# Evaluation Trade-Offs for Acyclic Conjunctive Queries

# Evaluation Trade-Offs for Acyclic Conjunctive Queries

**Ahmet Kara**
Universität Zürich, Switzerland

**Milos Nikolic**
University of Edinburgh, UK

**Dan Olteanu**
Universität Zürich, Switzerland

**Haozhe Zhang**
Universität Zürich, Switzerland

## Abstract

We consider the evaluation of acyclic conjunctive queries, where the evaluation time is decomposed into preprocessing time and enumeration delay. In a seminal paper at CSL'07, Bagan, Durand, and Grandjean showed that acyclic queries can be evaluated with linear preprocessing time and linear enumeration delay. If the query is free-connex, the enumeration delay becomes constant. Further prior work showed that constant enumeration delay can be achieved for arbitrary acyclic conjunctive queries at the expense of a preprocessing time that is characterised by the fractional hypertree width.

We introduce an approach that exposes a trade-off between preprocessing time and enumeration delay for acyclic conjunctive queries. The aforementioned prior works represent extremes in this trade-off space. Yet our approach also allows for the enumeration delay and the preprocessing time between these extremes, in particular the delay may lie between constant and linear time.

Our approach decomposes the given query into subqueries and achieves for each subquery a trade-off that depends on a parameter controlling the times for preprocessing and enumeration. The complexity of the query is given by the Pareto optimal points of a bi-objective optimisation program whose inputs are possible query decompositions and parameter values.

**2012 ACM Subject Classification** Theory of computation → Database query processing and optimization (theory); Information systems → Database views

**Keywords and phrases** acyclic queries, query evaluation, enumeration delay

**Digital Object Identifier** 10.4230/LIPIcs.CSL.2023.29

**Funding** This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588.

## 1 Introduction

The problem of query evaluation is central to databases, e.g., [19, 16, 12, 15]. Over the past five decades, extensive work has been conducted on finding algorithms for conjunctive query evaluation with increasingly lower computational complexity. This complexity is governed by notions of width, such as the treewidth [17], hypertree width [10, 13, 16], fractional edge cover number [2], and submodular width [14, 12]. Yet, this coarse analysis puts on the same par intrinsically hard queries and easy ones with the same asymptotic output size.

A finer analysis decomposes the complexity into preprocessing time and enumeration delay. The preprocessing time is the time to build a data structure that represents succinctly the query result. The enumeration delay is the maximum of three times: the time to output the first tuple in the query result, the time between outputting any two consecutive result tuples, and the time between the last result tuple and the end of the enumeration process [8].


▶ **Example 1.** The query $P(A, B) = R(A), S(B)$ computes the Cartesian product of the unary relations $R$ and $S$. The size of the query result is quadratic in the size of the relations, so any join algorithm needs at least quadratic time to compute this result. Yet the tuples in the query result can be enumerated with constant delay directly from the input relations.

The query $J(A, B) = R(A, C), S(B, C)$ computes the natural join of the binary relations $R$ and $S$ and projects away the join column $B$. It is conjectured that it is not possible to evaluate this query with constant enumeration delay after linear preprocessing time [3]. The query can be evaluated, however, with linear delay after no (or at most linear) preprocessing time [3] or with constant delay after quadratic preprocessing time [16].

Even though both queries $P$ and $J$ can have results of quadratic size, they differ in the amount of preprocessing required to guarantee constant-delay enumeration: Whereas $P$ does not need preprocessing, $J$ requires to compute the result in the preprocessing step. ⌟

There is a solid body of work on the trade-off between preprocessing time and enumeration delay for conjunctive queries. Any conjunctive query can be evaluated with $\mathcal{O}(N^{\mathsf{w}})$ preprocessing time and constant enumeration delay [16], where $N$ is the size of the input database and $\mathsf{w}$ is the width of the query; $\mathsf{w}$ generalises the fractional hypertree width [13] from Boolean to conjunctive queries with arbitrary free variables. Any acyclic conjunctive query admits linear preprocessing time and linear enumeration delay [3]. If the acyclic query is free-connex, the enumeration delay becomes constant; otherwise, the query cannot be evaluated with linear preprocessing time and constant enumeration delay, assuming the Boolean Matrix Multiplication conjecture [3]. A subclass of acyclic queries, called hierarchical, can be evaluated with $\mathcal{O}(N^{1+(\mathsf{w}-1)\epsilon})$ preprocessing time and $\mathcal{O}(N^{1-\epsilon})$ enumeration delay for any $\epsilon \in [0, 1]$ [11]. In this trade-off space, the preprocessing time ranges from $O(N)$ to $\mathcal{O}(N^{\mathsf{w}})$, while the enumeration delay ranges from $O(N)$ to constant. Conjunctive queries of bounded free-connex submodular width admit constant (in the database size, albeit non-polynomial in the query size) delay after a fixed-parameter tractable preprocessing step [5].

In this paper, we introduce an approach that defines a preprocessing-enumeration trade-off for *any* acyclic conjunctive query. Prior works are points in this trade-off space. Our approach can achieve lower query evaluation time than prior works in case only a fraction of the result is needed. For this, we pick one point in the trade-off space that defines the preprocessing time and the enumeration delay so that the overall computation time is the minimum across the entire space.

Our approach works as follows. Consider an acyclic query $Q$ with free variables $\mathcal{F}$ and a partial order $\omega$ of the variables of $Q$, such that the free variables come before the bound variables in $\omega$ [16]. We decompose $Q$ into queries that are *induced* by the free variables: For each free variable $X \in \mathcal{F}$, we construct an induced query $Q_X$, whose body is the join of all relations in Q subject to further simplifications by semi-join reductions and projections. The free variables of $Q_X$ are $X$ and the variables that come before $X$ in $\omega$ and on which $X$ depends (Section 2 defines dependent variables and Section 3 defines induced queries).

To enumerate the tuples in the result of $Q$, we use a chain of calls to the enumeration procedures for the induced queries in the order of the free variables $X_1, \ldots, X_n$ in $\omega$. We iterate over the result of $Q_{X_1}$ to enumerate the distinct values of $X_1$. We then iterate over the result of $Q_{X_2}$ to enumerate the distinct values of $X_2$ *given* a value for $X_1$. In general, we iterate over the result of $Q_{X_i}$ to enumerate the distinct values of $X_i$ given values for $X_1$ to $X_{i-1}$, as fixed by the iterators for $Q_{X_1}$ to $Q_{X_{i-1}}$. Once a tuple of values for all free variables is obtained, we backtrack.

For each induced query $Q_X$, we design a preprocessing-enumeration trade-off controlled by a parameter $\epsilon_X \in [0, 1]$ specific to $Q_X$. A low value for $\epsilon_X$ means low, down to linear preprocessing time at the expense of a high, up to linear delay. A high value for $\epsilon_X$ means

high preprocessing time to achieve a low, down to constant delay. To achieve a trade-off depending on $\epsilon_X$, we partition each relation on the join attributes such that the light parts have join values whose frequencies are below a threshold defined by $\epsilon_X$ while the heavy parts have join values with high frequencies. In the preprocessing phase, we then compute the query over the light parts of the relations. In the enumeration phase, we enumerate the tuples from the join of the parts where at least one part is heavy. The computation times for the two phases are given by $\mathcal{O}(N^{p(Q_X,\epsilon_X)})$ for preprocessing and $\mathcal{O}(N^{e(Q_X,\epsilon_X)})$ for enumeration delay, where $p(Q_X,\epsilon_X)$ and $e(Q_X,\epsilon_X)$ are called the preprocessing cost and enumeration cost, respectively. We can use the parameters for the induced queries to define a trade-off for the original query $Q$. To find a desired trade-off, we need to consider the set of possible variable orders for $Q$, as they may yield different sets of induced queries, as well as possible parameter values for the induced queries. Given a variable order $\omega$ for an acyclic query $Q$ and a set $\boldsymbol{\epsilon} = \{\epsilon_X \mid X \in \mathcal{F}\}$ of parameter values for the induced queries, we define the preprocessing and enumeration costs of $Q$ as the maximum preprocessing cost and enumeration cost, respectively, of the induced queries: $p(\omega,\boldsymbol{\epsilon}) = \max_{X\in\mathcal{F}}\ p(Q_X,\epsilon_X)$ and $e(\omega,\boldsymbol{\epsilon}) = \max_{X\in\mathcal{F}}\ e(Q_X,\epsilon_X)$.

Given two pairs $(p_1,e_1)$ and $(p_2,e_2)$, $(p_1,e_1) < (p_2,e_2)$ holds if (1) $p_1 \le p_2$, (2) $e_1 \le e_2$, and (3) at least one of the two inequalities is strict. A pair of preprocessing cost $p$ and enumeration cost $e$ is *Pareto optimal* if there are no trade-off parameters $\boldsymbol{\epsilon}$ and variable order $\omega$ such that $(p(\omega,\boldsymbol{\epsilon}), e(\omega,\boldsymbol{\epsilon})) < (p,e)$.

The Pareto optimal pairs of objective values of the following bi-objective *trade-off program* are possible preprocessing-enumeration costs for an acyclic query [1] $Q$ with free variables $\mathcal{F}$:

$$\begin{aligned} \text{minimise} \quad & \left( \max_{X\in\mathcal{F}}\ p(Q_X,\epsilon_X)\ ,\ \max_{X\in\mathcal{F}}\ e(Q_X,\epsilon_X) \right) \\ \text{subject to} \quad & \omega \text{ is a variable order for } Q \text{ and} \\ & Q_X \text{ is induced by } X \text{ wrt. } \omega,\ \forall X \in \mathcal{F} \text{ and} \\ & \epsilon_X \in [0,1],\ \forall X \in \mathcal{F} \end{aligned}$$

The minimisation of the program objective is based on the inequality ($<$) defined above. We denote the set of Pareto optimal pairs of objective values of this program by $\mu(Q)$. The complexity of our evaluation algorithm is given by any of these Pareto optimal values.

▶ **Theorem 2.** *Any acyclic conjunctive query $Q$ can be evaluated over a database of size $N$ with $\mathcal{O}(N^p)$ preprocessing time and $\mathcal{O}(N^e)$ enumeration delay, where $(p,e) \in \mu(Q)$.*

The exponents $p$ and $e$ are explained in detail in Sections 4.1 and 5.2. Our approach recovers prior works as corollaries, as they represent possible Pareto optimal pairs in $\mu(Q)$. It recovers the case of free-connex queries [3]:

▶ **Corollary 3.** *Any free-connex acyclic conjunctive query can be evaluated over a database of size $N$ with $\mathcal{O}(N)$ preprocessing time and $\mathcal{O}(1)$ enumeration delay.*

This is equivalent to stating that the pair $(1,0)$ of preprocessing and enumeration costs is in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program.

Our approach also recovers the more general result for arbitrary acyclic queries [3]:

[1] In this work we focus on acyclic queries with at least one free variable; in case of no free variables, there are no induced queries and the optimisation program is not well-defined. If an acyclic query has no free variables, it can be evaluated in linear time and the enumeration delay is trivially constant [3].

▶ **Corollary 4.** *Any acyclic conjunctive query can be evaluated over a database of size $N$ with $\mathcal{O}(N)$ preprocessing time and $\mathcal{O}(N)$ enumeration delay.*

This means that $(1, e) \in \mu(Q)$ for $e \leq 1$. Next, our approach recovers the result on constant enumeration delay from prior work on factorised representations for query results, when restricted to acyclic queries [16]:

▶ **Corollary 5.** *Any acyclic conjunctive query with fractional hypertree width $\mathsf{w}$ can be evaluated over a database of size $N$ with $\mathcal{O}(N^{\mathsf{w}})$ preprocessing time and $\mathcal{O}(1)$ enumeration delay.*

This corollary is implied by the fact that $(\mathsf{w}, 0) \in \mu(Q)$. Finally, our approach recovers the trade-off for hierarchical queries [11]:

▶ **Corollary 6.** *Any hierarchical query with fractional hypertree width $\mathsf{w}$ can be evaluated over a database of size $N$ with $\mathcal{O}(N^{1+(\mathsf{w}-1)\epsilon})$ preprocessing time and $\mathcal{O}(N^{1-\epsilon})$ enumeration delay for any $\epsilon \in [0, 1]$.*

We obtain the above result by showing that $(p, e) \in \mu(Q)$ with $p \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e \leq 1 - \epsilon$, for any $\epsilon \in [0, 1]$. In contrast to prior work [11], our approach uses different, more general evaluation strategies to account for the generality of acyclic queries. This generality comes at a price: Our trade-off does not have a closed-form expression for the computational complexity of preprocessing and enumeration as for hierarchical queries.

In this paper we use one approach to evaluate all induced queries. Our framework is however permissive and allows plugging in different evaluation strategies for different induced queries, e.g., evaluation strategies tailored specifically at path and star queries [7].

The structure of the paper is as follows. Section 2 introduces basic notions and tools. Section 3 explains our query decomposition technique. Sections 4 and 5 detail the preprocessing and enumeration phases. Section 6 compares our approach against two mainstream approaches by means of examples. Section 7 concludes with future work. Proofs of formal statements are deferred to Appendix A.

## 2 Preliminaries

### Data Model

A *schema* $\mathcal{X} = (X_1, \ldots, X_n)$ is a tuple of distinct variables. Each variable $X_i$ has a discrete domain $\mathsf{Dom}(X_i)$. We treat schemas and sets of variables interchangeably, assuming a fixed ordering of variables. A tuple $\mathbf{x}$ over schema $\mathcal{X}$ is an element from $\mathsf{Dom}(\mathcal{X}) = \mathsf{Dom}(X_1) \times \cdots \times \mathsf{Dom}(X_n)$.

A *relation* $R$ over schema $\mathcal{X}$ is a set of tuples over the same schema. The size of $R$ is given by the number of tuples in $R$. A database is a set of relations and has size given by the sum of the sizes of its relations.

Given a tuple $\mathbf{x}$ over schema $\mathcal{X}$ and $\mathcal{S} \subseteq \mathcal{X}$, $\mathbf{x}[\mathcal{S}]$ is the restriction of $\mathbf{x}$ onto $\mathcal{S}$. For a relation $R$ over $\mathcal{X}$, a schema $\mathcal{S} \subseteq \mathcal{X}$, and tuple $\mathbf{s} \in \mathsf{Dom}(\mathcal{S})$: $\sigma_{\mathcal{S}=\mathbf{s}}R = \{\, \mathbf{x} \mid \mathbf{x} \in R \wedge \mathbf{x}[\mathcal{S}] = \mathbf{s} \,\}$ is the set of tuples in $R$ that agree with $\mathbf{s}$ on the variables in $\mathcal{S}$; $\pi_{\mathcal{S}}R = \{\, \mathbf{x}[\mathcal{S}] \mid \mathbf{x} \in R \,\}$ is the set of restrictions of the tuples in $R$ to the variables in $\mathcal{S}$.

### Conjunctive Queries

A *conjunctive query* (CQ) is of the form

$$Q(\mathcal{F}) = R_1(\mathcal{X}_1), \ldots, R_n(\mathcal{X}_n).$$

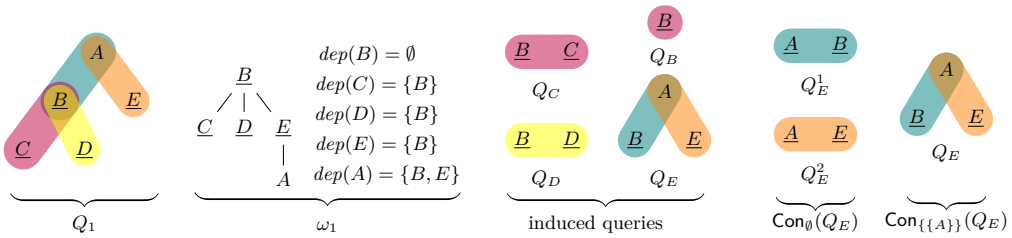



**Figure 1** Left to right: Hypergraph of the query $Q_1$ from Example 7; variable order $\omega_1$ for $Q_1$; the queries induced by the free variables in $\omega_1$; the $\emptyset$- and $\{\{A\}\}$-connected components of $Q_E$.

We denote by: $(R_i)_{i\in[n]}$ the relation symbols; $(R_i(\mathcal{X}_i))_{i\in[n]}$ the atoms; $vars(Q) = \bigcup_{i\in[n]} \mathcal{X}_i$ the set of variables; $free(Q) = \mathcal{F} \subseteq vars(Q)$ the set of free variables; and $atoms(Q) = \{R_i(\mathcal{X}_i) \mid i \in [n]\}$ the set of atoms in $Q$. The variables in $vars(Q) \setminus \mathcal{F}$ are called *bound*. Given a set $\mathcal{R}$ of atoms, we denote $vars(\mathcal{R}) = \bigcup_{R(\mathcal{X})\in\mathcal{R}} \mathcal{X}$.

The hypergraph of $Q$ is a multi-hypergraph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of variables in $Q$ and $\mathcal{E}$ contains a hyperedge over $\mathcal{X}$ for each atom $R(\mathcal{X})$ in $Q$. In the graphical representation of hypergraphs, we underline the free variables.

A set $\mathcal{V} \subseteq vars(Q)$ is called a *join v-set* if there are two distinct atoms $R_i(\mathcal{X}_i)$ and $R_j(\mathcal{X}_j)$ in $Q$ such that $\mathcal{X}_i \cap \mathcal{X}_j = \mathcal{V}$. We denote the set of join v-sets in $Q$ by $\mathsf{JVSets}(Q)$.

A CQ is *acyclic* (ACQ in short) if it admits a *join tree* where each node is an atom and if any two nodes have variables in common, then all nodes along the path between them also have these variables [19]. The query is *free-connex* acyclic if it is acyclic and stays acyclic after adding a fresh atom whose schema consists of the free variables of the query [6]. A query is *hierarchical* if for any two of its variables, either their sets of atoms are disjoint or one is contained in the other [18].

## Variable Orders

Given a CQ $Q$, two variables *depend* on each other if they occur in the same atom of the query. A *variable order* $\omega$ for $Q$ is a pair $(T, dep)$ [16] where:

- $T$ is a rooted forest with one node per variable in $Q$. The variables of each atom in $Q$ lie along the same root-to-leaf path in $T$. No bound variable is an ancestor of a free variable.
- The function $dep$ maps each variable $X$ to the subset of its ancestor variables in $T$ on which the variables in the subtree rooted at $X$ depend.

This type of variable orders were first introduced as d-tree extensions [16] and later called free-top variable orders [11]. We denote the set of variable orders of $Q$ by $\mathsf{VO}(Q)$. Using the language of hypertree decompositions, we call the set $\{X\} \cup dep(X)$ the *bag* of $\omega$ at $X$. In the graphical representation of variable orders, we underline the free variables.

▶ **Example 7.** Consider the query $Q_1(B, C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$. Figure 1 shows its hypergraph (left) and next to it a variable order $\omega_1$ for the query. Consider now the variant $Q_2(C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$ of $Q_1$ where the variable $B$ is bound. Its hypergraph is given in Figure 2 (left). The variable order $\omega_1$ from Figure 1 is not valid for $Q_2$, since the bound variable $B$ sits on top of the free variables $C$, $D$, and $E$. The variable order $\omega_2$ in Figure 2 is a valid variable order for $Q_2$. ⌟

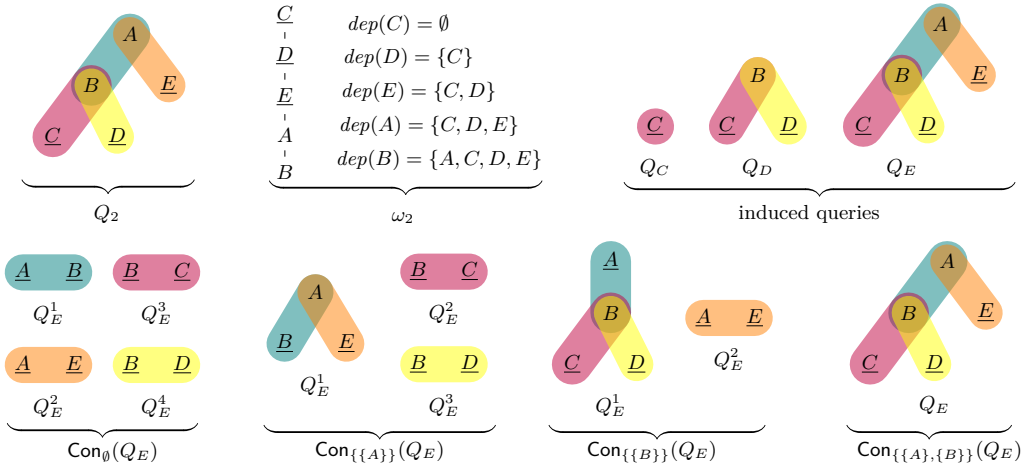



**Figure 2** First row, left to right: Hypergraph of the query $Q_2$ from Example 7; a variable order $\omega_2$ for $Q_2$; the queries induced by the free variables in $\omega_2$. Second row, left to right: The $\emptyset$-, $\{\{A\}\}$-, $\{\{B\}\}$-, and $\{\{A\},\{B\}\}$-connected components of $Q_E$.

## Width Measures

Given a CQ $Q$ and a set $\mathcal{V}$ of variables from $Q$, a *fractional edge cover* of $\mathcal{V}$ is a solution $\boldsymbol{\lambda} = (\lambda_{R(\mathcal{X})})_{R(\mathcal{X}) \in atoms(Q)}$ to the following linear program [2]:

$$\begin{aligned} \text{minimise} \quad & \sum_{R(\mathcal{X}) \in\, atoms(Q)} \lambda_{R(\mathcal{X})} \\ \text{subject to} \quad & \sum_{R(\mathcal{X}):\, X \in \mathcal{X}} \lambda_{R(\mathcal{X})} \geq 1 && \text{for all } X \in \mathcal{F} \text{ and} \\ & \lambda_{R(\mathcal{X})} \in [0, 1] && \text{for all } R(\mathcal{X}) \in atoms(Q) \end{aligned}$$

The optimal objective value of the above program is called the *fractional edge cover number* of $\mathcal{V}$ and denoted as $\rho^*_Q(\mathcal{V})$. We sometimes represent $Q$ by the set $\mathcal{E} = \{\mathcal{X} \mid \exists R(\mathcal{X}) \in atoms(Q)\}$ of the schemas of its atoms and write $\rho^*_{\mathcal{E}}(\mathcal{V})$. If $Q$ and $\mathcal{E}$ are clear from the context, we skip them in the expressions.

We define the *width* $\mathsf{w}(Q)$ of $Q$ as in prior work [11]:

$$\mathsf{w}(Q) = \min_{\omega \in \mathsf{VO}(Q)} \mathsf{w}(\omega), \text{ where}$$
$$\mathsf{w}(\omega) = \max_{X \in vars(Q)} \rho^*_Q(\{X\} \cup dep(X))$$

The measure $\mathsf{w}$ is equivalent to the fractional hypertree width when extended from Boolean to non-Boolean queries [16].

▶ **Example 8.** Consider the variable order $\omega_1$ for the query $Q_1$ in Figure 1. The bag at variable $A$ consists of the variables $A$, $B$, and $E$. It holds $\rho^*(\{A, B, E\}) = 2$, since we need two atoms to cover the variables in the bag. Similarly, for the bag $\{B, E\}$ at $E$, we have $\rho^*(\{B, E\}) = 2$. Each of the other bags $\mathcal{B}$ of the variable order can be covered by a single atom, therefore it holds $\rho^*(\mathcal{B}) = 1$. Hence, $\mathsf{w}(\omega_1) = 2$, which implies $\mathsf{w}(Q_1) \leq 2$. By checking other possible variable orders for $Q_1$, one can see that $\mathsf{w}(Q_1)$ equals 2.

Now, consider the variable order $\omega_2$ for the query $Q_2$ in Figure 2. The bag at $E$ consists of $C$, $D$, and $E$. We have $\rho^*(\{C, D, E\}) = 3$. It holds $\mathsf{w}(Q_2) = \mathsf{w}(\omega_2) = 3$. ⌟

### Partitioning

We partition relations based on the frequencies of their values. For a database $\mathcal{D}$, relation $R \in \mathcal{D}$ over schema $\mathcal{X}$, schema $\mathcal{S} \subset \mathcal{X}$, and threshold $\theta$, the pair $(R^{\mathcal{S}\to H}, R^{\mathcal{S}\to L})$ is a *partition* of $R$ on $\mathcal{S}$ with threshold $\theta$ if:

$$\text{(light part)} \quad R^{\mathcal{S}\to L} = \{\mathbf{x} \in R \mid \forall K \in \mathcal{D}, |\sigma_{\mathcal{S}=\mathbf{x}[\mathcal{S}]} K| < \theta\}$$
$$\text{(heavy part)} \quad R^{\mathcal{S}\to H} = R \setminus R^{\mathcal{S}\to L}$$

The relations $R^{\mathcal{S}\to H}$ and $R^{\mathcal{S}\to L}$ are called *heavy* and respectively *light* on the partition key $\mathcal{S}$. For $|\mathcal{D}| = N$ and a partition $(R^{\mathcal{S}\to H}, R^{\mathcal{S}\to L})$ of $R$ on $\mathcal{S}$ with threshold $\theta = N^\epsilon$ for $\epsilon \in [0,1]$, we have: (1) $\forall \mathbf{t} \in \pi_{\mathcal{S}} R^{\mathcal{S}\to L} : |\sigma_{\mathcal{S}=\mathbf{t}} R^{\mathcal{S}\to L}| < \theta = N^\epsilon$; and (2) $|\pi_{\mathcal{S}} R^{\mathcal{S}\to H}| \leq \frac{N}{\theta} = N^{1-\epsilon}$. The first bound follows from the light part condition. The second bound holds because each tuple in $\pi_{\mathcal{S}} R^{\mathcal{S}\to H}$ is paired with at least $\theta$ distinct tuples in at least one relation in the database. The database can contain at most $\frac{N}{\theta}$ such tuples over $\mathcal{S}$; otherwise, the database size exceeds $N$.

Given schemas $\mathcal{S}_1, \ldots, \mathcal{S}_n \subset \mathcal{X}$, an HL-signature *sig* for $R$ has the form $\{\mathcal{S}_1 \to s_1, \ldots, \mathcal{S}_n \to s_n\}$, where $s_i \in \{H, L\}$ for $i \in [n]$. The relation part $R^{sig}$ is defined as $\bigcap_{i\in[n]} R^{\mathcal{S}_i \to s_i}$.

### Computational Model

We consider the RAM model of computation. Each relation $R$ over schema $\mathcal{X}$ is implemented by a data structure that stores the tuples in $R$ using $O(|R|)$ space. This data structure can: (1) look up tuples in constant time; (2) enumerate all tuples in $R$ with constant delay; and (3) report $|R|$ in constant time. For a schema $\mathcal{S} \subset \mathcal{X}$, we use an index data structure that for any $\mathbf{s} \in \mathsf{Dom}(\mathcal{S})$ can: (4) enumerate all tuples in $\sigma_{\mathcal{S}=\mathbf{s}} R$ with constant delay; (5) check $\mathbf{s} \in \pi_{\mathcal{S}} R$ in constant time; and (6) return $|\sigma_{\mathcal{S}=\mathbf{s}} R|$ in constant time. Our complexity results are based on data complexity where the input query is assumed to be fixed.

## 3 Query Decomposition and Connected Components

Our approach decomposes queries following variable orders. Each query in such a decomposition is then split further into subqueries that are connected via specific join v-sets. In the following we introduce both concepts.

### 3.1 Decomposing A Query into Induced Queries

Given a CQ $Q(\mathcal{F})$, a variable order $\omega$ for $Q$, and $X \in \mathcal{F}$, we define the query $Q_X(\mathcal{F}_X)$ induced by $X$ wrt. $\omega$ as follows. The free variables of $Q_X$ are $\mathcal{F}_X = \{X\} \cup dep_\omega(X)$. The body of $Q_X$ is $\mathsf{bGYO}(Q, \mathcal{F}_X)$, where the function $\mathsf{bGYO}$ applies a variant of the GYO algorithm [20], which we call *bound-GYO* and explain next.

Given a CQ $Q$ and a variable set $\mathcal{F}_X$, bound-GYO replaces atoms in $Q$ by new atoms for fresh relations computed from the input relations. The algorithm repeats the following two rules as long as possible: (1) If a variable $Y \notin \mathcal{F}_X$ only occurs in one atom $R(\mathcal{X})$, replace $R(\mathcal{X})$ by the atom $R'(\mathcal{X} \setminus \{Y\})$, where $R'$ is obtained by projecting away $Y$ from $R$; (2) if there are two atoms $R(\mathcal{X})$ and $S(\mathcal{Y})$ with $\mathcal{X} \subseteq \mathcal{Y}$, remove the first atom and replace the second atom with a new atom $S'(\mathcal{Y})$ for a new relation $S'$ obtained from $S$ by filtering out the tuples that are not in $R$ ($S'$ is the result of the semi-join reduction of $S$ with $R$). All new relations in $\mathsf{bGYO}(Q, \mathcal{F}_X)$ can be computed in time linear in the size of the input database.

▶ **Example 9.** Consider the query $Q_1$ from Example 7. Figure 1 shows the hypergraphs of $Q_1$ and of the queries $Q_B$, $Q_C$, $Q_D$, and $Q_E$ induced by the free variables of $Q_1$ wrt. $\omega_1$. We explain the construction of $Q_E$. We first eliminate the variables $C$ and $D$ by computing $S' = \pi_B S$ and $T' = \pi_B T$. Then, we absorb $S'$ and $T'$ by computing $R' = (R \ltimes S') \ltimes T'$. We obtain $Q_E(B, E) = R'(A, B), U(A, E)$.

Now, consider the query $Q_2$ from Example 7. Figure 2 shows the hypergraph of $Q_2$ and of the queries induced by the free variables in $\omega_2$. The query $Q_E$ induced by $E$ is $Q_2$. The bound-GYO algorithm cannot apply any rule in this case, since all variables in $vars(Q_2) \setminus (\{E\} \cup dep_{\omega_2}(E)) = \{A, B\}$ are join variables and no atom has a schema that subsumes the schema of another atom. ⌟

## 3.2 Connected Components

Consider a CQ $Q(\mathcal{F})$ and a set $\mathcal{J} \subseteq \mathsf{JVSets}(Q)$ of join v-sets. A $\mathcal{J}$-path between two atoms $R(\mathcal{X})$ and $R'(\mathcal{X}')$ is a sequence $R_1(\mathcal{X}_1), \ldots, R_n(\mathcal{X}_n)$ of atoms such that $R_1(\mathcal{X}_1) = R(\mathcal{X})$, $R_n(\mathcal{X}_n) = R'(\mathcal{X}')$, and $\mathcal{X}_i \cap \mathcal{X}_{i+1} \in \mathcal{J}$ for $i \in [n-1]$. We say that a set $\mathcal{R} \subseteq atoms(Q)$ is *$\mathcal{J}$-connected* if there is a $\mathcal{J}$-path between any two atoms in $\mathcal{R}$. The set $\mathcal{R}$ is *maximally* $\mathcal{J}$-connected if it is $\mathcal{J}$-connected and any proper superset of $\mathcal{R}$ is not. Let $\mathcal{R}_1, \ldots, \mathcal{R}_k$ be the maximally $\mathcal{J}$-connected atom sets of $Q$. For each $i \in [k]$, consider the query $Q_i(\mathcal{F}_i)$ whose body consists of the atoms in $\mathcal{R}_i$ and whose free variable set $\mathcal{F}_i$ consists of $vars(\mathcal{R}_i) \cap (\mathcal{F} \cup \bigcup_{i \neq j} vars(\mathcal{R}_j))$. We call $Q_i(\mathcal{F}_i)$ a *$\mathcal{J}$-connected component* of $Q$ and denote the set of all $\mathcal{J}$-connected components of $Q$ by $\mathsf{Con}_{\mathcal{J}}(Q)$.

▶ **Example 10.** Consider the query $Q_E(B, E) = R'(A, B), U(A, E)$ depicted in Figure 1. We have $\mathsf{JVSets}(Q_E) = \{\{A\}\}$. Figure 1 shows the $\emptyset$- and the $\{\{A\}\}$-connected components of the query $Q_E$. The $\emptyset$-connected components of $Q_E$ are the queries $Q_E^1(A, B) = R'(A, B)$ and $Q_E^2(A, E) = U(A, E)$. The only $\{\{A\}\}$-connected component of $Q_E$ is $Q_E$ itself.

Now, consider the query $Q_E(C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$ depicted in Figure 2. In this case, we have $\mathsf{JVSets}(Q_E) = \{\{A\}, \{B\}\}$. Figure 2 shows the $\mathcal{J}$-connected components of $Q_E$ for any $\mathcal{J} \subseteq \{\{A\}, \{B\}\}$. The query has four $\emptyset$-connected components. Each of them contains in its body exactly one of the atoms in $Q_E$. The $\{\{B\}\}$-connected components are $Q_E^1(A, C, D) = R(A, B)$, $S(B, C)$, $T(B, D)$ and $Q_E^2(A, E) = U(A, E)$. The only $\{\{A\}, \{B\}\}$-connected component is $Q_E$. ⌟

# 4 Preprocessing

Given an ACQ $Q(\mathcal{F})$, a variable order $\omega$ for $Q$, and a parameter $\epsilon_X \in [0, 1]$ for each free variable $X$, we construct in the preprocessing stage a compact data structure that represents the result of $Q$. The data structure consists of the results of *skew-aware queries*, which are obtained from $Q$ by taking the frequencies of values in the input relations into account. We next explain the construction of skew-aware queries.

Given a query $Q_X$ induced by $X \in \mathcal{F}$ wrt. $\omega$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, consider the $\mathcal{J}$-connected components $\{Q_1, \ldots, Q_k\}$ of $Q_X$. For $i \in [k]$, the skew-aware query $S_{\mathcal{J}}(Q_i)$ is obtained from $Q_i$ by replacing each atom $R(\mathcal{X})$ in $Q_i$ with $R^{sig}(\mathcal{X})$, where $sig = \{\mathcal{V} \to L \mid \mathcal{V} \in \mathcal{J} \text{ and } \mathcal{V} \subseteq \mathcal{X}\} \cup \{\mathcal{V} \to H \mid \mathcal{V} \in \mathsf{JVSets}(Q_X) \setminus \mathcal{J} \text{ and } \mathcal{V} \subseteq \mathcal{X}\}$. That is, for any $\mathcal{V} \in \mathsf{JVSets}(Q_X)$ that is contained in schema $\mathcal{X}$ of $R$, it holds: The relation $R^{sig}$ is light on $\mathcal{V}$ if $\mathcal{V}$ is in $\mathcal{J}$ and heavy otherwise. The skew-aware query $S_{\mathcal{J}}(Q_i)$ has the same free variables as $Q_i$. Observe that each variable $Y$ that is free in at least two distinct skew-aware queries $S_{\mathcal{J}}(Q_i)$ and $S_{\mathcal{J}}(Q_j)$ must be part of a heavy join v-set. Otherwise, assume that all join

$\tau$(variable order $\omega$, free variables $\mathcal{F}$) : skew-aware queries

1 **let** $s_X = \emptyset$, $\forall X \in \mathcal{F}$
2 **foreach** $X \in \mathcal{F}$
3  **let** $Q_X$ be the query induced by $X$ wrt. $\omega$
4  **foreach** join v-set $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$
5  **let** $\{Q_1, \ldots, Q_k\}$ be the $\mathcal{J}$-connected components of $Q_X$
6  $s_X = s_X \cup \{\mathcal{J} \mapsto \{S_\mathcal{J}(Q_1), \ldots, S_\mathcal{J}(Q_k)\}\}$
7 **return** $\{s_X\}_{X \in \mathcal{F}}$

**Figure 3** Construction of skew-aware queries from a variable order $\omega$ and a variable set $\mathcal{F}$. The skew-aware query $S_\mathcal{J}(Q_i)$ results from $Q_i$ by replacing each relation in $Q_i$ by its part that is light on the join v-sets in $\mathcal{J}$ and heavy on all other join v-sets of the query $Q_X$ induced by $X$ wrt. $\omega$.

v-sets including $Y$ are light. By construction, this means that all such join v-sets are in $\mathcal{J}$. This implies that there is a $\mathcal{J}$-path between the atoms in $Q_i$ and $Q_j$ that include $Y$. This means that $Q_i$ and $Q_j$ are not maximally $\mathcal{J}$-connected, which contradicts our assumption that $Q_i$ and $Q_j$ are $\mathcal{J}$-connected components of $Q_X$.

▶ **Example 11.** The only $\{\{A\}\}$-connected component of the query $Q_E(B, E) = R'(A, B), U(A, E)$ in Figure 1 is $Q_E$ itself. The corresponding skew-aware query is $Q_E(B, E) = R'^{A \to L}(A, B), U^{A \to L}(A, E)$.

The $\emptyset$-connected components of $Q_E(C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$ in Figure 2 are $Q^1_E(A, B) = R(A, B)$, $Q^2_E(A, E) = U(A, E)$, $Q^3_E(B, C) = S(B, C)$, and $Q^4_E(B, D) = T(B, D)$. We obtain the skew-aware queries $Q^1_E(A, B) = R^{A \to H, B \to H}(A, B)$, $Q^2_E(A, E) = U^{A \to H}(A, E)$, $Q^3_E(B, C) = S^{B \to H}(B, C)$, and $Q^4_E(B, D) = T^{B \to H}(B, D)$. The $\{\{B\}\}$-connected components of $Q_E$ are $Q^1_E(A, C, D) = R(A, B), S(B, C), T(B, D)$ and $Q^2_E(A, E) = U(A, E)$. We obtain the skew-aware queries $Q^1_E(A, C, D) = R^{A \to H, B \to L}(A, B)$, $S^{B \to L}(B, C)$, $T^{B \to L}(B, D)$ and $Q^2_E(A, E) = U^{A \to H}(A, E)$. ⌟

The procedure $\tau(\omega, \mathcal{F})$ shown in Figure 3 constructs all skew-aware queries of the preprocessing stage. The procedure returns a set $\{s_X\}_{X \in \mathcal{F}}$ of maps, where $s_X$ maps each set of join v-sets of the query induced by $X$ wrt. $\omega$ to a set of skew-aware queries. We explain the construction in more detail. For each variable $X \in \mathcal{F}$, the procedure first constructs the query $Q_X$ induced by $X$ wrt. to $\omega$ (Line 3). For each $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, it creates the $\mathcal{J}$-connected components $\mathsf{Con}_\mathcal{J}(Q) = \{Q_1, \ldots, Q_k\}$ of $Q_X$ (Line 5). Then, it adds $\mathcal{J} \mapsto \{S_\mathcal{J}(Q_1), \ldots, S_\mathcal{J}(Q_k)\}$ to the map $s_X$, where each $S_\mathcal{J}(Q_i)$ is the skew-aware query obtained from $Q_i$ (Line 6). The procedure returns the set of maps $s_X$ with $X \in \mathcal{F}$ (Line 7).

Consider an ACQ $Q(\mathcal{F})$, a variable order $\omega$ for $Q$, $\epsilon_X \in [0, 1]$ for $X \in \mathcal{F}$, and a database of size $N$. In the preprocessing stage, we first eliminate all dangling tuples in the database using Yannakakis' algorithm [19]. For each free variable $X \in \mathcal{F}$, we then do the following. First, we compute the fresh relations in the induced query $Q_X$. Then, we partition the relations in $Q_X$ on each join v-set in $\mathsf{JVSets}(Q_X)$ using the threshold $N^{\epsilon_X}$. Finally, we compute the results of all skew-aware queries in $\tau(\omega, \mathcal{F})$.

We next show that the skew-aware queries in $\tau(\omega, \mathcal{F})$ represent the input query $Q(\mathcal{F})$.

▶ **Definition 12** (Skew-aware Composite Queries). *Given an ACQ $Q(\mathcal{F})$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q)$, consider the $\mathcal{J}$-connected components $\mathsf{Con}_\mathcal{J}(Q) = \{Q_1, \ldots, Q_k\}$ of $Q$ and the skew-aware queries $S_\mathcal{J}(Q_1)$, …, $S_\mathcal{J}(Q_k)$. The query $Q_\mathcal{J}(\mathcal{F}) = S_\mathcal{J}(Q_1), \ldots, S_\mathcal{J}(Q_k)$ is called a skew-aware composite query obtained from $Q$.*

Given an induced query $Q_X(\mathcal{F}_X)$, let $Q_X^{\cup}(\mathcal{F}_X)$ be the union of all skew-aware composite queries obtained from $Q_X$ and $Q_{\bowtie}(\mathcal{F}) = (Q_X^{\cup}(\mathcal{F}_X))_{X\in\mathcal{F}}$ the join of these unions. On any database, the result of $Q(\mathcal{F})$ is equal to the result of $Q_{\bowtie}(\mathcal{F})$:

▶ **Proposition 13.** *For any ACQ $Q(\mathcal{F})$, it holds $Q(\mathcal{F}) \equiv Q_{\bowtie}(\mathcal{F})$.*

## 4.1 Preprocessing Time

Consider the output $\{s_X\}_{X\in\mathcal{F}}$ of the procedure $\tau(\omega, \mathcal{F})$ in Figure 3. Assume that relations in the induced queries of $\omega$ are partitioned using the trade-off parameters $\{\epsilon_X\}_{X\in\mathcal{F}}$. The preprocessing time is dominated by the time to compute the skew-aware queries in $s_X(\mathcal{J})$ for any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$. Consider for some $X \in \mathcal{F}$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, a skew-aware query $\hat{Q}$ in $s_X(\mathcal{J})$. The atoms in the query are connected via light join keys. There are several ways to compute such a query. One strategy is to materialise the result of $\hat{Q}$ using factorised computation over all relations in the query [16]. Another strategy is to use a join plan where we join in one relation at a time using light join keys. We can also mix these strategies, i.e., we can use factorised computation over a subset of the relations and then join in, one by one, the remaining relations.

We formalise join strategies for a skew-aware query $\hat{Q}$ by the notion of a *cover*. A cover is a pair $(\mathcal{R}_1, \mathcal{R}_2)$ with $\mathcal{R}_1, \mathcal{R}_2 \subseteq \mathit{atoms}(\hat{Q})$ such that $\mathcal{R}_1$ is non-empty and $\mathcal{R}_1 \cup \mathcal{R}_2 = \mathit{atoms}(\hat{Q})$. We denote by $\mathsf{Cov}(\hat{Q})$ the set of all covers for $\hat{Q}$. A cover $(\mathcal{R}_1, \mathcal{R}_2)$ represents the strategy where we first use factorised computation to join the relations in $\mathcal{R}_1$ and then join in the relations in $\mathcal{R}_2$ using light join keys. Assuming that the database size is $N$ and the relations are partitioned using the threshold $N^{\epsilon_X}$, the time required by this strategy is $\mathcal{O}(N^{c+|\mathcal{R}_2|\epsilon_X})$, where $c = \max\{\mathsf{w}(\hat{Q}_1), \rho^*_{\hat{Q}_1}(\mathcal{V})\}$, $\mathcal{V} = \mathit{vars}(\mathcal{R}_1) \cap (\mathit{free}(\hat{Q}) \cup \mathit{vars}(\mathcal{R}_2))$, and $\hat{Q}_1$ is the query with free variables $\mathcal{V}$ whose body consists of the atoms in $\mathcal{R}_1$. The following cost measure $p$, called *preprocessing cost*, minimises the exponent of the computation time over all possible covers of the skew-aware query $\hat{Q}$:

$$p(\hat{Q}, \epsilon_X) = \min_{(\mathcal{R}_1,\mathcal{R}_2)\in\mathsf{Cov}(\hat{Q})} (c + |\mathcal{R}_2|\epsilon_X) \tag{1}$$

where $c$ is defined as above. The exponent of the overall preprocessing time is bounded by the maximal cost of any skew-aware query for the variable order $\omega$:

$$p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}}) = \max_{X\in\mathcal{F}} p(Q_X, \epsilon_X) \tag{2}$$

$$p(Q_X, \epsilon_X) = \max_{\mathcal{J}\subseteq\mathsf{JVSets}(Q)} \max_{\hat{Q}\in\mathsf{Con}_{\mathcal{J}}(Q)} p(\hat{Q}, \epsilon_X) \tag{3}$$

We state the preprocessing time of our approach using the preprocessing cost.

▶ **Proposition 14.** *Given an ACQ $Q(\mathcal{F})$, a variable order $\omega$ for $Q$, $\epsilon_X \in [0,1]$ for $X \in \mathcal{F}$, and a database of size $N$, the skew-aware queries in $\tau(\omega, \mathcal{F})$ can be computed in time $\mathcal{O}(N^p)$, where $p = p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$.*

▶ **Example 15.** Consider the query $Q_E(B, E) = R'(A, B), U(A, E)$ from Figure 1. For any $\epsilon_E \in [0, 1]$, a cover with minimal cost is $(\{R(A, B)\}, \{U(A, E)\})$, which implies $p(Q_E, \epsilon_E) = 1 + \epsilon_E$. The preprocessing cost for the query $Q_1$ from Example 7 is dominated by this cost. This leads to $\mathcal{O}(N^{1+\epsilon_E})$ overall preprocessing time.

Consider now the query $Q_E(C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$ from Figure 2. Depending on $\epsilon_E$, we can choose one of the two covers $(\{R(A, B)\}, \{S(B, C), T(B, D), U(A, E)\})$ and $(\{R(A, B), T(B, D), U(A, E)\}, \{S(B, C)\})$. This leads to $p(Q_E, \epsilon_E) = \min\{1 + 3\epsilon_E, 2 + \epsilon_E\}$. This cost dominates the preprocessing cost for the query $Q_2$ in Example 7. This implies $\mathcal{O}(N^{\min\{1+3\epsilon_E, 2+\epsilon_E\}})$ overall preprocessing time. ⌟

# 5 Enumeration

Consider an ACQ $Q(\mathcal{F})$ and a variable order $\omega$ for $Q$. In the preprocessing stage, we construct for each $X \in \mathcal{F}$, an induced query $Q_X$ with free variables $\{X\} \cup dep(X)$. In the enumeration phase, we consider $dep(X)$ as the *input* variables and $X$ as the *output* variable of $Q_X$. The main building block of our enumeration procedure is a mechanism that supports the enumeration of the $X$-values in the result of $Q_X$, given that the values of the input variables $dep(X)$ are fixed. Assume for now that we have such an enumeration mechanism for each induced query. We can enumerate the result of $Q$ as follows. We traverse the variables in $\omega$ in pre-order. The input variables of an induced query $Q_X$ are a subset of the variables above $X$ in $\omega$, so when we arrive at variable $X$, the input variables of $Q_X$ are already fixed by the output values of the induced queries for the variables above $X$. We use the enumeration mechanism of $Q_X$ to obtain the $X$-values paired with the fixed input values. Each tuple in the result of $Q$ is a concatenation of the output values of the induced queries. The overall enumeration delay for $Q$ is the sum of the enumeration delays for all induced queries.

▶ **Example 16.** Consider the query $Q_1$ and its variable order $\omega_1$ in Figure 1. The query $Q_1$ has four induced queries: $Q_B(B)$, $Q_C(B, C)$, $Q_D(B, D)$, and $Q_E(B, E)$. The query $Q_B$ has no input variables and the output variable $B$, whereas $Q_C$, $Q_D$, and $Q_E$ have the input variable $B$ and the output variables $C$, $D$, and respectively $E$. Following the variable order $\omega_1$, we enumerate the $B$-values from $Q_B$. For each such $B$-value $b$, we enumerate the $C$-, $D$-, and $E$-values that are paired with $b$ from $Q_C$, $Q_D$, and respectively $Q_E$. The concatenation of output values, one from each induced query, forms a tuple in the result of $Q_1$. The enumeration delay of $Q_1$ is the sum of the enumeration delays of the four induced queries. ⌟

## 5.1 Enumeration for an Induced Query

We now show the enumeration mechanism for an induced query $Q_X$. The preprocessing stage decomposes $Q_X$ into a set of skew-aware composite queries such that the result of $Q_X$ is the union of the results of these composite queries. We first show how to enumerate the result of one composite query and then the union of the results of all composite queries built for $Q_X$.

Consider a skew-aware composite query $Q_{\mathcal{J}}(\mathcal{F}) = Q_1(\mathcal{X}_1), \ldots, Q_k(\mathcal{X}_k)$ obtained from the $\mathcal{J}$-connected components of an induced query $Q_X(\mathcal{F})$ for some $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$. The query $Q_{\mathcal{J}}$ has the same free variables $\mathcal{F}$ as $Q_X$. We consider the output variable $O$ and input variables $\mathcal{I}$ of $Q_X$ as the output and respectively input variables of $Q_{\mathcal{J}}$. Given a tuple $\mathbf{i}$ over $\mathcal{I}$, we next show how to enumerate the $O$-values that are paired with $\mathbf{i}$ in the result of $Q_{\mathcal{J}}$.

**Case 1.** We first discuss the case that the output variable $O$ is a join variable in $Q_{\mathcal{J}}$. For each skew-aware query $Q'(\mathcal{X}') \in \{Q_1(\mathcal{X}_1), \ldots, Q_k(\mathcal{X}_k)\}$, we filter out all tuples in the result of $Q'$ that do not match the input tuple $\mathbf{i}$. The time to do this filtering is bounded by the size of the result of $Q'$ after fixing the input variables by $\mathbf{i}$. Then, we evaluate the skew-aware composite query $Q_{\mathcal{J}}$ over the filtered skew-aware queries.

By construction, each non-join variable is free, i.e., either an input or an output variable (non-join bound variables are removed by bound-GYO as shown in Section 3). Since $Q_{\mathcal{J}}$ has only one output variable $O$, which is a join variable in Case 1, all non-join variables of $Q_{\mathcal{J}}$ are input variables. These variables are fixed by the input tuple $\mathbf{i}$, so the output variable $O$ is the only free variable that is not fixed. This makes $Q_{\mathcal{J}}$ a free-connex query. Hence, for an input tuple $\mathbf{i}$, we can compute the first result tuple of $Q_{\mathcal{J}}$ in time linear in the output size of the filtered skew-aware queries and enumerate the remaining tuples with constant delay [3].

Given an input tuple, the enumeration delay for $Q_{\mathcal{J}}$ is thus linear in the size of the result of the skew-aware queries. Since the values of all non-join variables are fixed, this size is bounded by the size of the projections of the skew-aware queries onto their join variables:

$$\max_{\hat{Q} \in atoms(Q_{\mathcal{J}})} |\pi_{\hat{\mathcal{F}}} \hat{Q}|, \quad \text{where } \hat{\mathcal{F}} = free(\hat{Q}) \cap \bigcup_{\mathcal{V} \in \mathsf{JVSets}(Q_X)} \mathcal{V}. \tag{4}$$

The set $\hat{\mathcal{F}}$ consists of the free variables of $\hat{Q}$ that are part of the join v-sets of $Q_X$.

▶ **Example 17.** Consider the skew-aware composite query $Q(A, C, D, E) = Q_1(A, B)$, $Q_2(A, E)$, $Q_3(B, C)$, $Q_4(B, D)$, which joins the materialised skew-aware queries $Q_1$, $Q_2$, $Q_3$, and $Q_4$ and has the output variable $A$ and the input variables $C$, $D$, and $E$. For a given tuple $(c, d, e)$ over $(C, D, E)$, we show how to enumerate the $A$-values that are paired with $(c, d, e)$ in the result of $Q$. The output variable $A$ is a join variable. For each skew-aware query, we filter out all tuples that do not match $(c, d, e)$. For $Q_3$ and $Q_4$, we keep only the tuples that contain $c$ and respectively $d$, denoted as $Q_3(B, c)$ and $Q_4(B, d)$. For $Q_2$, we obtain $Q_2(A, e)$. The query $Q_1$ has no input variables, so no need for filtering. After the filtering, the query becomes $Q(A, c, d, e) = Q_1(A, B), Q_2(A, e), Q_3(B, c), Q_4(B, d)$. It is free-connex since $A$ is the only unfixed free variable. After the preprocessing stage that takes time bounded by the maximum of the sizes $|\pi_{A,B}(Q_1(A, B))|$, $|\pi_A(Q_2(A, E))|$, $|\pi_B(Q_3(B, C))|$, and $|\pi_B(Q_4(B, D))|$, we can enumerate the $A$-values with constant delay. ◿

**Case 2.** We now discuss the case where the output variable $O$ of the composite query $Q_{\mathcal{J}}$ is not a join variable. Let $Q_O(\mathcal{X}_O) \in \{Q_1(\mathcal{X}_1), \ldots, Q_k(\mathcal{X}_k)\}$ be the skew-aware query with $O \in \mathcal{X}_O$ and $\mathcal{F}_O \subseteq \mathcal{X}_O$ be the join variables in the schema of $Q_O$. We define a sub-query $Q_{\mathcal{F}_O}$ of $Q_{\mathcal{J}}$ by removing $Q_O$ from the body of $Q_{\mathcal{J}}$ and setting the schema of $Q_{\mathcal{F}_O}$ to be $(\mathcal{F} \setminus \mathcal{X}_O) \cup \mathcal{F}_O$, that is, removing the free variables of $Q_O$ from the schema of $Q_{\mathcal{J}}$ and adding the join variables $\mathcal{F}_O$ in $Q_O$. We set the new variables from $\mathcal{F}_O$ to be the output variables and other free variables, $(\mathcal{F} \setminus \mathcal{X}_O) \setminus \mathcal{F}_O$, to be the input variables of $Q_{\mathcal{F}_O}$.

These input variables of $Q_{\mathcal{F}_O}$ are a subset of the input variables $\mathcal{I}$ of $Q_{\mathcal{J}}$ since all variables in $\mathcal{F} \setminus \mathcal{X}_O$ are input variables of $Q_{\mathcal{J}}$ (the only output variable $O$ of $Q_{\mathcal{J}}$ is in $\mathcal{X}_O$ and removed). Hence, an input tuple $\mathbf{i}$ over $\mathcal{I}$ fixes the input variables of $Q_{\mathcal{F}_O}$. When these input variables are fixed, $Q_{\mathcal{F}_O}$ becomes free-connex. The reason is that by adding an atom with variables $\mathcal{X}_O$ to $Q_{\mathcal{F}_O}$, the query remains acyclic. All output variables of $Q_{\mathcal{F}_O}$ are join variables, so we can compute the $\mathcal{F}_O$-tuples in the result of $Q_{\mathcal{F}_O}$ as in the previous case. The computation time is as defined in Equation (4). Then, we can enumerate the distinct output of $Q_{\mathcal{J}}$ using the skew-aware query $Q_O$, as explained next.

For the given input tuple $\mathbf{i}$ over $\mathcal{I}$ and for each output $\mathcal{F}_O$-tuple $\mathbf{t}$ enumerated from $Q_{\mathcal{F}_O}$ for $\mathbf{i}$, all variables in $Q_O$ except $O$ are fixed by $\mathbf{i}$ and $\mathbf{t}$, therefore, the skew-aware query $Q_O$ supports constant-time lookups and constant-delay enumeration of the matching $O$-values in the result of $Q_O$. We decompose $Q_O$ into a union of queries instantiated for the distinct $\mathcal{F}_O$-tuples and the input tuple $\mathbf{i}$. From each query, instantiated for an $\mathcal{F}_O$-tuple $\mathbf{t}$ and the input tuple $\mathbf{i}$, we can enumerate with constant delay the $O$-values that are paired with $\mathbf{t}$ and $\mathbf{i}$ in the result of $Q_O$. The $O$-values from these instantiated queries might not be disjoint, so we cannot enumerate the $O$-values from each query one after the other. To enumerate the distinct $O$-values, we employ the Union algorithm [9]. The enumeration delay is linear in the sum of the enumeration delays of the instantiated queries. Since each instantiated query supports constant-delay enumeration, the enumeration delay over all instantiated queries is linear in the number of distinct $\mathcal{F}_O$-tuples, which is bounded by the size defined by Expression (4). Overall, the enumeration delay of the result of $Q_{\mathcal{J}}$ in this case is bounded by the size defined by Expression (4).

▶ **Example 18.** Consider the induced query $Q_E$ and its $\emptyset$-connected components in Figure 2. The skew-aware composite query is of the form: $Q_E^{\emptyset}(C, D, E) = Q_1(A, B), Q_2(A, E), Q_3(B, C), Q_4(B, D)$, where $Q_1, \ldots, Q_4$ are the skew-aware queries that are heavy on the join v-sets $\{A\}$ and $\{B\}$. The query has the input variables $C$ and $D$ and the output variable $E$. The output variable $E$ is not a join variable. We build a sub-query $Q_A$ from $Q_E^{\emptyset}$ by removing the query $Q_2$, which contains the output variable $E$, and setting the join variable $A$ to be the output variable of $Q_A$. The sub-query is then $Q_A(A, C, D) = Q_1(A, B), Q_3(B, C), Q_4(B, D)$. Given an input tuple $(c, d)$ over $(C, D)$, we retain only the tuples from the skew-aware queries $Q_1$, $Q_3$, and $Q_4$ that match $(c, d)$ to obtain the query $Q_A(A, c, d) = Q_1(A, B), Q_3(B, c), Q_4(B, d)$. The query $Q_A(A, c, d)$ is free-connex since $A$ is the only unfixed free variable. Preparing for the enumeration of the $A$-value from $Q_A$ takes time linear in the size of the filtered skew-aware queries in $Q_A$, $\mathcal{O}(\max(|\pi_{A,B}(Q_1(A, B))|, |\pi_B(Q_3(B, C))|, |\pi_B(Q_4(B, D))|))$. We can then enumerate the $A$-values in the result of $Q_A(A, c, d)$ with constant delay [3]. For each such $A$-value $a$, we can enumerate the $E$-values paired with $a$ in $Q_2(A, E)$ with constant delay. The output values of $Q_E^{\emptyset}$ are over the schema $(A, E)$ and might contain duplicates of $E$-values. We apply the Union algorithm [9] to enumerate only the distinct $E$-values with the delay linear in the number of distinct $A$-values, which is bounded by $\mathcal{O}(\pi_A(Q_1(A, B)))$. Since $|\pi_A(Q_1(A, B))| \leq |\pi_{A,B}(Q_1(A, B)|$, the delay for the enumeration of the output values of $Q_E^{\emptyset}$ is bounded by $\mathcal{O}(\max(|\pi_{A,B}(Q_1(A, B))|, |\pi_B(Q_3(B, C))|, |\pi_B(Q_4(B, D))|))$. ⌟

So far, we discussed the enumeration of the result of one skew-aware composite query. To enumerate the result of an induced query, we need to enumerate the union of the results of all skew-aware composite queries obtained from the induced query. Since these results might not be disjoint, we again apply the Union algorithm [9]. The enumeration delay is the maximum of the enumeration delays of all skew-aware composite queries.

Overall, the enumeration delay for a skew-aware composite query is bounded by the size of the projections of its skew-aware queries onto their join variables, as defined in Expression (4). The enumeration delay for an induced query is the maximum of the enumeration delays of its skew-aware composite queries, and the enumeration delay for the input ACQ is the maximum of the enumeration delays of its induced queries. Hence, the enumeration delay of an ACQ $Q(\mathcal{F})$ is given by:

$$\max_{X \in \mathcal{F}} \max_{\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)} \max_{\hat{Q} \in \mathsf{Con}_{\mathcal{J}}(Q_X)} |\pi_{\hat{\mathcal{F}}} S_{\mathcal{J}}(\hat{Q})|, \tag{5}$$

where $S_{\mathcal{J}}(\hat{Q})$ is the skew-aware query obtained from $\hat{Q}$ and $\hat{\mathcal{F}} = \mathit{free}(\hat{Q}) \cap \bigcup_{\mathcal{V} \in \mathsf{JVSets}(Q_X)} \mathcal{V}$.

## 5.2 Enumeration Delay

Let $\{s_X\}_{X \in \mathcal{F}}$ be the output of the procedure $\tau(\omega, \mathcal{F})$ in Figure 3 for some variable order $\omega$ and a set $\mathcal{F}$ of free variables. Assume that the relations are partitioned using parameter values $\{\epsilon_X\}_{X \in \mathcal{F}}$. Consider a skew-aware query $Q \in s_X(\mathcal{J})$ for some $X \in \mathcal{F}$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$. Expression (5) implies that the overall delay is bounded by the maximal size of the projection of the result of such a query $Q$ onto $\hat{\mathcal{F}} = \mathit{free}(Q) \cap \bigcup_{\mathsf{JVSet} \in \mathsf{JVSets}(Q_X)} \mathsf{JVSet}$, i.e., the free variables of $Q$ that are part of join v-sets of $Q_X$. We give three bounds on the exponent $e$ of the size complexity: 1) One bound on $e$ is given by $p(Q, \epsilon_X)$, where $p$ is the preprocessing cost defined in Equation (1). 2) Another bound is $\rho^*_{Q_X}(\hat{\mathcal{F}})$, i.e., the fractional edge cover number of $\hat{\mathcal{F}}$. 3) The relations in $Q$ are heavy on the join v-sets of $Q_X$ that are included in $\hat{\mathcal{F}}$. This means that for each relation $R$ in $Q$ and $\mathsf{JVSet} \in \mathsf{JVSets}(Q_X)$, we have $|\pi_{\mathsf{JVSet}} R| \leq N^{1-\epsilon_X}$.

Hence, we can bound $e$ by $\rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})(1-\epsilon_X)$, i.e., the minimal number of join v-sets covering all variables in $\hat{\mathcal{F}}$ multiplied by $1-\epsilon_X$. We define the enumeration cost $e$ of $Q_X$, $Q$, and $\epsilon_X$ as the minimum of these three bounds:

$$e(Q_X, Q, \epsilon_X) = \min\{\, p(Q, \epsilon_X),\ \rho^*_{Q_X}(\hat{\mathcal{F}}),\ \rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})(1-\epsilon_X)\,\}, \tag{6}$$

The overall enumeration cost is given by the maximum $e(Q_X, Q, \epsilon_X)$ over all induced and skew-aware queries:

$$e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}}) = \max_{X\in\mathcal{F}} e(Q_X, \epsilon_X) \tag{7}$$

$$e(Q_X, \epsilon_X) = \max_{\mathcal{J}\subseteq\mathsf{JVSets}(Q_X)} \max_{\hat{Q}\in\mathsf{Con}_{\mathcal{J}}(Q_X)} e(Q_X, \hat{Q}, \epsilon_X) \tag{8}$$

We state the delay of our approach for a given variable order and trade-off parameters.

▶ **Proposition 19.** *Given an ACQ $Q(\mathcal{F})$, a variable order $\omega$ for $Q$, $\epsilon_X \in [0,1]$ for $X \in \mathcal{F}$, and a database of size $N$, the result of $Q$ can be enumerated from the queries in $\tau(\omega, \mathcal{F})$ with $\mathcal{O}(N^e)$ delay, where $e = e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$.*

▶ **Example 20.** Consider the induced query $Q_E$ from Figure 2 and the skew-aware composite query $Q^{\emptyset}_E(C,D,E) = Q_1(A,B), Q_2(B,C), Q_3(B,D), Q_4(A,E)$ obtained from its $\emptyset$-connected components. The query $Q^{\emptyset}_E(C,D,E)$ has the join variables $\hat{\mathcal{F}} = \{A,B\}$. Let $\epsilon_E \in [0,1]$.

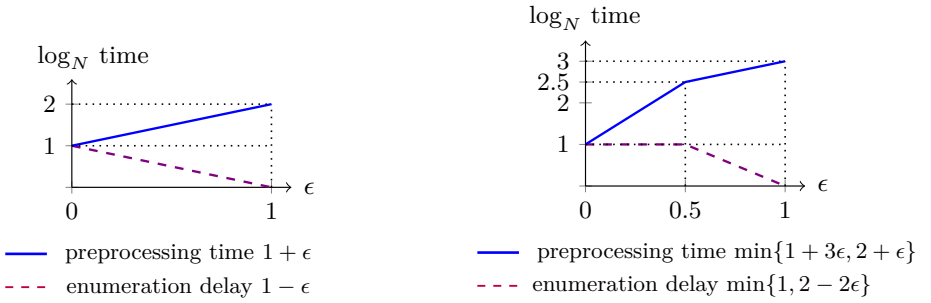
We compute the enumeration cost $e(Q_E, Q_1, \epsilon_E)$ of $Q_1$ as defined in Equation (6). Since the body of $Q_1$ consists of a single atom that contains $A$ and $B$, we have $p(Q_1, \epsilon_E) = 1$ and $\rho^*_{Q_E}(A,B) = 1$. Since $A$ and $B$ can be covered by two join v-sets of $Q_E$, we get $\rho^*_{\mathsf{JVSets}(Q_E)}(A,B)(1-\epsilon_E) = 2(1-\epsilon_E) = 2-2\epsilon_E$. Overall, we obtain $e(Q_E, Q_1, \epsilon_E) = \min\{1, 2-2\epsilon\}$. The query $Q_2$ consists of one atom. The only join variable in its schema is $B$. Hence, the enumeration cost of $Q_2$ is $e(Q_E, Q_2, \epsilon_E) = 1 - 1\epsilon_E$. Similarly, the enumeration cost for $Q_3$ is $e(Q_E, Q_3, \epsilon_E) = 1 - 1\epsilon_E$. The skew-aware query $Q_4$ has no join variables. Its enumeration cost is therefore constant.

The maximum of the enumeration costs for the four skew-aware queries considered above is $\max\{\min\{1, 2-2\epsilon_E\}, 1-\epsilon_E, 0\} = \min\{1, 2-2\epsilon_E\}$. This cost dominates the enumeration costs of all induced queries of the query $Q_2$ in Example 7. This implies $\mathcal{O}(N^{\min\{1,2-2\epsilon\}})$ overall enumeration delay. ⌟

## 6 Benefits of Our Approach

In this section we exemplify the benefits of our approach versus two mainstream approaches dubbed lazy and eager. The lazy approach invests no time in the preprocessing phase at the expense of linear enumeration delay [3]. The eager approach constructs a factorised representation of the query result in time $\mathcal{O}(N^{\mathsf{w}})$, where $\mathsf{w}$ is the width of the query, after which it needs constant enumeration delay [16]. In case we only need to enumerate a fraction of the query result, our approach can be asymptotically faster than both competing approaches. Furthermore, if the fraction is known in advance, we can derive the trade-off parameters for the lowest overall complexity to compute this fraction of the query result.

▶ **Example 21.** Consider the query $Q_1(B,C,D,E) = R(A,B), S(B,C), T(B,D), U(A,E)$ from Example 7 whose hypergraph is depicted in Figure 1 (left). The query has width $\mathsf{w} = 2$ (see Example 8). Assume that the input relations have size $N$. Our approach achieves $\mathcal{O}(N^{1+\epsilon})$ preprocessing time (Example 15) and $\mathcal{O}(N^{1-\epsilon})$ enumeration delay for any $\epsilon \in [0,1]$, as depicted in Figure 4 (top left). These complexities are obtained by using the




| $\gamma$ | 0 | $\frac{1}{2}$ | 1 | $1\frac{1}{2}$ | 2 | $2\frac{1}{2}$ | 3 |
|---|---|---|---|---|---|---|---|
| lazy | 1 | $1\frac{1}{2}$ | 2 | $2\frac{1}{2}$ | 3 | $3\frac{1}{2}$ | 4 |
| eager | 2 | 2 | 2 | 2 | 2 | $2\frac{1}{2}$ | 3 |
| ours | 1 | $1\frac{1}{4}$ | $1\frac{1}{2}$ | $1\frac{3}{4}$ | 2 | $2\frac{1}{2}$ | 3 |
| $\epsilon$ | 0 | $\frac{1}{4}$ | $\frac{1}{2}$ | $\frac{3}{4}$ | 1 | 1 | 1 |

| $\gamma$ | 0 | $\frac{1}{2}$ | 1 | $1\frac{1}{2}$ | 2 | $2\frac{1}{2}$ | 3 |
|---|---|---|---|---|---|---|---|
| lazy | 1 | $1\frac{1}{2}$ | 2 | $2\frac{1}{2}$ | 3 | $3\frac{1}{2}$ | 4 |
| eager | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| ours | 1 | $1\frac{1}{2}$ | 2 | $2\frac{1}{2}$ | $2\frac{2}{3}$ | $2\frac{5}{6}$ | 3 |
| $\epsilon$ | 0 | 0 | 0 | 0 | $\frac{2}{3}$ | $\frac{5}{6}$ | 1 |

**Figure 4** Top: Evaluation trade-offs for the query $Q_1$ from Example 21 (left) and the query $Q_2$ from Example 22 (right). Bottom: The overall evaluation times achieved by the lazy, eager, and our approaches in case we want to enumerate $N^\gamma$ output tuples of $Q_1$ (left) and of $Q_2$ (right). The last row gives the $\epsilon$ values at which we achieve the complexities of our approach.

same trade-off parameter value $\epsilon$ for all induced queries. The lazy approach achieves linear enumeration delay after constant preprocessing time, while the eager approach achieves constant enumeration delay after $\mathcal{O}(N^2)$ preprocessing time. Our approach recovers the lazy approach[2] at $\epsilon = 0$ and the eager approach at $\epsilon = 1$. For any $\epsilon \in (0, 1)$, our approach allows for new trade-offs between preprocessing and enumeration.

The query result has at most $N^3$ tuples. Assume that we want to compute $N^\gamma$ tuples from the query result for $0 \leq \gamma \leq 3$. The second to fourth rows of the bottom left table in Figure 4 give the exponents of the overall evaluation times achieved by the lazy, eager, and our approaches for different values of $\gamma$. The last row gives the $\epsilon$ values at which we achieve the complexities of our approach. For instance, for $\gamma = \frac{1}{2}$, the lazy approach needs $\mathcal{O}(N \cdot N^{\frac{1}{2}}) = \mathcal{O}(N^{1\frac{1}{2}})$, the eager approach needs $\mathcal{O}(N^2 + N^{\frac{1}{2}}) = \mathcal{O}(N^2)$, and our approach needs $\mathcal{O}(N^{\frac{1}{4}} + N^{\frac{3}{4}}N^{\frac{1}{2}}) = \mathcal{O}(N^{1\frac{1}{4}})$ time. In case $\gamma$ is equal to $\frac{1}{2}$, 1, or $1\frac{1}{2}$, the computation time of our approach is strictly better than of the lazy and eager approaches. For the other cases shown in the table, our approach recovers the best of the other two approaches. ⌟

▶ **Example 22.** Consider the variant $Q_2$ of the query $Q_1$ from Example 21 where the variable $B$ is bound. Its hypergraph is shown in Figure 2 (left). This query has width 3 (Example 8). Assume that the input relations have size $N$. We explained in Examples 15 and 20 that our approach achieves $\mathcal{O}(N^{\min\{1+3\epsilon, 2+\epsilon\}})$ preprocessing time and $\mathcal{O}(N^{\min\{1,2-2\epsilon\}})$ enumeration delay for any $\epsilon \in [0, 1]$, cf. Figure 4 (top right). Just like in case of $Q_1$, we obtain these complexities by using the same trade-off parameter value for all induced queries. We recover prior work using $\epsilon = 0$ (lazy) and $\epsilon = 1$ (eager). For $\epsilon \in (\frac{1}{2}, 1)$, we obtain new trade-offs.

[2] Our approach requires at least linear preprocessing time. In case the delay is linear, the preprocessing time is also linear and we can shift it to the enumeration of the first tuple to match the constant preprocessing time of the lazy approach.

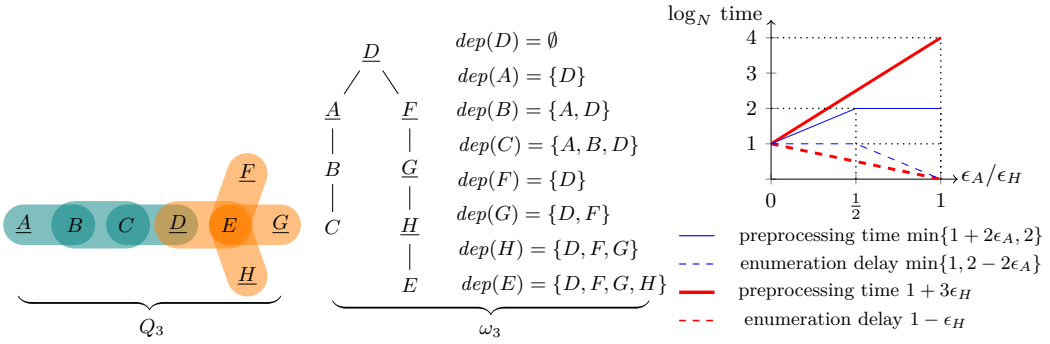



**Figure 5** Left to right: Hypergraph of the query $Q_3$ in Example 23; variable order $\omega_3$ for $Q_3$; trade-offs for the induced queries $Q_A$ (thin blue lines) and $Q_H$ (thick red lines).

There can be at most $N^3$ tuples in the result of $Q_2$. The bottom right table in Figure 4 gives the exponents of the overall computation times for the lazy, eager and our approaches for computing $N^\gamma$ tuples in the query result. In case $\gamma$ equals 2 or $2\frac{1}{2}$, the overall computation time of our approach is strictly lower than of the other approaches. In all other cases considered in the table, our approach recovers the best of the other two approaches. ⌟

The next example illustrates two aspects of our approach. First, even though our approach may not achieve novel trade-offs for some induced queries for a given acyclic query $Q$, it may still achieve new trade-offs for the entire query $Q$. Second, in case we are given a budget for one of the preprocessing and enumeration costs, we can pick the trade-off parameters so as to optimise for the other cost.

▶ **Example 23.** Consider the query $Q_3(A, D, F, G, H) = R(A, B), S(B, C), T(C, D), U(D, E), V(E, F), W(E, G), X(E, H)$ visualised in Figure 5 (left) and its variable order $\omega_3$ in Figure 5 (middle). The query has width 4. We focus on the queries induced by the variables $A$ and $H$, since their complexities dominate the overall complexity of $Q_3$. The query induced by $A$ is the path query $Q_A(A, D) = R(A, B), S(B, C), T'(C, D)$ and has width 2. The query induced by $H$ is the star query $Q_H(D, F, G, H) = U'(D, E), V(E, F), W(E, G), X(E, H)$ and has width 4. The relations $T'$ and $U'$ are computed by bound-GYO from $T$ and $U$ through semi-join reductions with $U$ and $T$, respectively. For both induced queries, the lazy approach achieves linear enumeration delay after constant preprocessing time. The eager approach achieves constant enumeration delay after $\mathcal{O}(N^2)$ preprocessing time for $Q_A$ and after $\mathcal{O}(N^4)$ preprocessing time for $Q_H$. For any $\epsilon_H \in [0, 1]$, our approach evaluates $Q_H$ with $\mathcal{O}(N^{1+3\epsilon_H})$ preprocessing time and $\mathcal{O}(N^{1-\epsilon_H})$ enumeration delay, as visualised with thick red lines in Figure 5 (right). For any $\epsilon_A \in [0, 1]$, it takes $\mathcal{O}(N^{\min\{1+2\epsilon_A, 2\}})$ preprocessing time and $\mathcal{O}(N^{\min\{1, 2-2\epsilon_A\}})$ enumeration delay for $Q_A$, as depicted with thin blue lines in Figure 5 (top right). For both queries, it recovers the lazy approach at $\epsilon_A = \epsilon_H = 0$ and the eager approach at $\epsilon_A = \epsilon_H = 1$. For any $\epsilon_H \in (0, 1)$, it allows for new trade-offs for $Q_H$ beyond the aforementioned ones of the existing approaches. For instance, for $\epsilon_H = \frac{1}{2}$, it achieves $\mathcal{O}(N^{2\frac{1}{2}})$ preprocessing time, which is less than the preprocessing time of the eager approach and $\mathcal{O}(N^{\frac{1}{2}})$ enumeration delay, which is less than the delay of the lazy approach.

Our approach does not achieve new trade-offs for the induced query $Q_A$, but it still achieves new trade-offs for the whole query $Q_3$. The lazy and eager approaches achieve for $Q_3$ the same trade-offs as for $Q_H$. Our approach achieves $\mathcal{O}(N^p)$ preprocessing time and $\mathcal{O}(N^e)$ delay, where $p = \max\{\min\{1 + 2\epsilon_A, 2\}, 1 + 3\epsilon_H\}$ and $e = \max\{\min\{1, 2 - 2\epsilon_A\}, 1 - \epsilon_H\}$. This means that for $\epsilon_A = 1$ and $\epsilon_H \in (\frac{1}{2}, 1)$, we obtain new trade-offs for $Q_3$.

Now, assume that we have a preprocessing cost budget of $2\frac{1}{2}$, i.e., we can afford $\mathcal{O}(N^{2\frac{1}{2}})$ preprocessing time. By setting $\epsilon_H = \frac{1}{2}$, we achieve the lowest enumeration cost of $\frac{1}{2}$ for $Q_H$. The preprocessing cost of $Q_A$ does not reach $2\frac{1}{2}$ for any $\epsilon_A$. So, one possibility is to set $\epsilon_A = 1$ to obtain an enumeration cost of 0 for $Q_A$ and an overall enumeration cost of $\frac{1}{2}$. If we have an enumeration cost budget of $\frac{1}{4}$, we can set $\epsilon_H = \frac{3}{4}$ to obtain the lowest preprocessing cost of $3\frac{1}{4}$ for $Q_H$. Since the preprocessing cost of $Q_A$ never reaches $3\frac{1}{4}$, we set $\epsilon_A = 1$ to achieve 0 enumeration cost for $Q_A$ and an overall enumeration cost of $\frac{1}{4}$. ⌟

## 7 Conclusion

In this paper we introduce an evaluation approach for acyclic conjunctive queries that trades off between the preprocessing time and the enumeration delay. This trade-off space includes points representing prior works and also points corresponding to novel evaluation strategies where the enumeration delay lies between linear and constant. Our approach can be extended to arbitrary conjunctive queries: in the preprocessing phase, we either fully materialise induced queries with cycles or do nothing. For such cyclic induced queries, the trade-off space of our approach thus only consists of two points where the delay is constant or as high as needed to compute the cycles in the query. In future work, we would like to generalise our approach to queries with aggregates over arbitrary semirings [1].

## References


1 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions Asked Frequently. In *PODS*, pages 13–28, 2016.

2 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *CSL*, pages 208–222, 2007.

4 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020.

5 Christoph Berkholz and Nicole Schweikardt. Constant delay enumeration with fpt-preprocessing for conjunctive queries of bounded submodular width. In *MFCS*, pages 58:1–58:15, 2019.

6 Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013.

7 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Enumeration algorithms for conjunctive queries with projection. In *ICDT*, pages 14:1–14:17, 2021.

8 Arnaud Durand and Etienne Grandjean. First-order Queries on Structures of Bounded Degree are Computable with Constant Delay. *ACM Trans. Comput. Logic*, 8(4):21, 2007.

9 Arnaud Durand and Yann Strozecki. Enumeration complexity of logical query problems with second-order variables. In *CSL*, pages 189–202, 2011.

10 Georg Gottlob, Matthias Lanzinger, Reinhard Pichler, and Igor Razgon. Complexity analysis of generalized and fractional hypertree decompositions. *J. ACM*, 68(5):38:1–38:50, 2021.

11 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in Static and Dynamic Evaluation of Hierarchical Queries. In *PODS*, pages 375–392, 2020.

12 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 *PODS*, pages 429–444, 2017.

13 Dániel Marx. Approximating fractional hypertree width. *ACM Trans. Alg.*, 6(2):29:1–29:17, 2010.

14 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.

15 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.

16 Dan Olteanu and Jakub Závodný. Size Bounds for Factorised Representations of Query Results. *ACM TODS*, 40(1):2:1–2:44, 2015.

17 Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

18 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

19 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

20 Clement T. Yu and Meral Z. Ozsoyoglu. An algorithm for tree-query membership of a distributed query. In *COMPSAC*, pages 306–312, 1979.


# A Appendix: Proofs

## A.1 Proof of Theorem 2

Consider an ACQ $Q(\mathcal{F})$ and a database of size $N$. Assume that the pair $(p, e)$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program in Section 1. This means that there is a variable order $\omega$ for $Q$ and a set $\{\epsilon_X\}_{X\in\mathcal{F}}$ of trade-off parameters such that $p = p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (2)) and $e = e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (7)). By Propositions 14 and 19, the query $Q$ can be evaluated with $\mathcal{O}(N^p)$ preprocessing time and $\mathcal{O}(N^e)$ delay.

## A.2 Proof of Corollary 3

Consider a free-connex ACQ $Q(\mathcal{F})$. We show that $(1, 0)$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of our trade-off program in Section 1.

Since $Q$ is free-connex acyclic, it admits a variable order $\omega$ of width 1 [4]. Consider an arbitrary free variable $X$ in $\omega$. Since the width of $\omega$ is 1, the variables $\{X\} \cup dep_\omega(X)$ are covered by a single atom of $Q$. This means that the body of the induced query $Q_X$ consists of a single atom whose variables are free, i.e., the induced query is of the form $Q_X(\mathcal{X}) = R(\mathcal{X})$. This implies that $\mathsf{JVSets}(Q_X) = \emptyset$. The only $\emptyset$-connected component of $Q_X$ is $Q_X$ itself. This means $p(Q_X, \epsilon_X) = 1$ (Equation (1)) for any $\epsilon_X \in [0, 1]$ and $e(Q_X, Q_X, \epsilon_X) = 0$ (Equation (6)) for $\epsilon_X = 1$. Since $X$ is an arbitrary free variable in $\omega$, we derive that the preprocessing cost $p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (2)) is 1 and the enumeration cost $e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (7)) is 0 for parameter values $\{\epsilon_X = 1\}_{X\in\mathcal{F}}$. No variable order can admit lower preprocessing or enumeration cost. This implies $(1, 0) \in \mu(Q)$.

## A.3 Proof of Corollary 4

Consider an arbitrary ACQ $Q(\mathcal{F})$. We show that $(1, e)$ with $e \leq 1$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program in Section 1.

Given an arbitrary variable order $\omega$ for $Q$, consider the induced query $Q_X$ for any free variable $X$ in $\omega$. For any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, let $\hat{Q}$ be an arbitrary $\mathcal{J}$-connected component of $Q_X$. Consider the cover $(\{R(\mathcal{X})\}, \mathcal{R})$ for $\hat{Q}$, where $R(\mathcal{X})$ is an arbitrary atom in $\hat{Q}$ and $\mathcal{R} = atoms(\hat{Q}) \setminus \{R(\mathcal{X})\}$. This means that the preprocessing cost $p(\hat{Q}, \epsilon_X)$ (Equation (1)) is upper-bounded by $1 + |\mathcal{R}|\epsilon_X$ for any $\epsilon_X \in [0, 1]$. This implies $p(\hat{Q}, 0) = 1$. By definition, $e(Q_X, \hat{Q}, 0)$ (Equation (6)) is upper-bounded by $p(\hat{Q}, 0) = 1$. Since $\omega$ and $X$ were chosen arbitrarily, we derive that the preprocessing cost $p(\omega, \{\epsilon_X = 0\}_{X\in\mathcal{F}})$ (Equation (2)) is 1 and the enumeration cost $e(\omega, \{\epsilon_X = 0\}_{X\in\mathcal{F}})$ (Equation (7)) is at most 1. Overall, we conclude that $(1, e)$ with $e \leq 1$ is included in $\mu(Q)$.

## A.4 Proof of Corollary 5

Consider an ACQ $Q(\mathcal{F})$ of width $\mathsf{w}$. We show that $(\mathsf{w}, 0)$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of our trade-off program in Section 1.

Consider a variable order $\omega$ for $Q$ of width $\mathsf{w}$. Let $Q_X(\mathcal{F}_X)$ be the query induced by a variable $X \in \mathcal{F}$. By construction, $\rho^*_{Q_X}(\mathcal{F}_X) \leq \mathsf{w}$. For any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, let $\hat{Q}(\hat{\mathcal{F}})$ be an arbitrary $\mathcal{J}$-connected component of $Q_X$. Consider the cover $(atoms(\hat{Q}), \emptyset)$ for $\hat{Q}$. Note that $Q_1$ as used in Equation (1) is equal to $\hat{Q}$. It follows from the structure of $Q_X$ and $\hat{Q}$ that $\mathsf{w}(\hat{Q}) \leq \mathsf{w}$ and $\rho^*_{\hat{Q}}(\hat{\mathcal{F}}) \leq \rho^*_{Q_X}(\mathcal{F}_X) \leq \mathsf{w}$. This implies that is $p(\hat{Q}, \epsilon_X) \leq \mathsf{w}$ (Equation (1)) for any $\epsilon_X \in [0, 1]$. By definition, $e(Q_X, \hat{Q}, \epsilon_X)$ (Equation (6)) is upper-bounded by $k(1 - \epsilon_X)$, for some $k \geq 0$. By setting $\epsilon_X = 1$, we obtain $e(Q_X, \hat{Q}, 1) = 0$. Since $\omega$ and $X$ were chosen arbitrarily, we conclude that the preprocessing cost is $p(\omega, \{\epsilon_X = 1\}_{X \in \mathcal{F}}) = \mathsf{w}$ (Equation (2)) and the enumeration cost is $e(\omega, \{\epsilon_X = 1\}_{X \in \mathcal{F}}) = 0$ (Equation (7)). This implies that $(\mathsf{w}, 0)$ is included in $\mu(Q)$.

## A.5 Proof of Corollary 6

Consider a hierarchical query $Q(\mathcal{F})$ of width $\mathsf{w}$. We show that for any $\epsilon \in [0, 1]$, $(p, e)$ with $p \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e \leq 1 - \epsilon$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program in Section 1.

Let $\omega$ be a variable order for $Q$ of width $\mathsf{w}$ and $\epsilon \in [0, 1]$. Consider the query $Q_X(\mathcal{F}_X)$ induced by a variable $X \in \mathcal{F}$ such that $\rho^*_Q(\mathcal{F}_X) = \mathsf{w}$, where $\mathcal{F}_X = \{X\} \cup dep_\omega(X)$. Hierarchical queries stay hierarchical after removing variables or atoms. Hence, $Q_X(\mathcal{F}_X)$ is hierarchical. It follows from the construction of $Q_X$ and the shape of hierarchical queries that the number of atoms in $Q_X$ is $\mathsf{w}$. For any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, consider an arbitrary $\mathcal{J}$-connected component $\hat{Q}$ of $Q_X$. The query $\hat{Q}$ is hierarchical and the number of atoms in $\hat{Q}$ is at most $\mathsf{w}$. Consider the cover $C = (\{R(\mathcal{X})\}, \mathcal{R})$ for $\hat{Q}$, where $R(\mathcal{X})$ is an arbitrary atom in $\hat{Q}$ and $\mathcal{R} = atoms(\hat{Q}) \setminus \{R(\mathcal{X})\}$. Since the width of a query with a single atom and the fractional edge cover of a variable set included in a single atom are at most one, the cost of $C$ is at most $1 + |\mathcal{R}|\epsilon = 1 + (\mathsf{w} - 1)\epsilon$. It follows, $p(\hat{Q}, \epsilon) \leq 1 + (\mathsf{w} - 1)\epsilon$ (Equation (1)). Let $\hat{\mathcal{F}}$ (as defined in Equation (6)) be the free variables of $\hat{Q}$ that are included in the join v-sets of $Q_X$. Since $Q_X$ is hierarchical, there must be a single join v-set in $Q_X$ that subsumes $\hat{\mathcal{F}}$. Hence, $e(Q_X, \hat{Q}, \epsilon)$ is upper-bounded by $\rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})(1 - \epsilon) \leq 1 - \epsilon$. We conclude that $p(\omega, \{\epsilon_X = \epsilon\}_{X \in \mathcal{F}}) \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e(\omega, \{\epsilon_X = \epsilon\}_{X \in \mathcal{F}}) \leq 1 - \epsilon$. This implies that $(p, e)$ with $p \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e \leq 1 - \epsilon$ is included in $\mu(Q)$.

## A.6 Proof of Proposition 13

Consider an ACQ $Q(\mathcal{F})$ and a variable order $\omega$ for $Q$. Let $Q_X(\mathcal{F}_X)$ be the query induced by $X \in \mathcal{F}$. In the preprocessing stage, we partition the relations of $Q_X$ into disjoint parts. Each skew-aware composite query obtained from $Q_X$ computes the join of a combination of these parts. Hence, the union $Q^\cup_X(\mathcal{F}_X)$ of all skew-aware composite queries is the result of the induced query $Q_X(\mathcal{F}_X)$. The query $Q_\bowtie(\mathcal{F})$ is defined by the join of the induced queries. Each induced query $Q_X(\mathcal{F}_X)$ computes the projection of the result of $Q$ onto $\mathcal{F}_X$. The union of the free variables of the induced queries covers exactly the free variables $\mathcal{F}$ of $Q$. Hence, on any database, the result of $Q_\bowtie(\mathcal{F})$ is equal to the result of $Q(\mathcal{F})$.

## A.7 Proof of Proposition 14

Consider an acyclic query $Q(\mathcal{F})$, a variable order $\omega$ for $Q$, trade-off parameters $\{\epsilon_X\}_{X\in\mathcal{F}}$, and a database of size $N$. Let $\{s_X\}_{X\in\mathcal{F}}$ be the output of $\tau(\omega,\mathcal{F})$ (Figure 3). In the preprocessing stage, we first eliminate the dangling tuples in the database. Since $Q$ is acyclic, this can be done in $\mathcal{O}(N)$ time [19]. For any free variable $X$, we compute the fresh relations in the induced query $Q_X$ by computing projections and semi-joins. These operations can be executed in $\mathcal{O}(N)$ time. Partitioning the relations can also be done in linear time.

The preprocessing time is dominated by the time to compute the skew-aware queries in $\{s_X\}_{X\in\mathcal{F}}$. Given $X \in \mathcal{F}$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, consider a skew-aware query $\hat{Q}$ in $s_X(\mathcal{J})$. Let $(\mathcal{R}_1,\mathcal{R}_2)$ be a cover of $\hat{Q}$ such that $c+|\mathcal{R}_2|\epsilon_X$ is minimal, where $c$ is defined as in the preprocessing cost $p(\hat{Q},\epsilon_X)$ (Equation (1)). Using factorised computation [16], we join the relations in $\mathcal{R}_1$ in $\mathcal{O}(N^c)$ time. Let relation $S$ be the result of this join. Then, we use a left-deep join plan to join $S$, one by one, with the relations in $\mathcal{R}_2$. In each step, we take a relation from $\mathcal{R}_2$ that shares a light join v-set with the variables covered so far. Given an intermediate result $S'$, we join $S'$ with such a relation $R$ from $\mathcal{R}_2$ as follows: We iterate over the tuples in $S'$ and for each such tuple, we iterate over the matching tuples in $R$. Since $R$ is light on the join v-set, this can be done in $\mathcal{O}(|S|\cdot N^{\epsilon_X})$ time. Since $\mathcal{R}_1$ and $\mathcal{R}_2$ cover all relations in $\hat{Q}$ and these relations are connected via light join v-sets, this gives overall $\mathcal{O}(N^{c+|\mathcal{R}_2|\epsilon_X})$ computation time for $\hat{Q}$. By definition of the preprocessing cost $p(\omega,\{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (2)), the overall computation time for the induced queries $Q_X$ is $\mathcal{O}(N^p)$.

## A.8 Proof of Proposition 19

Consider an acyclic query $Q(\mathcal{F})$, a variable order $\omega$ for $Q$, trade-off parameters $\{\epsilon_X\}_{X\in\mathcal{F}}$, and a database of size $N$. Let $\{s_X\}_{X\in\mathcal{F}}$ be the output of $\tau(\omega,\mathcal{F})$. As explained in Section 5 (Equation (5)), the enumeration delay of our approach is bounded by:

$$\max_{X\in\mathcal{F}} \max_{\mathcal{J}\subseteq\mathsf{JVSets}(Q_X)} \max_{\hat{Q}\in\mathsf{Con}_{\mathcal{J}}(Q_X)} |\pi_{\hat{\mathcal{F}}} S_{\mathcal{J}}(\hat{Q})|,$$

where $S_{\mathcal{J}}(\hat{Q})$ is the skew-aware query obtained from $\hat{Q}$ and $\hat{\mathcal{F}} = \mathit{free}(\hat{Q}) \cap \bigcup_{\mathcal{V}\in\mathsf{JVSets}(Q_X)} \mathcal{V}$.

We give bounds on the exponent $e$ in the size complexity $\mathcal{O}(N^e)$ of $\pi_{\hat{\mathcal{F}}} S_{\mathcal{J}}(\hat{Q})$. One bound is the fractional edge cover number $\rho^*_{Q_X}(\hat{\mathcal{F}})$ [2]. Another bound is given by the preprocessing cost $p(\hat{Q},\epsilon_X)$ (Equation (1)) for $\hat{Q}$. The join v-sets of $Q_X$ that are included in $\hat{\mathcal{F}}$ are heavy, which means that the number of distinct tuples over each of these join v-sets must be bounded $N^{1-\epsilon_X}$. This implies that $e$ is bounded $k - k\epsilon_X$ where $k$ is the minimal number of join v-sets covering all variables in $\hat{\mathcal{F}}$. The number $k$ can be expressed by $\rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})$. From the definition of the enumeration cost $e(\omega,\{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (7)), we conclude that $e$ is bounded by $e = e(\omega,\{\epsilon_X\}_{X\in\mathcal{F}})$.