arXiv:2406.05536v3 [cs.DB] 3 Jul 2024

Output-Optimal Algorithms for Acyclic Join-Aggregate Queries

Xiao Hu
University of Waterloo
xiaohu@uwaterloo.ca

Abstract

The classic Yannakakis framework proposed in 1981 is still the state-of-the-art approach for tackling acyclic join-aggregate queries [**28, 19**] defined over commutative semi-rings. It has been shown that the time complexity of the Yannakakis framework is $O(N + \mathrm{OUT})$ for any free-connex join-aggregate query, where N is the input size of the database and OUT is the output size of the query result. This is already output-optimal. However, only a general upper bound $O(N \cdot \mathrm{OUT})$ on the time complexity of the Yannakakis framework is known for the remaining class of acyclic but non-free-connex queries.

We first show a lower bound $\Omega\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ for computing an acyclic join-aggregate query by *semi-ring algorithms*, where outw is identified as the *out-width* of the input query, N is the input size of the database, and OUT is the output size of the query result. For example, $\mathsf{outw} = 2$ for the chain matrix multiplication query, and $\mathsf{outw} = k$ for the star matrix multiplication query with k relations. We give a tighter analysis of the Yannakakis framework and show that the Yannakakis framework is already output-optimal on the class of *aggregate-hierarchical* queries. However, for the large remaining class of non-aggregate-hierarchical queries, such as chain matrix multiplication, the Yannakakis framework requires $\Theta(N \cdot \mathrm{OUT})$ time. We next explore a hybrid version of the Yannakakis framework and present an output-optimal algorithm for computing any general acyclic join-aggregate query within $\widetilde{O}\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ time, matching the out-width-dependent lower bound up to a poly-logarithmic factor. To our knowledge, this is the first polynomial improvement for computing acyclic join-aggregate queries since 1981.

1 Introduction

We study the class of join-aggregate queries defined over commutative semi-rings, which has wide applications in data analytical tasks. The Yannakakis framework proposed in 1981 is still the state-of-the-art approach for tackling acyclic join-aggregate queries [**28, 19**]. A few fundamental questions remain after 40 years: What is the minimum number of semi-ring operations required for computing an acyclic join-aggregate query? What is the theoretical limitation of the classic Yannakakis framework? What is an output-optimal algorithm for computing acyclic join-aggregate queries? In this paper, we will answer these challenging questions.

1.1 Problem Definition

Join Queries. A (natural) *join* is defined as a hypergraph $q = (\mathcal{V}, \mathcal{E})$, where the set of vertices $\mathcal{V} = \{x_1, \dots, x_\ell\}$ model the *attributes* and the set of hyperedges $\mathcal{E} = \{e_1, \dots, e_k\} \subseteq 2^{\mathcal{V}}$ model the *relations*. Let $\mathrm{dom}(x)$ be the *domain* of attribute $x \in \mathcal{V}$. Let $\mathrm{dom}(X) = \prod_{x \in X} \mathrm{dom}(x)$ be the *domain* of a subset $X \subseteq \mathcal{V}$ of attributes. An *instance* of q is a set of relations $\mathcal{R} = \{R_e : e \in \mathcal{E}\}$. Each relation R_e consists of a set of *tuples*, where each tuple is an assignment that assigns a value from $\mathrm{dom}(x)$ to x for every attribute $x \in e$. If every relation R_e is distinct, then $\mathcal{Q}$ does not contain *self-joins*. The *full join result* of q on $\mathcal{R}$, denoted as $q(\mathcal{R})$, is defined as

$$q(\mathcal{R}) = \{t \in \mathrm{dom}(\mathcal{V}) : \forall e \in \mathcal{E}, \pi_e t \in R_e\},$$

i.e., all combinations of tuples, one from each relation, such that they share the same values on their common attributes.

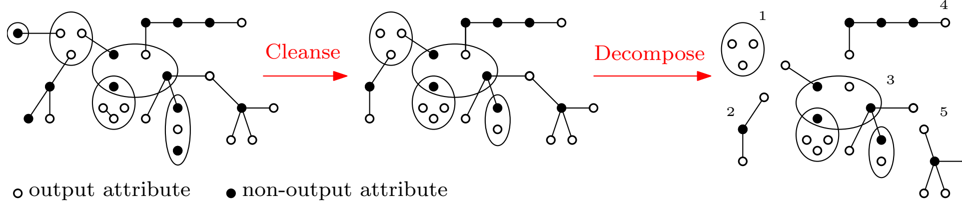

Figure 1: An illustration of the CLEANSE and DECOMPOSE procedures [16].

For $q = (\mathcal{V}, \mathcal{E})$, a fractional edge covering is defined as a function $\rho : \mathcal{E} \to [0, 1]$ such that $\sum_{e:A\in e} \rho(e) \geq 1$ holds for each attribute $A \in \mathcal{V}$. The fractional edge covering number of q, denoted as $\rho^*(q)$, is defined as the minimum sum of weight over all possible fractional edge coverings for q, i.e.,

$$\rho^*(q) = \min_{\rho:\rho \text{ is a fractional edge covering for } q} \sum_{e\in\mathcal{E}} \rho(e).$$

Join-Aggregate Queries. To study join-aggregate queries, we consider *annotated relations* [**13, 19**]. Let $(\mathbb{R}, \oplus, \otimes)$ be a commutative semi-ring. Every tuple t is further associated with an *annotation* $w(t) \in \mathbb{R}$. The annotation of a full join result $t \in q(\mathcal{R})$ is $w(t) := \bigotimes_{e\in\mathcal{E}} w(\pi_e t)$. A *join-aggregate query* is defined as a triple $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, where $q = (\mathcal{V}, \mathcal{E})$ is a (natural) join query, and $\mathbf{y} \subseteq \mathcal{V}$ is the set of *output attributes* (a.k.a. *free attributes*). For simplicity, let $\bar{\mathbf{y}} = \mathcal{V} - \mathbf{y}$ be the set of non-output attributes (a.k.a. *existential attributes*). The *query result* of $\mathcal{Q}$ on $\mathcal{R}$, denoted as $\mathcal{Q}(\mathcal{R})$, is defined as

$$\mathcal{Q}(\mathcal{R}) = \oplus_{\bar{\mathbf{y}}} q(\mathcal{R}) = \left\{ (t_{\mathbf{y}}, w(t_{\mathbf{y}})) : t_{\mathbf{y}} \in \pi_{\mathbf{y}} q(\mathcal{R}), w(t_{\mathbf{y}}) = \bigoplus_{t\in q(\mathcal{R}):\pi_{\mathbf{y}} t = t_{\mathbf{y}}} w(t) \right\},$$

In plain language, a join-aggregate query (semantically) first computes the full join result $q(\mathcal{R})$ and the annotation of each result, which is the $\otimes$-aggregate of the tuples comprising the join result. Then it partitions $q(\mathcal{R})$ into groups by the attributes in $\mathbf{y}$. Finally, for each group, it computes the $\oplus$-aggregate of the annotations of the join result in that group. Join-aggregate queries include many commonly seen database queries as special cases. For example, if we ignore the annotations, then it becomes a join-project query $\pi_{\mathbf{y}} q(\mathcal{R})$, also known as a *conjunctive query*. If we take $\mathbb{R}$ be the domain of integers and set $w(t) = 1$ for every tuple t, it becomes the `COUNT(*) GROUP BY` $\mathbf{y}$ query; in particular, if $\mathbf{y} = \emptyset$, the query computes the full join size $|q(\mathcal{R})|$. If we take $\mathcal{V} = \{A, B, C\}$ with $\mathbf{y} = \{A, C\}$, and $\mathcal{E} = \{\{A, B\}, \{B, C\}\}$, it becomes the matrix multiplication problem.

We use $N = \sum_{e\in\mathcal{E}} |R_e|$ to denote the input size and $\text{OUT} = |\mathcal{Q}(\mathcal{R})|$ to denote the output size. Let $\mathcal{Q}_{\mathbf{y}} = (\mathbf{y}, \{e \cap \mathbf{y} : e \in \mathcal{E}\})$ be the sub-query derived by output attributes. For any instance of input size N, it has been shown [**3**] that the largest output size is $\Theta\left(N^{\rho^*(\mathcal{Q}_{\mathbf{y}})}\right)$. We study the data complexity of this problem by considering the query size (i.e., k and ℓ) as constants and measuring the time complexity of algorithms by data-dependent quantities, such as N and OUT. Similar to [**25**], we confine ourselves to algorithms that work with semi-ring elements as an abstract type. We can only copy them from existing semi-ring elements or combine them using semi-ring operations.

Other Notations. In a join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, we use $\mathcal{E}_A = \{e \in \mathcal{E} : A \in e\}$ to denote the set of relations containing attribute A. An attribute $A \in \mathcal{V}$ is *unique* if it appears only in one relation, i.e., $|\mathcal{E}_A| = 1$, and *joint* otherwise. Given an instance $\mathcal{R}$, for a subset of attributes $\mathbb{A} \subseteq \mathcal{V}$, we define the *active domain* of $\mathbb{A}$ as the collection of tuples in $\text{dom}(\mathbb{A})$ that appears in at least one full join result of q on $\mathcal{R}$, i.e., $\pi_{\mathbb{A}} q(\mathcal{R})$. For any $n \in \mathbb{Z}^+$, we use $[n]$ to denote $\{1, 2, \cdots, n\}$. For a pair of sets S_1 and S_2, we use $S_1 - S_2 = \{x \in S_1 : x \notin S_2\}$ to denote the set minus operation.

Classification of Join-Aggregate Queries. We introduce the following three important classes of join-aggregate queries that will be frequently discussed throughout the paper.

- **Acyclic Query [7, 11, 17]** There are many equivalent definitions of acyclic queries, and we use the one based on generalized join tree [**17**]. A *generalized relation* R_e is defined on the projection of one

input relation $R_{e'}$ for $e' \in \mathcal{E}$ onto a subset of attributes $e \subseteq e'$. A join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ is acyclic if there exists a tree $\mathcal{T}$ such that (1) each node in $\mathcal{T}$ corresponds to an input relation or a generalized relation; (2) every input relation in $\mathcal{E}$ corresponds to a node in $\mathcal{T}$; (3) for every attribute $A \in \mathcal{V}$, the set of nodes containing it forms a connected subtree of $\mathcal{T}$. $\mathcal{T}$ is called a *generalized join tree* of $\mathcal{Q}$.

- **Free-connex Query** [**4**] A join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ is free-connex if $\mathcal{Q}$ is acyclic and $(\mathcal{V}, \mathcal{E} \cup \{\mathbf{y}\}, \mathbf{y})$ is also acyclic.
- **Hierarchical Query** [**26**] A join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ is hierarchical if for any pair of attributes $A, B \in \mathcal{V}$, either $\mathcal{E}_A \subseteq \mathcal{E}_B$, or $\mathcal{E}_B \subseteq \mathcal{E}_A$, or $\mathcal{E}_A \cap \mathcal{E}_B = \emptyset$.

A hierarchical query must be acyclic, and a free-connex query must also be acyclic. However, there is no containment relationship between free-connex and hierarchical queries. See Figure 2.

1.2 Previous Upper Bounds

The Yannakakis framework can be slightly adapted for computing acyclic join-aggregate queries [**28, 19**]. As shown in Algorithm 7, it picks an arbitrary join tree[1] $\mathcal{T}$ for $\mathcal{Q}$ rooted at node r. It first removes all *dangling tuples* in the input instance $\mathcal{R}$, i.e., those won't appear in any full join result, in $O(N)$ time via a bottom-up and then a top-down phase of semi-joins. If OUT $= 0$, i.e., $\mathcal{Q}(\mathcal{R}) = \emptyset$, all input tuples will be removed as dangling tuples. After being done with semi-joins, the Yannakakis framework performs joins and aggregations in a bottom-up way. Specifically, it takes two relation R_e and $R_{e'}$ such that e is a leaf and e' is the parent of e, aggregate over non-output attributes that only appear in e but not in e' by replacing R_e with $\oplus_{\bar{\mathbf{y}} \cap (e - e')} R_e$, and replaces $R_{e'}$ with $R_e \bowtie R_{e'}$. Then R_e is removed, and the step repeats until only one relation remains, say R_r. It will output $\oplus_{\bar{\mathbf{y}} \cap e} R_r$ as the final result. The running time of the Yannakakis framework is proportional to the largest intermediate join size (after dangling tuples are removed), which is no larger than the full join size. Hence, the Yannakakis framework is always better than the naive solution of computing the full join results and then projecting out non-output attributes. We note that the largest intermediate join size could differ drastically on different *query plans*, i.e., each query plan of the Yannakakis framework corresponds to a specific rooted join tree together with a specific sequence of joins and projections in the bottom-up computation.

If $\mathcal{Q}$ is *free-connex*, there is a query plan that only generates at most $O(\text{OUT})$ intermediate join results [**19, 5**], hence free-connex queries can be computed in $O(N + \text{OUT})$ time. Note that any acyclic full join query is free-connex. Yannakakis gave an upper bound of $O(N \cdot \text{OUT})$ on the largest intermediate join size for non-free-connex queries. For matrix multiplication query,

$$\mathcal{Q}_{\mathsf{matrix}} = \bigoplus_{B} R_1(A, B) \bowtie R_2(B, C),$$

the simplest acyclic but non-free-connex query, this bound has been improved to $O\left(N \cdot \sqrt{\text{OUT}}\right)$ [**2**] by a better analysis. This is also tight since there are instances with intermediate join result (which is also the full join result for $\mathcal{Q}_{\mathsf{matrix}}$) as large as $\Theta\left(N \cdot \sqrt{\text{OUT}}\right)$. This bound also extends to star queries (a.k.a. star matrix multiplication),

$$\mathcal{Q}_{\mathsf{star}} = \bigoplus_{B} R_1(A_1, B) \bowtie R_2(A_2, B) \bowtie \cdots \bowtie R_k(A_k, B),$$

on which the tight bound is $O\left(N \cdot \text{OUT}^{1-\frac{1}{k}}\right)$. But, for line queries (a.k.a. chain matrix multiplication),

$$\mathcal{Q}_{\mathsf{line}} = \bigoplus_{A_2, A_3, \cdots, A_k} R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie \cdots \bowtie R_k(A_k, A_{k+1})$$

this bound $O(N \cdot \text{OUT})$ is already tight [**16**].

[1] Below, we always use "join tree" to denote "generalized join tree" for simplicity.

People have incorporated generalized hypertree decomposition techniques [**14**] together with worst-case optimal join algorithms [**23, 27**] into the Yannakakis framework to handle cyclic join-aggregate queries. Khamis et al. [**20**] showed that an arbitrary join-aggregate query $\mathcal{Q}$ can be computed in $O\left(N^{\#\mathsf{subw}} \cdot \mathrm{OUT}\right)$ time, where $\#\mathsf{subw} \geq 1$ is the #sub-modular width of $\mathcal{Q}$ [**22**]. And $\#\mathsf{subw} = 1$ if and only if $\mathcal{Q}$ is acyclic. If restricting the generalized hypertree decompositions to be free-connex, one can also compute an arbitrary join-aggregate query $\mathcal{Q}$ in $O\left(N^{\#\mathsf{fc\text{-}subw}} + \mathrm{OUT}\right)$ time, where $\#\mathsf{fc\text{-}subw}$ is the #free-connex sub-modular width of $\mathcal{Q}$ [**20**]. Moreover, $\#\mathsf{fc\text{-}subw} = 1$ if and only if $\mathcal{Q}$ is free-connex. It is not hard to see that $\#\mathsf{fc\text{-}subw} \geq \#\mathsf{subw}$ for any join-aggregate query. These two results are incomparable unless we know the value of OUT.

There are some other works using fast matrix multiplication to speed up conjunctive queries [**2, 10, 1, 16**] or graph pattern search (as a special case of self-joins) [**8, 18, 9**], but these techniques cannot be applied to general join-aggregate queries. Other systematic works also study sparse chain matrix multiplication [**12, 24, 6, 21**], but without any theoretical guarantees on the output-optimality. We won't pursue these dimensions further.

1.3 Previous Lower Bounds

Before reviewing the lower bounds, we mention two necessary primitives proposed for join-project queries [**16**], which have been adapted to join-aggregate queries below.

Cleanse$(\mathcal{Q}, \mathcal{R})$**.** For a join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ and an instance $\mathcal{R}$, the cleanse procedure iteratively (i) removes a unique non-output attribute $A \in \mathcal{V} - \mathbf{y}$ (suppose $\mathcal{E}_A = \{e\}$) and replaces relation R_e with $\bigoplus_A R_e$; or (ii) removes a relation $e \in \mathcal{E}$ if there is another relation $e' \in \mathcal{E}$ such that $e \subseteq e'$ and updates the annotation of each tuple $t \in R_{e'}$ by $w(t) \otimes w\left(\pi_e t\right)$.

The complete procedure is described in Algorithm 6, which takes $O(N)$ time. $\mathcal{Q}$ is called *cleansed* if no more attribute or relation can be removed, and *non-cleansed* otherwise. In a cleansed query $\mathcal{Q}$, every unique attribute must be an output attribute.

Decompose$(\mathcal{Q}, \mathcal{R})$**.** The *existential connectivity* of a join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ is defined as a graph $G^{\exists}_{\mathcal{Q}}$, where each $e \in \mathcal{E}$ is a vertex, and there is an edge between $e, e' \in \mathcal{E}$ if they share some non-output attribute(s), i.e., $e \cap e' - \mathbf{y} \neq \emptyset$. Let $\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_h \subseteq \mathcal{E}$ be the connected components of $G^{\exists}_{\mathcal{Q}}$. Each $\mathcal{E}_i$ defines a sub-query $\mathcal{Q}_i = (\bigcup_{e \in \mathcal{E}_i} e, \mathcal{E}_i, \bigcup_{e \in \mathcal{E}_i} e \cap \mathbf{y})$ and a sub-instance $\mathcal{R}_i = \{R_e : e \in \mathcal{E}_i\}$. For a cleansed query $\mathcal{Q}$, every sub-query $\mathcal{Q}_i$ is also cleansed.

Now, we are ready to review a non-trivial lower bound of computing acyclic join-aggregate queries by semi-ring algorithms [**16**], which is built on the notion of *free-width* of the input query:

Definition 1 (free-width [**16**])**.** *For any acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, its free-width* $\mathsf{freew}(\mathcal{Q})$ *is defined as follows:*

- *If $\mathcal{Q}$ is existentially disconnected with $\mathcal{Q}_1, \mathcal{Q}_2, \cdots, \mathcal{Q}_h$ as sub-queries defined by $G^{\exists}_{\mathcal{Q}}$,*
$$\mathsf{freew}(\mathcal{Q}) = \max_{i \in [h]} \mathsf{freew}(\mathcal{Q}_i).$$
- *If $\mathcal{Q}$ is existentially connected but not cleansed,* $\mathsf{freew}(\mathcal{Q}) = \mathsf{freew}(\mathcal{Q}')$ *where $\mathcal{Q}'$ is the cleansed version of $\mathcal{Q}$.*
- *If $\mathcal{Q}$ is existentially connected and cleansed,*
$$\mathsf{freew}(\mathcal{Q}) = \max_{S \subseteq \mathcal{E}: \forall e \in S, e \cap \mathcal{V}_\bullet \neq \emptyset} |S|,$$
where $\mathcal{V}_\bullet$ denotes the set of unique attributes in $\mathcal{Q}$.

Lemma 1 ([**16**])**.** *For an arbitrary acyclic join-aggregate query $\mathcal{Q}$ without self-joins, given any $1 \leq N$ and $1 \leq \mathrm{OUT} \leq N^{\mathsf{freew}}$, there is an instance $\mathcal{R}$ of input size $\Theta(N)$ and output size $\Theta(\mathrm{OUT})$ such that any semi-ring algorithm computing $\mathcal{Q}(\mathcal{R})$ requires $\Omega\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{freew}}} + \mathrm{OUT}\right)$ time, where* freew *is the free-width of $\mathcal{Q}$.*

Join-Aggregate Query	Yannakakis	Hybrid Yannakakis
Matrix	$\Theta\left(N \cdot \sqrt{\mathrm{OUT}}\right)$ [28]	
Star	$\Theta\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$ [28]	
Aggregate-Hierarchical	$\Theta\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right)$ [28]	
Line	$\Theta(N \cdot \mathrm{OUT})$ [28]	$\Theta(N \cdot \sqrt{\mathrm{OUT}})$
Acyclic		$\Theta\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$

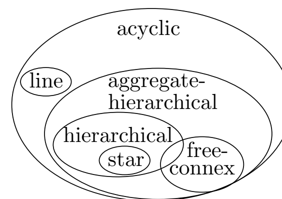

Figure 2: The left table compares the Yannakakis framework and our new algorithm. N is the input size and OUT is the output size. k is the number of relations. outw is the out-width of the input query (see Definition 1). The right figure shows the relationship between different classes of queries.

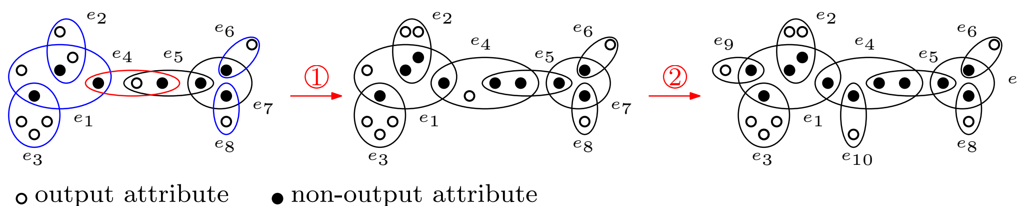

Figure 3: An illustration of free-width and out-width of a join-aggregate query. The example query has $\mathsf{freew} = 5$, with five relations (as shown in blue) that exclusively contain some output attribute(s). But, it has $\mathsf{outw} = 6$ with an optimal fractional edge covering for $\mathcal{Q}_{\mathbf{y}}$, where both blue and red relations are assigned with weight 1. ① and ② are illustrated for the SEPARATE procedure in Section 4.

1.4 Our Results

We focus on acyclic join-aggregate queries where the output attributes can be arbitrary. Our results are summarized in Figure 2.

New Lower Bound. We start with proving a higher lower bound by identifying a new query-dependent quantity noted as *out-width* for the input acyclic join-aggregate query. This lower bound has been matched by our new upper bound (to be introduced soon).

Definition 2 (out-width)**.** *For any acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, its out-width $\mathsf{outw}(\mathcal{Q})$ is defined as follows:*

- *If $\mathcal{Q}$ is existentially disconnected with $\mathcal{Q}_1, \mathcal{Q}_2, \cdots, \mathcal{Q}_h$ as sub-queries defined by $G^{\exists}_{\mathcal{Q}}$,*

$$\mathsf{outw}(\mathcal{Q}) = \max_{i \in [h]} \mathsf{outw}(\mathcal{Q}_i).$$

- *If $\mathcal{Q}$ is existentially connected but not cleansed, $\mathsf{outw}(\mathcal{Q}) = \mathsf{outw}(\mathcal{Q}')$ where $\mathcal{Q}'$ is the cleansed version of $\mathcal{Q}$.*
- *If $\mathcal{Q}$ is existentially connected and cleansed, $\mathsf{outw}(\mathcal{Q}) = \rho^*(\mathcal{Q}_{\mathbf{y}})$, where $\mathcal{Q}_{\mathbf{y}} = (\mathbf{y}, \{e \cap \mathbf{y} : e \in \mathcal{E}\})$ is the sub-query derived by output attributes.*

Remark 1. In Appendix A, we show $\mathsf{outw}(\mathcal{Q}) = 1$ if and only if $\mathcal{Q}$ is free-connex; and $\mathsf{freew}(\mathcal{Q}) \leq \mathsf{outw}(\mathcal{Q})$ for any acyclic query $\mathcal{Q}$. Moreover, $\mathsf{freew}(\mathcal{Q}_{\mathsf{star}}) = \mathsf{outw}(\mathcal{Q}_{\mathsf{star}}) = k$, and $\mathsf{freew}(\mathcal{Q}_{\mathsf{line}}) = \mathsf{outw}(\mathcal{Q}_{\mathsf{line}}) = 2$. In Figure 3, we give an example of $\mathcal{Q}$ on which $\mathsf{freew}(\mathcal{Q}) < \mathsf{outw}(\mathcal{Q})$.

Theorem 1. *For an arbitrary existentially-connected acyclic join-aggregate query $\mathcal{Q}$ without self-joins, given any $1 \leq N$ and $\mathrm{OUT} \leq N^{\mathsf{outw}}$, there exists an instance $\mathcal{R}$ of input size $\Theta(N)$ and output size $\Theta(\mathrm{OUT})$ such*

that any semi-ring algorithm for computing $\mathcal{Q}(\mathcal{R})$ *requires at least* $\Omega\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ *time, where* outw *is the out-width of* $\mathcal{Q}$.

Theorem 2. *For an arbitrary acyclic join-aggregate query* $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ *without self-joins, given any* $1 \le N$ *and* $\mathrm{OUT} \le N^{\rho^*(\mathcal{Q}_\mathbf{y})}$ *with the fractional edge covering number* $\rho^*(\mathcal{Q}_\mathbf{y})$ *of* $\mathcal{Q}_\mathbf{y} = (\mathbf{y}, \{e \cap \mathbf{y} : e \in \mathcal{E}\})$*, there exists an instance* $\mathcal{R}$ *of input size* $\Theta(N)$ *and output size* $\Theta(\mathrm{OUT})$ *such that any semi-ring algorithm for computing* $\mathcal{Q}(\mathcal{R})$ *requires at least* $\Omega\left(\min\left\{N^{\mathsf{outw}}, N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right\}\right)$ *time, where* outw *is the out-width of* $\mathcal{Q}$.

New Upper Bound. As observed in [**2**], Yannakakis framework is optimal on star queries by matching the lower bound in Lemma 1. In Appendix B.1, we extend the optimality of the Yannakakis framework to general aggregate-hierarchical queries:

Definition 3 (Aggregate-Hierarchical Query)**.** *An acyclic join-aggregate query* $\mathcal{Q}$ *is aggregate-hierarchical if the cleansed version of every connected sub-query in* $G^{\exists}_{\mathcal{Q}}$ *is hierarchical.*

However, Yannakakis framework is $\Theta(\sqrt{N})$-factor away from optimal on any non-aggregate-hierarchical query, such as line queries. This paper shows new upper bound for acyclic but non-aggregate-hierarchical queries. Our main contribution is an output-optimal algorithm for computing existentially-connected queries, as shown in Theorem 3:

Theorem 3. *For an arbitrary existentially-connected acyclic join-aggregate query* $\mathcal{Q}$*, and any instance* $\mathcal{R}$ *of input size* N *and output size* OUT*, the* $\mathcal{Q}(\mathcal{R})$ *can be computed within* $\widetilde{O}\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ *time, where* outw *is the out-width of* $\mathcal{Q}$.

For a general acyclic join-aggregate query $\mathcal{Q}$ and an arbitrary instance $\mathcal{R}$, we can apply the Cleanse and Decompose procedures as pre-processing. Then, it suffices to compute the query results for each connected sub-query and combine them via join as the post-processing step. The whole framework is described in Algorithm 1. The pre-processing and post-processing steps only take linear time regarding the input and output size. Combining this observation with Theorem 3, we obtain Theorem 4.

Theorem 4. *For an arbitrary acyclic join-aggregate query* $\mathcal{Q}$*, and any instance* $\mathcal{R}$ *of input size* N *and output size* OUT*, the* $\mathcal{Q}(\mathcal{R})$ *can be computed within* $\widetilde{O}\left(N \cdot \max_{i \in [h]} |\pi_{\mathbf{y}_i} \mathcal{Q}(\mathcal{R})|^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ *time, where* $\mathcal{Q}_1, \mathcal{Q}_2, \cdots, \mathcal{Q}_h$ *are the connected sub-queries defined by* $G^{\exists}_{\mathcal{Q}}$*, and* $\mathbf{y}_i$ *is set of output attributes in* $\mathcal{Q}_i$ *for* $i \in [h]$.

Remark 2. For Theorem 3, as it is inherently challenging to compute or estimate OUT efficiently, we follow the same framework as [**16**] by computing an $O(1)$-approximation of OUT on the fly, which only increases the overall complexity by a logarithmic factor. More specifically, we obtain $\widetilde{\mathrm{OUT}}$ of OUT such that $\widetilde{\mathrm{OUT}} \le \mathrm{OUT} \le 2 \cdot \widetilde{\mathrm{OUT}}$, when invoking algorithms in Sections 2, 3 and 4.

Remark 3. For Theorem 4, consider an arbitrary sub-query $\mathcal{Q}_i$ in $G^{\exists}_{\mathcal{Q}}$, which is existentially connected. For an arbitrary instance $\mathcal{R}$ of input size N and output size OUT, the largest output size for $\mathcal{Q}_i$ is $\Theta\left(N^{\mathsf{outw}(\mathcal{Q}_i)}\right)$. On the other hand, $|\pi_{\mathbf{y}_i} \mathcal{Q}(\mathcal{R})| \le |\pi_{\mathbf{y}} \mathcal{Q}(\mathcal{R})| = \mathrm{OUT}$. Together with the fact that $\mathsf{outw}(\mathcal{Q}_i) \le \mathsf{outw}(\mathcal{Q})$ for $i \in [h]$, we have

$$N \cdot \max_{i \in [h]} |\pi_{\mathbf{y}_i} \mathcal{Q}(\mathcal{R})|^{1-\frac{1}{\mathsf{outw}}} \le \min\left\{N^{\mathsf{outw}}, N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right\}.$$

Hence, the upper bound in Theorem 7 is optimal by matching the lower bound in Theorem 2.

2 Line Query

We start with the line queries (a.k.a. chain matrix multiplication):

$$\bigoplus_{A_2, A_3, \cdots, A_k} R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie \cdots \bowtie R_k(A_k, A_{k+1}).$$

Suppose we are given $\widetilde{\mathrm{OUT}}$ such that $\widetilde{\mathrm{OUT}} \le \mathrm{OUT} \le 2 \cdot \widetilde{\mathrm{OUT}}$. Our algorithm consists of five steps:

Algorithm 1: AcyclicJoinAggregate($\mathcal{Q}, \mathcal{R}$)

Input : an acyclic join-aggregate query $\mathcal{Q}$ and instance $\mathcal{R}$;
Output: query result $\mathcal{Q}(\mathcal{R})$;
1 $(\mathcal{Q}_1, \mathcal{R}_1), (\mathcal{Q}_2, \mathcal{R}_2), \cdots, (\mathcal{Q}_h, \mathcal{R}_h) \leftarrow$ Decompose($\mathcal{Q}, \mathcal{R}$);
2 **foreach** $i \in [h]$ **do**
3 $\quad (\mathcal{Q}_i, \mathcal{R}_i) \leftarrow$ Cleanse($\mathcal{Q}_i, \mathcal{R}_i$) ▶ Algorithm 6;
4 $\quad (\mathcal{Q}'_i, \mathcal{R}'_i) \leftarrow$ Separate($\mathcal{Q}_i, \mathcal{R}_i$); ▶ Section 4;
5 $\quad \mathcal{S}_i \leftarrow$ HybridYannakakis($\mathcal{Q}'_i, \mathcal{R}'_i$); ▶ Algorithm 8;
6 **return** $\Join_{i \in [h]} \mathcal{S}_i$;

Algorithm 2: Line$\left(\mathcal{Q}_{\mathsf{line}}, \mathcal{R}, \tilde{\mathrm{OUT}}\right)$

Input : A line query $\mathcal{Q}_{\mathsf{line}}$, an instance $\mathcal{R}$ of input size N and output size OUT, and parameter $\tilde{\mathrm{OUT}}$ such that $\tilde{\mathrm{OUT}} \le \mathrm{OUT} \le 2 \cdot \tilde{\mathrm{OUT}}$;
Output: Query result $\mathcal{Q}_{\mathsf{line}}(\mathcal{R})$;
1 **foreach** $i \in [k-1]$ **do**
2 $\quad$ **if** $i = 1$ **then** $T_1(A_1, A_2) \leftarrow R_1(A_1, A_2)$;
3 $\quad$ **else** $T_i(A_1, A_{i+1}) \leftarrow \oplus_{A_i} S_{i-1}(A_1, A_i) \Join R_i\,(A_i, A_{i+1})$;
4 $\quad A^{\mathsf{light}}_{i+1} \leftarrow \left\{a \in \mathrm{dom}(A_{i+1}) : \left|\sigma_{A_{i+1}=a} T_i\right| \le \sqrt{\tilde{\mathrm{OUT}}}\right\}$;
5 $\quad A^{\mathsf{heavy}}_{i+1} \leftarrow \left\{a \in \mathrm{dom}(A_{i+1}) : \left|\sigma_{A_{i+1}=a} T_i\right| > \sqrt{\tilde{\mathrm{OUT}}}\right\}$;
6 $\quad R^{\mathsf{light}}_i \leftarrow R_i \ltimes A^{\mathsf{light}}_{i+1}$;
7 $\quad R^{\mathsf{heavy}}_i \leftarrow R_i \ltimes A^{\mathsf{heavy}}_{i+1}$;
8 $\quad \mathcal{Q}_i \leftarrow \oplus_{A_2, A_3, \cdots, A_k} \left(\Join_{j \in [i-1]} R^{\mathsf{light}}_j\right) \Join R^{\mathsf{heavy}}_i \Join \left(\Join_{j=i+1}^{k} R_j\right)$;
9 $\quad S_i(A_1, A_{i+1}) \leftarrow T_i(A_1, A_{i+1}) \ltimes A^{\mathsf{light}}_{i+1}$;
10 $\mathcal{Q}_* \leftarrow \oplus_{A_2, A_3, \cdots, A_k} \left(\Join_{j \in [k-1]} R^{\mathsf{light}}_j\right) \Join R_k$;
11 **return** $\mathcal{Q}_1 \oplus \mathcal{Q}_2 \oplus \cdots \oplus \mathcal{Q}_{k-1} \oplus \mathcal{Q}_*$;

Step 1: Compute data statistics. For each value $a \in \mathrm{dom}(A_2)$, we define its *degree* as the number of tuples in R_1 that displays value a in attribute A_2, i.e., $\triangle(a) = |\sigma_{A_2=a} R_1|$. A value $a \in \mathrm{dom}(A_2)$ is *heavy* if $\triangle(a) > \sqrt{\tilde{\mathrm{OUT}}}$, and *light* if $1 \le \triangle(a) \le \sqrt{\tilde{\mathrm{OUT}}}$. Let $A^{\mathsf{heavy}}_2, A^{\mathsf{light}}_2$ be the set of heavy and light values in A_2. Let $R^{\mathsf{heavy}}_1 = R_1 \ltimes A^{\mathsf{heavy}}_2$ and $R^{\mathsf{light}}_1 = R_1 \ltimes A^{\mathsf{light}}_2$ be the set of heavy and light tuples in R_1 respectively. Then, we iteratively partition relations by ordering $R_2, R_3, \cdots, R_k$. Suppose relation R_j is partitioned into R^{heavy}_j and R^{light}_j, for every $j \in [i-1]$. We next partition relation R_i as follows. For each value $a \in \mathrm{dom}(A_{i+1})$, we define its *degree* as the number of distinct values in $\mathrm{dom}(A_1)$ that can be joined with a via tuples in relations $R^{\mathsf{light}}_1, R^{\mathsf{light}}_2, \cdots, R^{\mathsf{light}}_{i-1}$, which is denoted as

$$\triangle(a) = \left|\pi_{A_1}\left\{\left(\Join_{j \in [i-1]} R^{\mathsf{light}}_j\right) \Join (\sigma_{A_{i+1}=a} R_i)\right\}\right|$$

A value $a \in \mathrm{dom}(A_{i+1})$ is *heavy* if $\triangle(a) > \sqrt{\tilde{\mathrm{OUT}}}$, and *light* if $1 \le \triangle(a) \le \sqrt{\tilde{\mathrm{OUT}}}$. Let $A^{\mathsf{heavy}}_{i+1}, A^{\mathsf{light}}_{i+1}$ be the set of heavy, light values in A_i respectively. Let $R^{\mathsf{heavy}}_i = R_i \ltimes A^{\mathsf{heavy}}_{i+1}$, and $R^{\mathsf{light}}_i = R_i \ltimes A^{\mathsf{light}}_{i+1}$ be the set of heavy, light tuples in R_i respectively. Note that there could be some tuples undefined.

We compute $\Delta(\cdot)$ as described in Algorithm 2. For simplicity, we define two intermediate relations:

$$T_i\,(A_1, A_{i+1}) = \bigoplus_{A_2, A_3, \cdots, A_i} \left(\Join_{j \in [i-1]} R^{\mathsf{light}}_j\right) \Join R_i,$$
$$S_i(A_1, A_{i+1}) = \bigoplus_{A_2, A_3, \cdots, A_i} \left(\Join_{j \in [i]} R^{\mathsf{light}}_j\right)$$

and compute them in a recursive way as follows (with $T_1 = R_1$):

$$T_i(A_1, A_{i+1}) = \bigoplus_{A_i} S_{i-1}(A_1, A_i) \bowtie R_i(A_i, A_{i+1})$$
$$S_i(A_1, A_{i+1}) = T_i(A_1, A_{i+1}) \ltimes A_{i+1}^{\mathsf{light}}$$

Once we have computed T_i for some $i \in \{1, 2, \cdots, k-1\}$. We can identify the heavy and light values in A_{i+1}, i.e., A_{i+1}^{heavy} and A_{i+1}^{light}. We can use $A_{i+1}^{\mathsf{heavy}}, A_{i+1}^{\mathsf{light}}$ to partition R_i into $R_i^{\mathsf{heavy}}, R_i^{\mathsf{light}}$. Then, S_i can be computed based on T_i and A_{i+1}^{light}. Furthermore, T_{i+1} can be computed based on S_i and R_{i+1}.

We next analyze the cost. T_1 can be computed within $O(N)$ time and $|T_1| = O(N)$. Consider any $i \in \{2, 3, \cdots, k\}$. As there are N tuples in R_i, and each of them can be joined with at most $\sqrt{\tilde{\mathrm{OUT}}}$ tuples in S_{i-1}, implied by the definition of R_{i-1}^{light}, T_i can be computed within $O\left(N \cdot \sqrt{\mathrm{OUT}}\right)$ time. Also, $|T_i| = O\left(N \cdot \sqrt{\mathrm{OUT}}\right)$. The cost of computing S_i is $O(|T_i|) = O\left(N \cdot \sqrt{\mathrm{OUT}}\right)$.

Step 2: Partition $\mathcal{Q}_{\mathsf{line}}$. After all relations are partitioned, we partition $\mathcal{Q}_{\mathsf{line}}$ into k sub-queries:

$$\mathcal{Q}_i = \oplus_{A_2, A_3, \cdots, A_k} \left(\bowtie_{j \in [i-1]} R_j^{\mathsf{light}}\right) \bowtie R_i^{\mathsf{heavy}} \bowtie \left(\bowtie_{j=i+1}^{k} R_j\right)$$

for $i \in [k-1]$, and $\mathcal{Q}_* = \oplus_{A_2, A_3, \cdots, A_k} \left(\bowtie_{j \in [k-1]} R_j^{\mathsf{light}}\right) \bowtie R_k$.

Step 3: Compute $\mathcal{Q}_i$ for each $i \in [k-1]$. We invoke the Yannakakis framework over a join tree rooted at R_1 and compute $\mathcal{Q}_i$:

$$\oplus_{A_2} R_1^{\mathsf{light}} \bowtie \cdots \left(\oplus_{A_{i+1}} R_i^{\mathsf{heavy}} \bowtie \left(\oplus_{A_{i+2}} R_{i+1} \bowtie \cdots (\oplus_{A_k+1} R_k \bowtie R_{k+1})\right)\right)$$

After all dangling tuples are removed, each value in $\mathrm{dom}(A_{k+1})$ can be joined with some tuple in R_i^{heavy}, hence can be joined with at least $\sqrt{\tilde{\mathrm{OUT}}}$ values in A_1, implied by the definition of A_{i+1}^{heavy}. As there are OUT results in total, the active domain size of A_{k+1} is at most $O\left(\sqrt{\mathrm{OUT}}\right)$. The Yannakakis framework materializes an intermediate relation $\pi_{A_j, A_{j+1}, A_{k+1}} \mathcal{Q}_i$ in each step, for $j = k, k-1, \cdots, 1$, whose size can be bounded by

$$\left|\pi_{A_j, A_{j+1}, A_{k+1}} \mathcal{Q}_i\right| \le |R_j(A_j, A_{j+1})| \cdot \left|\pi_{A_{k+1}} \mathcal{Q}_i\right| = O\left(N \cdot \sqrt{\mathrm{OUT}}\right)$$

Hence, this step takes $O\left(N \cdot \sqrt{\mathrm{OUT}}\right)$ time.

Step 4: Compute $\mathcal{Q}_*$. Note that we have computed $S_{k-1}(A_1, A_k)$ in Step 1. We next simply compute $\oplus_{A_k} S_{k-1}(A_1, A_k) \bowtie R_k(A_k, A_{k+1})$ for $\mathcal{Q}_*$. As there are N tuples in R_k and each of them can be joined with at most $\sqrt{\tilde{\mathrm{OUT}}}$ tuples in S_{k-1}, the total number of intermediate join result is $O\left(N \cdot \sqrt{\mathrm{OUT}}\right)$. Hence, this step takes $O\left(N \cdot \sqrt{\mathrm{OUT}}\right)$ time. It is equivalent to invoking the Yannakakis framework over a join tree rooted at R_k.

Step 5: Aggregate all subqueries. Finally, we aggregate the results of all subqueries. As each subquery produces at most OUT results, the total number of results is $O(\mathrm{OUT})$. Hence, this step takes $\widetilde{O}(\mathrm{OUT})$ time.

Theorem 5. *For $\mathcal{Q}_{\mathsf{line}}$ and an arbitrary instance $\mathcal{R}$ of input size N and output size* OUT*, if a $O(1)$-approximation of* OUT *is known, the $\mathcal{Q}_{\mathsf{line}}(\mathcal{R})$ can be computed in $\widetilde{O}\left(N \cdot \sqrt{\mathrm{OUT}}\right)$ time.*

3 Fat Star Query

We next consider a *fat star* query $\mathcal{Q}_{\mathsf{fstar}}$, which is a generalization of the star queries (a.k.a. star matrix multiplication):

$$\bigoplus_{B_1, B_2, \cdots, B_k} R_0(B_1, B_2, \cdots, B_k) \bowtie R_1(A_1, B_1) \bowtie \cdots \bowtie R_k(A_k, B_k)$$

Moreover, $\mathsf{outw}(\mathcal{Q}_{\mathsf{fstar}}) = k$. Now, suppose we are given $\tilde{\mathrm{OUT}}$ such that $\tilde{\mathrm{OUT}} \le \mathrm{OUT} \le 2 \cdot \tilde{\mathrm{OUT}}$. Our algorithm consists of five steps:

Step 1: Compute data statistics. Consider an arbitrary $i \in [k]$. We first compute for each value $b \in \mathrm{dom}(B_i)$ its *degree* $d_i(b)$ in R_i, defined as $d_i(b) = |\sigma_{B_i=b} R_i|$. A value $b \in \mathrm{dom}(B_i)$ is *heavy* if $d_i(b) > \tilde{\mathrm{OUT}}^{\frac{1}{k}}$, and *light* otherwise. Let $B_i^{\mathsf{heavy}}, B_i^{\mathsf{light}}$ be the set of heavy and light tuples in attribute B_i respectively. Let $R_i^{\mathsf{heavy}} = R_i \ltimes B_i^{\mathsf{heavy}}, R_i^{\mathsf{light}} = R_i \ltimes B_i^{\mathsf{light}}$ be the set of heavy and light tuples in relation R_i respectively. This step can be done within $O(N)$ time. We point out two straightforward observations:

Lemma 2. *For any tuple $t \in R_0$, $\prod_{i\in[k]} d_i(\pi_{B_i} t) \le \mathrm{OUT}$.*

Lemma 3. *For any $b \in \mathrm{dom}(B_i)$, $d_i(b) \cdot |\sigma_{B_i=b} \mathcal{X}_i| \le \mathrm{OUT}$, where $\mathcal{X}_i = \oplus_{\bar{\mathbf{y}}-\{B_i\}} R_0 \Join \left(\Join_{j\in[k]-\{i\}} R_j\right)$.*

Step 2: Partition $\mathcal{Q}_{\mathsf{fstar}}$. Similarly, we can partition $\mathcal{Q}_{\mathsf{fstar}}$ into the $k+1$ sub-queries:

$$\mathcal{Q}_i = \bigoplus_{B_1,B_2,\cdots,B_k} R_0 \Join \left(\Join_{j\in[i-1]} R_j^{\mathsf{light}}\right) \Join R_i^{\mathsf{heavy}} \Join \left(\Join_{j=i+1}^{k} R_j\right), \text{ for } i \in [k]$$

$$\mathcal{Q}_* = \bigoplus_{B_1,B_2,\cdots,B_k} R_0 \Join \left(\Join_{j\in[k]} R_j^{\mathsf{light}}\right).$$

Step 3: Compute $\mathcal{Q}_i$ for each $i \in [k]$. We invoke the Yannakakis framework over a join tree rooted at R_i. To analyze the cost of this step, we need to unravel the execution of Yannakakis framework on $\mathcal{Q}_i$, which computes

- $R_{\mathsf{new}}(B_i, A_1, \cdots, A_{i-1}, A_{i+1}, \cdots, A_k) = \bigoplus_{B_1,\cdots,B_{i-1},B_{i+1},\cdots,B_k} R_0 \Join \left(\Join_{j\in[i-1]} R_j^{\mathsf{light}}\right) \Join \left(\Join_{j=i+1}^{k} R_j\right)$;
- $\bigoplus_{B_i} R_i^{\mathsf{heavy}}\left(A_i, B_i^{\mathsf{heavy}}\right) \Join R_{\mathsf{new}}\left(B_i, A_1, \cdots, A_{i-1}, A_{i+1}, \cdots, A_k\right)$;

After removing dangling tuples, each tuple $t \in R_0$ can be joined with at least $d_i(\pi_{B_i} t) > \tilde{\mathrm{OUT}}^{\frac{1}{k}}$ tuples in R_i^{heavy}. Implied by Lemma 2, each tuple $t \in R_0$ can participate in at most $O\left(\mathrm{OUT}^{1-\frac{1}{k}}\right)$ join results. Hence, the number of intermediate join result materialized for computing R_{new} is at most $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$. Moreover, each value $b \in \mathrm{dom}(B_i)$ can be joined with $d_i(b) > \tilde{\mathrm{OUT}}^{\frac{1}{k}}$ tuples in R_i^{heavy}. Implied by Lemma 3, each $b \in B_i^{\mathsf{heavy}}$ can appear in at most $O\left(\mathrm{OUT}^{1-\frac{1}{k}}\right)$ tuples in R_{new}. As there are at most N tuples in R_i^{heavy}, the number of intermediate join result materialized is at most $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$. Together, this steps takes $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$ time.

Step 4: Compute $\mathcal{Q}_*$. For each $i \in [k]$, we compute the following intermediate query result

$$S_i\left(B_i, A_1, \cdots, A_{i-1}, A_{i+1}, \cdots, A_k\right) = \bigoplus_{B_1,\cdots,B_{i-1},B_{i+1},\cdots,B_k} R_0 \Join \left(\Join_{j\in[k]-\{i\}} R_j^{\mathsf{light}}\right).$$

We compute for each value $b \in \mathrm{dom}(B_i)$, its degree $\triangle(b)$ in relation S_i, defined as $\triangle(b) = |\sigma_{B_i=b} S_i|$. A value $b \in \mathrm{dom}(B_i)$ is *large* if $\triangle(b) > 2 \cdot \tilde{\mathrm{OUT}}^{1-\frac{1}{k}}$, and *small* otherwise. Let $B_i^{\mathsf{large}}, B_i^{\mathsf{small}}$ be the set of large and small values in B_i^{light} respectively. Let $R_i^{\mathsf{large}} = R_i^{\mathsf{light}} \ltimes B_i^{\mathsf{large}}, R_i^{\mathsf{small}} = R_i^{\mathsf{light}} \ltimes B_i^{\mathsf{small}}$ be the set of large and small tuples in R_i^{light} respectively. This way, we can further reduce $\mathcal{Q}_*$ into following $k+1$ sub-queries:

$$\mathcal{Q}_{*i} = \bigoplus_{B_i} R_i^{\mathsf{small}}\left(A_i, B_i^{\mathsf{small}}\right) \Join S_i\left(B_i, A_1, \cdots, A_{i-1}, A_{i+1}, \cdots, A_k\right)$$

for $i \in [k]$, and $\mathcal{Q}_{**} = \oplus_{B_1,B_2,\cdots,B_k} R_0 \Join \left(\Join_{j\in[k]} R_j^{\mathsf{large}}\right)$. However, we don't need to compute $\mathcal{Q}_{**}$ due to the following fact:

Algorithm 3: $\textsc{FatStar}\left(\mathcal{Q}_{\mathsf{fstar}}, \mathcal{R}, \tilde{\mathrm{OUT}}\right)$

Input : A fat star query $\mathcal{Q}_{\mathsf{fstar}}$, an instance $\mathcal{R}$ of input size N and output size OUT, and parameter $\tilde{\mathrm{OUT}}$ such that $\tilde{\mathrm{OUT}} \le \mathrm{OUT} \le 2 \cdot \tilde{\mathrm{OUT}}$;

Output: Query result $\mathcal{Q}_{\mathsf{fstar}}(\mathcal{R})$;

1 **foreach** $i \in [k]$ **do**

2 $\quad B_i^{\mathsf{light}} \leftarrow \left\{ b \in \mathrm{dom}(B_i) : 1 \le |\sigma_{B_i = b} R_i| \le \tilde{\mathrm{OUT}}^{\frac{1}{k}} \right\}$;

3 $\quad B_i^{\mathsf{heavy}} \leftarrow \left\{ b \in \mathrm{dom}(B_i) : |\sigma_{B_i = b} R_i| > \tilde{\mathrm{OUT}}^{\frac{1}{k}} \right\}$;

4 $\quad R_i^{\mathsf{light}} \leftarrow R_i \ltimes B_i^{\mathsf{light}}$;

5 $\quad R_i^{\mathsf{heavy}} \leftarrow R_i \ltimes B_i^{\mathsf{heavy}}$;

6 $\quad \mathcal{Q}_i \leftarrow \bigoplus_{B_1, B_2, \cdots, B_k} R_0 \bowtie \left(\bowtie_{j=1}^{i-1} R_j^{\mathsf{light}}\right) \bowtie R_i^{\mathsf{heavy}} \bowtie \left(\bowtie_{j=i+1}^{k} R_j\right)$;

7 **foreach** $i \in [k]$ **do**

8 $\quad S_i \leftarrow \bigoplus_{B_1, \cdots, B_{i-1}, B_{i+1}, \cdots, B_k} R_0 \bowtie \left(\bowtie_{j \in [k]-\{i\}} R_j^{\mathsf{light}}\right)$;

9 $\quad B_i^{\mathsf{small}} \leftarrow \left\{ b \in B_i^{\mathrm{light}} : |\sigma_{B_i = b} S_i| \le \tilde{\mathrm{OUT}}^{1-\frac{1}{k}} \right\}$;

10 $\quad R_i^{\mathsf{small}} \leftarrow R_i^{\mathrm{light}} \ltimes B_i^{\mathsf{small}}$;

11 $\quad \mathcal{Q}_{*i} \leftarrow \oplus_{B_i} R_i^{\mathsf{small}} \bowtie S_i$;

12 **return** $\mathcal{Q}_1 \oplus \cdots \oplus \mathcal{Q}_{k-1} \oplus \mathcal{Q}_{1*} \oplus \cdots \oplus \mathcal{Q}_{k*}$;

Lemma 4. $\mathcal{Q}_{**} = \emptyset$.

Proof of Lemma 4. Consider an arbitrary $i \in [k]$. After removing all dangling tuples, each value in $\mathrm{dom}(A_i)$ appears in at least $2 \cdot \tilde{\mathrm{OUT}}^{1-\frac{1}{k}}$ query results via some value in $\mathrm{dom}(B_i)$. Hence, the active domain size of A_i is at most $\frac{\mathrm{OUT}}{2 \cdot \tilde{\mathrm{OUT}}^{1-\frac{1}{k}}} \le \frac{\mathrm{OUT}}{2 \cdot (\frac{\mathrm{OUT}}{2})^{1-\frac{1}{k}}} \le \mathrm{OUT}^{\frac{1}{k}}$. Suppose $\mathcal{Q}_{**} \neq \emptyset$. Let t be an arbitrary join result of the underlying join. Hence, $\pi_{B_i} t \in B_i^{\mathsf{large}}$ for every $i \in [k]$. Consider an arbitrary $j \in [k]$. The tuple $t' = \pi_{B_j} t$ appears in at most

$$\left|\sigma_{B_j = t'} S_j\right| \le \prod_{i \in [k]-\{j\}} |\mathrm{dom}(A_i)| \le \left(\mathrm{OUT}^{\frac{1}{k}}\right)^{k-1} = \mathrm{OUT}^{1-\frac{1}{k}} \le 2 \cdot \tilde{\mathrm{OUT}}^{1-\frac{1}{k}}$$

distinct tuples in S_j, contradicting the fact that $\pi_{B_j} t \in B_j^{\mathsf{large}}$. Hence, such a join result t does not exist, and $\mathcal{Q}_{**} = \emptyset$. □

We show that S_i can be computed in $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$ time. There are at most N tuples in R_0. Each tuple $t \in R_0$ can be joined with at most $\prod_{j \in [k]-\{i\}} d_j\left(\pi_{B_j} t\right) \le \tilde{\mathrm{OUT}}^{1-\frac{1}{k}}$ tuples in $\left(\bowtie_{j \in [k]-\{i\}} R_j^{\mathsf{light}}\right)$. Hence, the intermediate join size as well as the input size of S_i, can be bounded by $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$. All other data statistics can be computed in $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$ time. For computing $\mathcal{Q}_{*i}$, we note that each tuple $t \in R_i^{\mathsf{small}}$ can be joined with at most $\tilde{\mathrm{OUT}}^{1-\frac{1}{k}}$ tuples in S_i. As there are at most N tuples in R_i^{small}, the number of intermediate join result materialized is at most $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$.

Step 5: Aggregate all subqueries. We aggregate the results of all subqueries. As each subquery produces at most OUT results, and there are $O(1)$ subqueries, this step takes $\widetilde{O}(\mathrm{OUT})$ time.

Theorem 6. *For $\mathcal{Q}_{\mathsf{fstar}}$ of k relations and an instance $\mathcal{R}$ of input size N and output size* OUT*, if a $O(1)$-approximation of* OUT *is known, the $\mathcal{Q}_{\mathsf{fstar}}(\mathcal{R})$ can be computed in $\widetilde{O}\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$ time.*

4 Acyclic Query

4.1 Framework

Component 1: Separate$(\mathcal{Q}, \mathcal{R})$. We start with transforming an arbitrary acyclic query $\mathcal{Q}$ into a *well-separated* one with nice properties:

Definition 4 (Well-separated Query). *An acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ is well-separated, if it is cleansed and existentially connected while satisfying the following two conditions:*

① *every output attribute in $\mathbf{y}$ is a unique attribute;*

② *for each relation $e \in \mathcal{E}$ with $e \cap \mathbf{y} \neq \emptyset$, there exists some other relation $e' \in \mathcal{E} - \{e\}$ such that $e - \mathbf{y} \subseteq e'$.*

Lemma 5. *For any cleansed and existentially-connected acyclic join-aggregate query $\mathcal{Q}$ and an instance $\mathcal{R}$ of input size N, there is an algorithm that can construct within $O(N)$ time a well-separated acyclic join-aggregate query $\mathcal{Q}'$ and an instance $\mathcal{R}'$ of input size $\Theta(N)$, such that $\mathsf{outw}(\mathcal{Q}) = \mathsf{outw}(\mathcal{Q}')$ and $\mathcal{Q}(\mathcal{R}) = \mathcal{Q}'(\mathcal{R}')$.*

The high-level idea behind the transformation is as follows. See a running example in Figure 3. Let $\mathbf{y}_\bullet \subseteq \mathbf{y}$ be the set of unique output attributes.

① As proved in Lemma 18, it is always feasible to find a subset $\mathcal{E}_\mathbf{y} \subseteq \mathcal{E}$ of outw relations, such that for each output attribute $A \in \mathbf{y}$, there exists some relation $e \in \mathcal{E}_\mathbf{y}$ with $A \in e$. We assign each attribute $A \in \mathbf{y}$ to an arbitrary relation $e \in \mathcal{E}_\mathbf{y}$ with $A \in e$. Let $\pi : \mathbf{y} \to \mathcal{E}_\mathbf{y}$ be the assignment function. For each joint output attribute $A \in \mathbf{y} - \mathbf{y}_\bullet$, we simply add a unique attribute x_A to relation $\pi(A)$ and "turn" A into a non-output attribute. To preserve the equivalence of the query results, we force a one-to-one mapping between $\mathrm{dom}(A)$ and $\mathrm{dom}(\pi(A))$. The annotations of all tuples stay the same.

② Now we are left with a cleansed and existentially connected acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, such that each output attribute $x \in \mathbf{y}$ is a unique attribute. Consider an arbitrary relation $e \in \mathcal{E}$ with $e \cap \mathbf{y} \neq \emptyset$. If there exists no other relation $e' \in \mathcal{E}$ such that $e - \mathbf{y} \subseteq e'$, we just add another relation e'' such that $e \cap e'' = e \cap \mathbf{y}$ and the remaining attribute in $e'' - e$ is a unique output attributes, and "turn" all attributes in $e \cap e''$ into non-output attributes. Similarly, to preserve the equivalence of the query results, we force a one-to-one mapping between the domain of attributes in $e \cap e''$ and that of attribute in $e'' - e$. The annotation of each tuple in such newly added relations is always 1.

It can be easily checked that in both steps of transformation, the out-width of the queries does not change. If there is an algorithm that can compute an arbitrary instance $\mathcal{R}'$ for $\mathcal{Q}'$ of input size N and output size OUT within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}(\mathcal{Q}')}} + \mathrm{OUT}\right)$ time, then there is an algorithm that can compute an arbitrary instance $\mathcal{R}$ for $\mathcal{Q}$ of input size $\Theta(N)$ and output size OUT within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}(\mathcal{Q})}} + \mathrm{OUT}\right)$ time. Below, we focus on well-separated queries. The complete procedure is given in Algorithm 4. Suppose we are given $\tilde{\mathrm{OUT}}$ such that $\tilde{\mathrm{OUT}} \le \mathrm{OUT} \le 2 \cdot \tilde{\mathrm{OUT}}$.

Component 2: Join Tree and Edge Label. Let $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ be an arbitrary well-separated query. Since $\mathcal{Q}$ is cleansed, every unique attribute is an output attribute. Moreover, as $\mathcal{Q}$ is well-separated, each output attribute is also unique. Let $\mathcal{E}_\bullet = \{e \in \mathcal{E} : e \cap \mathbf{y} \neq \emptyset\}$ denote the set of relations that contain some (unique) output attribute. In an arbitrary join tree $\mathcal{T}$ of $\mathcal{Q}$, a node e_1 is a neighbor to another node $e_2 \in \mathcal{T}$ if there is an edge between e_1, e_2 in $\mathcal{T}$. For a node e, let $\mathcal{N}_e$ denote the set of neighbors of e in $\mathcal{T}$. A node is a *leaf* if $|\mathcal{N}_e| = 1$. We show an important observation for any well-separate join-aggregate query:

Lemma 6. *For any well-separated query $\mathcal{Q}$, there exists a join tree $\mathcal{T}$ for $\mathcal{Q}$ such that $\mathcal{E}_\bullet$ is exactly the set of leaf nodes of $\mathcal{T}$.*

For a pair of incident nodes e_1, e_2, we use $\{e_1, e_2\}$ to denote the undirected edge between them, use (e_1, e_2) (resp. (e_2, e_1)) to denote the directed edge from e_1 to e_2 (resp. from e_2 to e_1). Removing the edge $\{e_1, e_2\}$ separates $\mathcal{T}$ into two connected subtrees $\mathcal{T}_{e_1,e_2}$ and $\mathcal{T}_{e_2,e_1}$, which contains e_1 and e_2 separately. Moreover, $\mathcal{T}_{e_1,e_2}$ defines a join-project query over relations residing in this subtree as:

$$\mathcal{Q}_{e_1,e_2}(\mathcal{R}) := \pi_{\mathbf{y} \cup (e_1 \cap e_2)} \left(\Join_{e \in \mathcal{T}_{e_1,e_2}} R_e \right)$$

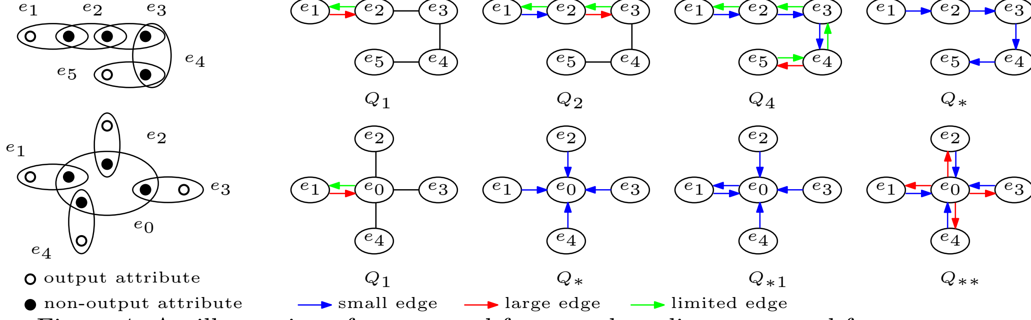

Figure 4: An illustration of our general framework on line query and fat star query.

For simplicity, we define $\phi_{e_1,e_2} = \frac{|\mathcal{E}_\bullet \cap \mathcal{T}_{e_1,e_2}|}{\mathsf{outw}}$ as the *fraction* of the number of relations containing some output attribute(s) from the subtree $\mathcal{T}_{e_1,e_2}$.

We next introduce the "label" to edges in $\mathcal{T}$, with respect to the instance $\mathcal{R}$. An edge (e_1, e_2) is *large* if $|\sigma_{e_1\cap e_2 = t}\mathcal{Q}_{e_1,e_2}(\mathcal{R})| > \tilde{\mathrm{OUT}}^{\phi_{e_1,e_2}}$ holds for every tuple $t \in \pi_{e_1\cap e_2} R_{e_1}$, and *small* if $|\sigma_{e_1\cap e_2 = t}\mathcal{Q}_{e_1,e_2}(\mathcal{R})| \leq \tilde{\mathrm{OUT}}^{\phi_{e_1,e_2}}$ holds for every tuple $t \in \pi_{e_1\cap e_2} R_{e_1}$. Furthermore, we identify a more constrained label for small edges. An edge (e_1, e_2) is *limited* if $|\pi_{\mathbf{y}}\mathcal{Q}_{e_1,e_2}(\mathcal{R})| \leq \tilde{\mathrm{OUT}}^{\phi_{e_1,e_2}}$. Note that a limited edge must be small, but not vice versa.

Component 3: Easy Instance for Yannakakis Framework. We point out a critical observation on the connection between small edges and the Yannakakis framework (see Section 4.2):

Lemma 7. *For a well-separated join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, given any instance $\mathcal{R}$ of input size N and output size* OUT*, if there is a join tree $\mathcal{T}$ with a leaf node $e_1 \in \mathcal{E}_\bullet$ whose unique in-going edge is small, the $\mathcal{Q}(\mathcal{R})$ can be computed within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ time, where* outw *is the out-width of $\mathcal{Q}$.*

Component 4: Partition Procedure. However, such a leaf node with an in-going small edge may not always exist. Also, it could be time-consuming to "check" whether an edge is small or not, since the corresponding subquery may need to materialize a large number of intermediate results. The most interesting part of this framework is an efficient way to partition the input instance such that every sub-instance is "easy", i.e., admits a join tree containing a leaf node with the unique in-going edge as small. More specifically, we aim to obtain (in Section 4.3):

Lemma 8. *For a well-separated query $\mathcal{Q}$ and an arbitrary instance $\mathcal{R}$, there is an algorithm that can partition $\mathcal{R}$ into $O(1)$ sub-instances $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_\ell$ within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right)$ time such that $\mathcal{Q}(\mathcal{R}) = \bigoplus_{i\in[\ell]} \mathcal{Q}(\mathcal{R}_i)$, and for each $\mathcal{R}_i$, there exists a join tree $\mathcal{T}$ with a leaf node $e \in \mathcal{E}_\bullet$ such that the edge $(*, e)$ is small, where* outw *is the out-width of $\mathcal{Q}$ and where $\mathcal{E}_\bullet = \{e \in \mathcal{E} : e \cap \mathbf{y} \neq \emptyset\}$.*

Combine Lemma 5, Lemma 8 and Lemma 7, we obtain:

Theorem 7. *For any existentially-connected acyclic join-aggregate query $\mathcal{Q}$ and an arbitrary instance $\mathcal{R}$ of input size N and output size* OUT*, if a $O(1)$-approximation of* OUT *is known, the $\mathcal{Q}(\mathcal{R})$ can be computed in $\widetilde{O}\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ time, where* outw *is the out-width of $\mathcal{Q}$.*

4.2 Proof of Lemma 7

We simply invoke the Yannakakis framework along the join tree $\mathcal{T}$ rooted at e_1. Let e_2 be the unique node incident to e_1. As edge (e_2, e_1) is small, every tuple $t \in \mathrm{dom}(e_1 \cap e_2)$ appears in no more than $\tilde{\mathrm{OUT}}^{1-\frac{1}{\mathsf{outw}}}$ query result in $\mathcal{Q}_{e_2,e_1}$, since $|\mathcal{E}_\bullet \cap \mathcal{T}_{e_2,e_1}| = |\mathcal{E}_\bullet| - |\mathcal{E}_\bullet \cap \mathcal{T}_{e_1,e_2}| = \mathsf{outw} - 1$. Consider any node e with its parent e' in $\mathcal{T}$ (rooted at e_1). It is easy to see that every tuple $t' \in \mathrm{dom}(e \cap e')$ appears in no more than $\tilde{\mathrm{OUT}}^{1-\frac{1}{\mathsf{outw}}}$ query result in $\mathcal{Q}_{e,e'}$. Suppose not, assume there exists some tuple $t' \in \mathrm{dom}(e \cap e')$ appears in more than

$\tilde{\text{OUT}}^{1-\frac{1}{\text{outw}}}$ query result in $\mathcal{Q}_{e,e'}$. After removing all dangling tuples, there must exist some full join result t such that $\pi_{e \cap e'} = t'$. We note that $\pi_{e_1 \cap e_2} t \in \pi_{e_1 \cap e_2} R_{e_1}$; and moreover, the tuple $\pi_{e_1 \cap e_2} t$ will appear in at least $\tilde{\text{OUT}}^{1-\frac{1}{\text{outw}}}$ query result in $\mathcal{Q}_{e_1,e_2}$ via t', contradicting the fact that edge (e_2, e_1) is small.

Now, we are ready to analyze the cost of Yannakakis framework on $\mathcal{T}$ (rooted at e_1). Each time, it equivalently picks an internal node e' whose children are all leaves, say $\mathcal{L}$, and materializes $R_{e'} \bowtie (\bowtie_{e \in \mathcal{L}} \mathcal{Q}_{e,e'})$. As there are at most N tuples in $R_{e'}$, and each tuple can be joined with at most $\tilde{\text{OUT}}^{1-\frac{1}{\text{outw}}}$ tuples in $\bowtie_{e \in \mathcal{L}} \mathcal{Q}_{e,e'}$, the number of intermediate join result materialized is always bounded by $O\left(N \cdot \text{OUT}^{1-\frac{1}{\text{outw}}}\right)$. Hence, Yannakakis framework takes $O\left(N \cdot \text{OUT}^{1-\frac{1}{\text{outw}}}\right)$ time.

4.3 Proof of Lemma 8

At last, we come to the most technical part of Algorithm 4. We start with an input instance $\mathcal{R}$ and an unlabeled join tree $\mathcal{T}$. We iteratively partition $\mathcal{R}$ until all sub-instances fall into the "easy case" as captured by Lemma 7 in **line 5-9**. In general, we can always apply two rules to label edges in **line 10** and **line 11-12**, as described in Lemma 9 and Lemma 10.

Lemma 9 (Large-Reverse-Limited)**.** *For a pair of incident nodes $\{e_1, e_2\}$, if edge (e_1, e_2) is large, edge (e_2, e_1) must be limited.*

Lemma 10 (Limited-Imply-Limited)**.** *For any node e_1 with its neighbors $\mathcal{N}_{e_1}$, if edge (e_3, e_1) is limited for every node $e_3 \in \mathcal{N}_{e_1}$ except $e_2 \in \mathcal{N}_{e_1}$, then edge (e_1, e_2) must be limited.*

If no more edges can be labeled, we can further partition the input instance $\mathcal{R}$ as **line 13-18**. Consider an unlabeled edge (e_1, e_2) such that (e_3, e_1) is small for every node $e_3 \in \mathcal{N}_{e_1} - \{e_2\}$. We first compute $\mathcal{Q}_{e_1,e_2}$ by invoking the Yannakakis framework with the join tree $\mathcal{T}_{e_1,e_2}$ rooted at e_1. A tuple $t \in \text{dom}(e_1 \cap e_2)$ is *heavy* if it can be joined with at least $\tilde{\text{OUT}}^{\phi_{e_1,e_2}}$ query result in $\mathcal{Q}_{e_1,e_2}$, and *light* otherwise. We also partition tuples in R_{e_1} as *heavy* and *light*, by attributes $e_1 \cap e_2$. Now, we can partition the instance into two sub-instances, which contains heavy and light tuples in R_{e_1} separately, and two copies of $\mathcal{T}$ in which edge (e_1, e_2) is labeled as large and small separately. We continue applying this procedure recursively to every sub-instance, until all of them falls into the case in **line 5-9**. At last, we just aggregate all these sub-queries. See two examples in Figure 4.

Example 1. *For $\mathcal{Q}_{\mathsf{line}}$ with $k = 5$, $\mathcal{E}_\bullet = \{e_1, e_5\}$. We start with an arbitrary leaf node say e_1, and partition R_1 into $R_1^{\mathsf{heavy}}, R_1^{\mathsf{light}}$. For $\mathcal{Q}_1$, edge (e_1, e_2) is large. Implied by Lemma 9, edge (e_2, e_1) is limited. We next choose e_2. As edge (e_1, e_2) is small, we further compute $\mathcal{Q}_{e_2,e_3}$ (i.e., S_2) and partition R_2 into $R_2^{\mathsf{heavy}}, R_2^{\mathsf{light}}$. We continue this process until obtaining k sub-queries as in Section 2. Consider an arbitrary sub-query $\mathcal{Q}_i$ in which edge (e_i, e_{i+1}) is large. Implied by Lemma 9, edge (e_{i+1}, e_i) is limited. Applying Lemma 10 to edges $(e_i, e_{i-1}), (e_{i-1}, e_{i-2}), \cdots, (e_2, e_1)$ iteratively, we conclude that edge (e_2, e_1) is limited. Consider the remaining sub-instance $\mathcal{Q}_*$ such that (e_i, e_{i+1}) is small for every $i \in [4]$. It automatically falls into the "easy case", since $e_5 \in \mathcal{E}_\bullet$ is a leaf node.*

Example 2. *For $\mathcal{Q}_{\mathsf{fstar}}$ with $k = 4$, $\mathcal{E}_\bullet = \{e_1, e_2, e_3, e_4\}$. We start with an arbitrary leaf node say e_1, and partition R_1 into R_1^{heavy} and R_1^{light}. For $\mathcal{Q}_1$, edge (e_1, e_0) is small. We continue to visit e_2 and further partition R_2. After visiting all leaf nodes, we are left with a sub-instance in which (e_0, e_i) is large for every $i \in [4]$. Implied by Lemma 9, (e_i, e_0) is limited for every $i \in [4]$. We pick an unlabeled edge (e_0, e_1) and further partition R_{e_0} by $\mathcal{Q}_{e_0,e_1}$ (i.e., S_1). For $\mathcal{Q}_{*1}$, edge (e_0, e_1) is small. It continues to consider (e_0, e_2) and further partitions the remaining sub-instance. At last, we are left with $\mathcal{Q}_{**}$, where (e_0, e_i) is labeled as large for every $i \in [4]$. Implied by Lemma 9, edge (e_i, e_0) must be limited for every $i \in [4]$. Implied by Lemma 10, edge (e_0, e_4) must be limited, which contradicts the fact that (e_0, e_4) is large. Hence, $\mathcal{Q}_{**} = \emptyset$.*

Correctness. Although the Algorithm 4 is non-deterministic, we can show that Algorithm 4 quickly terminates after running the while-loop at line 4-19 for at most $2 \cdot |\mathcal{E}|$ iterations.

Lemma 11. *Algorithm 4 terminates after running the while-loop at line 3-20 for at most $2 \cdot |\mathcal{E}|$ iterations.*

Algorithm 4: HybridYannakakisWithOUT ($\mathcal{Q}, \mathcal{R}, \tilde{\text{OUT}}$)

Input : A well-separated query $\mathcal{Q}$, an instance $\mathcal{R}$ of input size N and output size OUT, and parameter $\tilde{\text{OUT}}$ such that $\tilde{\text{OUT}} \le \text{OUT} \le 2 \cdot \tilde{\text{OUT}}$;

Output: A set of sub-instances as described in Lemma 8;

1 $\mathcal{T} \leftarrow$ a unlabeled join tree of $\mathcal{Q}$ whose leaf nodes correspond to $\mathcal{E}_\bullet = \{e \in \mathcal{E} : e \cap \mathbf{y} \neq \emptyset\}$;
2 $\mathcal{P} \leftarrow \{(\mathcal{T}, \mathcal{R})\}$, $\mathcal{S} \leftarrow \emptyset$;
3 **while** $\mathcal{P} \neq \emptyset$ **do**
4   **foreach** $(\mathcal{T}, \mathcal{R}) \in \mathcal{P}$ **do**
5   **if** $\exists$ *a leaf node* e *in* $\mathcal{T}$ *s.t. edge* $(*, e)$ *is small* **then**
6       $\mathcal{P} \leftarrow \mathcal{P} - \{(\mathcal{T}, \mathcal{R})\}$;
7       $\mathcal{L} \leftarrow \mathcal{Q}(\mathcal{R})$ computed by Yannakakis framework along $\mathcal{T}$ rooted at e;
8   **if** $\mathcal{S} = \emptyset$ **then** $\mathcal{S} \leftarrow \mathcal{L}$;
9   **else** $\mathcal{S} \leftarrow \mathcal{S} \oplus \mathcal{L}$;
10   **if** $\exists$ *unlabeled* (e_1, e_2) *in* $\mathcal{T}'$ *s.t.* (e_2, e_1) *is large* **then** label (e_1, e_2) as limited;
11   **if** $\exists$ *unlabeled* (e_1, e_2) *in* $\mathcal{T}'$ *s.t.* (e_3, e_1) *is limited for every* $e_3 \in \mathcal{N}_{e_1} - \{e_2\}$ **then**
12   label (e_1, e_2) as limited;
13   **if** $\exists$ *unlabeled* (e_1, e_2) *s.t.* (e_3, e_1) *is small for every* $e_3 \in \mathcal{N}_{e_1} - \{e_2\}$ **then**
14       $H_{e_1,e_2} \leftarrow \left\{t \in \text{dom}(e_1 \cap e_2) : |\sigma_{e_1 \cap e_2 = t} \mathcal{Q}_{e_1,e_2}| > \tilde{\text{OUT}}^{\phi_{e_1,e_2}}\right\}$;
15       $\mathcal{R}_1 \leftarrow \mathcal{R} - \{R_{e_1}\} + \{R_{e_1} \ltimes H_{e_1,e_2}\}$;
16       $\mathcal{R}_2 \leftarrow \mathcal{R} - \{R_{e_1}\} + \{R_{e_1} \triangleright H_{e_1,e_2}\}$;
17       $\mathcal{T}_1, \mathcal{T}_2 \leftarrow \mathcal{T}$ with (e_1, e_2) labeled as large, small separately;
18       $\mathcal{P} \leftarrow \mathcal{P} - \{(\mathcal{T}, \mathcal{R})\} + \{(\mathcal{T}_1, \mathcal{R}_1), (\mathcal{T}_2, \mathcal{R}_2)\}$;
19 **return** $\mathcal{S}$;

Lemma 12 (Not-All-Large)**.** *For an arbitrary connected subtree $\mathcal{T}_1$ of $\mathcal{T}$, let $\mathcal{T}_2$ be the resulted subtree after removing $\mathcal{T}_1$ from $\mathcal{T}$. If $\mathcal{T}_2$ is disconnected, then there must exist a pair of nodes $e_1 \in \mathcal{T}_1, e_2 \in \mathcal{T}_2$ such that (e_1, e_2) is an edge in $\mathcal{T}$ but not large.*

Proof of Lemma 11. In each iteration of the while-loop, for every sub-instance that do not fall into the base case, one unlabeled edge gets labeled. Below, we will prove that for a sub-instance with an arbitrary partial labeling, if no labeling rule can be applied further, it must fall into the base case at line 5. As there are most $2 \cdot |\mathcal{E}|$ unlabeled edges in the beginning, the while-loop runs at most $2 \cdot |\mathcal{E}|$ iterations until all edges are labeled. Hence, Algorithm 4 will terminate after running the while-loop for at most $2 \cdot |\mathcal{E}|$ iterations. Consider any sub-instance $\mathcal{R}'$ with an arbitrary partial labeling on $\mathcal{T}$. By contradiction, we assume that there exists no unlabeled edge to which line 10, or line 11-12, or line 13-18 of Algorithm 4 can be applied. In Algorithm 5, we show how to identify a leaf node $e \in \mathcal{T}$ such that $(*, e)$ is small, hence this sub-instance always falls into the easy case.

The high-level idea of Algorithm 5 is to conceptually remove nodes in $\mathcal{T}$, until a single node is left, which is exactly the leaf node as desired. We root $\mathcal{T}$ at an arbitrary node $r \in \mathcal{E}_\bullet \cap \mathcal{T}$. Let p_e be the parent of e in this rooted tree. If there exists a node e such that the edge (e, p_e) is large, but the edge $(e', p_{e'})$ is small for every node e' residing in the subtree rooted at e, we simply remove the whole subtree $\mathcal{T}_{e,p_e}$ and continue. Otherwise, every (e, p_e) is small, including the edge $(*, r)$ incident to r, hence r is returned.

It remains to show that line 5 will always be triggered by Algorithm 5. In the execution, $\mathcal{T}$ is always a connected subtree. Moreover, $\mathcal{T} \cap \mathcal{E}_\bullet \neq \emptyset$. To show this, we need an important observation in Lemma 12. By contradiction, assume $\mathcal{T} \cap \mathcal{E}_\bullet = \emptyset$. Let $\mathcal{T}'$ be the subtree(s) removed from the initial join tree. As $\mathcal{T}$ is connected and $\mathcal{T} \cap \mathcal{E}_\bullet = \emptyset$, $\mathcal{T}'$ must be disconnected. Moreover, for each pair of nodes $e_1 \in \mathcal{T}$ and $e_2 \in \mathcal{T}'$, the subtree $\mathcal{T}_{e_2,e_1}$ has been removed due to the fact that edge (e_1, e_2) is large. This way, Lemma 12 is violated on $\mathcal{T}$, coming to a contradiction. Hence, $\mathcal{T} \cap \mathcal{E}_\bullet \neq \emptyset$ always holds in the execution process. This means that it is always feasible to root $\mathcal{T}$ at some node in $\mathcal{T} \cap \mathcal{E}_\bullet$, and further shrink $\mathcal{T}$. As $\mathcal{T}$ has a limited size, it will finally end with the case that every edge (e, p_e) is small for any $e \in \mathcal{T}$, i.e., triggering line 5. ☐

Time Complexity. When an unlabeled edge gets labeled in **line 13-18**, the associated instance $\mathcal{R}$ will

Algorithm 5: IDENTIFYLEAF($\mathcal{Q}, \mathcal{R}, \mathcal{T}$)

Input : A well-separated query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, an instance $\mathcal{R}$ and a labeled join tree $\mathcal{T}$ with leaf nodes $\mathcal{E}_\bullet = \{e \in \mathcal{E} : e \cap \mathbf{y} \neq \emptyset\}$, in which no more edges can be labeled by line 10-18 in Algorithm 4;

Output: A leaf node as described in Lemma 7;

```
1 while true do
2   Root T at an arbitrary node r ∈ T ∩ E•;
3   foreach e ∈ T do p_e ← the parent node of e;
4   if ∃ a node e ∈ T such that (e, p_e) is large but (e', p_e') is small for every node e' in the subtree
      rooted at e then
5     T ← T − T_{e,p_e};
6   else return r;
```

be partitioned into two sub-instances $\mathcal{R}_1, \mathcal{R}_2$. As each edge will be labeled once, and there are $O(1)$ edges to be labeled, the total number of sub-instances returned at last is still $O(1)$. The cost of Algorithm 4 is dominated by that of computing $\mathcal{Q}_{e_1,e_2}$. We invoke the Yannakakis framework along the join tree $\mathcal{T}_{e_1,e_2}$ rooted at e_1. Following the similar analysis of Lemma 7, for an arbitrary node $e_3 \in \mathcal{N}_{e_1} - \{e_2\}$, the subquery $\mathcal{Q}_{e_3,e_1}$ can be computed within $O\left(N \cdot \mathrm{OUT}^{\phi_{e_3,e_1}}\right)$ time. Then, $\mathcal{Q}_{e_1,e_2}$ is computed as follows: $\mathcal{Q}_{e_1,e_2} = \pi_{\mathbf{y} \cup (e_1 \cap e_2)} R_{e_1} \bowtie \left(\bowtie_{e_3 \in \mathcal{N}_{e_1} - \{e_2\}} \mathcal{Q}_{e_3,e_1}\right)$. There are at most N tuples in R_{e_1}. Each of them can be joined with at most $\prod_{e_3 \in \mathcal{N}_{e_1} - \{e_2\}} \tilde{\mathrm{OUT}}^{\phi_{e_3,e_1}} = \tilde{\mathrm{OUT}}^{\phi_{e_1,e_2}} \leq \tilde{\mathrm{OUT}}^{1-\frac{1}{\mathsf{outw}}}$ tuples in $\left(\bowtie_{e_3 \in \mathcal{N}_{e_1} - e_2} \mathcal{Q}_{e_3,e_1}\right)$, hence the $\mathcal{Q}_{e_1,e_2}$ can be computed within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right)$ time. All other statistics can be computed within asymptotically same time complexity. From Lemma 8, line 13-19 will be invoked at most $O(1)$ times. Hence, the total time complexity of Algorithm 4 can be bounded by $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right)$.

5 Lower Bound

In [**2**], it has been shown that for the sparse matrix multiplication problem, and parameters $N, \mathrm{OUT} \leq N^2$, there exists an instance $\mathcal{R}$ of input size $\Theta(N)$ and output size $\Theta(\mathrm{OUT})$ such that any semi-ring algorithm has to incur a cost of $\Omega\left(N \cdot \sqrt{\mathrm{OUT}}\right)$. The similar argument can be extended to star queries. We will also rely on this existing lower bound to establish our new lower bound.

Lemma 13 ([**2**])**.** *For the star join-aggregate query $\mathcal{Q}_{\mathsf{star}}$ of k relations, parameters $N, \mathrm{OUT} \leq N^k$, there exists an instance $\mathcal{R}$ of input size N and output size OUT such that any semi-ring algorithm for computing $\mathcal{Q}(\mathcal{R})$ requires at least $\Omega\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$ time.*

Proof of Lemma 13. We construct a hard instance $\mathcal{R}$ of input size $\Theta(N)$ and output size OUT for $\mathcal{Q}_{\mathsf{star}} = \oplus_B R_1(A_1, B) \bowtie R_2(A_2, B) \bowtie \cdots \bowtie R_k(A_k, B)$ as follows. There are $\mathrm{OUT}^{1/k}$ distinct values in each attribute A_i for $i \in [k]$. There are $N \cdot \mathrm{OUT}^{-1/k}$ distinct values in attribute B. Each relation R_i is a Cartesian product between A_i and B. It can be checked that each relation contains exactly N tuples. Any semi-ring algorithm has to compute all full join result, which is as large as $\Theta\left(N \cdot \mathrm{OUT}^{1-1/k}\right)$ for this hard instance, hence requires at least $\Omega\left(N \cdot \mathrm{OUT}^{1-1/k}\right)$ time. □

Proof of Theorem 1. Consider an arbitrary existentially-con-nected acyclic join-aggregate query $\mathcal{Q}$, with parameters $1 \leq N$ and $\mathrm{OUT} \leq N^{\mathsf{outw}}$. We show a reduction from $\mathcal{Q}$ to $\mathcal{Q}_{\mathsf{star}}$ with outw relations. Suppose we are given a hard instance $\mathcal{R}$ for $\mathcal{Q}_{\mathsf{star}}$. We distinguish two cases. First, we assume $\mathcal{Q}$ is cleansed for simplicity.

Implied by Lemma 19, let $S \subseteq \mathbf{y}$ be the set of output attributes such that no pair of them appears in the same relation, and $|S| = \mathsf{outw}$. Each output attribute $A \in \mathbf{y} - S$ contains one distinct value $\{*\}$.

Each output attribute $A \in S$ contains $\mathrm{OUT}^{\frac{1}{\mathsf{outw}}}$ distinct values. Each non-output attribute $B \in \bar{\mathbf{y}}$ contains $N \cdot \mathrm{OUT}^{-\frac{1}{\mathsf{outw}}}$ distinct values. For each relation R_e, $|e \cap S| \leq 1$. The projection of R_e onto all non-output attributes contains tuples in a form of $(b_i, b_i, \cdots, b_i)$, for $i \in \left[N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right]$. The projection of R_e onto all output attributes is the full Cartesian product. Moreover, $R_e = (\pi_{\mathbf{y}} R_e) \times (\pi_{\bar{\mathbf{y}}} R_e)$. For each output attribute $A \in S$, we choose an arbitrary relation $e \in \mathcal{E}$ such that $A \in e$. Note that all chosen relations are also distinct. Let $\mathcal{E}_S$ be the set of chosen relations. We specify an arbitrary one-to-one mapping from relations in $\mathcal{Q}_{\mathsf{star}}$ and relations in $\mathcal{E}_S$, say R_i corresponds to S_i. From our construction above, there is also a one-to-one mapping between tuples in R_i and S_i. We just set the annotation of a tuple $t \in S_i$ as the same as $t' \in R_i$, if t corresponds to t. For every remaining relation R_e in $\mathcal{R}$, we simply set the annotation of each tuple as 1. As $|e \cap S| \leq 1$ for each relation $e \in \mathcal{E}$, it can be checked that each relation contains at most N tuples. The output size of this query is exactly OUT.

It is not hard to see $\oplus_{\mathcal{V}-S} \mathcal{Q}(\mathcal{R}) = \mathcal{Q}_{\mathsf{star}}(\mathcal{R}_{\mathsf{star}})$. Any semi-ring algorithm that can compute $\mathcal{Q}(\mathcal{R})$ within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right)$ time, can also compute $\mathcal{Q}_{\mathsf{star}}(\mathcal{R}_{\mathsf{star}})$ within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}}\right)$ time. Hence, this automatically follows Lemma 13.

Now, we consider the second case where $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ is not cleansed. Let $\mathcal{Q}' = (\mathcal{V}', \mathcal{E}', \mathbf{y}')$ be the cleansed version of $\mathcal{Q}$, and $\mathcal{R}'$ be the instance constructed for $\mathcal{Q}'$ as the first case. Implied by the cleanse procedure, $\mathbf{y} = \mathbf{y}'$. Each non-output attribute $B \in \mathcal{V} - \mathcal{V}'$ contains one distinct value $\{*\}$. For each $e' \in \mathcal{E} - \mathcal{E}'$, there must exist a relation $e \in \mathcal{E}'$ such that $e' \subseteq e$. The relation $R_{e'}$ is just a projection of R_e onto attribute e', where each tuple has its annotation as 1. The argument for the first case applies here. ☐

Proof of Theorem 2. Consider an arbitrary acyclic join-aggre-gate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, with parameters $1 \leq N$ and $\mathrm{OUT} \leq N^{\rho^*(\mathcal{Q}_{\mathbf{y}})}$. Let outw be the out-width of $\mathcal{Q}$. The case when $\mathcal{Q}$ is existentially connected has been proved by Theorem 1. We next assume that $\mathcal{Q}$ is existentially disconnected. Let $\mathcal{Q}_1, \mathcal{Q}_2, \cdots, \mathcal{Q}_h$ be the connected sub-queries in $G^{\exists}_{\mathcal{Q}}$. Without loss of generality, assume $\mathsf{outw}(\mathcal{Q}_1) = \mathsf{outw}$. We can construct an instance as Theorem 1 with parameters N and $\min\{\mathrm{OUT}, N^{\mathsf{outw}}\}$. Suppose $\mathcal{Q}_1 = (\mathcal{V}_1, \mathcal{E}_1, \mathbf{y}_1)$. More specifically, each attribute in $\mathcal{V} - \mathcal{V}_1$ contains one distinct value $\{*\}$. Consider an arbitrary relation $e \in \mathcal{E} - \mathcal{E}_1$. If $e \cap \mathcal{V}_1 = \emptyset$, relation R_e only contains one tuple in a form of $(*, *, \cdots, *)$. Otherwise, $e \cap \mathcal{V}_1 \subseteq \mathbf{y}$. In this case, $R_e = (\pi_{\mathbf{y}} R_e) \times (\pi_{\bar{\mathbf{y}}} R_e)$. Note that each tuple in R_e has its annotation as 1. The following argument in Theorem 1 applies here. ☐

References

[**1**] A. Abboud, K. Bringmann, N. Fischer, and M. Künnemann. The time complexity of fully sparse matrix multiplication. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4670–4703. SIAM, 2024.

[**2**] R. R. Amossen and R. Pagh. Faster join-projects and sparse matrix multiplications. In *Proceedings of the 12th International Conference on Database Theory*, pages 121–126. ACM, 2009.

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

[**4**] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.

[**5**] N. Bakibayev, T. Kocisky, D. Olteanu, and J. Zavodny. Aggregation and ordering in factorised databases. In *Proc. International Conference on Very Large Data Bases*, 2013.

[**6**] H. Barthels, M. Copik, and P. Bientinesi. The generalized matrix chain algorithm. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 138–148, 2018.

[**7**] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[**8**] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*, pages 223–234. Springer, 2014.

[**9**] M. Dalirrooyfard, S. Mathialagan, V. V. Williams, and Y. Xu. Listing cliques from smaller cliques. *arXiv preprint arXiv:2307.15871*, 2023.

[**10**] S. Deep, X. Hu, and P. Koutris. Fast join project query evaluation using matrix multiplication. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1213–1223, 2020.

[**11**] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[**12**] S. S. Godbole. On efficient computation of matrix chain products. *IEEE Transactions on Computers*, 100(9):864–866, 1973.

[**13**] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. ACM Symposium on Principles of Database Systems*, 2007.

[**14**] M. Grohe and D. Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):1–20, 2014.

[**15**] X. Hu. Cover or pack: New upper and lower bounds for massively parallel joins. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 181–198, 2021.

[**16**] X. Hu. Fast matrix multiplication for query processing. *Proceedings of the ACM on Management of Data*, 2(2):1–25, 2024.

[**17**] M. Idris, M. Ugarte, and S. Vansummeren. The dynamic Yannakakis algorithm: Compact and efficient query processing under updates. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2017.

[**18**] C. Jin, V. V. Williams, and R. Zhou. Listing 6-cycles. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 19–27. SIAM, 2024.

[**19**] M. R. Joglekar, R. Puttagunta, and C. Ré. AJAR: Aggregations and joins over annotated relations. In *Proc. ACM Symposium on Principles of Database Systems*, 2016.

[**20**] M. A. Khamis, R. R. Curtin, B. Moseley, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. Functional aggregate queries with additive inequalities. *ACM Transactions on Database Systems (TODS)*, 45(4):1–41, 2020.

[**21**] C. Lin, W. Luo, Y. Fang, C. Ma, X. Liu, and Y. Ma. On efficient large sparse matrix chain multiplication. *Proceedings of the ACM on Management of Data*, 2(3):1–27, 2024.

[**22**] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM (JACM)*, 60(6):1–51, 2013.

[**23**] H. Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 111–124. ACM, 2018.

[**24**] K. Nishida, Y. Ito, and K. Nakano. Accelerating the dynamic programming for the matrix chain product on the gpu. In *2011 Second International Conference on Networking and Computing*, pages 320–326. IEEE, 2011.

[**25**] R. Pagh and F. Silvestri. The input/output complexity of triangle enumeration. In *Proc. ACM Symposium on Principles of Database Systems*, 2014.

[**26**] D. Suciu, D. Olteanu, C. Ré, and C. Koch. Probabilistic databases, synthesis lectures on data management. *Morgan & Claypool*, 2011.

[**27**] T. L. Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. In *Proc. International*

Conference on Database Theory, 2014.
[**28**] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. International Conference on Very Large Data Bases*, pages 82–94, 1981.

A Missing Materials in Section 1

Algorithm 6: CLEANSE($\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y}), \mathcal{R}$)

1 Let $\mathcal{T}$ be a join tree of $\mathcal{Q}$;
2 **while** *visit nodes a bottom-up way (excluding the root)* **do**
3   **foreach** *node e visited* **do**
4     $e' \leftarrow$ the parent of e;
5     $R_{e'} \leftarrow R_{e'} \ltimes R_e$;
6 **while** *visit nodes a top-down way (excluding the root)* **do**
7   **foreach** *node e visited* **do**
8     $e' \leftarrow$ the parent of e;
9     $R_e \leftarrow R_e \ltimes R_{e'}$;
10 **while true do**
11   **if** $\exists B \in \bar{\mathbf{y}}$ *s.t.* $|\mathcal{E}_B| = 1$ **then**
12     $R_e \leftarrow \oplus_B R_e$;
13     $e \leftarrow e - \{B\}$;
14     $\mathcal{V} \leftarrow \mathcal{V} - \{B\}$;
15   **if** $\exists e, e' \in \mathcal{E}$ *s.t.* $e \subseteq e'$ **then**
16     $R_{e'} \leftarrow R_{e'} \bowtie R_e$;
17     $\mathcal{E} \leftarrow \mathcal{E} - \{e\}$;
18   **if** *$\mathcal{Q}$ does not change from last iteration* **then break**;
19 **return** updated $\mathcal{Q}, \mathcal{R}$;

Algorithm 7: YANNAKAKIS($\mathcal{Q} = (\mathcal{V}, \mathcal{E}), \mathbf{y}, \mathcal{R}$) [**28, 19**]

1 Let $\mathcal{T}$ be a join tree of $\mathcal{Q}$ rooted at r;
2 **while** *visit nodes a bottom-up way (excluding the root r)* **do**
3   **foreach** *node e visited* **do**
4     $e' \leftarrow$ the parent of e;
5     $R_{e'} \leftarrow R_{e'} \ltimes R_e$;
6 **while** *visit nodes a top-down way (excluding the root r)* **do**
7   **foreach** *node e visited* **do**
8     $e' \leftarrow$ the parent of e;
9     $R_e \leftarrow R_e \ltimes R_{e'}$;
10 **while** *visit nodes a bottom-up way (excluding the root r)* **do**
11   **foreach** *node e visited* **do**
12     $e' \leftarrow$ the parent of e;
13     $R_e \leftarrow \oplus_{\bar{\mathbf{y}} \cap (e - e')} R_e$;
14     $R_{e'} \leftarrow R_e \bowtie R_{e'}$;
15 **return** $\oplus_{\bar{\mathbf{y}} \cap r} R_r$ for the root r;

Lemma 14. *For an acyclic join-aggregate query $\mathcal{Q}$, $\mathsf{outw}(\mathcal{Q}) = 1$ if and only if $\mathcal{Q}$ is free-connex.*

Lemma 15 ([**4**], Lemma 21)**.** *For any free-connex join-aggregate $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, there exists a join tree $\mathcal{T}'$ for $(\mathcal{V}, \mathcal{E} \cup \mathcal{E}_{\mathbf{y}}, \mathbf{y})$ and a subset of relations $\mathcal{E}_{\mathsf{con}} \subseteq \mathcal{E} \cup \mathcal{E}_{\mathbf{y}}$ such that the corresponding nodes of $\mathcal{E}_{\mathsf{con}}$ form a connex subtree of $\mathcal{T}'$, i.e., $\mathcal{E}_{\mathsf{con}}$ includes the root of $\mathcal{T}'$, $\mathbf{y} = \bigcup_{e \in \mathcal{E}_{\mathsf{con}}} e$ and the subtree is connected.*

Proof of Lemma 14. We mention an equivalent definition of free-connex join-aggregate queries based on join tree, in Lemma 15.

If Direction. For a free-connex join-aggregate query $\mathcal{Q}$, if applying the cleanse procedure, we will iteratively remove all non-output attributes and be left with $\mathcal{E}_{\mathsf{con}}$. Let $\mathcal{Q}_{\mathsf{con}} = \left(\bigcup_{e \in \mathcal{E}_{\mathsf{con}}} e, \mathcal{E}_{\mathsf{con}}, \mathbf{y}\right)$. As $\mathcal{E}_{\mathsf{con}}$ only contains output attribute, in the existential-connectivity graph of $G^{\exists}_{\mathcal{Q}_{\mathsf{con}}}$, every single relation forms a connected subquery with out-width 1. Hence, $\mathcal{Q}$ has out-width as 1.

Only-If Direction. Consider an arbitrary acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ with $\mathsf{outw}(\mathcal{Q}) = 1$. Let $\mathcal{Q}' = (\mathcal{V}', \mathcal{E}', \mathbf{y})$ be the cleansed version of $\mathcal{Q}$. Note that $\mathsf{outw}(\mathcal{Q}') = 1$. Moreover, for each connected subquery $\mathcal{Q}'_i$ in $G^{\exists}_{\mathcal{Q}'}$, $\mathsf{outw}(\mathcal{Q}'_i) = 1$. We note that $\mathcal{Q}'_i$ is cleansed, is existentially connected, and has $\mathsf{outw}(\mathcal{Q}'_i) = 1$, $\mathcal{Q}'_i$ must be a single relation with all attributes as output attributes. Putting all sub-queries together, $\mathcal{Q}'$ must be an acyclic full join query. We next build a join tree for $\mathcal{Q}$ as follows.

- We build a join tree $\mathcal{T}'$ for $\mathcal{Q}'$.
- For each relation $e \in \mathcal{E}'$, if there is some non-output unique attribute from e, we create a relation e' such that e' contain all non-output unique attribute from e, as well as all remaining attributes in e, and add e' as a child of e.
- We revisit the cleanse procedure on $\mathcal{Q}$ in Algorithm 6. For each relation e removed in line 20 due to the fact that $e \subseteq e'$ for some $e' \in \mathcal{E}$, we add e (or the subtree rooted at e) as a child of e'.

Let $\mathcal{T}$ be the resulted tree. It can be easily checked that $\mathcal{T}$ is a valid join tree for $\mathcal{Q}$, satisfying all properties in Lemma 15. Hence, $\mathcal{Q}$ is free-connex, which completes the whole proof. ☐

Lemma 16. *For any acyclic query $\mathcal{Q}$, $\mathsf{freew}(\mathcal{Q}) \le \mathsf{outw}(\mathcal{Q})$.*

Proof. If $\mathcal{Q}$ is non-cleansed, then $\mathsf{freew}(\mathcal{Q}) = \mathsf{freew}(\mathcal{Q}')$ and $\mathsf{outw}(\mathcal{Q}) = \mathsf{outw}(\mathcal{Q}')$ for the cleansed version $\mathcal{Q}'$. Below, it suffices to assume that $\mathcal{Q}$ is cleansed. Let $\mathcal{E}' \subseteq \mathcal{E}$ be the set of relations that contain a unique output attribute, such that $|\mathcal{E}'| = \mathsf{freew}(\mathcal{Q})$. We note that any fractional edge covering of $\mathcal{Q}$ must assign weight 1 to every relation in $\mathcal{E}'$; otherwise, some unique output attribute is not covered. Hence, $\mathsf{outw}(\mathcal{Q}) \ge |\mathcal{E}'| = \mathsf{freew}(\mathcal{Q})$. ☐

B Yannakakis Framework Revisited

B.1 Aggregate-Hierarchical Query

Theorem 8. *For any aggregate-hierarchical query $\mathcal{Q}$ and an instance $\mathcal{R}$ of input size N and output size OUT, there is a query plan of Yannakakis framework that can compute the query result $\mathcal{Q}(\mathcal{R})$ within $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{\mathsf{outw}}} + \mathrm{OUT}\right)$ time.*

Given an aggregate-hierarchical query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ and an instance $\mathcal{R}$ of input size N and output size OUT, we can first apply the cleanse and decompose procedures. It suffices to compute the results for each connected sub-query first and then combine them over all sub-queries via join. These pre-processing and post-processing steps take $O(N + \mathrm{OUT})$ time. In the remaining, we assume that $\mathcal{Q}$ is cleansed, existentially connected and hierarchical.

We mention that such a query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ has an attribute tree $\mathcal{H}$, such that (i) there is a one-to-one correspondence between attributes in $\mathcal{V}$ and nodes in $\mathcal{H}$; (ii) each relation corresponds to a leaf-to-root path; (iii) every leaf node (that corresponds to a unique attribute) must be an output attribute. Note that $|\mathcal{E}| = \mathsf{outw}$. In $\mathcal{H}$, let $\mathcal{H}_x$ be the subtree of $\mathcal{H}$ rooted at attribute $x \in \mathcal{V}$. Let $\mathcal{Q}_x$ be the sub-query derived by relations that contain attribute x, i.e., $\mathcal{Q}_x = (\mathcal{V}_x, \mathcal{E}_x, \mathbf{y}_x)$, where $\mathcal{V}_x = \bigcup_{e \in \mathcal{E}_x} e$ and $\mathbf{y}_x = \mathbf{y} \cap \mathcal{V}_x$. Let $\mathsf{path}(x_1, x_2)$ denote the set of nodes lying on the path between x_1 and x_2 in $\mathcal{H}$. We build a join tree $\mathcal{T}$ for $\mathcal{Q}$ as follows. Consider the children of the root attribute in $\mathcal{H}$. Let $\{\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_j\}$ be a partition of $\mathcal{E}$ such that all relations in $\mathcal{E}_i$ shares one common child attribute. We build a join tree $\mathcal{T}_i$ for each group $\mathcal{E}_i$, and then add $\mathcal{T}_i$ for every $i < j$ as the last $(j-1)$ child nodes of the root of $\mathcal{T}_j$. See an example in Figure 5.

We apply the Yannakakis framework along such a join tree by traversing nodes in a post order. Below, we aim to bound the number of intermediate join results materialized by this specific query plan of Yannakakis framework. For simplicity, we define

$$f_{q,\mathbf{y},\mathcal{R}}(x) = \left|\pi_{\mathsf{path}(x,r)\cup(\mathbf{y}\cap\mathcal{H}_x)} q(\mathcal{R})\right|$$

for an attribute $x \in \mathcal{V}$. The number of intermediate join result that materialized is exactly $O\left(\max_{x\in\mathcal{H}} f_{q,\mathbf{y},\mathcal{R}}(x)\right)$. Let $\mathbb{I}(\mathcal{Q}, N, \text{OUT})$ denote the class of all input instances of input size N and output size OUT for $\mathcal{Q}$. We can further rewrite the size bound above as:

$$\max_{\mathcal{R}\in\mathbb{I}(\mathcal{Q},N,\text{OUT})} \max_{x\in\mathcal{H}} f_{q,\mathbf{y},\mathcal{R}}(x) \le \max_{x\in\mathcal{H}} \max_{\mathcal{R}\in\mathbb{I}(\mathcal{Q},N,\text{OUT})} f_{q,\mathbf{y},\mathcal{R}}(x)$$

Then, it suffices to prove for an arbitrary attribute $x \in \mathcal{H}$:

$$\max_{\mathcal{R}\in\mathbb{I}(\mathcal{Q},N,\text{OUT})} f_{q,\mathbf{y},\mathcal{R}}(x) \le N \cdot \text{OUT}^{1-\frac{1}{|\mathcal{E}_x|}}$$

As $|\mathcal{E}_x| \le \mathsf{outw}$, we come to the desired result. Let's take a closer look at the sub-query $f_{q,\mathbf{y},\mathcal{R}}(x)$ derived, which is captured by the class of *generalized star query* below. Implied by Lemma 17, we complete the whole proof.

Definition 5 (Generalized Star Query). *A cleansed, existentially-connected and hierarchical query* $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ *is a* generalized star *if* $\mathcal{V} - \mathbf{y} \subseteq \bigcap_{e\in\mathcal{E}} e$.

Lemma 17. *For a generalized star query* $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ *of* k *relations, given any parameter* $1 \le N$ *and* $\text{OUT} \le N^k$*, we have* $\max_{\mathcal{R}\in\mathbb{I}(\mathcal{Q},N,\text{OUT})} |\Join_{e\in\mathcal{E}} R_e| = O\left(N \cdot \text{OUT}^{1-\frac{1}{k}}\right)$.

Proof. We distinguish two cases. If $\bigcap_{e\in\mathcal{E}} e \subseteq \mathbf{y}$, $|\Join_{e'\in\mathcal{E}} R_{e'}| = \text{OUT}$ holds for an arbitrary instance $\mathcal{R} \in \mathbb{I}(\mathcal{Q}, N, \text{OUT})$. As $\text{OUT} \le \prod_{e\in\mathcal{E}} |R_e|$, this result automatically holds. In the following, we assume $\bigcap_{e\in\mathcal{E}} e - \mathbf{y} \neq \emptyset$. Let $\mathbf{z} = \bigcap_{e\in\mathcal{E}} e \cap \mathbf{y}$. Our proof consists of two steps:

Step 1. Consider a derived sub-query $\mathcal{Q}' = (\mathcal{V}', \mathcal{E}', \mathbf{y}')$ where $\mathcal{V}' = \mathcal{V} - \mathbf{z}$, $\mathcal{E}' = \{e \cap \mathcal{V}' : e \in \mathcal{E}\}$ and $\mathbf{y}' = \mathbf{y} - \mathbf{z}$. We will prove

$$\max_{\mathcal{R}\in\mathbb{I}(\mathcal{Q}',N,\text{OUT})} |\Join_{e\in\mathcal{E}} R_e| = O\left(N \cdot \text{OUT}^{1-\frac{1}{|\mathcal{E}'|}}\right)$$

with a similar argument made in [**2**].

Let $\bigcap_{e\in\mathcal{E}'} e = \{B\}$ be the unique non-output attribute appearing in all relations from $\mathcal{Q}'$. Suppose $\text{dom}(B) = \{b_1, b_2, \cdots, b_\ell\}$. We introduce a variable Δ_i for each value b_i to denote the number of input tuples that display b_i in attribute B. A closely related sub-problem is to produce as many as full join result. If $|\sigma_{B=b_i} R_e| \neq |\sigma_{B=b_i} R_{e'}|$ for any pair of relations $e \neq e'$, there always another solution where $|\sigma_{B=b_i} R_e| = |\sigma_{B=b_i} R_{e'}| = \frac{1}{2} \cdot (|\sigma_{B=b_i} R_e| + |\sigma_{B=b_i} R_{e'}|)$. It can be easily shown that the transformed solution satisfies the input size constraint while producing larger (or at least no smaller) number of full join result. Moreover, the largest number of full join result can be produced is $\left(\frac{\Delta_i}{|\mathcal{E}'|}\right)^{|\mathcal{E}'|}$. Hence, our overall optimization problem can be optimized as follows:

$$\begin{aligned}
\text{max.} \quad & \left(\frac{1}{|\mathcal{E}'|}\right)^{|\mathcal{E}'|} \cdot \sum_{i\in[\ell]} \Delta_i^{|\mathcal{E}'|} \\
\text{subject to.} \quad & \sum_{i\in[\ell]} \Delta_i \le N \\
& \Delta_i \le \text{OUT}^{\frac{1}{|\mathcal{E}'|}}, \forall i \in [\ell]
\end{aligned}$$

is a valid solution with larger number of full join result produced. The optimal solution $N \cdot \text{OUT}^{1-\frac{1}{|\mathcal{E}'|}}$ is achieved when $\Delta_i = \text{OUT}^{\frac{1}{|\mathcal{E}'|}}$ and $\ell = \frac{N}{\text{OUT}^{\frac{1}{|\mathcal{E}'|}}}$.

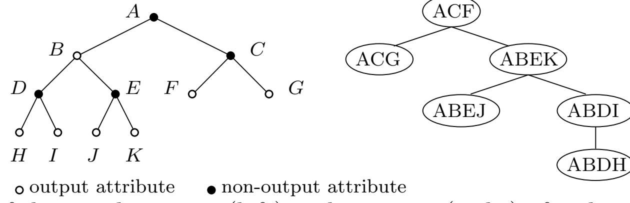

Figure 5: An illustration of the attribute tree (left) and join tree (right) of a cleansed, existentially connected and hierarchical join-aggregate query $\mathcal{Q} = \oplus_{A,C,D,E} R_1(A,B,D,H) \bowtie R_2(A,B,D,I) \bowtie R_3(A,B,E,J) \bowtie R_4(A,B,E,K) \bowtie R_5(A,C,F) \bowtie R_6(A,C,G)$.

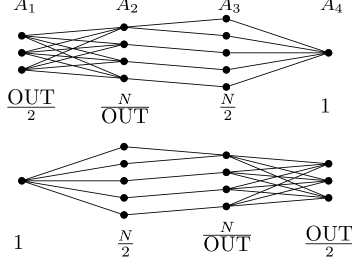

Figure 6: An illustration of hard instance for Yannakakis framework on line-3 query [**16**].

Step 2. Consider an arbitrary instance $\mathcal{R}$ of $\mathcal{Q}$. Let $z_1, z_2, \cdots, z_\ell$ be the values in the active domain of $\mathbf{z}$ in $\mathcal{R}$. Let $N_i = \sum_{j\in[\ell]} |\sigma_{\mathbf{z}=z_i} R_j|$ and $\mathrm{OUT}_i = \prod_{j\in[\ell]} |\sigma_{\mathbf{z}=z_i} R_j|$. Implied by **Step 1**, the largest number of full join result can be produced is at most $O\left(N_i \cdot \mathrm{OUT}_i^{1-\frac{1}{k}}\right)$. Given the following two constraints

$$\sum_{i\in[\ell]} N_i = N, \text{ and } \sum_{i\in[\ell]} \mathrm{OUT}_i = \mathrm{OUT},$$

the largest number of full join result produced can be bounded by

$$\sum_{i\in[\ell]} N_i \cdot \mathrm{OUT}_i^{1-\frac{1}{|\mathcal{E}|}} \le \sum_{i\in[\ell]} N_i \cdot \mathrm{OUT}^{1-\frac{1}{k}} \le N \cdot \mathrm{OUT}^{1-\frac{1}{|\mathcal{E}|}}.$$

As $|\mathcal{E}| = k$, this is exactly $O\left(N \cdot \mathrm{OUT}^{1-\frac{1}{k}}\right)$. □

B.2 Non-Aggregate-Hierarchical Query

As shown in [**16**], Yannakakis framework indeed incurs $\Theta(N \cdot \mathrm{OUT})$ time for line queries. We first revisit the hard instance constructed for line-3 query as shown in Figure 6. Consider an arbitrary acyclic but non-aggregate-hierarchical join-aggregate query $\mathcal{Q}$. Let $\mathcal{Q}'$ be its cleansed version. Let $\mathcal{Q}_1, \mathcal{Q}_2, \cdots, \mathcal{Q}_h$ be the connected sub-queries in $G^{\exists}_{\mathcal{Q}'}$. As $\mathcal{Q}$ is acyclic, every sub-query $\mathcal{Q}_i$ is also acyclic. As $\mathcal{Q}$ is non-aggregate-hierarchical, at least one sub-query is not hierarchical, say $\mathcal{Q}_1$. There must exist a pair of attributes x, x' and three relations e, e', e'' such that $x \in e \cap e' - e''$ and $x' \in e' \cap e'' - e$. Let $\mathcal{T}$ be a join tree of $\mathcal{Q}_1$. As $\mathcal{Q}_1$ is cleansed, every leaf node must contain some unique attribute.

It is always feasible to find a pair of leaf nodes e_1, e_n in $\mathcal{T}$ such that $e, e'' \in \mathsf{path}(e_1, e_n)$, $e \in \mathsf{path}(e_1, e'')$ and $e'' \in \mathsf{path}(e', e_n)$. It could be possible that $e = e_1$ or $e'' = e_n$. Let A_1, A_{n+1} be an arbitrary unique output attribute in e_1, e_n separately. As $\mathcal{Q}_1$ is existentially connected, there must exists a subset S of m relations in $\mathsf{path}(e_1, e_n)$ (including e_1 and e_n) and a subset of non-output attributes $B_1, B_2, \cdots, B_m$ such that $B_1 \in e_1$, $B_n \in e_n$, $B_i, B_{i+1} \in e_i$ for some relation $e_i \in \mathsf{path}(e_1, e_n)$, and there exists no relation e' such that $B_i, B_j \in e'$ for any $|j - i| > 1$. Otherwise, $\mathcal{Q}_1$ is not existentially connected.

Give an instance $\mathcal{R}_{\mathsf{line}}$ of line-3 query $\mathcal{Q}_{\mathsf{line}}$, we construct an instance $\mathcal{R}$ for $\mathcal{Q}$ as follows. We set $\mathrm{dom}(x) = \{*\}$ for every attribute $x \in \mathcal{V} - \{C_1, C_2, B_1, B_2, \cdots, B_m\}$. We use C_1 to simulate A_1, use C_2 to simulate

Algorithm 8: HybridYannakakis($\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y}), \mathcal{R}$)

Input : A well-separated query $\mathcal{Q}$ and an instance $\mathcal{R}$;
Output: Query result $\mathcal{Q}(\mathcal{R})$;

1 **if** $|\mathcal{E}| = 1$, *say* $\mathcal{E} = \{e\}$ **then return** $\oplus_{\bar{\mathbf{y}}} R_e$;
2 $\mathcal{E}_{\mathbf{y}} \leftarrow \{e \in \mathcal{E} : e \cap \mathbf{y} \neq \emptyset\}$;
3 $e \leftarrow$ an arbitrary relation in $\in \mathcal{E}_{\mathbf{y}}$;
4 Put an ordering on elements in $\pi_{e \cap \mathbf{y}} R_e$ as $a_1, a_2, a_3, \cdots$;
5 $S_e^{(0)} \leftarrow R_e$, $i \leftarrow 1$;
6 **while true do**
7     $S_e^{(i)} \leftarrow \emptyset$;
8   **foreach** $t \in S_e^{(i-1)}$ **do**
9   Suppose $\pi_{e \cap \mathbf{y}} t = a_j$;
10         $t' \leftarrow$ a tuple with $\pi_{e \cap \mathbf{y}} t' = a_{\lfloor \frac{j+1}{2} \rfloor}$ and $\pi_A t' = \pi_A t$ for any attribute $A \in e - \mathbf{y}$;
11         $S_e^{(i)} \leftarrow S_e^{(i)} \cup \{t'\}$;
12   **if** $\left|\pi_{e \cap \mathbf{y}} S_e^{(i)}\right| = 1$ **then break**;
13     $i \leftarrow i + 1$;
14 $R_{e'} \leftarrow R_{e'} \ltimes S_e^{(i)}$;
15 $\mathcal{Q}' \leftarrow (\mathcal{V} - e \cap \mathbf{y}, \mathcal{E} - \{e\}, \mathbf{y} - e)$;
16 $\mathcal{J}^{(i)} \leftarrow \left(\pi_{e \cap \mathbf{y}} S_e^{(i)}\right) \times$ HybridYannakakis $(\mathcal{Q}', \mathcal{R} - \{R_e\})$;
17 **while** $i > 0$ **do**
18     $\mathcal{J}^{(i-1)} \leftarrow$ HybridYannakakisWithOUT $\left(\mathcal{Q}, \mathcal{R} - \{R_e\} + \left\{S_e^{(i-1)}\right\}, 2 \cdot |\mathcal{J}^{(i)}|\right)$; ▶ Algorithm 4;
19     $i \leftarrow i - 1$;
20 **return** $\mathcal{J}^{(0)}$;

A_4, B_1 to simulate A_2 and all remaining attributes $B_2, B_3, \cdots, B_m$ to simulate A_3. For any relation e, such that $|e \cap \{B_2, B_3, \cdots, B_m\}| > 1$, the projection of R_e onto $\{B_2, B_3, \cdots, B_m\}$ should be a one-to-one mapping. The argument for line-3 query can be applied for $\mathcal{Q}$ similarly, i.e., any query plan of the Yannakakis framework requires $\Theta(N \cdot \mathrm{OUT})$ time.

C Missing Proofs in Section 4

Lemma 18. *There exists a subset $\mathcal{E}_{\mathbf{y}} \subseteq \mathcal{E}$ of relations such that $|\mathcal{E}_{\mathbf{y}}| = \mathsf{outw}$ and for each output attribute $A \in \mathbf{y}$, there exists some relation $e \in \mathcal{E}_{\mathbf{y}}$ with $A \in e$.*

Proof of Lemma 18. Consider the sub-query $\mathcal{Q}_{\mathbf{y}} = (\mathbf{y}, \{e \cap \mathbf{y} : e \in \mathcal{E}\})$ derived by output attributes. As $\mathcal{Q}$ is acyclic, $\mathcal{Q}_{\mathbf{y}}$ is also acyclic. As shown in [**15**], every acyclic query has an optimal fractional edge covering ρ^* that is also integral, i.e., $\rho^*(e) = 1$ or $\rho^*(e) = 0$ for any relation $e \in \mathcal{E}$. Let $\mathcal{E}_{\mathbf{y}} \subseteq \mathcal{E}$ be the set of relations for which $\rho^*(e) = 1$ holds for every $e \in \mathcal{E}_{\mathbf{y}}$. For every attribute $A \in \mathbf{y}$, there must exist a relation $e \in \mathcal{E}_{\mathbf{y}}$ such that $A \in e$. Implied by the definition of out-width, $|\mathcal{E}_{\mathbf{y}}| = \mathsf{outw}$. ☐

Lemma 19. *For an existentially-connected and cleansed acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ of out-width outw, there exists a subset S of outw attributes such that no pair of them appear in the same relation from $\mathcal{E}$.*

Proof of Lemma 19. Consider the sub-query $\mathcal{Q}_{\mathbf{y}} = (\mathbf{y}, \{e \cap \mathbf{y} : e \in \mathcal{E}\})$ derived by output attributes, which is also acyclic, and an arbitrary join tree $\mathcal{T}$. Initially, we set $S = \emptyset$. As shown in [**15**], the following greedy strategy leads to an optimal fractional edge covering that is also integral. It iteratively performs the following two procedures: (i) removes a relation e if there exists another relation e' such that $e \subseteq e'$; (ii) if there exists

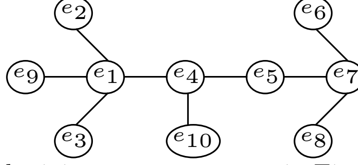

Figure 7: A join tree constructed for the join-aggregate query in Figure 3, where $\mathcal{E}_\bullet = \{e_2, e_9, e_3, e_{10}, e_6, e_8\}$.

a relation e containing some unique attribute, we remove relation e as well as all attributes in e, and add an arbitrary attribute in e to S. It can be easily checked that $|S| = \mathsf{outw}$, and no pair of them appears in the same relation from $\mathcal{E}$. □

Proof of Lemma 5. Below, we show the separate procedure with two steps. See Figure 3.

Step 1. Consider a cleansed and existentially connected acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$. Let $\mathbf{y}_\bullet \subseteq \mathbf{y}$ be the set of unique output attributes. Let $\mathbf{y} - \mathbf{y}_\bullet$ be the set of joint output attributes.

Suppose $\mathcal{E}_\mathbf{y} \subseteq \mathcal{E}$ is the subset of relations identified in Lemma 18. We assign each attribute $A \in \mathbf{y}$ to an arbitrary relation $e \in \mathcal{E}_\mathbf{y}$ with $A \in e$. Let $\pi : \mathbf{y} \to \mathcal{E}_\mathbf{y}$ be the assignment function.

We construct a query $\mathcal{Q}' = (\mathcal{V}', \mathcal{E}', \mathbf{y})$ based on $\mathcal{Q}$ as follows. First, $\mathcal{V}'$ includes all attributes in $\mathcal{V}$. For each output attribute $A \in \mathbf{y} - \mathbf{y}_\bullet$, we add a distinct attribute $x_A \notin \mathcal{V}$ to $\mathcal{V}'$. For each relation $e \in \mathcal{E}$, we add a relation e_π to $\mathcal{E}'$ as follows. If $e \in \mathcal{E}_\mathbf{y}$, e_π includes all attributes in e as well as the new attribute x_A for each joint output attribute $A \in e \cap (\mathbf{y} - \mathbf{y}_\bullet)$. Otherwise, e_π includes all attributes in e by replacing every joint output attribute $A \in e \cap (\mathbf{y} - \mathbf{y}_\bullet)$ with x_A if there exists any. More precisely,

$$e_\pi = \begin{cases} e + \left(\bigcup_{A \in e \cap (\mathbf{y} - \mathbf{y}_\bullet): \pi(A) = e} \{x_A\}\right) & \text{if } e \in \mathcal{E}_\mathbf{y} \\ e - \mathbf{y} + \{x_A : A \in e \cap (\mathbf{y} - \mathbf{y}_\bullet)\} & \text{otherwise} \end{cases}$$

It is not hard to see $\mathsf{outw}(\mathcal{Q}) = \mathsf{outw}(\mathcal{Q}')$. Moreover, each output attribute in $\mathbf{y}$ is unique in $\mathcal{Q}'$.

Given an arbitrary instance $\mathcal{R}$ for $\mathcal{Q}$, we construct an instance $\mathcal{R}'$ for $\mathcal{Q}'$ as follows. For each relation $e \in \mathcal{E}$, we add a relation R_{e_π} to $\mathcal{R}'$ by distinguishing the following three cases:

- **Case 1:** $e \cap \mathbf{y} = \emptyset$ or $e \cap (\mathbf{y} - \mathbf{y}_\bullet) = \emptyset$. In this case, $e = e_\pi$, and we just add R_e to $\mathcal{R}'$.
- **Case 2:** $e \cap (\mathbf{y} - \mathbf{y}_\bullet) \neq \emptyset$ and $e \in \mathcal{E}_\mathbf{y}$. For each tuple $t \in R_e$, we add a new tuple t' to R_{e_π} with the same annotation, such that $\pi_e t' = \pi_e t$, and $\pi_{x_A} t' = \pi_A t$ for each $A \in e \cap (\mathbf{y} - \mathbf{y}_\bullet)$.
- **Case 3:** $e \cap (\mathbf{y} - \mathbf{y}_\bullet) \neq \emptyset$ and $e \notin \mathcal{E}_\mathbf{y}$. For each tuple $t \in R_e$, we add a new tuple t' to R_{e_π} with the same annotation, such that $\pi_{e-\mathbf{y}} t' = \pi_{e-\mathbf{y}} t$, and $\pi_{x_A} t' = \pi_A t$ for each $A \in e \cap -\mathbf{y}$.

From our construction above, there is a one-to-one correspondence between $\mathcal{Q}(\mathcal{R})$ and $\mathcal{Q}'(\mathcal{R}')$.

Step 2. Now, suppose we are given a cleansed and existentially connected acyclic join-aggregate query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$, such that $|\{e \in \mathcal{E} : x \in e\}| = 1$ holds for each output attribute $x \in \mathbf{y}$. Let $\mathbf{y}_* = \{x \in \mathbf{y} : x \in e, \nexists e' \in \mathcal{E}, \text{ such that } e - \mathbf{y} \subseteq e'\}$. We next construct a well-separated query $\mathcal{Q}' = (\mathcal{V}', \mathcal{E}', \mathbf{y})$ as follows. First, $\mathcal{V}'$ include all attributes in $\mathcal{V}$. For each output attribute $A \in \mathbf{y}_*$, we add a distinct attribute $z_A \notin \mathcal{V}$ to $\mathcal{V}'$. For each relation $e \in \mathcal{E}$ with $e \cap \mathbf{y}^* = \emptyset$, we add $e' = e$ to $\mathcal{E}'$. For each relation $e \in \mathcal{E}$ with $e \cap \mathbf{y}^* \neq \emptyset$, we add $e' = e - \mathbf{y} + \{z_A : A \in e \cap \mathbf{y}_*\}$, and $e'' = (e \cap \mathbf{y}^*) + \{z_A : A \in e \cap \mathbf{y}_*\}$ to $\mathcal{E}'$. It is not hard to see that $\mathsf{outw}(\mathcal{Q}) = \mathsf{outw}(\mathcal{Q}')$. Moreover, each relation $e'' \in \mathcal{E}'$ with $e'' \cap \mathbf{y} \neq \emptyset$ is now an ear in $\mathcal{Q}'$.

Given an arbitrary instance $\mathcal{R}$ for $\mathcal{Q}$, we construct an instance $\mathcal{R}'$ for $\mathcal{Q}'$ as follows. Consider an arbitrary relation $e \in \mathcal{E}$.

- **Case 1:** $e \cap \mathbf{y}^* = \emptyset$. In this case, we just add R_e to $\mathcal{R}'$.
- **Case 2:** $e \cap \mathbf{y}^* \neq \emptyset$. For each tuple $t \in R_e$, we add a new tuple t' to $R_{e'}$ with the same annotation, such that $\pi_{e-\mathbf{y}} t = \pi_{e-\mathbf{y}} t'$ and $\pi_A t = \pi_{z_A} t'$ for each attribute $A \in e \cap \mathbf{y}^*$. For each tuple $t \in \pi_{e \cap \mathbf{y}^*} R_e$, we add a new tuple t'' to $R_{e'}$ with annotation as 1, such that $\pi_{e \cap \mathbf{y}^*} t = \pi_{e \cap \mathbf{y}^*} t''$ and $\pi_A t = \pi_{z_A} t''$ for each attribute $A \in e \cap \mathbf{y}^*$.

From our construction above, there is a one-to-one correspondence between $\mathcal{Q}(\mathcal{R})$ and $\mathcal{Q}'(\mathcal{R}')$. $\square$

Proof of Lemma 6. By definition, $|\mathcal{E}_\bullet| = \mathsf{outw}$. Let $\mathcal{T}$ be a join tree of $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, \mathbf{y})$ built as follows: (i) we first build a join tree $\mathcal{T}'$ for relations in $\mathcal{E} - \mathcal{E}_\bullet$; (ii) for each relation $e \in \mathcal{E}_\bullet$, we pick an arbitrary relation $e' \in \mathcal{E} - \mathcal{E}_\bullet$ such that $e - \mathbf{y}_\bullet \subseteq e'$ and add e as a child of e'. It can be easily checked that $\mathcal{T}$ is a valid join tree for $\mathcal{Q}$. See Figure 7. We next show that $\mathcal{E}_\bullet$ is exactly the set of leaf nodes of $\mathcal{T}$. By the construction above, each relation $e_1 \in \mathcal{E}_\bullet$ is added as a leaf of $\mathcal{T}$. On the other hand, consider a leaf node e_2 in $\mathcal{T}$ but $e_2 \in \mathcal{E} - \mathcal{E}_\bullet$. Let e_3 be the unique node incident to e_2. Implied by the property of join tree, there must be $e_3 - \mathbf{y}_\bullet \subseteq e_2$. If $e_3 \cap \bullet = \emptyset$, $e_3 \subseteq e_2$, which contradicts the fact that $\mathcal{Q}$ is cleansed. Otherwise, $e_3 \in \mathcal{E}_\bullet$, which contradicts the assumption $e_3 \notin \mathcal{E}_\bullet$. Together, $\mathcal{E}_\bullet$ is exactly the set of leaf nodes of $\mathcal{T}$. $\square$

Proof of Lemma 9. Consider an arbitrary tuple $t \in \mathrm{dom}(e_1 \cap e_2)$. As (e_1, e_2) is large, t can be joined with at least $\mathrm{OUT}^{\phi_{e_1,e_2}}$ query result of $\mathcal{Q}_{e_1,e_2}$. After removing dangling tuples, every tuple $t' \in \pi_{\mathbf{y}} \mathcal{Q}_{e_2,e_1}$ can appear together with at least $\mathrm{OUT}^{\phi_{e_1,e_2}}$ tuples from $\pi_{\mathbf{y}-e_1} \mathcal{Q}_{e_1,e_2}$ in the final query result. This way, $|\pi_{\mathbf{y}} \mathcal{Q}_{e_2,e_1}| \leq \mathrm{OUT}^{1-\phi_{e_1,e_2}} = \mathrm{OUT}^{\phi_{e_2,e_1}}$. Hence, (e_2, e_1) is limited. $\square$

Proof of Lemma 10. For $\mathcal{Q}_{e_1,e_2}$, we observe the following:

$$|\pi_{\mathbf{y}} \mathcal{Q}_{e_1,e_2}| \leq \prod_{e_3 \in \mathcal{N}_{e_1} - \{e_2\}} |\pi_{\mathbf{y}} \mathcal{Q}_{e_3,e_1}| \leq \prod_{e_3 \in \mathcal{N}_{e_1} - \{e_2\}} \mathrm{OUT}^{\phi_{e_3,e_1}} \leq \mathrm{OUT}^{\phi_{e_1,e_2}}$$

where the first inequality follows that $\bigcup_{e \in \mathcal{T}_{e_1,e_2}} (e \cap \mathbf{y}) = \bigcup_{e_3 \in \mathcal{N}_{e_1} - \{e_2\}} \bigcup_{e \in \mathcal{T}_{e_1,e_2}} (e \cap \mathbf{y})$ and the second inequality follows that $\sum_{e_3 \in \mathcal{N}_{e_1} - \{e_2\}} \phi(e_3, e_1) = \phi(e_1, e_2)$. By definition, (e_1, e_2) must be limited. $\square$

Proof of Lemma 12. Let $\mathcal{E}'$ be the subset of nodes in $\mathcal{T}_1$ that are incident to some node in $\mathcal{T}_2$. For each node $e \in \mathcal{E}'$, let p_e be the unique node from $\mathcal{T}_2$ incident to e. By contradiction, we assume that edge (p_e, e) is large for every $e \in \mathcal{E}'$. Implied by Lemma 9, edge (e, p_e) is limited for every $e \in \mathcal{E}'$. Consider an arbitrary $e' \in \mathcal{E}'$. We observe that for every tuple $t \in \mathrm{dom}(e' \cap p_{e'})$, it can be joined with at most

$$\pi_{e \in \mathcal{E}' - \{e'\}} \mathrm{OUT}^{\phi_{e,p_e}} \leq \mathrm{OUT}^{\sum_{e \in \mathcal{E}' - \{e'\}} \phi_{e,p_e}} = \mathrm{OUT}^{\phi_{p_{e'},e'}}$$

query results in $\mathcal{Q}_{p_{e'},e'}$, contradicting the fact that edge $(p_{e'}, e')$ is large. $\square$