C})}{\max(1,|\mathsf{ans}(Q)|)})$ for classes defined with cardinality constraints and with acyclic degree constraints, matching the complexity established in previous work with more involved techniques [17, 5, 9]. Our approach is elementary. Intuitively, we see the trace of our WCOJ algorithm as a tree whose leaves are either conflicts or solutions. The sampling problem hence reduces to uniformly sampling "interesting" leaves in a tree, without fully exploring it. It turns out that this is easy to do by adapting an algorithm from Rosenbaum [14] as long as one has a way of overestimating the number of interesting leaves in each subtree. We show that this can be done for join queries using the knowledge we have on how to compute worst case bounds. The only technical blackbox we use to establish this result is (a weak form of) Friedgut's inequality [6]. We recover in particular the recent result from [17] on sampling join queries under acyclic degree constraints with an elementary proof.

Organisation of the paper. We give some necessary notations and preliminaries in Section 2. Section 3 contains the description of our branch and bound algorithm and a simple analysis of its complexity. We then show that this is enough to establish worst case optimality for classes defined with cardinality constraints and acyclic degree constraints in Section 4. Finally, Section 5 shows that the branch and bound algorithm can easily be turned into a sampling algorithm achieving the same complexity as previous work in a simpler and more modular way. Due to space constraints, some (easy) proofs have been moved to the appendix. We identify them by adding ($\star$) in the statements.

2 Preliminaries

Notations. We assume the reader familiar with the basic vocabulary of database theory and mostly introduce notations in this section. Given two sets X and D, we denote by D^X the set of *tuples* over variables X and domain D, that is, the set of mappings from X to D. We denote by $\langle\rangle$ the empty tuple, that is, the only element of $D^\emptyset$ and by $\langle x \leftarrow d\rangle$ the tuple on variable $\{x\}$ that maps x to d. For $\tau \in D^X$ and $\sigma \in D^Y$ with $X \cap Y = \emptyset$, we denote by $\tau \cup \sigma$ the tuple mapping $z \in X \cup Y$ to $\tau(z)$ if $z \in X$ and $\sigma(z)$ otherwise.

A *relation* R is a subset of D^X. Given $\tau \in D^X$ and $Y \subseteq X$, we denote by $\tau_{|Y}$ the restriction of τ to Y, that is, the tuple such that $\tau_{|Y}(y) = \tau(y)$ for every $y \in Y$. For $R \subseteq D^X$, we write $R_{|Y}$ for $\{\tau_{|Y} \mid \tau \in R\}$. From now on when mentioning a relation, for example R, we assume that X_R is the set of variables on which it is defined, i.e. $R \subseteq D^{X_R}$. Let $\tau \in D^Y$, we denote by $R[\tau]$ the relation $\{\sigma_{|X_R - Y} \mid \sigma \in R,\ \sigma_{|Y} = \tau_{|X_R}\}$. That is $R[\tau]$ is obtained by filtering out every tuple of R that do not agree with τ on the variables Y.

A *join query* Q over variables X and domain D is a set of relations such that for every $R \in Q$, $R \subseteq D^{X_R}$ for some $X_R \subseteq X$. Observe that, as it is often done in the literature about WCOJ algorithms, we slightly deviate from the usual database setting which separates the data from the query. We can still see a join query as a usual full conjunctive query with hypergraph $(X, \{X_R \mid R \in Q\})$ and the data, that is, the tuples of each $R \in Q$.

The *answer set of* Q, denoted by $\mathsf{ans}(Q)$, is defined as the set of tuples $\tau \in D^X$ such that for every $R \in Q$, $\tau_{|X_R} \in R$. The *join problem* is the problem of outputing $\mathsf{ans}(Q)$ given Q as input. The *data size* of Q, denoted as $\|Q\|$, is defined as the number of tuples in its relations, $\|Q\| = \sum_{R\in Q} |R|$. Given $Y \subseteq X$, we denote by $Q_{|Y}$ the join query defined as $\{R_{|Y\cap X_R} \mid R \in Q\}$. Given $\tau \in D^Y$, we define $Q[\tau] = \{R[\tau] \mid R \in Q\}$. We say that τ is *inconsistent with* Q if $Q[\tau]$ contains an empty relation; τ is otherwise *consistent with* Q. We make the following observation that will be crucial for the rest of this paper:

▶ **Lemma 1** ($\star$). *For every $\tau \in D^Y$, $\tau \in \mathsf{ans}(Q_{|Y})$ iff τ is consistent with Q.*

In this paper, we will always make the assumption that the domain D of a join query is its *active domain*, that is, the set of values that appear in at least one relation. Moreover, we assume that every value in this active domain is encoded with $\mathcal{O}(\log|D|)$ bits. While this is a reasonable assumption, it may not be completely realistic in practice (for example when using string values). We can still enforce this condition with linear preprocessing by reencoding the domain using a perfect hash function [4].

Worst-case optimal join. In this section, we give an abstract definition of what we call a worst-case optimal join algorithm. Let $H = (X, E)$ be a hypergraph and $\mathcal{C}$ be a class of join queries with hypergraph H. We define the *worst case of* $\mathcal{C}$, denoted by $\mathsf{wc}(\mathcal{C})$ as $\mathsf{wc}(\mathcal{C}) = \sup_{Q \in \mathcal{C}} |\mathsf{ans}(Q)|$. An algorithm is a *worst-case optimal join for* $\mathcal{C}$ if, on input $Q \in \mathcal{C}$, it outputs $\mathsf{ans}(Q)$ in time $\tilde{\mathcal{O}}(\mathsf{wc}(\mathcal{C}) \times p(n, m))$, where $\tilde{\mathcal{O}}(\cdot)$ hides polylog factors, p is a polynomial independent on $\mathcal{C}$ and $m = |E|$ and $n = |X|$ are parameters that only depend on the structure of the query and not on the content of the relations. Observe that these parameters are constants with respect to $\mathcal{C}$ so it may be unecessary to add the $p(n, m)$ factor in the desired complexity. However, we want to stress out that the complexity of our algorithm (and of the existing worst-case optimal join algorithms) behaves reasonably with respect to n and m. Of course, for this definition to make sense, one needs $\mathsf{wc}(\mathcal{C})$ to be finite. Many such classes have been studied in the literature and many worst-case optimal join algorithms have been proposed. In this paper, we will focus on the two main classes that have been considered: classes defined via cardinality constraints and classes defined via degree constraints.

Cardinality Constraints. Let $H = (X, E)$ be a hypergraph verifying $\bigcup E = X$ (every node is covered by a hyperedge) and let $\mathbf{N} \in \mathbb{N}^E$. We let $\mathcal{C}_H(\leqslant \mathbf{N})$ be the class of join queries Q on hypergraph H such that for every $e \in E$, there is R_e in Q s.t. $X_{R_e} = e$ and $|R_e| \leqslant \mathbf{N}(e)$. We say that $\mathcal{C}_H(\leqslant \mathbf{N})$ is a class defined via *cardinality constraints* because it puts a bound on the cardinality of (the intersection of) the involved relations. Clearly, $\mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N})) \leqslant \prod_{e \in E} \mathbf{N}(e) < \infty$. Actually, one can get a sharper almost optimal upper bound on $\mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))$ using a result by Grohe and Marx [7] (optimality was proven by Atserias, Grohe and Marx in [1]) and which has later been known as the AGM bound. For example, one can show that for the triangle hypergraph $H_\Delta = \{e_1, e_2, e_3\}$ where $e_1 = \{1, 2\}$, $e_2 = \{2, 3\}$ and $e_3 = \{1, 3\}$, $\mathsf{wc}(\mathcal{C}_{H_\Delta}(\leqslant \mathbf{N})) \leqslant \sqrt{N(e_1)N(e_2)N(e_3)}$. We delay the precise presentation of such bounds to Section 5 where we are interested in sampling answer from conjunctive queries. One strength of our worst case optimal join approach compared to previous work is that we do not need to have an understanding of the worst case bound to prove its worst case optimality.

Degree constraints. Another class of join queries which received attention in the literature on worst-case optimal joins is the class of queries defined with degree constraints. Given two sets $A \subseteq B$, a *degree constraint* is a triplet of the form $(A, B, N_{B|A})$ with $N_{B|A} > 0$. A relation R_e on variables $e \supseteq B$ respects the degree constraint $(A, B, N_{B|A})$ if and only if $\max_{\tau \in D^A} |R[\tau]_{|B}| \leqslant N_{B|A}$. It is a generalisation of cardinality constraints since a cardinality constraint can be seen as a degree constraint of the form $(\emptyset, B, N_B)$. It can also be seen as a generalisation of functional dependencies since a functional dependency $X \to y$ can be seen as the $(X, \{y\}, 1)$ degree constraint. Let $H = (X, E)$ be a hypergraph and DC be a set of degree constraints of the form $(A, B, N_{B|A})$ with $A \subseteq B \subseteq X$. Each degree constraint

$\delta \in DC$ is associated with an hyperedge $e_\delta \in E$ with $e_\delta \supseteq B$ which guards it. We let $\mathcal{C}_H(DC)$ be the class of queries Q on hypergraph H such that for every $\delta = (A, B, N_{B|A}) \in DC$, there is an atom R in Q such that $X_R = e_\delta$ and R respects δ.

Observe that it may happen that $\mathsf{wc}(\mathcal{C}_H(DC)) = +\infty$. In this paper, we are only interested in classes where this does not happen. This is often enforced by assuming that $\bigcup E = X$ and that for every $e \in H$, at least one constraint in DC is a cardinality constraint of the form $(\emptyset, e, N_e)$ with guard e that has hence to be respected by a relation R with $X_R = e$. In this case, as before, $\mathsf{wc}(\mathcal{C}_H(DC)) \leqslant \prod_e N_e < +\infty$. Here again, more precise upper bounds are known on $\mathsf{wc}(\mathcal{C}_H(DC))$ but they will not be necessary for our worst case optimal join algorithm and we delay this discussion to Section 5 where we will need them.

3 Branch and bound algorithm for join queries

In this section, we propose a simple branch and bound algorithm to compute join queries and provide an easy upper bound on its complexity. The algorithm can be seen as an instance of GenericJoin from [13] but it is given in an extremely simple form and its analysis is elementary. Written in this way, the algorithm is not worst case optimal but a simple algorithmic trick will allow us to recover known results, presented in Section 4.

The algorithm, whose pseudo code is given in Algorithm 1, is a simple recursive search: assume a fixed order $(x_1, \dots, x_n)$ is given on variables X. We find the answers of Q by setting variables sequentially according to this order, trying each possible value in the domain. Whenever the current partial assignment is inconsistent with Q, it is not further expanded. If every variable is assigned and the assignment is consistent with Q, then it is output.

Algorithm 1 An algorithm to compute join queries.

procedure $WCJ(Q, \tau)$
    if $Q[\tau]$ contains an empty relation **then return**
    $i \leftarrow$ last variable assigned by τ
    if $i = n$ **then** output τ. **return**
    for $d \in D$ **do** $WCJ(Q, \tau \cup \langle x_{i+1} \leftarrow d \rangle)$
end procedure

Correction of the algorithm. Starting with a call $WCJ(Q, \langle\rangle)$, every recursive call is of the form $WCJ(Q, \tau)$ where τ is a tuple in D^{X_i} where $X_i := \{x_1, \dots, x_i\}$. We claim that for every τ which assigns variables X_i, then $WCJ(Q, \tau)$ outputs $\tau \cup \sigma$ for every answer σ of $Q[\tau]$. The proof is by induction on i. If $i = n$, then τ is output if and only if $Q[\tau]$ does not contain the empty relation, which by Lemma 1 means that τ is an answer of Q. Now assume $i < n$. If τ is inconsistent with Q then nothing is output, this is coherent with our induction hypothesis since $Q[\tau]$ contains an empty relation, meaning that any tuple σ so that $\sigma_{|X_i} = \tau$ is not in $\mathsf{ans}(Q)$. Otherwise, by induction, $WCJ(Q, \tau \cup \langle x_{i+1} \leftarrow d \rangle)$ outputs $\tau \cup \langle x_{i+1} \leftarrow d \rangle \cup \sigma$ for every $\sigma \in \mathsf{ans}(Q[\tau \cup \langle x_{i+1} \leftarrow d \rangle])$, that is, for every $\sigma \in \mathsf{ans}(Q[\tau])$. It completes the induction and it directly follows that $WCJ(Q, \langle\rangle)$ outputs $\mathsf{ans}(Q)$.

Number of recursive calls. We claim that Algorithm 1 does at most $(1 + |D|) \cdot \sum_{i \leqslant n} |\mathsf{ans}(Q_{|X_i})|$ recursive calls. Indeed, as stated before, every recursive call is of the form $WCJ(Q, \tau)$ where τ is a tuple of D^{X_i}. In the first case, assume that Q is consistent with τ, which means in particular that τ is in $\mathsf{ans}(Q_{|X_i})$ by Lemma 1. Hence, there are at most $\sum_{i \leqslant n} |\mathsf{ans}(Q_{|X_i})|$ recursive calls of this type. In the second case, assume that τ is inconsistent with Q. Then the recursive call with parameters (Q, τ) has been issued from a call of the form (Q, τ') where $\tau = \tau' \cup \langle x_i \leftarrow d \rangle$ for some $d \in D$. In particular, τ' is consistent with Q, otherwise, such a recursive call would not have happened. Hence, $\tau' \in \mathsf{ans}(Q_{|X_{i-1}})$ and there are at most $|D|$ possible τ for a given $\tau' \in \mathsf{ans}(Q_{|X_{i-1}})$. Therefore, there are at most $|D| \cdot \sum_{i \leqslant n} |\mathsf{ans}(Q_{|X_i})|$ recursive calls of this form, this in total, $(|D|+1) \sum_{i \leqslant n} |\mathsf{ans}(Q_{|X_i})|$ recursive calls.

Efficient implementation. Now we explain how, using a very simple data structure, one can assume that each recursive call is executed in $\tilde{\mathcal{O}}(m)$ where m is the number of atoms in Q. The only non trivial thing is to check whether $Q[\tau]$ contains an empty relation. To do that, we simply assume that every relation is given sorted in lexicographical order, for the attribute order $x_1, \dots, x_n$. This could be obtained via a preprocessing that is quasi linear in the data (or linear in the RAM model, but since we ignore polylogarithmic factors, it does not matter much). Now observe that if R is a relation of Q and τ a tuple in D^{X_i}, then all tuples from $R[\tau]$ are consecutively stored in the table. Hence we can represent $R[\tau]$ by keeping two pointers p_1, p_2 on the tuples of R: one towards the first tuple and one towards the last tuple in $R[\tau]$. To check whether $R[\tau]$ is consistent, it is enough to check that $p_1 \leqslant p_2$. To go from the representation of $R[\tau]$ to the representation of $R[\tau \cup \langle x_{i+1} \leftarrow d \rangle]$, we simply need to find the first and last tuple between p_1 and p_2 where $x_{i+1} = d$. This can be done via a binary search in time $\mathcal{O}(\mathsf{log}|R|)$. Hence, each recursive join can be executed in time $\mathcal{O}(m \mathsf{log} \|Q\|)$, that is, $\tilde{\mathcal{O}}(m)$. A slightly more involved data structure would allow us to compute in time $\mathcal{O}(m)$ by representing R as a trie as in [16]. We just proved:

▶ **Theorem 2.** *Given a join query Q on domain D with m atoms and $(x_1, \dots, x_n)$ an order on the variables of Q, $WCJ(Q, \langle\rangle)$ computes $\mathsf{ans}(Q)$ in time $\tilde{\mathcal{O}}(m|D| \cdot \sum_{i \leqslant n} |\mathsf{ans}(Q_{|X_i})|)$, where $X_i = \{x_1, \dots, x_i\}$.*

4 Worstcase optimality

4.1 Prefix closed classes

To show that Algorithm 1 is worst case optimal on a class $\mathcal{C}$ of instances, we need to bound Theorem 2 by $\tilde{\mathcal{O}}(\mathsf{wc}(\mathcal{C}))$. Of course, this will not be true for any class of instances but it turns out that we can easily do so on classes defined by cardinality constraints or by acyclic degree constaints. Theorem 2 motivates the following definition: a class $\mathcal{C}$ is *prefix closed for the order $\pi = (x_1, \dots, x_n)$* if and only if for every $i \leqslant n$ and $Q \in \mathcal{C}$, $|\mathsf{ans}(Q_{|X_i})| \leqslant \mathsf{wc}(\mathcal{C})$. Indeed, if $\mathcal{C}$ is prefix closed for order π, then computing $\mathsf{ans}(Q)$ for $Q \in \mathcal{C}$ using Algorithm 1 with order π will take $\tilde{\mathcal{O}}(mn \cdot |D| \cdot \mathsf{wc}(\mathcal{C}))$, where D is the domain of Q.

▶ **Theorem 3.** *For every class $\mathcal{C}$ that is prefix closed for order $(x_1, \dots, x_n)$ and join query $Q \in \mathcal{C}$ with n variables and m relations, $WCJ(Q, \langle\rangle)$ returns $\mathsf{ans}(Q)$ in time $\tilde{\mathcal{O}}(nm \cdot |D| \cdot \mathsf{wc}(\mathcal{C}))$.*

While m and n are considered constant in our setting, we cannot assume so for $|D|$. Hence, Theorem 2 and prefix closedness will not be enough to establish worst case optimality of Algorithm 1. That said, we present a simple trick in Section 4.2 which allows us to circumvent this issue easily. The main classes for which worst case optimal algorithms are known are prefix closed, at least for one order. Even if cardinality constraints are less general than degree constraints, we start by showing it for the former as a warmup, even if the proof is essentially the same for the latter:

▶ **Theorem 4.** *Let $\mathcal{C}_H(\leqslant \mathbf{N})$ be a class of join queries defined for hypergraph $H = (X, E)$ and cardinality constraints $\mathbf{N} \subseteq \mathbb{N}^E$. Then $\mathcal{C}_H(\leqslant \mathbf{N})$ is prefix closed for every order.*

Proof. Let $Q \in \mathcal{C}_H(\leqslant \mathbf{N})$, $(x_1, \ldots, x_n)$ be an order on X and $i \leqslant n$. We need to show that $\mathsf{ans}(Q_{|X_i}) \leqslant \mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))$. To do so, we construct $Q^* \in \mathcal{C}_H(\leqslant \mathbf{N})$ such that $|\mathsf{ans}(Q_{|X_i})| = |\mathsf{ans}(Q^*)|$. Since $Q^* \in \mathcal{C}_H(\leqslant \mathbf{N})$, we have by definition that $\mathsf{ans}(Q^*) \leqslant \mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))$, hence $\mathsf{ans}(Q_{|X_i}) \leqslant \mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))$.

Assume that Q is on domain $D \neq \emptyset$ and let $d \in D$ be some fixed element of D. We denote by $d^Y \in D^Y$ the tuple defined as $d^Y(y) = d$ for every $y \in Y$. Let $R \in Q_{|X_i}$. By definition, $R = R_e|_{X_i}$ for some $e \in E$. Hence, $|R| \leqslant |R_e| \leqslant \mathbf{N}(e)$. We define $R_e^* \subseteq D^e$ as $R \times \{d^{e \setminus X_i}\}$, that is, we extend every tuple from R to variables e by setting every missing variable to d. Clearly, $|R^*| = |R| \leqslant |R_e| \leqslant \mathbf{N}(e)$. Hence the query Q^* defined as $\{R^* \mid R \in Q_{|X_i}\}$ is in $\mathcal{C}_H(\leqslant \mathbf{N})$. Moreover, we clearly have $\mathsf{ans}(Q^*) = \mathsf{ans}(Q_{|X_i}) \times \{d^{X \setminus X_i}\}$, therefore $|\mathsf{ans}(Q^*)| = |\mathsf{ans}(Q_{|X_i})|$ as needed to complete the proof. ◀

We now generalise the previous result to classes defined via degree constraints. Observe however that such classes may not always be prefix closed, or sometimes only for some particular order. For example, consider the query $Q = R(x_3, x_1) \wedge S(x_3, x_2)$ and consider the class $\mathcal{C}$ respecting functional dependencies $x_3 \to x_1$ and $x_3 \to x_2$ and cardinality constraints $|R| \leqslant N$ and $|S| \leqslant N$. Clearly, $\mathsf{wc}(\mathcal{C}_Q) \leqslant N$ since once x_3 is fixed, so are x_1 and x_2. Now, consider an instance Q^* where $R^* = S^* = \{(i, i) \mid 0 < i \leqslant N\}$. It is easy to see that $Q^* \in \mathcal{C}$ and that $Q^*_{|\{x_1,x_2\}}$ has $N^2 > \mathsf{wc}(\mathcal{C}_Q)$ answers. The previous example is not prefix closed for (x_1, x_2, x_3) because we chose an order that goes in the wrong direction with regard to the functional dependencies. One can check that $\mathcal{C}_Q$ is prefix closed for the order (x_3, x_2, x_1).

This motivates the following definition: for $H = (X, E)$ a hypergraph and DC to be a set of degree constraints. We define the *dependency graph* G_{DC} as the graph whose vertex set is X and there is an edge $u \to v$ if and only if there is a degree constraint $(A, B, N_{B|A})$ in DC with $u \in A$ and $v \in B \setminus A$. We say that DC is acyclic if G_{DC} is acyclic. In this case, an order $(x_1, \ldots, x_n)$ is said to be *compatible* with DC if this is a topological sort of G_{DC}. Unsurprisingly, this allows to prove the following generalisation of Theorem 4:

▶ **Theorem 5.** *Let $\mathcal{C}_H(DC)$ be a class of join queries defined for hypergraph $H = (X, E)$ and acyclic degree constraints DC. Then $\mathcal{C}_H(DC)$ is prefix closed for every order compatible with DC.*

Proof. The proof is very similar to the proof of Theorem 4. Let $Q \in \mathcal{C}_H(DC)$ and $i \leqslant n$. We construct Q^* as in Theorem 4. We still have $|\mathsf{ans}(Q_{|X_i})| = |\mathsf{ans}(Q^*)|$. We only have to check that $Q^* \in \mathcal{C}_H(DC)$. Let $(A, B, N_{B|A}) \in DC$ be a cardinality constraint. By definition, it is respected by an atom R of Q on variable $e \supseteq B$. We claim that $R^* \in Q^*$ also respects δ. Indeed $X_i \cap e \subseteq A$, then for every $\tau \in D^A$, there is at most one tuple in $R^*[\tau]$ which is $\tau \times d^{e \setminus X_i}$, hence $|R^*[\tau]|_{|B}| \leqslant 1 \leqslant N_{B|A}$. Otherwise, since the order is compatible with DC, $A \subseteq X_i$. Hence $R^*[\tau] = R_{|X_i}[\tau] \times d^{e \setminus X_i}$. In particular $|R^*[\tau]| = |R_{|X_i}[\tau]| \leqslant |R[\tau]|$. Hence projecting out on B, $|R_{|X_i}[\tau]_{|B}| \leqslant |R[\tau]_{|B}| \leqslant N_{B|A}$ since R respects the degree constraint $(A, B, N_{B|A})$. Hence, R^* also respects this degree constraint. Since this reasoning works for every $R^* \in Q^*$, we conclude that $Q^* \in \mathcal{C}_H(DC)$. Hence $|\mathsf{ans}(Q_{|X_i})| = |\mathsf{ans}(Q^*)| \leqslant \mathsf{wc}(\mathcal{C}_H(DC))$, which is what we needed to prove. ◀

A direct corollary of Theorems 3 and 5 is that Algorithm 1 is almost worst case optimal on classes defined by acyclic degree constraints.

▶ **Corollary 6.** *Let $\mathcal{C}_H(DC)$ be a class of join queries defined for hypergraph $H = (X, E)$, $m = |E|, n = |X|$ and acyclic degree constraints DC. Assume $x_1, \ldots, x_n$ is an order compatible with DC. Then for every $Q \in \mathcal{C}_H(DC)$ $WCJ(Q, \langle\rangle)$ returns* $\mathsf{ans}(Q)$ *in time* $\tilde{\mathcal{O}}(mn|D| \cdot \mathsf{wc}(\mathcal{C}_H(DC)))$.

Observe that in order to prove worst case optimality of Algorithm 1 in Corollary 6, we have not used any knowledge on the actual value of $\mathsf{wc}(\mathcal{C}_H(DC))$, which makes our approach simpler than existing analysis of worst case optimal join algorithms. While such knowledge is also not necessary in the analysis of Leapfrog Trie Join [16], the complexity analysis given in this paper is more complicated.

4.2 Binarisation

We have seen that Algorithm 1 achieves $\tilde{\mathcal{O}}(mn|D| \cdot \mathsf{wc}(\mathcal{C}))$ complexity when $\mathcal{C}$ is prefix closed, which does not qualify as a worst case optimal join yet. The extra $|D|$ factor comes from the fact that we are testing every possible value of $d \in D$ for each variable, even if many of them will directly lead to inconsistencies. We could overcome this issue by exploring only relevant values, using for example the trie join algorithm from [16] which allows to enumerate values present in the intersection of every relation in time $O(\log|D|)$ or Hash indices as in [12]. While these techniques are interesting for practical implementation, our goal in this paper is to use as little technical tools as possible. Hence, we present here a new simple technique to remove the extra $|D|$ factor. The main idea is that instead of testing every value in the domain for each variable, we fix its value bit by bit. This could be implemented directly by modifying Algorithm 1 or, as we chose to present it, by transforming any join query Q on domain D with n variables into a join query $\widetilde{Q}^b$ with $n \cdot b$ variables where $b = \lceil \log|D| \rceil$ variables on domain $\{0, 1\}$ such that the answers of $\widetilde{Q}^b$ are in one-to-one correspondence with the answers of Q. We do this by reencoding each element of the domain D in binary.

More formally, let Q be a join query on variables X and domain D. Without loss of generality, we assume that $D = \{1, \ldots, d\}$ for some d and we let $b = \lceil \log d \rceil$ to be the number of bits needed to encode every element of D. We represent each element k in D by the binary number $\widetilde{k}^b$ representing k and written with b bits. For $1 \leqslant i \leqslant b$, let $\widetilde{k}^b[i]$ be the i^{th} bit of a binary representation of $k \in D$. The function $\widetilde{\cdot}^b$ is a bijection between D and its image.

We now lift the functions $\widetilde{\cdot}^b$ to pairs of bijections over tuples, relations and then over join queries. For a set of variables Y, we denote by $\widetilde{Y}^b$ the set $\{y^i \mid y \in Y, 1 \leqslant i \leqslant b\}$, that is, the set containing b distinct copies of each variable of Y. For $\tau \in D^Y$, we define $\widetilde{\tau}^b$ as follows: for every $y \in Y$ and $i \in [b]$, $\widetilde{\tau}^b(y^i) = \widetilde{\tau(y)}^b[i]$. Given a relation $R \subseteq D^Y$ we let $\widetilde{R}^b = \{\widetilde{\tau}^b \mid \tau \in R\}$. Finally, given a join query Q over variables X and domain D, we let $\widetilde{Q}^b = \{\widetilde{R}^b \mid R \in Q\}$. Obviously, the answers of $\widetilde{Q}^b$ are in one to one correspondence with the answers of Q. Moreover, we have that the cardinalities of relations are invariant under this transformation, i.e. $|\widetilde{R}^b| = |R|$. Applying Theorem 2 on $\widetilde{Q}^b$ directly yields the following:

▶ **Theorem 7.** *Given a join query Q on domain $D \subseteq [2^b]$ with m atoms, $(x_1, \ldots, x_n)$ an order on the variables of Q, $WCJ(\widetilde{Q}^b, \langle\rangle)$ with order $(x_1^1, \ldots, x_1^b, \ldots, x_n^1, \ldots, x_n^b)$ computes* $\mathsf{ans}(Q)$ *in time* $\tilde{\mathcal{O}}(m \sum_{i \leqslant n} \sum_{j \leqslant b} |\mathsf{ans}(\widetilde{Q}^b_{|X_i^j})|)$ *where* $X_i^j = \{x_1^1, \ldots, x_1^b, \ldots, x_i^1, \ldots, x_i^j\}$.

To show worst case optimality, it remains to bound $\mathsf{max}_{i,j}|\mathsf{ans}(\widetilde{Q}^b_{|X_i^j})|$ by $\mathsf{wc}(\mathcal{C})$. We do this by showing that in the case of acyclic degree constraint, $\widetilde{Q}^b$ belongs to a class $\widetilde{\mathcal{C}}^b$ defined by acyclic degree constraints where $\mathsf{wc}(\widetilde{\mathcal{C}}^b) \leqslant \mathsf{wc}(\mathcal{C})$ and such that if $\mathcal{C}$ is prefix closed for $x_1, \ldots, x_n$ then $\widetilde{\mathcal{C}}^b$ is prefix closed for $x_1^1, \ldots, x_1^b, \ldots, x_n^1, \ldots, x_n^b$. The idea is to binarise the degree constraints as follows: for $b \in \mathbb{N}$ and degree constraint $\delta = (A, B, N)$, we denote by $\widetilde{\delta}^b$ the degree constraint $(\widetilde{A}^b, \widetilde{B}^b, N)$ and for a set DC of degree constraints, let $\widetilde{DC}^b := \{\widetilde{\delta}^b \mid \delta \in DC\}$. We show:

▶ **Lemma 8.** *Let DC be a set of degree constraints, H a hypergraph and $b \in \mathbb{N}$. For every $Q \in \mathcal{C}_H(DC)$ on domain $D \subseteq [2^b]$, we have $\widetilde{Q}^b \in \mathcal{C}_{\widetilde{H}^b}(\widetilde{DC}^b)$. Moreover, $\mathsf{wc}(\mathcal{C}_{\widetilde{H}^b}(\widetilde{DC}^b)) \leqslant \mathsf{wc}(\mathcal{C}_H(DC))$. Finally, if DC is acyclic and $x_1, \dots, x_n$ is an order compatible with DC, then $\widetilde{DC}^b$ is acyclic and $x_1^1, \dots, x_1^b, \dots, x_n^1, \dots, x_n^b$ is an order compatible with $\widetilde{DC}^b$.*

Proof. The first part of the statement follows from the following observation: let $\delta = (A, B, N)$ be a degree constraint and R a relation on variable $e \supseteq B$ which respects δ, then $\widetilde{R}^b$ respects $\widetilde{\delta}^b$. Indeed, let τ be an assignment of $\widetilde{A}^b$ and let τ' be the corresponding assignment of A on domain 2^b defined as $\tau'(x) = \sum_{i=1}^{b} 2^{i-1}\tau(x^i)$. Then it is easy to see that $R[\tau']|_Y$ is in one to one correspondence with $\widetilde{R}^b[\tau]|_{\widetilde{Y}^b}$ by using the same encoding. In particular, $|\widetilde{R}^b[\tau]|_{\widetilde{Y}^b}| = |R[\tau']|_Y \leqslant N$.

Now let $Q' \in \mathcal{C}_{\widetilde{H}^b}(\widetilde{DC}^b)$ be a query on domain D. We let Q to be the query on hypergraph H on domain D^b where each relation R' of Q' on variables $\widetilde{Y}^b$ is transformed into a relation R on variables Y as follows: for a tuple $\tau' \in R'$, we build the tuple $\tau \in R$ by taking for each $y \in Y$, $\tau(y) = \bigtimes_{j \leqslant b} \tau'(y^j)$. It is easy to see that if R' respects degree constraint $\widetilde{\delta}^b$, then R respects δ and that $\mathsf{ans}(Q)$ and $\mathsf{ans}(Q')$ are in one-to-one correspondence. Hence $Q \in \mathcal{C}_H(DC)$ and then $\mathsf{wc}(\mathcal{C}_{\widetilde{H}^b}(\widetilde{DC}^b)) \leqslant \mathsf{wc}(\mathcal{C}_H(DC))$.

Finally, by definition, it is clear that there is an edge in G_{DC} between x and y if and only if there is an edge between x^i and y^j for every $i, j \leqslant b$ in $G_{\widetilde{DC}^b}$. Assume towards a contradiction that there is a path from x_i^j to x_k^ℓ for some $i \geqslant k$ in $G_{\widetilde{DC}^b}$. Then there is necessarily a path from x_i to x_k in G_{DC} from what precedes, which contradicts the fact that $x_1, \dots, x_n$ is a topological sort of G_{DC}. ◀

A direct consequence of Lemma 8 is that $\mathcal{C}_{\widetilde{H}^b}(\widetilde{DC}^b)$ is prefix closed when DC is acyclic and its worst case is not greater than the worst case of $\mathcal{C}_H(DC)$. Since for any $Q \in \mathcal{C}_H(DC)$, the domain of $\widetilde{Q}^b$ is two and has $bn = \tilde{O}(n)$ variables, we have a worst case optimal join (up to logarithmic factors) algorithm for $\mathcal{C}_H(DC)$:

▶ **Corollary 9.** *Let $\mathcal{C}_H(DC)$ be a class of join queries defined for hypergraph $H = (X, E)$, $m = |E|, n = |X|$ and acyclic degree constraints DC. Assume $x_1, \dots, x_n$ is an order compatible with DC. Then for every $Q \in \mathcal{C}_H(DC)$ on domain $D \subseteq [2^b]$, $WCJ(\widetilde{Q}^b, \langle\rangle)$ on order $x_1^1, \dots, x_1^b, \dots, x_n^1, \dots, x_n^b$ returns $\mathsf{ans}(Q)$ in time $\tilde{O}(mn \cdot \mathsf{wc}(\mathcal{C}_H(DC)))$.*

There is a slight abuse in the statement of Corollary 9 as the algorithm does not directly return $\mathsf{ans}(Q)$ but a binary representation of each tuple in $\mathsf{ans}(Q)$. However, it is straightforward to turn each answer of $\widetilde{Q}^b$ back to the corresponding answer of Q in $\tilde{O}(1)$.

Comparison with Generic Join and Leapfrog Triejoin. Our algorithm is quite similar to *Generic Join*[13, Algoritm 3] and *Leapfrog Trie Join* [16], which can already be seen as a particular case of Generic Join. Similarly, Algorithm 1 can be seen as a degenerated form of Generic Join. The main difference in the approach is that both Generic Join and Trie Join use a specific algorithm (trie join algorithm and m-way sort merge respectively) to ensure that a variable x is branched only on values that would not introduce any inconsistency. We circumvent this need by using binarisation instead, which can be seen, from a higher perspective, as simply branching on the bits of each value instead. In turn, this approach could be emulated directly on the query (without binarizing the query explicitly) which would result in an algorithm that is very close to Generic Join, that is, a branching algorithm which efficiently branch only on the values of x that do not contradict the query right away. That said, this approach would then blur the simple complexity analysis we provide in this paper where we can plainly see that the complexity of the algorithm depends on the prefix closed property of the class and the size of the domain (which can be reduced via binarization).

The main novelty in our work is the complexity analysis. The analysis for Generic Join from [13] relies on the knowledge of the value of the worst case, known as the AGM bound for cardinality constraints and polymatroid bound for acyclic degree constraints. While proofs of these bounds can be found in numerous references (see [15] for a survey), they add a layer of complexity in the understanding of why these simple branch and bound strategies achieve worst case optimality. Our analysis instead relies on a very easy to check property of the classes considered, namely the prefix closedness property: "forgetting" variables in the tables will not allow to create an instance having more answers than the worst case. Proving that classes defined by cardinality constraints or acyclic degree constraints are prefix closed is elementary and very natural. Our approach is closer in spirit with the one taken in [16] where the concept of *renumbering* is introduced, and the analysis of the runtime is bounded by the runtime of the algorithm on a normalised instance where some values have been changed. While the approach is similar, we feel that prefix closedness is an easier notion.

5 Uniform Sampling

There has been significant work around extending WCOJ algorithms into sampling algorithms. It is known that if $Q \in \mathcal{C}$ for $\mathcal{C}$ a class of queries defined by cardinality constraints [5, 9] or by acyclic degree constraints [17], one can uniformly sample $\tau \in \mathsf{ans}(Q)$ in time of $\tilde{O}(\frac{|\mathsf{wc}(\mathcal{C})|}{\max(1,|\mathsf{ans}(Q)|)})$. We recover both results using an elementary algorithm, adaptated from [14] to uniformly sample leaves from a tree without exploring it completely. This approach is similar to the one used in [9] and in [3] but our approach is more modular, making it easier to adapt to the more general case of acyclic degree constraints, the probabilities calculation being much more elementary as the one presented in [5].

5.1 Efficiently sampling the leaves of a tree

Our core technique relies on the problem of sampling uniformly a leaf of a rooted tree. We aim at designing an algorithm which avoids exploring the tree exhaustively. This question has already been addressed by Rosenbaum in [14] where he proposes an algorithm exploring the tree in a top-down manner, and whenever it encouters a leaf, either returns it or fails. To remove bias toward subtrees having many leaves in the algorithm, he guides the search with upper bounds on the number of leaves of each subtree, obtained from the depth and the branching size of the tree.

We adapt Rosenbaum's algorithm in a slightly more general setting. Indeed, in our approach, the leaves of the tree will correspond to cases where the recursion of Algorithm 1 stops. In this case, either a solution is found and we are interested in the leaf, or a inconsistency is found and we want to reject the leaf. We adapt the algorithm to be able to work on a tree when we want to sample only a subset of its leaves. Moreover, the upper bounds on the leaves used in [14] is too coarse for our purposes, we therefore describe the algorithm by using oracle calls to a function (over-)estimating this number of leaves. This motivates the following definition:

▶ **Definition 10.** *Let T be a rooted tree. A* leaf estimator upb *for T is a function mapping nodes of T to positive values such that:*

1. *for every node t with children $t_1, \dots, t_n$, $\mathsf{upb}(t) \geqslant \sum_{i=1}^{n} \mathsf{upb}(t_i)$; we call functions with this property* tree-superadditive*; and*
2. *if t is a leaf, then $\mathsf{upb}(t) \in \{0, 1\}$.*

We denote by $\mathsf{upb}(T)$ the value of $\mathsf{upb}(r)$ where r is the root of T. Now, given a tree T and a leaf estimator upb for T, we say that a leaf ℓ of T is a 1-*leaf of T* if and only if $\mathsf{upb}(\ell) = 1$ and denote by $\mathsf{leaves}_1(T)$ the set of 1-leaves of T. Our goal is to uniformly sample $\ell \in \mathsf{leaves}_1(T)$. Observe that since upb is a tree-superadditive function, for any node t, we have that $\mathsf{upb}(t)$ is an upper bound on the number of 1-leaves below t. We define $\mathsf{children}(t)$ as the function that, given a node t, returns the list of its direct children.

We define our algorithm recursively as follows:

Algorithm 2 A variation of the Rosenbaum algorithm [14].

Sample a leaf in the subtree root in t as follows:

- if t is a leaf belonging to $\mathsf{leaves}_1(T)$, output the leaf with probability 1;
- it t is any other leaf, fail with probability 1; and
- if t has children $t_1, \dots, t_n$, recursively sample a 1-leaf in t_i with probability $\frac{\mathsf{upb}(t_i)}{\mathsf{upb}(t)}$ and return it if the recursive call in t_i succeeds and fail otherwise. Note that we may directly fail without recursively sampling with probability $1 - \sum_i \frac{\mathsf{upb}(t_i)}{\mathsf{upb}(t)}$.

▶ **Theorem 11.** *Let T be a tree rooted in r and upb a leaf estimator for T. Let out be the output of Algorithm 2 on input r. Then, for any leaf $\ell \in \mathsf{leaves}_1(T)$, we have that Algorithm 2 is a uniform Las Vegas sampler with guarantees:*

$$\Pr(\mathsf{out} = \ell) = \frac{1}{\mathsf{upb}(T)} \qquad \textit{and} \qquad \Pr(\mathsf{out} = \mathsf{fail}) = 1 - \frac{|\mathsf{leaves}_1(T)|}{\mathsf{upb}(T)}$$

Algorithm 2 consists of $\mathcal{O}(B \cdot \mathsf{depth}(T))$ calls to upb, where B is the branching size of the tree, in $\mathcal{O}(\mathsf{depth}(T))$ calls to the $\mathsf{children}$ function.

Proof. We proceed by induction on the depth of the tree T. If the tree T is of depth 1, then it can be one of two cases: either it belongs to $\mathsf{leaves}_1(T)$ and then $|\mathsf{leaves}_1(T)| = 1$ and therefore it is trivial to see that the algorithm samples this leaf with probability $\frac{1}{\mathsf{upb}(T)} = 1$, or it does not belong to $\mathsf{leaves}_1(T)$ and then there is nothing to sample, so the algorithm fails inevitably.

Supposing that the property holds for a tree T' of depth at most k, if we now have a tree T of depth $k+1$, then it has children $(t_1, \dots, t_n)$ each of depth at most k. Then, by induction, Algorithm 2 samples from a given t_i with probability $\frac{\mathsf{upb}(t_i)}{\mathsf{upb}(t)}$. If the recursive call has succeeded, then the algorithm has sampled a leaf from t_i with probability $\frac{1}{\mathsf{upb}(t_i)}$. Since the random choices are independent, the probability of outputing this leaf from t is $\frac{\mathsf{upb}(t_i)}{\mathsf{upb}(t)} \times \frac{1}{\mathsf{upb}(t_i)} = \frac{1}{\mathsf{upb}(t)}$.

The complexity statement is straightforward. We need to evaluate the number of selected leaves in each subtree along a path from the root to a leaf, leading to $\mathcal{O}(B \cdot \mathsf{depth}(T))$ calls to upb and for each node we visit, we need to find the list of children, leading to $\mathcal{O}(\mathsf{depth}(T))$ calls to $\mathsf{children}$. ◀

We can extend Theorem 11 with the following corollary:

▶ **Corollary 12** (⋆). *Given a tree T with branching size B and oracle access to a leaf estimator function* $\mathsf{upb}(\cdot)$*, we can sample the leaves in* $\mathsf{leaves}_1(T)$ *with uniform probability* $\frac{1}{|\mathsf{leaves}_1(T)|}$*, when* $|\mathsf{leaves}_1(T)| > 0$ *or answer that* $|\mathsf{leaves}_1(T)| = 0$*. This is done by repeating Algorithm 2 an expected* $\mathcal{O}(\frac{\mathsf{upb}(T)}{\max(1,|\mathsf{leaves}_1(T)|)})$ *number of times and thus with an expected number of calls to* upb *in* $\mathcal{O}(\frac{\mathsf{upb}(T)}{\max(1,|\mathsf{leaves}_1(T)|)} \cdot B \cdot \mathsf{depth}(T))$ *and to* $\mathsf{children}$ *in* $\mathcal{O}(\frac{\mathsf{upb}(T)}{\max(1,|\mathsf{leaves}_1(T)|)} \cdot \mathsf{depth}(T))$*.*

5.2 Applying Algorithm 2 to join queries

There is a strong link between the tree structure used in the aforementioned sampling method and Algorithm 1. Assume that we want to sample uniformly the results of the query Q with relations over variables $X = \{x_1, \dots, x_n\}$ and domain D. For this, we can follow the structure of the execution of Algorithm 1 when it uses the order $(x_1, \dots, x_n)$ on variables. The execution of Algorithm 1 naturally constructs a tree structure whose nodes are some assignments of D^{X_i} for some $i \in [0, n]$ corresponding to the input of recursive calls. We call this tree the *trace tree of* Q and denote it by T_Q. In T_Q, an assignment $\tau \in D^{X_i}$ is the *parent* of another assignment τ' when $\tau' = \tau \cup \langle x_{i+1} \leftarrow d\rangle$. Among all the possible assignments, the ones that are nodes in T_Q are those that are consistent with Q or those that are inconsistent with Q but have a parent that is consistent with Q. Figure 1 depicts such a tree: assignments that are inconsistent with Q are labelled $\bot$, those that are elements of the answer set are labelled $\top$. Moreover, the assignment corresponding to a node in Figure 1 can simply be read off the path from the root to that particular node.

The leaves of T_Q that we want to sample in this tree are simply the elements of D^{X_n} that are consistent with Q, namely the solutions of Q. Algorithm 2 can then be applied to T_Q. Thus we call *Q-estimator* a function $\mathsf{q\_upb}$ on the nodes of T_Q that verifies:

- when τ is a node of T_Q in D^{X_i} that is consistent with Q, $\mathsf{q\_upb}(\tau) \geqslant \sum_{d\in D} \mathsf{q\_upb}(\tau \cup \langle x_{i+1} \leftarrow d\rangle)$ (i.e. $\mathsf{q\_upb}$ is tree-superadditive),
- $\mathsf{q\_upb}(\tau) = 1$ when $\tau \in \mathsf{ans}(Q)$
- $\mathsf{q\_upb}(\tau) = 0$ when τ is inconsistent with Q.

▶ **Theorem 13.** *Given a Q-estimator* $\mathsf{q\_upb}(\tau)$ *that can be evaluated in time t for every τ, it is possible to uniformly sample* $\mathsf{ans}(Q)$ *with expected time* $\mathcal{O}(\frac{\mathsf{q\_upb}(\langle\rangle)}{\max(1,|\mathsf{ans}(Q)|)} \cdot |X| \cdot |D| \cdot t)$*.*

Proof. This is a consequence of Corollary 12. The depth of T_Q is $|X|$ and its branching size is $|D|$. ◀

5.3 Tree-superadditive worst-case bounds

It now remains to define Q-estimators when Q belongs to $\mathcal{C}_H(\leqslant \mathbf{N})$ or to $\mathcal{C}_H(DC)$. This will allow us to recover the sampling results from the literature [5, 9, 17] by using Theorem 13 and the binarisation technique Section 4.2. As the class $\mathcal{C}_H(DC)$ generalises the class $\mathcal{C}_H(\leqslant \mathbf{N})$, we could have only treated the first case. We think however that the $\mathcal{C}_H(\leqslant \mathbf{N})$ being simpler, it conveys more intuitions. Our main tool is a simple consequence of an inequality by Friedgut [6].

▶ **Lemma 14** (⋆, Friedgut, [6, Lemma 3.3]). *For every finite sets I and J, every family of positive real numbers $(\omega_j)_{j\in J}$ so that $\sum_{j\in J} \omega_j \geqslant 1$, and every family of positive real numbers $(a_{i,j})_{i\in I,j\in J}$, we have:*

$$\sum_{i\in I} \prod_{j\in J} a_{i,j}^{\omega_j} \leqslant \prod_{j\in J} \left(\sum_{i\in I} a_{i,j}\right)^{\omega_j} .$$

Cardinality constraints. Let $\mathcal{C}_H(\leqslant \mathbf{N})$ be a class of join queries defined for hypergraph $H = (X, E)$ (with $X = \{x_1, \ldots, x_n\}$ and $E = \{e_1, \ldots, e_m\}$) and cardinality constraints $\mathbf{N} \subseteq \mathbb{N}^E$. In [1], Atserias, Grohe and Marx show that $\mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))$ can be computed from the solutions of the following linear program:

$$\mathsf{min} \sum_{j}^{m} \omega_j \mathsf{log}(N(e_j)) \ s.t., \forall i = 1, \ldots, n \sum_{j: x_i \in E_j} \omega_j \geqslant 1$$

and prove $\mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))$ is $\prod_{j=1}^{m} N(e_j)^{\omega_j}$ up to polylogarithmic factors. The vector ω is called a *fractional cover* of H.

Let us fix a fractional cover ω of H, for a query Q in $\mathcal{C}_H(\leqslant \mathbf{N})$. For every $j \in [1, m]$, we let $R_j \in Q$ be a relation such that $|R_j| \leqslant N(e_j)$ (which exists by definition of $\mathcal{C}_H(\leqslant \mathbf{N})$). We take as Q-estimator $\mathsf{agm\_upb}(\tau)$ as follows:

$$\mathsf{agm\_upb}(\tau) = \begin{cases} 0 & \text{when } \tau \text{ is inconsistent with } Q \\ \prod_{j=1}^{m} |R_j[\tau]|^{\omega_j} & \text{otherwise} \end{cases}.$$

When τ is inconsistent with Q, then, by definition, $\mathsf{agm\_upb}(\tau) = 0$. Furthermore, when τ is in $\mathsf{ans}(Q)$, for every j, $|R_j[\tau]| = 1$ and therefore, $\mathsf{agm\_upb}(\tau) = 1$. To show that $\mathsf{agm\_upb}$ is a Q-estimator, we finally need to show that for every τ that is consistent with Q, we have $\mathsf{agm\_upb}(\tau) \geqslant \sum_{d \in D} \mathsf{agm\_upb}(\tau \cup \langle x_{i+1} \leftarrow d \rangle)$.

For every j we have:

- $|R_j[\tau]| = \sum_{d \in D} \left|R_j[\tau \cup \langle x_{i+1} \leftarrow d \rangle]\right|$ when x_{i+1} is in $X_{R_j} - X_i$
- $|R_j[\tau]| = |R_j[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|$ for every $d \in D$ otherwise.

We let $K = \{k \in [1, m] \mid x_{i+1} \in X_{R_j} - X_i\}$ and $L = [1, m] \setminus K$. Since $\bigcup E = X$, we must have $K \neq \emptyset$. We thus have:

$$\begin{aligned} \sum_{d \in D} \mathsf{agm\_upb}(\tau \cup \langle x_{i+1} \leftarrow d \rangle) &= \sum_{d \in D} \prod_{k \in K} |R_k[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|^{\omega_k} \times \prod_{l \in L} |R_l[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|^{\omega_l} \\ &= \sum_{d \in D} \prod_{k \in K} |R_k[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|^{\omega_k} \times \prod_{l \in L} |R_l[\tau]|^{\omega_l} \\ &= \prod_{l \in L} |R_l[\tau]|^{\omega_l} \times \sum_{d \in D} \prod_{k \in K} |R_k[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|^{\omega_k} \end{aligned}$$

Then $\mathsf{agm\_upb}(\tau) \geqslant \sum_{d \in D} \mathsf{agm\_upb}(\tau \cup \langle x_{i+1} \leftarrow d \rangle)$ follows from:

$$\sum_{d \in D} \prod_{k \in K} \left|R_k[\tau \cup \langle x_{i+1} \leftarrow d \rangle]\right|^{\omega_k} \leqslant \prod_{k \in K} \left(\sum_{d \in D} \left|R_k[\tau \cup \langle x_{i+1} \leftarrow d \rangle]\right| \right)^{\omega_k}.$$

By definition of K and since (ω_j) is a fractional cover of H, we have $\sum_{k \in K} \omega_k \geqslant 1$. Hence we can directly get the bound using Lemma 14.

▶ **Theorem 15.** *Given $\mathcal{C}_H(\leqslant \mathbf{N})$ a class of join queries defined for hypergraph $H = (X, E)$ and cardinality constraints $\mathbf{N} \subseteq \mathbb{N}^E$, for every query Q in $\mathcal{C}_H(\leqslant \mathbf{N})$, it is possible to uniformly sample* $\mathsf{ans}(Q)$ *with expected time* $\widetilde{\mathcal{O}}(\frac{\mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))}{\mathsf{max}(1, |\mathsf{ans}(Q)|)} \cdot |X| \cdot \mathsf{log}(|D|) \cdot |E|)$.

Proof. Given Q with active domain D, we let $b = \lceil \mathsf{log}(|D|) \rceil$ and we are going to sample $\widetilde{Q}^b$. As we have seen in Section 4.2, $\mathsf{ans}(Q)$ and $\mathsf{ans}(\widetilde{Q}^b)$ can be considered to be the same set. Moreover $\widetilde{Q}^b$ belongs to a class $\widetilde{\mathcal{C}}^b$ defined by cardinality constraints where $\mathsf{wc}(\widetilde{\mathcal{C}}^b) \leqslant \mathsf{wc}(\mathcal{C}_H(\leqslant \mathbf{N}))$. Therefore, there is a $\widetilde{Q}^b$-estimator $\mathsf{agm\_upb}(\cdot)$. Using the data structure described in Section 3 to represent $R[\tau]$ or a trie structure annotated with cardinalities to represent every relation of Q, we can compute $\left|R[\tau \cup \langle x_{i+1} \leftarrow d \rangle]\right|$ in $O(\mathsf{log}|R|)$. Therefore, computing $\mathsf{agm\_upb}(\cdot)$ takes time $\widetilde{\mathcal{O}}(|E|)$. Finally, by definition of $\mathsf{agm\_upb}(\cdot)$, we have $\mathsf{agm\_upb}(\langle\rangle) \leqslant \mathsf{wc}(\widetilde{\mathcal{C}}^b)$. This allows us to apply Theorem 13 and yields the claimed complexity. ◀

Polymatroid. Let $\mathcal{C}_H(DC)$ be a class of join queries defined for hypergraph $H = (X, E)$ (with $X = \{x_1, \ldots, x_n\}$ and $E = \{e_1, \ldots, e_m\}$) and acyclic degree constraints $(A_\delta, B_\delta, N_\delta)_{\delta \in DC}$. The polymatroid bound is a generalisation of the AGM bound that can be formulated on acyclic degree constraints with the solutions of the following linear program:

$$\min \sum_{\delta \in DC} \omega_\delta \log(N_\delta) \quad s.t., \forall x \in X, \sum_{\delta : x \in B_\delta \setminus A_\delta} \omega_\delta \geqslant 1 \ . \tag{1}$$

For this program to have a solution, we need to assume that for every $x \in X$, there is at least one constraint δ such that $x \in B_\delta \setminus A_\delta$. As stated in Section 2, to ensure that acyclic degree constraints induce a finite worst case, we generally assume that $\bigcup E = X$ and that for each $e \in E$, we have at least one cardinality constraint $(\emptyset, e, N_e)$. We assume this condition to be met here. Now for any solution ω_δ of the previous program, it has been shown [10] that $\mathsf{wc}(\mathcal{C}_H(DC))$ is $\prod_{\delta \in DC} N_\delta^{\omega_\delta}$ up to some polylogarithmic factors.

Let us fix a solution ω of (1), and an order $(x_1, \ldots, x_n)$ on X compatible with DC. For a query Q in $\mathcal{C}_H(DC)$ and $\delta = (A_\delta, B_\delta, N_\delta)$ in DC, we let R_δ to be a relation in Q that respects δ and such that $X_{R_\delta} = e_\delta$ (in other words, the relation guards this constraint). To lighten notations, we denote by $R'_\delta = R_\delta|_{B_\delta}$, since the degree constraint is applied to this projection of R_δ and not R_δ itself. Moreover, given τ in D^{X_i}, we let:

$$N_\delta[\tau] = \begin{cases} 0 & \text{if } \tau \text{ is inconsistent with } Q \\ N_\delta & \text{if } A_\delta \setminus X_i \neq \emptyset \\ |R'_\delta[\tau]| & \text{otherwise} \end{cases}$$

We now take as Q-estimator $\mathsf{pm\_upb}(\tau)$ as follows:

$$\mathsf{pm\_upb}(\tau) = \prod_{\delta \in DC} N_\delta[\tau]^{\omega_\delta} \ .$$

When τ is inconsistent with Q, by definition of $N_\delta[\tau]$, $\mathsf{pm\_upb}(\tau) = 0$. If τ is in $\mathsf{ans}(Q)$, since for every δ, $A_\delta \setminus X_n = \emptyset$, $N_\delta[\tau] = |R'_j[\tau]| = 1$ and therefore, $\mathsf{pm\_upb}(\tau) = 1$. Proving that $\mathsf{pm\_upb}$ is a Q-estimator finally requires to show that for every τ that is consistent with Q, we have $\mathsf{pm\_upb}(\tau) \geqslant \sum_{d \in D} \mathsf{pm\_upb}(\tau \cup \langle x_{i+1} \leftarrow d \rangle)$.

We let $K = \{\delta \in DC \mid x_{i+1} \in B_\delta \setminus A_\delta\}$ and $L = DC \setminus K$. As stated before, for the linear program to have a solution, we assumed $K \neq \emptyset$. We make two observations:

1. For $\delta \in K$, $N_\delta[\tau] = |R'_\delta[\tau]|$ and for any $d \in D$, $N_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle] = |R'_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|$,
2. For $d \in D$, $N_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle] \leqslant N_\delta[\tau]$.

The first observation follows directly from the facts that $x_{i+1} \in B_\delta \setminus A_\delta$ by definition of K and that $x_1, \ldots, x_n$ is compatible with DC which implies that for any $\delta \in K$, $A_\delta \subseteq X_i$. For the second inequality, first assume $A_\delta \setminus X_{i+1} \neq \emptyset$. Then both sides are equal to N_δ. Now assume $A_\delta \setminus X_{i+1} = \emptyset$. Then $N_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle] = |R'_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|$ and either $N_\delta[\tau] = |R'_\delta[\tau]|$, then the equality is clear, or $N_\delta[\tau] = N_\delta$ and the inequality follows from the fact that R_δ respects δ by definition. Then we obtain:

$$\begin{aligned} \sum_{d \in D} \mathsf{pm\_upb}(\tau \cup \langle x_{i+1} \leftarrow d \rangle) &= \sum_{d \in D} \prod_{\delta \in K} |R'_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|^{\omega_\delta} \times \prod_{\delta \in L} N_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle]^{\omega_\delta} \\ &\leqslant \sum_{d \in D} \prod_{\delta \in K} |R'_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|^{\omega_\delta} \times \prod_{\delta \in L} N_\delta[\tau]^{\omega_\delta} \\ &= \prod_{\delta \in L} N_\delta[\tau]^{\omega_\delta} \times \sum_{d \in D} \prod_{\delta \in K} |R'_\delta[\tau \cup \langle x_{i+1} \leftarrow d \rangle]|^{\omega_\delta} \end{aligned}$$

We hence get $\sum_{d\in D} \mathsf{pm\_upb}(\tau \cup \langle x_{i+1} \leftarrow d\rangle) \leqslant \mathsf{pm\_upb}(\tau)$ by showing that:

$$\sum_{d\in D}\prod_{\delta\in K}\left|R'_\delta[\tau\cup\langle x_{i+1}\leftarrow d\rangle]\right|^{\omega_\delta} \leqslant \prod_{\delta\in K}\left(\sum_{d\in D}\left|R'_\delta[\tau\cup\langle x_{i+1}\leftarrow d\rangle\right|\right)^{\omega_\delta} = \prod_{\delta\in K} R'_\delta[\tau]^{\omega_\delta} = \prod_{\delta\in K} N_\delta[\tau]^{\omega_\delta} \ .$$

The above inequality is a consequence of Lemma 14. Indeed, $\sum_{\delta\in K}\omega_\delta \geqslant 1$ by definition of K and because of the constraints ω_δ verifies in the linear program. We then conclude with this result, obtained exactly in the same way as Theorem 15.

▶ **Theorem 16.** *Given $\mathcal{C}_H(DC)$ a class of join queries defined for hypergraph $H = (X, E)$ and acyclic degree constraints DC, for every query Q in $\mathcal{C}_H(DC)$, it is possible to uniformly sample* $\mathsf{ans}(Q)$ *with expected time* $\mathcal{O}(\frac{\mathsf{wc}(\mathcal{C}_H(DC))}{\max(1,|\mathsf{ans}(Q)|)} \cdot |X| \cdot \log(|D|) \cdot |E|)$.

References

1 Albert Atserias, Martin Grohe, and Dániel Marx. Size Bounds and Query Plans for Relational Joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013. doi:10.1137/110859440.

2 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM. doi:10.1145/800105.803397.

3 Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In Carsten Lutz and Jean Christoph Jung, editors, *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, volume 155 of *LIPIcs*, pages 7:1–7:18. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICDT.2020.7.

4 Zbigniew J Czech, George Havas, and Bohdan S Majewski. Perfect hashing. *Theoretical Computer Science*, 182(1-2):1–143, 1997. doi:10.1016/S0304-3975(96)00146-6.

5 Shiyuan Deng, Shangqi Lu, and Yufei Tao. 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*, pages 99–111. ACM, 2023. doi:10.1145/3584372.3588666.

6 Ehud Friedgut. Hypergraphs, entropy, and inequalities. *The American Mathematical Monthly*, 111(9):749–760, 2004. URL: http://www.jstor.org/stable/4145187.

7 Martin Grohe and Dániel Marx. Constraint Solving via Fractional Edge Covers. *ACM Transactions on Algorithms*, 11(1):1–20, 2014. doi:10.1145/2636918.

8 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 Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 429–444. ACM, 2017. doi:10.1145/3034786.3056105.

9 Kyoungmin Kim, Jaehyun Ha, George Fletcher, and Wook-Shin Han. Guaranteeing the Õ(AGM/OUT) Runtime for Uniform Sampling and Size Estimation over Joins. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 113–125. ACM, 2023. doi:10.1145/3584372.3588676.

10 Hung Q. Ngo. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 111–124. ACM, 2018. doi:10.1145/3196959.3196990.

11 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 37–48. ACM, 2012. doi:10.1145/2213556.2213565.

12 Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):1–40, 2018. doi:10.1145/3180143.

13 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013. doi:10.1145/2590989.2590991.

14 Paul R. Rosenbaum. Sampling the Leaves of a Tree with Equal Probabilities. *Journal of the American Statistical Association*, 88(424):1455–1457, 1993. doi:10.1080/01621459.1993.10476433.

15 Dan Suciu. Applications of information inequalities to database theory problems. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*, pages 1–30. IEEE, 2023. doi:10.1109/LICS56636.2023.10175769.

16 Todd Veldhuizen. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. *Proceedings of the 17th International Conference on Database Theory (ICDT), Athens, Greece, 2014*, 17(13):96–106, 2014. doi:10.5441/002/ICDT.2014.13.

17 Ru Wang and Yufei Tao. Join Sampling Under Acyclic Degree Constraints and (Cyclic) Subgraph Sampling. In Graham Cormode and Michael Shekelyan, editors, *27th International Conference on Database Theory (ICDT 2024)*, volume 290 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICDT.2024.23.

A Proof of Lemma 1

▶ **Lemma 1** ($\star$). *For every $\tau \in D^Y$, $\tau \in \mathsf{ans}(Q_{|Y})$ iff τ is consistent with Q.*

Proof. It is simply a reformulation: if $\tau \in \mathsf{ans}(Q_{|Y})$ then it means that for every $R \in Q$, $\tau_{|X_R \cap Y} \in R_{|Y}$. In particular, $R[\tau]$ is not empty. Hence τ is consistent with Q. Conversely, if τ is consistent with Q, then for every $R \in Q$, $R[\tau]$ is not empty. That is, there exists some tuple $\sigma \in R$ such that $\sigma_{|Y} = \tau_{|X_R \cap Y}$. In other words, $\tau_{|X_R \cap Y} \in R_{|Y}$ for every $R \in Q$, hence, $\tau \in \mathsf{ans}(Q_{|Y})$. ◀

B Proof of Corollary 12

▶ **Corollary 12** ($\star$). *Given a tree T with branching size B and oracle access to a leaf estimator function $\mathsf{upb}(\cdot)$, we can sample the leaves in $\mathsf{leaves}_1(T)$ with uniform probability $\frac{1}{|\mathsf{leaves}_1(T)|}$, when $|\mathsf{leaves}_1(T)| > 0$ or answer that $|\mathsf{leaves}_1(T)| = 0$. This is done by repeating Algorithm 2 an expected $\mathcal{O}(\frac{\mathsf{upb}(T)}{\max(1,|\mathsf{leaves}_1(T)|)})$ number of times and thus with an expected number of calls to upb in $\mathcal{O}(\frac{\mathsf{upb}(T)}{\max(1,|\mathsf{leaves}_1(T)|)} \cdot B \cdot \mathsf{depth}(T))$ and to $\mathsf{children}$ in $\mathcal{O}(\frac{\mathsf{upb}(T)}{\max(1,|\mathsf{leaves}_1(T)|)} \cdot \mathsf{depth}(T))$.*

Proof. We first treat the case where $|\mathsf{leaves}_1(T)| = n > 0$. Algorithm 2 can either fail or produce a leaf that has been sampled with uniform probability. It is a *Las Vegas* algorithm. It samples $\mathsf{leaves}_1(T)$ of T with uniform probability $\frac{1}{\mathsf{upb}(T)}$. When $|\mathsf{leaves}_1(T)| = n > 0$, it thus outputs some data with probability $\frac{n}{\mathsf{upb}(T)}$ or fails with probability $1 - \frac{n}{\mathsf{upb}(T)}$. Now if we repeat the algorithm until it succeeds, as each $\mathsf{leaves}_1(T)$ has the same probability to be outputted in one repetition, they all have the same probability to be outputted at the end of this process. In a nutshell, this procedure uniformly chooses amongst the $\mathsf{leaves}_1(T)$ of T which all have probability $\frac{1}{n}$ to be outputted. Moreover, the number of repetitions of Algorithm 2 until it succeeds follows a geometric distribution and its expected value is thus $\frac{\mathsf{upb}(T)}{n}$.

The case where $|\mathsf{leaves}_1(T)| = 0$ is a little trickier. Since Algorithm 2 can only fail, repeating it would result in an infinite loop. We can circumvent this in one of two ways.

Either we start a full exploration of T in parallel and if it does not find any $\mathsf{leaves}_1(T)$ in T, we stop running Algorithm 2 (if Algorithm 2 returns a 1-leaf, we stop the exploration). A second method would be to improve Algorithm 2 by maintaining the parts of T that have already been explored and by updating the values of $\mathsf{upb}(t)$ for each subtree that is explored by using the information from its children. Eventually, the algorithm explores T entirely. Indeed parts of T that have been explored are known not to contain data and have thus probability 0 to be explored again. In the end, updating the $\mathsf{upb}(t)$ value of each node will result in having $\mathsf{upb}(T) = 0$, meaning that $\mathsf{leaves}_1(T)$ is empty. We then stop the search. Both methods would cost a time of $\mathcal{O}(\mathsf{upb}(T))$. ◀

▶ Remark 17. Notice that the second method presented in the proof may also be useful when $|\mathsf{leaves}_1(T)|$ is small compared to $\mathsf{upb}(T)$. Indeed, updating $\mathsf{upb}(\cdot)$ at each failure of the algorithm increases the probability of success of the next iteration of the algorithm resulting in an overall improvement in the speed of convergence.

C Proof of Friedgut's Lemma

In this section, we prove the lemma that we use in Section 5.3, that we restate below for convenience.

▶ **Lemma 14** (⋆, Friedgut, [6, Lemma 3.3]). *For every finite sets I and J, every family of positive real numbers $(\omega_j)_{j\in J}$ so that $\sum_{j\in J}\omega_j \geqslant 1$, and every family of positive real numbers $(a_{i,j})_{i\in I, j\in J}$, we have:*

$$\sum_{i\in I}\prod_{j\in J} a_{i,j}^{\omega_j} \leqslant \prod_{j\in J}\left(\sum_{i\in I} a_{i,j}\right)^{\omega_j} .$$

Proof. This is a consequence of the *Generalised Weighted Entropy Lemma* in [6, Lemma 3.3]. The statement of this lemma is depends on the following objects:

- A hypergraph $H = (X, E)$.
- A finite set L.
- A family of subsets of X $(F_l)_{l\in L}$. Let $e_l = e\cap F_l$ for every $e \in E$. And let $E_l = \{e_l \mid e \in E\}$.
- A family of weights $W = (w_l)_{l\in L}$: w_l associates a positive real number to the elements of E_l.
- A family of positive real numbers $A = (\alpha_l)_{l\in L}$ so that for every $x \in X$, $\sum_{l|x\in F_l}\alpha_l \geqslant 1$.

Then it states that:

$$\sum_{e\in E}\prod_{l\in L} w_l(e_l) \leqslant \prod_{l\in L}\left(\sum_{e_l\in E_l} w_l(e_l)^{1/\alpha_l}\right)^{\alpha_l} .$$

The statement of the lemma we want to prove is obtained by setting:

- $X = I$, $E = \{\{i\} \mid i \in I\}$,
- $L = J$ and for every $j \in J$, $F_j = I$, as a consequence for every $e \in E$ and $j \in J$, $e_j = e$, and thus $E_j = E$.
- For each $j \in J$, we let $w_j(\{i\}) = a_{i,j}^{\omega_j}$.
- Finally, we let $A = (\omega_j)_{j\in J}$. As for every $i \in I$ and $j \in J$, we have that $i \in F_j$, the hypothesis that A must satisfy is a consequence of the hypothesis $\sum_{j\in J}\omega_j \geqslant 1$.

In this setting, the *Generalised Weighted Entropy Lemma* gives us:

$$\sum_{i\in I}\prod_{j\in J} a_{i,j}^{\omega_j} \leqslant \prod_{j\in J}\left(\sum_{i\in I}(a_{i,j}^{\omega_j})^{1/\omega_j}\right)^{\omega_j} = \prod_{j\in J}\left(\sum_{i\in I} a_{i,j}\right)^{\omega_j} .$$

Which is the expected inequality. ◀