Two-Attribute Skew Free, Isolated CP Theorem, and Massively Parallel Joins

Miao Qiao
University of Auckland
Auckland, New Zealand
miao.qiao@auckland.ac.nz

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

ABSTRACT

This paper presents an algorithm to process a multi-way join with load $\tilde{O}(n/p^{2/(\alpha\phi)})$ under the MPC model, where n is the number of tuples in the input relations, α the maximum arity of those relations, p the number of machines, and ϕ a newly introduced parameter called the *generalized vertex packing number*. The algorithm owes to two new findings. The first is a *two-attribute skew free* technique to partition the join result for parallel computation. The second is an *isolated cartesian product theorem*, which provides fresh graph-theoretic insights on joins with $\alpha \geq 3$ and generalizes an existing theorem on $\alpha = 2$.

CCS CONCEPTS

• **Information systems → Join algorithms**; • **Theory of computation → Massively parallel algorithms**.

KEYWORDS

Joins; Conjunctive Queries; MPC Algorithms; Parallel Computing

ACM Reference Format:
Miao Qiao and Yufei Tao. 2021. Two-Attribute Skew Free, Isolated CP Theorem, and Massively Parallel Joins. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3452021.3458321

1 INTRODUCTION

Massively-parallel computation systems such as Hadoop [7] and Spark [1] are designed to leverage hundreds or even thousands of machines to accomplish a computation task on data of a gigantic volume. Performance bottleneck these systems is communication rather than CPU computation. Understanding the communication complexities of fundamental database problems has become an active research area [2, 3, 5, 6, 10–12, 14, 15, 20].

1.1 Problem Definition

This paper studies parallel algorithms for processing natural joins.

Joins. Denote by **att** a countably infinite set where each element is an *attribute*. We will assume a total order on **att**, and use $A \prec B$ to represent the fact "attribute A ranking before attribute B". Denote by **dom** an infinite set where each element is a *value*.

A *tuple* over a set $\mathcal{U} \subseteq$ **att** is a function $\boldsymbol{u} : \mathcal{U} \to$ **dom**. Alternatively, we may represent a tuple as $(a_1, a_2, ..., a_{|\mathcal{U}|})$ where a_i is the output of $\boldsymbol{u}$ on the i-th ($1 \leq i \leq |\mathcal{U}|$) smallest attribute in $\mathcal{U}$ (according to $\prec$). Given a non-empty $\mathcal{V} \subseteq \mathcal{U}$, define $\boldsymbol{u}[\mathcal{V}]$ as the tuple $\boldsymbol{v}$ over $\mathcal{V}$ such that $\boldsymbol{v}(X) = \boldsymbol{u}(X)$ for every $X \in \mathcal{V}$; we say that $\boldsymbol{v}$ is a *projection* of $\boldsymbol{u}$.

A *relation* is a set $\mathcal{R}$ of tuples over the same set $\mathcal{U}$ of attributes. $\mathcal{U}$ is the *scheme* of $\mathcal{R}$, denoted as $scheme(\mathcal{R}) = \mathcal{U}$. Define $arity(\mathcal{R}) = |scheme(\mathcal{R})|$, referred to as the *arity* of $\mathcal{R}$. $\mathcal{R}$ is *unary* if $arity(\mathcal{R}) = 1$, or *binary* if $arity(\mathcal{R}) = 2$.

A *join query* is defined as a set $\mathcal{Q}$ of relations. The query result $Join(\mathcal{Q})$ is the relation:

$$\left\{\text{tuple } \boldsymbol{u} \text{ over } attset(\mathcal{Q}) \;\middle|\; \forall \mathcal{R} \in \mathcal{Q},\ \boldsymbol{u}[scheme(\mathcal{R})] \in \mathcal{R}\right\}$$

where $attset(\mathcal{Q}) = \bigcup_{\mathcal{R} \in \mathcal{Q}} scheme(\mathcal{R})$. If $\mathcal{Q}$ includes only two relations $\mathcal{R}$ and $\mathcal{S}$, we may also represent $Join(\mathcal{Q})$ as $\mathcal{R} \bowtie \mathcal{S}$. Define

$$\begin{aligned} n &= \sum_{\mathcal{R} \in \mathcal{Q}} |\mathcal{R}| \\ k &= |attset(\mathcal{Q})| \qquad (1) \\ \alpha &= \max_{\mathcal{R} \in \mathcal{Q}} arity(\mathcal{R}). \qquad (2) \end{aligned}$$

In particular, n is the *input size* of $\mathcal{Q}$. We will treat $|\mathcal{Q}|$ and k (hence, α) as constants, and assume $\alpha \geq 2$ (a query with $\alpha = 1$ has been optimally solved; see Section 3).

Computation model. We will work under the *massively parallel computation* (MPC) model, which has become a standard model for studying parallel join algorithms nowadays [6, 8, 11, 12, 14, 20]. At the beginning, the input relations of $\mathcal{Q}$ are distributed onto p machines, each of which stores $O(n/p)$ tuples. An algorithm can perform only a constant number of *rounds*, each with two phases:

- **Phase 1:** Each machine performs local computation, and prepares the messages to be sent to other machines.
- **Phase 2:** The machines exchange messages (which must be prepared in Phase 1).

Each tuple in $Join(\mathcal{Q})$ must reside on at least one machine when the algorithm terminates. The *load* of a round is the maximum number of words received by a machine in that round. The *load* of the algorithm is the maximum load of all rounds. The main challenge in algorithm design is to minimize the load.

load	source	remark
$\tilde{O}(n/p^{\frac{1}{\lvert\mathcal{Q}\rvert}})$	[3]	the hyper-cube (HC) algorithm
$\tilde{O}(n/p^{\frac{1}{k}})$	[6]	the BinHC algorithm
$\tilde{O}(n/p^{\frac{1}{\psi}})$	[14]	the KBS algorithm; ψ = the edge quasi-packing number (Appendix H)
$\tilde{O}(n/p^{\frac{1}{\rho}})$	[12, 20]	ρ = the fractional edge-covering number (Section 3.1); the algorithm is applicable only to $\alpha = 2$
$\tilde{O}(n/p^{\frac{1}{\rho}})$	[8]	for acyclic queries
$\tilde{O}(n/p^{\frac{2}{\alpha\phi}})$	ours	ϕ = the generalized vertex-packing number (Section 4); the algorithm subsumes [12, 20] when $\alpha = 2$
$\tilde{O}(n/p^{\frac{2}{\alpha\phi-\alpha+2}})$	ours	for α-uniform queries
$\tilde{O}(n/p^{\frac{2}{k-\alpha+2}})$	ours	for symmetric queries

Table 1: Comparison of all the known generic algorithms

We assume $p \leq \sqrt{n}$. Unless otherwise stated, by "an algorithm having load at most L", we mean that its load is at most L with probability at least $1 - 1/p^c$ where $c > 0$ can be set to an arbitrarily large constant. Each value in **dom** is assumed to fit in a word. Notation $\tilde{O}(.)$ hides a factor polylogarithmic to p. Given an integer $x \geq 1$, $[x]$ denotes the set $\{1, ..., x\}$.

1.2 Previous Work

Afrati and Ullman [3] developed the *hyper-cube* (HC) algorithm that answers a query $\mathcal{Q}$ with load $O(n/p^{1/\lvert\mathcal{Q}\rvert})$ deterministically. Extending HC with random binning ideas, Beame, Koutris, and Suciu [5] obtained the *BinHC* algorithm with load $\tilde{O}(n/p^{1/k})$.

Improving their earlier work [6], Koutris, Beame, and Suciu [14] gave an algorithm — called *KBS* henceforth — with load $\tilde{O}(n/p^{1/\psi})$, where ψ is the *edge quasi-packing number* of $\mathcal{Q}$ (Appendix H).

There has been work dedicated to queries involving only relations with arity at most 2, due to their special importance in *subgraph enumeration*[1]. Ketsman and Suciu [12] were the first to solve such queries $\mathcal{Q}$ with load $\tilde{O}(n/p^{1/\rho})$, where ρ is the *fractional edge covering number* of $\mathcal{Q}$ (Section 3.1). A simpler algorithm with the same load was presented by Tao [20].

Recently, Hu [8] presented an algorithm for answering any acyclic query[2] with load $\tilde{O}(n/p^{1/\rho})$.

The above algorithms are *generic* because they support either arbitrary queries [3, 14] or queries in a broad class [8, 12, 20]. There are algorithms designed for specific joins, e.g., star joins [3], cycle joins [14], clique joins [14], line joins [3, 14], Loomis-Whitney joins [14], etc.

We refer the reader to [2, 10, 11] for algorithms that can achieve a load sensitive to the output size $\lvert Join(\mathcal{Q})\rvert$.

On the lower bound side, Atserias, Grohe, and Marx [4] showed that $\lvert Join(\mathcal{Q})\rvert$ can reach $\Omega(n^\rho)$ but is always bounded by $O(n^\rho)$. This implies [14] that any algorithm must incur a load of $\Omega(n/p^{1/\rho})$ in the worst case. Furthermore, Hu [8] proved that $\Omega(n/p^{1/\tau})$ is another worst-case lower bound, where τ is the query's *factional edge packing number* (Section 3.1); she also described a class of queries whose τ values are strictly larger than ρ. It is clear from the above discussion that the upper and lower bounds have matched for (i) queries with $\alpha = 2$ and (ii) acyclic queries. Matching the lower bounds for an arbitrary query is still open.

The lower bounds mentioned earlier apply to algorithms that perform an arbitrary (constant) number of rounds. In [14], Koutris, Beame, and Suciu showed that $\Omega(n/p^{1/\psi})$ is a lower bound for *single*-round algorithms, subject to certain constraints.

Regarding other computational models, the (natural join) problem has been optimally settled in RAM [16, 17, 21], while external-memory (EM; a.k.a. I/O-efficient) algorithms have been developed for specific joins (see [9, 10, 18] and the references therein). There exists a reduction [14] for converting an MPC algorithm to work in the EM model. The reduction also applies to the algorithms developed in this paper.

1.3 Our Results

We will describe an algorithm (Theorem 8.2) that answers an arbitrary query $\mathcal{Q}$ with load

$$\tilde{O}\left(n/p^{\frac{2}{\alpha\phi}}\right) \tag{3}$$

where $\alpha \geq 2$ is the maximum arity and ϕ is the *generalized vertex packing number* of $\mathcal{Q}$ that will be formally introduced in Section 4. For $\alpha = 2$, ϕ can be proven equal to ρ; thus, our load matches the lower bound $\Omega(n/p^{1/\rho})$. For $\alpha \geq 3$, the algorithm improves the previous work on certain (non-trivial) query classes.

One notable class is the k*-choose-*α join $\mathcal{Q}$, which has $\binom{k}{\alpha}$ relations (recall that k is the number of attributes), each having a scheme that is a unique combination of α attributes. Currently, the best solution to a k-choose-α join with $\alpha \in [3, k-2]$[3] is the KBS algorithm [14], which (as mentioned) requires a load of $\tilde{O}(n/p^{1/\psi})$, with the edge quasi-packing number ψ at least $k-\alpha+1$ (Appendix H). On the other hand, $\mathcal{Q}$ has a generalized vertex packing number $\phi = k/\alpha$ (Section 4). The load of our algorithm is $\tilde{O}(n/p^{2/k})$, and is already better than $\tilde{O}(n/p^{1/(k-\alpha+1)})$ for $\alpha < k/2 + 1$.

[1]Namely, find all the occurrences of a subgraph pattern in a graph.
[2]Specifically, alpha-acyclic queries, which generalize berge-acylic and r-hierarchical queries.

[3]With $\alpha = k - 1$, $\mathcal{Q}$ is the Loomis-Whitney join and has been solved optimally. The case $\alpha \leq 2$ is left out because, in general, all queries with $\alpha \leq 2$ have been settled. The case $\alpha = k$ is not interesting either because $\mathcal{Q}$ has only one relation.

We can in fact prove a stronger result. A more general class is the α-*uniform* join, where all the relations in a query $\mathcal{Q}$ have arity α. For such $\mathcal{Q}$, the load of our algorithm can be further bounded as (Theorem 9.1)

$$\tilde{O}\left(n/p^{\frac{2}{\alpha\phi-\alpha+2}}\right). \tag{4}$$

Hence, our load for a k-choose-α join is actually $\tilde{O}(n/p^{2/(k-\alpha+2)})$, which strictly improves the KBS algorithm as long as $\alpha < k$.

Yet another notable class is the *symmetric* join, each being an α-uniform join $\mathcal{Q}$ with an additional constraint that each attribute in *attset*($\mathcal{Q}$) belongs to the same number of relations. The k-choose-α join is a proper subset of the symmetric join. To see this, consider the cycle join [14] where $\alpha = 2$ and $\mathcal{Q}$ contains k relations with schemes $\{A_1, A_2\}, \{A_2, A_3\}, ..., \{A_{k-1}, A_k\}, \{A_k, A_1\}$, respectively. A cycle join is symmetric but not a k-choose-2 join if $k > 3$.

The value ϕ for a symmetric join is always k/α (Section 4). By (4), our algorithm answers a symmetric query with load $\tilde{O}(n/p^{2/(k-\alpha+2)})$. This has an interesting implication. In general, $\rho \geq k/2$ holds on all queries with $\alpha \leq 2$. Hence, a query on binary relations must incur a load of $\Omega(n/p^{2/k})$ (see the lower bound discussion in Section 1.2). As $\tilde{O}(n/p^{2/(k-\alpha+2)})$ is $o(n/p^{2/k})$ for $\alpha \geq 3$, *every* symmetric query with $\alpha \geq 3$ is inherently easier than *every* query with $\alpha \leq 2$ (and the same value of k). No existing algorithms can achieve such a separation.

Table 1 presents a summary of our results in comparison with the existing ones.

A lower bound remark. As mentioned, our algorithm is optimal for $\alpha = 2$. For $\alpha \geq 3$, the load of our algorithm in (3) cannot be significantly improved in the following sense: no algorithm can achieve a load of $o(n/p^{2/(\alpha\phi)})$ in general. To explain, consider a class of queries $\mathcal{Q}$ constructed as follows. Let $A_1, ..., A_{k/2}$ and $B_1, ..., B_{k/2}$ be $k \geq 6$ distinct attributes. $\mathcal{Q}$ has $2 + k/2$ relations: (i) one with scheme $\{A_1, ..., A_{k/2}\}$ and another with scheme $\{B_1, ..., B_{k/2}\}$, and (ii) a relation with scheme $\{A_i, B_i\}$ for each $i \in [k/2]$. For every such $\mathcal{Q}$, $\alpha = k/2$ and $\phi = 2$. As shown in [8], every algorithm requires a load of $\Omega(n/p^{2/k})$ processing $\mathcal{Q}$. Notice that $\Omega(n/p^{2/k}) = \Omega(n/p^{2/(\alpha\phi)})$, i.e., $\Omega(n/p^{2/(\alpha\phi)})$ is also a lower bound on the load. Our algorithm is thus optimal on this class of queries.

2 OVERVIEW OF OUR TECHNIQUES

We will first review two standard techniques and then discuss the new techniques developed in this paper.

Standard 1: Skew free. Consider an arbitrary relation $\mathcal{R} \in \mathcal{Q}$ and any non-empty $\mathcal{V} \subseteq$ *scheme*($\mathcal{R}$). Given a tuple $\boldsymbol{v}$ over $\mathcal{V}$, define $f_{\mathcal{V}}(\boldsymbol{v}, \mathcal{R})$ as the number of tuples $\boldsymbol{u} \in \mathcal{R}$ satisfying $\boldsymbol{v} = \boldsymbol{u}[\mathcal{V}]$; we will refer to $f_{\mathcal{V}}(\boldsymbol{v}, \mathcal{R})$ as the $\mathcal{V}$*-frequency* of $\boldsymbol{v}$ in $\mathcal{R}$.

Suppose that we assign a *share* of $p_A \geq 1$ to each attribute $A \in$ *attset*($\mathcal{Q}$) subject to

$$\prod_{A \in attset(\mathcal{Q})} p_A \quad \leq \quad p. \tag{5}$$

Relation $\mathcal{R} \in \mathcal{Q}$ is *skew free* if

$$f_{\mathcal{V}}(\boldsymbol{v}, \mathcal{R}) \quad \leq \quad \frac{n}{\prod_{A \in \mathcal{V}} p_A} \tag{6}$$

holds for every non-empty $\mathcal{V} \subseteq$ *scheme*($\mathcal{R}$) and any tuple $\boldsymbol{v}$ over $\mathcal{V}$. $\mathcal{Q}$ is *skew free* if all relations in $\mathcal{Q}$ are skew free.

Standard 2: BinHC and the heavy-light technique. Suppose that the share p_A of every attribute $A \in$ *attset*($\mathcal{Q}$) has been decided. Beame, Koutris, and Suciu [6] proved that the BinHC algorithm (Section 1.2) solves a skew free query with load

$$\tilde{O}\left(\max_{\mathcal{R} \in \mathcal{Q}} \frac{n}{\prod_{A \in scheme(\mathcal{R})} p_A}\right). \tag{7}$$

When $\mathcal{Q}$ is not skew free, a common approach [12, 14, 20] is to resort to the following algorithmic paradigm. First, design a number of sub-queries. Then, choose the (attribute) shares appropriately to make every sub-query skew free and, thus, solvable by BinHC. What differentiates different algorithms is how to determine the sub-queries such that they

- **(Objective 1)** together produce precisely *Join*($\mathcal{Q}$), and
- **(Objective 2)** incur a small load under BinHC.

Fulfilling these objectives is non-trivial. In [14], Koutris, Beame, and Suciu presented a method that we refer to as the *heavy-light technique*, as outlined below. Let $\lambda > 0$ be some real value. Call a value $x \in$ **dom**

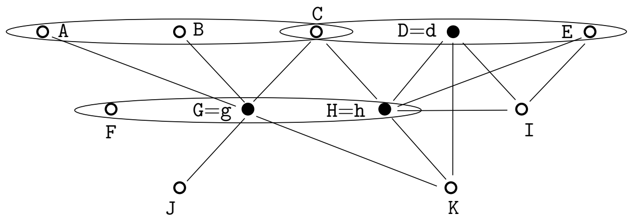
- *heavy*, if there exist a relation $\mathcal{R} \in \mathcal{Q}$ and an attribute $A \in$ *scheme*($\mathcal{R}$) such that $\mathcal{R}$ has at least n/λ tuples $\boldsymbol{u}$ with $\boldsymbol{u}(A) = x$;
- *light*, otherwise.

For every possible $\mathcal{U} \subseteq$ *attset*($\mathcal{Q}$), create a sub-query $\mathcal{Q}_{\mathcal{U}}$ to produce all the tuples $\boldsymbol{u} \in$ *Join*($\mathcal{Q}$) such that $\boldsymbol{u}(A)$ is heavy if $A \in \mathcal{U}$ or light otherwise. Clearly, there are 2^k sub-queries (see (1) for k); and the union of their results is *Join*($\mathcal{Q}$). This achieves Objective 1.

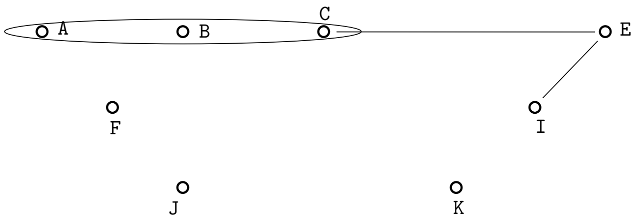
How about Objective 2? Let us fix a $\mathcal{U} \subseteq$ *attset*($\mathcal{Q}$) and concentrate on an arbitrary relation $\mathcal{R} \in \mathcal{Q}$. As far as $\mathcal{Q}_{\mathcal{U}}$ is concerned, we only need to consider those tuples $\boldsymbol{u} \in \mathcal{R}$ such that, for each $A \in$ *scheme*($\mathcal{R}$), $\boldsymbol{u}(A)$ is heavy if $A \in \mathcal{U}$ and light otherwise; denote by $\mathcal{R}'$ the set of such tuples. To apply BinHC on $\mathcal{Q}_{\mathcal{U}}$, we must set the shares to make $\mathcal{R}'$ skew free. This, however, can be difficult because we do not have control over the $\mathcal{V}$-frequency of a tuple for any subset $\mathcal{V} \subseteq$ *scheme*($\mathcal{R}'$) with $|\mathcal{V}| \geq 2$.

Koutris, Beame, and Suciu [14] circumvented the issue by setting $\lambda = p$ and fixing the share p_A to 1 for each attribute $A \in \mathcal{U}$. Interestingly, $\mathcal{R}'$ is guaranteed skew free *regardless of* the shares p_A of $A \notin \mathcal{U}$. This follows from two observations: (i) trivially, every heavy value can appear in at most $|\mathcal{R}'| \leq n$ tuples, and (ii) every light value can belong to $\tilde{O}(n/p) = \tilde{O}(n/\prod_{A \notin \mathcal{U}} p_A)$ tuples, noticing that $\prod_{A \notin \mathcal{U}} p_A \leq p$. The KBS algorithm achieves load $\tilde{O}(n/p^{1/\psi})$ by optimizing the shares p_A of $A \notin \mathcal{U}$.

The above approach fails to work for $\lambda \ll p$, whereas we often need $\lambda = p^c$ for some $c < 1$ to improve upon $\tilde{O}(n/p^{1/\psi})$. To deal with the issue, Ketsman and Suciu [12] and Tao [20] refined the heavy-light technique, but their refinement heavily relies on the premise $\alpha \leq 2$.



(a) The original query ($\rho = \phi = 5$ and $\psi = 9$)



(b) A residual query for the plan $(\{\mathsf{D}\}, \{(\mathsf{G}, \mathsf{H})\})$

Figure 1: Illustration of the proposed techniques

New 1: Two-attribute skew free. For a relation $\mathcal{R} \in \mathcal{Q}$, the skew free condition demands that $\mathcal{V}$-frequencies be low for all non-empty $\mathcal{V} \subseteq scheme(\mathcal{R})$. The is a stringent requirement and limits the applicability of BinHC.

Our first idea is to relax the requirement. We say that $\mathcal{R}$ is *two-attribute skew free* as long as (6) holds for every non-empty $\mathcal{V} \subseteq scheme(\mathcal{R})$ with $|\mathcal{V}| \leq 2$. $\mathcal{Q}$ is *two-attribute skew free* if all its relations are two-attribute skew free. As will be proved later (Lemma 3.5), a two-attribute skew free query $\mathcal{Q}$ can be answered by BinHC with load

$$\tilde{O}\Big(\max_{\mathcal{R}\in\mathcal{Q}}\Big(\min_{\substack{\mathcal{V}\subseteq scheme(\mathcal{R})\\ |\mathcal{V}|\leq 2}} \frac{n}{\prod_{A\in\mathcal{V}} p_A}\Big)\Big). \tag{8}$$

Relaxation of the skew free constraint is a matter of tradeoff. On the one hand, the load in (8) is higher than (7); but on the other hand, we gain greater flexibility in assigning shares. Fortunately, we can compensate for the loss in (8) with an enhanced heavy-light technique that is made possible by the new skew-free definition, as outlined below.

New 2: Two-attribute heavy-light. Given a $\lambda > 0$, define a value $x \in \mathbf{dom}$ to be light/heavy in the same way as before. In addition, we say that a value pair $(y, z) \in \mathbf{dom} \times \mathbf{dom}$ is:

- *heavy*, if there exist a relation $\mathcal{R} \in \mathcal{Q}$ and two distinct attributes $Y, Z \in scheme(\mathcal{R})$ such that the $\{Y, Z\}$-frequency of tuple (y, z) in $\mathcal{R}$ is at least n/λ^2;
- *light*, otherwise.

Define a *plan* as:

$$\boldsymbol{P} \quad = \quad \Big(\{X_1, ..., X_a\}, \{(Y_1, Z_1), ..., (Y_b, Z_b)\}\Big) \tag{9}$$

where $a \geq 0$, $b \geq 0$, $X_1, ..., X_a, Y_1, ..., Y_b, Z_1, ..., Z_b$ are distinct attributes in $attset(\mathcal{Q})$, and $Y_j \prec Z_j$ for each $j \in [b]$. Since $|attset(\mathcal{Q})|$ is a constant, only $O(1)$ plans exist.

Let us concentrate on a plan $\boldsymbol{P}$. Define $\mathcal{H} = \{X_1, ..., X_a, Y_1, ..., Y_b, Z_1, ..., Z_b\}$. We issue sub-queries to extract all the tuples $\boldsymbol{u} \in Join(\mathcal{Q})$ satisfying:

- $\boldsymbol{u}(X_i)$ is heavy for all $i \in [a]$;
- $\boldsymbol{u}(A)$ is light for any attribute $A \notin \{X_1, ..., X_a\}$;
- $(\boldsymbol{u}(Y_i), \boldsymbol{u}(Z_i))$ is heavy for all $i \in [b]$.
- $(\boldsymbol{u}(A), \boldsymbol{u}(B))$ is light for any distinct attributes $A, B \notin \mathcal{H}$.

The union of all sub-queries' results is precisely $Join(\mathcal{Q})$ (Lemma 5.2).

For an illustration, Figure 1(a) shows a query $\mathcal{Q}$ with $attset(\mathcal{Q}) = \{\mathsf{A}, \mathsf{B}, ..., \mathsf{K}\}$. Each segment represents a binary relation, e.g., $\{\mathsf{A}, \mathsf{G}\}$. Each ellipse represents a relation of arity 3, e.g., $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$. $\mathcal{Q}$ has thirteen binary relations and three arity-3 relations.

Consider the plan $\boldsymbol{P} = (\{\mathsf{D}\}, \{(\mathsf{G}, \mathsf{H})\})$. Each sub-query $\mathcal{Q}'$ issued for $\boldsymbol{P}$ assigns (i) a heavy value — assumed d below — to D, and (ii) a heavy value pair — assumed (g, h) — to (G, H). $\mathcal{Q}'$ returns all and only the tuples $\boldsymbol{u} \in Join(Q)$ such that:

- $\boldsymbol{u}(\mathsf{D}) = \mathsf{d}$;
- $(\boldsymbol{u}(\mathsf{G}), \boldsymbol{u}(\mathsf{H})) = (\mathsf{g}, \mathsf{h})$;
- $\boldsymbol{u}(A)$ is light for every attribute $A \in \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{I}, \mathsf{J}, \mathsf{K}\}$ (this implies that both g and h are light);
- $(\boldsymbol{u}(A), \boldsymbol{u}(B))$ is light for any distinct attributes $A, B \in \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}, \mathsf{F}, \mathsf{I}, \mathsf{J}, \mathsf{K}\}$.

The relations in $\mathcal{Q}'$ only need to contain the *relevant* tuples. For example, let $\mathcal{R}_{\{\mathsf{G},\mathsf{J}\}}$ be the relation in $\mathcal{Q}$ with scheme $\{\mathsf{G}, \mathsf{J}\}$. In $\mathcal{Q}'$, the corresponding relation $\mathcal{R}'_{\{\mathsf{G},\mathsf{J}\}}$ includes only the tuples $\boldsymbol{v} \in \mathcal{R}_{\{\mathsf{G},\mathsf{J}\}}$ such that $\boldsymbol{v}(\mathsf{G}) = \mathsf{g}$ and $\boldsymbol{v}(\mathsf{J})$ is light. As another example, let $\mathcal{R}_{\{\mathsf{A},\mathsf{B},\mathsf{C}\}}$ be the relation in $\mathcal{Q}$ with scheme $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$. The corresponding relation $\mathcal{R}'_{\{\mathsf{A},\mathsf{B},\mathsf{C}\}}$ in $\mathcal{Q}'$ includes only the tuples $\boldsymbol{v} \in \mathcal{R}_{\{\mathsf{A},\mathsf{B},\mathsf{C}\}}$ such that $\boldsymbol{v}(\mathsf{A})$, $\boldsymbol{v}(\mathsf{B})$, $\boldsymbol{v}(\mathsf{C})$ are light, and so are the value pairs $(\boldsymbol{v}(\mathsf{A}), \boldsymbol{v}(\mathsf{B}))$, $(\boldsymbol{v}(\mathsf{B}), \boldsymbol{v}(\mathsf{C}))$, $(\boldsymbol{v}(\mathsf{A}), \boldsymbol{v}(\mathsf{C}))$.

Since attributes D, G, and H have been fixed to specific values, they can be removed, giving rise to a *residual query* as is shown in Figure 1(b). The 3-arity relation with scheme $\{\mathsf{C}, \mathsf{D}, \mathsf{E}\}$, for instance, now becomes a binary relation with scheme $\{\mathsf{C}, \mathsf{E}\}$. Similarly, three isolated unary relations (on attributes F, J, and K) have been created. We resolve this residual query in three (conceptual) steps:

1. **(Non-unary join)** compute $\mathcal{J}_1$, the join result of the non-unary relations with schemes $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, $\{\mathsf{C}, \mathsf{E}\}$, and $\{\mathsf{E}, \mathsf{I}\}$.
2. **(Isolated CP)** compute $\mathcal{J}_2$, the cartesian product (CP) of the three isolated unary relations.
3. **(Final CP)** compute $\mathcal{J}_1 \times \mathcal{J}_2$, the result of the residual query.

Set $\lambda = p^{1/(\alpha\phi)}$ (ϕ is the generalized vertex-packing number of $\mathcal{Q}$; Section 4). By assigning a share λ to each attribute in $\{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}, \mathsf{I}\}$, we guarantee that *every non-unary relation is two-attribute skew free.* Hence, BinHC can be used to perform Step (1) with load $\tilde{O}(n/\lambda^2) = \tilde{O}(n/p^{2/(\alpha\phi)})$, by virtue of (8).

Ensuring load $\tilde{O}(n/p^{2/(\alpha\phi)})$ for Steps (2) and (3), however, requires new insight into the mathematical structure of the problem, as explained next.

New 3: A new isolated cartesian product theorem. The load of Steps (2) and (3) critically depends on $|\mathcal{J}_2|$, namely, the CP size of the isolated unary relations (e.g., those on F, J, and K in Figure 1(b)). If $|\mathcal{J}_2|$ *were* small for every residual query of the plan $\boldsymbol{P}$, we could bound the load to be $\tilde{O}(n/p^{2/(\alpha\phi)})$ easily. Unfortunately, this is not true: $|\mathcal{J}_2|$ can vary significantly for different residual queries.

We will establish an *isolated cartesian product theorem*, showing that the *average* $|\mathcal{J}_2|$ *of all residual queries dedicated to* $\boldsymbol{P}$ *is sufficiently small.* This permits global optimization to allocate more (resp. less) machines to those residual queries with larger (resp. smaller) $|\mathcal{J}_2|$, thereby guaranteeing a load of $\tilde{O}(n/p^{2/(\alpha\phi)})$ for every residual query. The theorem generalizes an earlier result on binary relations in [13, 20]. Its establishment, which constitutes the most important technical contribution of this paper, owes heavily to the newly introduced ϕ. Our argument is considerably different from (and actually subsumes) the ones in [13, 20], and sheds new light on the join problem.

3 PRELIMINARIES

3.1 Hypergraphs & Edge Coverings/Packings

A *hypergraph* $\mathcal{G}$ is a pair $(\mathcal{V}, \mathcal{E})$ such that (i) $\mathcal{V}$ is a finite set where each element is a *vertex*, and (ii) $\mathcal{E}$ is a set of *hyperedges* — or just *edges* — each being a non-empty subset of $\mathcal{V}$. For each $e \in \mathcal{E}$, the size $|e|$ is the edge's *arity*; and e is *unary* if $|e| = 1$. A vertex in $\mathcal{V}$ is *exposed* if it belongs to no edges. Our discussion will concentrate on hypergraphs without exposed vertices.

Let $\mathcal{W}$ be a function mapping $\mathcal{E}$ to values in $[0, 1]$. We call $\mathcal{W}(e)$ the *weight* of $e \in \mathcal{E}$ (under $\mathcal{W}$) and $\sum_{e \in \mathcal{E}} W(e)$ the *weight* of $\mathcal{W}$. Given a vertex $X \in \mathcal{V}$, we call $\sum_{e \in \mathcal{E}: X \in e} \mathcal{W}(e)$ the *weight* of X (under $\mathcal{W}$). $\mathcal{W}$ is a *fractional edge covering* of $\mathcal{G}$ if the weight of every vertex $X \in \mathcal{V}$ is at least 1. The *fractional edge covering number* of $\mathcal{G}$ — denoted as $\rho(\mathcal{G})$ — is the minimum weight of all fractional edge coverings of $\mathcal{G}$. $\mathcal{W}$ is a *fractional edge packing* of $\mathcal{G}$ if the weight of every vertex $X \in \mathcal{V}$ is at most 1. The *fractional edge packing number* of $\mathcal{G}$ — denoted as $\tau(\mathcal{G})$ — is the maximum weight of all fractional edge packings of $\mathcal{G}$.

Example. The hypergraph $\mathcal{G}$ in Figure 1(a) has a fractional edge covering number $\rho(\mathcal{G}) = 5$, achieved by the function $\mathcal{W}$ that maps {D, K}, {G, J}, {I, E}, {A, B, C}, and {F, G, H} to 1, and the other edges to 0. $\mathcal{G}$ has a fractional edge packing number $\tau(\mathcal{G}) = 4.5$, achieved by the function $\mathcal{W}$ that maps {D, H}, {D, K}, and {K, H} to 0.5, {E, I}, {G, J}, and {A, B, C} to 1, and the other edges to 0. □

Lemma 3.1. *If* $\alpha = \max_{e \in \mathcal{E}} |e|$, *then* $\alpha \cdot \rho(\mathcal{G}) \geq |\mathcal{V}|$.

Proof. Let $\mathcal{W}$ be a fractional edge covering of $\mathcal{G}$ with the minimum weight. Thus, $\alpha \cdot \rho(\mathcal{G}) = \alpha \sum_{e \in \mathcal{E}} \mathcal{W}(e) \geq \sum_{e \in \mathcal{E}} |e| \mathcal{W}(e) = \sum_{X \in |\mathcal{V}|} \sum_{e \in \mathcal{E}: X \in e} \mathcal{W}(e) \geq \sum_{X \in |\mathcal{V}|} 1 = |\mathcal{V}|$. □

Given a subset $\mathcal{U}$ of $\mathcal{V}$, we define the subgraph *induced* by $\mathcal{U}$ as the hypergraph $(\mathcal{U}, \mathcal{E}')$ where

$$\mathcal{E}' \quad = \quad \{\mathcal{U} \cap e \mid e \in \mathcal{E} \land \mathcal{U} \cap e \neq \emptyset\}.$$

3.2 Query Hypergraph and AGM Bound

We say that a query $\mathcal{Q}$ is *clean* if no two relations in $\mathcal{Q}$ share the same scheme. A clean $\mathcal{Q}$ defines a hypergraph $\mathcal{G} = (attset(\mathcal{Q}), \mathcal{E})$ where $\mathcal{E} = \{scheme(\mathcal{R}) \mid \mathcal{R} \in \mathcal{Q}\}$. For each edge $e \in \mathcal{E}$, let $\mathcal{R}_e$ represent the relation $\mathcal{R} \in \mathcal{Q}$ with $e = scheme(\mathcal{R})$.

Lemma 3.2 ([4]). *Let* $\mathcal{W}$ *be a fractional edge covering of the hypergraph* $\mathcal{G} = (attset(\mathcal{Q}), \mathcal{E})$ *defined by a clean query* $\mathcal{Q}$. *Then,* $|Join(\mathcal{Q})| \leq \prod_{e \in \mathcal{E}} |\mathcal{R}_e|^{\mathcal{W}(e)}$.

The above is commonly known as the *AGM bound*.

3.3 MPC Building Blocks

Cartesian products (CP). Consider a query $\mathcal{Q} = \{\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_t\}$ with $t \geq 2$ where the relations have mutually disjoint schemes. Thus, $Join(\mathcal{Q}) = \mathcal{R}_1 \times \mathcal{R}_2 \times ... \times \mathcal{R}_t$; we will use $CP(\mathcal{Q})$ as an alternative representation of this cartesian product.

Lemma 3.3 ([13]). *We can compute* $CP(\mathcal{Q})$ *with load*

$$O\left(\max_{\text{non-empty } \mathcal{Q}' \subseteq \mathcal{Q}} \frac{|CP(\mathcal{Q}')|^{\frac{1}{|\mathcal{Q}'|}}}{p^{\frac{1}{|\mathcal{Q}'|}}}\right) \tag{10}$$

using p machines.

Lemma 3.4 ([12, 13]). $\mathcal{Q}_1$ *and* $\mathcal{Q}_2$ *are queries whose input sizes are bounded by N. If*

- $Join(\mathcal{Q}_1)$ *can be computed with load* $\tilde{O}(N/p_1^{1/t_1})$ *using* p_1 *machines and*
- $Join(\mathcal{Q}_2)$ *with load* $\tilde{O}(N/p_2^{1/t_2})$ *using* p_2 *machines*

then we can compute $Join(\mathcal{Q}_1) \times Join(\mathcal{Q}_2)$ *using* $p_1 p_2$ *machines with load* $\tilde{O}(N/\min\{p^{1/t_1}, p^{1/t_2}\})$.

Skew-free queries. Suppose that we have assigned a share p_A to every attribute $A \in attset(\mathcal{Q})$ subject to the condition in (5). In Appendix A, we prove:

Lemma 3.5. *When* $\mathcal{Q}$ *is two-attribute skew free (Section 2), the BinHC algorithm of [6] answers* $\mathcal{Q}$ *with a load not exceeding* (8), *repeated here for the reader's convenience:*

$$\tilde{O}\Big(\max_{\mathcal{R} \in \mathcal{Q}} \Big(\min_{\substack{\mathcal{V} \subseteq scheme(\mathcal{R}) \\ |\mathcal{V}| \leq 2}} \frac{n}{\prod_{A \in \mathcal{V}} p_A}\Big)\Big).$$

4 GENERALIZED VERTEX PACKINGS

Let $\mathcal{F}$ be a function mapping $\mathcal{V}$ to the values in $(-\infty, 1]$ (note that the output of $\mathcal{F}$ can be negative). Define the *weight* of $\mathcal{F}$ as $\sum_{X \in \mathcal{V}} \mathcal{F}(X)$. For each edge $e \in \mathcal{E}$, we call $\sum_{X \in e} \mathcal{F}(X)$ the *weight* of e (under $\mathcal{F}$).

We say that $\mathcal{F}$ is a *generalized vertex packing* of $\mathcal{G}$ if the weight of every edge $e \in \mathcal{E}$ is at most 1. The *generalized vertex-packing number* of $\mathcal{G}$ — denoted as $\phi(\mathcal{G})$ — is the maximum weight of all the generalized vertex packings of $\mathcal{G}$.

Example. The hypergraph $\mathcal{G}$ in Figure 1(a) has a generalized vertex packing number $\phi(\mathcal{G}) = 5$, as is given by the function $\mathcal{F}$ that maps B to −1, D, E, G, and H to 0, and the other vertices to 1. □

Next, we will prove several properties that will be useful to our analysis. Our discussion will revolve around a special linear program about $\mathcal{G}$:

The **characterizing program** of $\mathcal{G}$

Variables: x_e for each $e \in \mathcal{E}$

Maximize: $\sum_{e \in \mathcal{E}} x_e(|e| - 1)$ subject to

- for each $A \in \mathcal{V}$, $\sum_{e \in \mathcal{E}: A \in e} x_e \le 1$, and
- for each $e \in \mathcal{E}$, $x_e \ge 0$.

The above program is always feasible (e.g., setting $x_e = 0$ for all $e \in \mathcal{E}$), and is always bounded (the first bullet implies $x_e \le 1$). It thus has an optimal solution, which we denote as $\overline{\phi}(\mathcal{G})$.

Example. For the hypergraph $\mathcal{G}$ in Figure 1(a), an optimal solution to the characterizing program sets x_e to 1 for $e = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, $\{\mathsf{F}, \mathsf{G}, \mathsf{H}\}$, $\{\mathsf{D}, \mathsf{K}\}$, and $\{\mathsf{E}, \mathsf{I}\}$, and to 0 for the other edges. This assignment achieves the value $\overline{\phi}(\mathcal{G}) = 6$ for the objective function. □

Lemma 4.1. $\phi(\mathcal{G}) + \overline{\phi}(\mathcal{G}) = |\mathcal{V}|$.

Proof. Consider the dual of the characterizing program:

Variables: y_A for each $A \in \mathcal{V}$

Minimize: $\sum_{A \in \mathcal{V}} y_A$ subject to

- for each $e \in \mathcal{E}$, $\sum_{A \in e} y_A \ge |e| - 1$, and
- for each $A \in \mathcal{V}$, $y_A \ge 0$.

By the duality of linear programming (LP), the optimal value of the objective function in the dual program is identical to that in the characterizing program.

Let $\mathcal{F}$ be a generalized vertex packing of $\mathcal{G}$ of the maximum weight. Define $y_A = 1 - \mathcal{F}(A)$ for each $A \in \mathcal{V}$. We will prove that the assignment $\{y_A \mid A \in \mathcal{V}\}$ must be an optimal solution of the dual program, i.e., $\overline{\phi}(\mathcal{G}) = \sum_A y_A$.

For each $e \in \mathcal{E}$, $\Sigma_{A \in e} y_A = \Sigma_{A \in e}(1 - \mathcal{F}(A)) = |e| - \sum_{A \in e} \mathcal{F}(A) \ge |e| - 1$. This, together with the fact that $y_A \ge 0$ for all $A \in \mathcal{V}$, indicates that $\{y_A \mid A \in \mathcal{V}\}$ is a feasible assignment (for the dual).

Suppose that there is another a feasible assignment $\{y'_A \mid A \in \mathcal{V}\}$ able to make the objective function even lower, namely, $\sum_A y'_A < \sum_A y_A$. Let us design a function $\mathcal{F}'$ by setting $\mathcal{F}'(A) = 1 - y'_A$ for each $A \in \mathcal{V}$. Clearly, $\mathcal{F}'(A) \le 1$ for every $A \in \mathcal{V}$. Furthermore, for each $e \in \mathcal{E}$, $\sum_{A \in e} \mathcal{F}'(A) = \Sigma_{A \in e}(1 - y'_A) = |e| - \Sigma_{A \in e} y'_A \le 1$. Therefore, $\mathcal{F}'$ is a generalized vertex packing. However, $\sum_A \mathcal{F}'(A) = \sum_A(1 - y'_A) > \sum_A(1 - y_A) = \sum_A \mathcal{F}(A)$, which contradicts the definition of $\mathcal{F}$.

It follows from the above discussion that $\overline{\phi}(\mathcal{G}) = \sum_A y_A = \sum_A(1 - \mathcal{F}(A)) = |\mathcal{V}| - \sum_A \mathcal{F}(A) = |\mathcal{V}| - \phi(\mathcal{G})$, which establishes the lemma. □

Lemma 4.2. *If every edge in $\mathcal{G}$ contains two vertices, $\phi(\mathcal{G}) = \rho(\mathcal{G})$.*

Proof. When $|e| = 2$ for every $e \in \mathcal{E}$, $\overline{\phi}(\mathcal{G})$ is exactly the fractional edge packing number $\tau(\mathcal{G})$ (Section 3.1). As proved in Theorem 2.2.7 of [19], such a $\mathcal{G}$ must have the property that $\rho(\mathcal{G}) + \tau(\mathcal{G}) = |\mathcal{V}|$. The lemma thus follows from Lemma 4.1. □

Lemma 4.3. *If $\mathcal{G}$ is the hypergraph defined by a symmetric query $\mathcal{Q}$, $\phi(\mathcal{G}) = k/\alpha$, where α and k are given in* (1) *and* (2)*, respectively.*

Proof. We will prove:

$$\rho(\mathcal{G}) + \overline{\phi}(\mathcal{G}) \quad \le \quad k \tag{11}$$

which will establish the lemma by the following argument. Assume without loss of generality that every vertex in $\mathcal{G}$ is covered by c edges. The number of edges in $\mathcal{G}$ must be ck/α. Consider the function $\mathcal{W}$ that maps each edge to $1/c$. $\mathcal{W}$ is clearly a fractional edge covering with weight k/α. On the other hand, Lemma 3.1 tells us $\rho(\mathcal{G}) \ge k/\alpha$, which leads to $\rho(\mathcal{G}) = k/\alpha$. It follows from (11) that $\overline{\phi}(\mathcal{G}) \le k(1 - \frac{1}{\alpha})$. On the other hand, consider the following assignment to the variables in the characterizing program: $\{x_e = 1/c \mid e \in \mathcal{E}\}$. This assignment satisfies all constraints, and achieves $\sum_{e \in \mathcal{E}} x_e(|e| - 1) = \frac{ck}{\alpha} \frac{1}{c}(\alpha - 1) = k(1 - \frac{1}{\alpha})$. This implies $\overline{\phi}(\mathcal{G}) = k(1 - \frac{1}{\alpha})$, which by Lemma 4.1 leads to $\phi(\mathcal{G}) = k/\alpha$.

It remains to prove (11), which (by Lemma 4.1) is equivalent to showing $\rho(\mathcal{G}) \le \phi(\mathcal{G})$. For this purpose, let us recall two standard concepts from the fractional graph theory [19]. Let $\mathcal{F}'$ be a function that maps $\mathcal{V}$ to the set of real values in $[0, 1]$. $\mathcal{F}'$ is a *fractional vertex packing* of $\mathcal{G}$ if $\sum_{A \in e} \mathcal{F}'(A) \le 1$ for every $e \in \mathcal{E}$. The *fractional vertex-packing number* of $\mathcal{G}$ is the maximum $\sum_{A \in \mathcal{V}} \mathcal{F}'(A)$ of all the fractional vertex packings $\mathcal{F}'$. By the duality of LP, the fractional vertex-packing number is precisely $\rho(\mathcal{G})$. The inequality $\rho(\mathcal{G}) \le \phi(\mathcal{G})$ follows from the definition of $\phi(\mathcal{G})$ (at the beginning of Section 4), and the fact that any fractional vertex packing is a generalized vertex packing. □

5 2-ATTRIBUTE HEAVY-LIGHT TAXONOMY

In Sections 5-7, we consider that the input query $\mathcal{Q}$ is (i) clean (Section 3.2), and (ii) *unary-free*, namely, each relation in $\mathcal{Q}$ has arity at least 2.

Denote by $\mathcal{G} = (attset(\mathcal{Q}), \mathcal{E})$ the hypergraph defined by $\mathcal{Q}$ (Section 3.2). Set λ to an arbitrary positive real value. In the manner explained in Section 2, each value $x \in \mathbf{dom}$ can be classified into light/heavy, and so can each value pair $(x, y) \in \mathbf{dom} \times \mathbf{dom}$.

Configurations. Consider a plan $\boldsymbol{P}$ as defined in (9). As before, set $\mathcal{H} = \{X_1, ..., X_a, Y_1, ..., Y_b, Z_1, ..., Z_b\}$ for some $a, b \ge 0$. Given a $\mathcal{U} \subseteq \mathcal{H}$ and a tuple $\boldsymbol{u}$ over $\mathcal{U}$, we say that $(\mathcal{U}, \boldsymbol{u})$ is a *$\mathcal{U}$-configuration* of $\boldsymbol{P}$ if

- $\boldsymbol{u}(X_i)$ is heavy for every $X_i \in \mathcal{U}, i \in [a]$;
- $\boldsymbol{u}(Y_j)$ is light for every $Y_j \in \mathcal{U}, j \in [b]$;
- $\boldsymbol{u}(Z_j)$ is light for every $Z_j \in \mathcal{U}, j \in [b]$;
- $(\boldsymbol{u}(Y_j), \boldsymbol{u}(Z_j))$ is heavy for every $(Y_j, Z_j) \in \mathcal{U} \times \mathcal{U}, j \in [b]$ (if either $Y_j \notin U$ or $Z_j \notin U$, this condition does not apply to j).

When $\mathcal{U} = \mathcal{H}$, $(\mathcal{U}, \boldsymbol{u})$ is a *full configuration* of $\boldsymbol{P}$.

Proposition 5.1. *$\boldsymbol{P}$ has at most $\lambda^{|\mathcal{H}|}$ full configurations.*

Proof. There can be at most λ values on each X_i ($i \in [a]$), and at most λ^2 value pairs on each (Y_j, Z_j) ($j \in [b]$). Hence, the number of full configurations is at most $\lambda^a \cdot (\lambda^2)^b = \lambda^{|\mathcal{H}|}$. □

Residual queries. Given a full configuration $(\mathcal{H}, \boldsymbol{h})$ of plan $\boldsymbol{P}$, we will formulate a *residual query* $\mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$ which returns exactly the set of tuples $\boldsymbol{u} \in Join(\mathcal{Q})$ "consistent" with $(\mathcal{H}, \boldsymbol{h})$, meaning that:

- $\boldsymbol{u}(A) = \boldsymbol{h}(A)$ for each $A \in \mathcal{H}$;
- $\boldsymbol{u}(A)$ is light for any attribute $A \notin \mathcal{H}$;

- $(\boldsymbol{u}(A), \boldsymbol{u}(B))$ is light for any distinct attributes $A, B \notin \mathcal{H}$.

Formally, an edge $e \in \mathscr{E}$ is *active* on $\boldsymbol{P}$ if e has at least one attribute outside $\mathcal{H}$. For an active edge e, let $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ be the *residual relation* of e that fulfills all the conditions below:

- $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ is over $e' = e \setminus \mathcal{H}$;
- $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ collects the projection $\boldsymbol{v}[e']$ of all the tuples $\boldsymbol{v} \in \mathcal{R}_e$ (see Section 3.2 for the definition of $\mathcal{R}_e$) satisfying
  - $\boldsymbol{v}(A) = \boldsymbol{h}(A)$ for each $A \in e \cap \mathcal{H}$;
  - $\boldsymbol{v}(A)$ is light for each $A \in e'$;
  - $(\boldsymbol{v}(A), \boldsymbol{v}(B))$ is light for any distinct $A, B \in e'$.

We can now formulate the residual query as:

$$\mathcal{Q}'(\mathcal{H}, \boldsymbol{h}) \quad = \quad \left\{ \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h}) \;\middle|\; e \in \mathscr{E},\ e \text{ active on } \boldsymbol{P} \right\}. \qquad (12)$$

Example. Consider the query $\mathcal{Q}$ in Figure 1(a) and the plan $\boldsymbol{P} = (\{\mathsf{D}\}, \{(\mathsf{G}, \mathsf{H})\})$. Hence, $\mathcal{H} = \{\mathsf{D}, \mathsf{G}, \mathsf{H}\}$. Consider the full configuration $(\mathcal{H}, \boldsymbol{h})$ where $\boldsymbol{h} = (\mathsf{d}, \mathsf{g}, \mathsf{h})$.

All edges of the graph in Figure 1(a) are active except $\{\mathsf{D}, \mathsf{H}\}$. Set $e = \{\mathsf{G}, \mathsf{J}\}$; thus $e' = \{\mathsf{J}\}$. The residual relation $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ collects all tuples of the form $(\mathsf{g}, x) \in \mathcal{R}_e$ where x is light. For another example, set $e = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$; thus $e' = e$. The residual relation $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ collects all tuples $(x, y, z) \in \mathcal{R}_e$ such that the values x, y, z and the value pairs $(x, y), (x, z), (y, z)$ are all light. □

The next lemma, proved in Appendix B, shows that we can find the entire $\mathit{Join}(\mathcal{Q})$ by answering each residual query correctly:

Lemma 5.2.

$$\mathit{Join}(\mathcal{Q}) = \bigcup_{\boldsymbol{P}} \left(\bigcup_{\text{full config. } (\mathcal{H},\, \boldsymbol{h}) \text{ of } \boldsymbol{P}} \mathcal{Q}'(\mathcal{H}, \boldsymbol{h}) \times \{\boldsymbol{h}\} \right). \qquad (13)$$

Completing a configuration. Consider a $\mathcal{U}$-configuration of $\boldsymbol{P}$: $(\mathcal{U}, \boldsymbol{u})$. We say that a full configuration $(\mathcal{H}, \boldsymbol{h})$ of $\boldsymbol{P}$ *completes* $(\mathcal{U}, \boldsymbol{u})$ if $\boldsymbol{u} = \boldsymbol{h}[\mathcal{U}]$. The following lemma states that $(\mathcal{U}, \boldsymbol{u})$ cannot be completed by too many full configurations:

Lemma 5.3. *$(\mathcal{U}, \boldsymbol{u})$ can be completed by $O(\lambda^{|\mathcal{H} \setminus \mathcal{U}|})$ full configurations of $\boldsymbol{P}$.*

Proof. We will design a tuple $\boldsymbol{h}$ over $\mathcal{H}$ to make the full configuration $(\mathcal{H}, \boldsymbol{h})$ complete $(\mathcal{U}, \boldsymbol{u})$. Clearly, $\boldsymbol{h}(A) = \boldsymbol{u}(A)$ for every $A \in \mathcal{U}$. It remains to design $\boldsymbol{h}(A)$ for $A \in \mathcal{H} \setminus \mathcal{U}$.

(1) For every $i \in [a]$ with $X_i \notin \mathcal{U}$, $\boldsymbol{h}(X_i)$ can be any heavy value.
(2) For every $j \in [b]$ with $Y_j \in \mathcal{U}$ but $Z_j \notin \mathcal{U}$, $\boldsymbol{h}(Z_j)$ can be any light value that makes $(\boldsymbol{u}(Y_j), \boldsymbol{h}(Z_j))$ heavy.
(3) For every $j \in [b]$ with $Y_j \notin \mathcal{U}$ but $Z_j \in \mathcal{U}$, $\boldsymbol{h}(Y_j)$ can be any light value that makes $(\boldsymbol{h}(Y_j), \boldsymbol{u}(Z_j))$ heavy.
(4) For every $j \in [b]$ with $Y_j \notin \mathcal{U}$ and $Z_j \notin \mathcal{U}$, $\boldsymbol{h}(Y_j)$ and $\boldsymbol{h}(Z_j)$ can be any light values that make $(\boldsymbol{h}(Y_j), \boldsymbol{h}(Z_j))$ heavy.

We claim:

- for (1), there are at most λ ways to set $\boldsymbol{h}(X_i)$;
- for (2), there are at most $\lambda \cdot |\mathcal{Q}|$ ways to set $\boldsymbol{h}(Z_j)$;
- for (3), there are at most $\lambda \cdot |\mathcal{Q}|$ ways to set $\boldsymbol{h}(Y_j)$;
- for (4), there are at most λ^2 ways to set $(\boldsymbol{h}(Y_j), \boldsymbol{h}(Z_j))$.

The first and the last bullets are obvious because there are at most λ heavy values and λ^2 heavy value pairs. As the second and third bullets are symmetric, we will prove only the former. Fix an arbitrary j satisfying the condition in (2); denote by z a possible value for $\boldsymbol{h}(Z_j)$. As $(\boldsymbol{u}(Y_j), z)$ is heavy, at least n/λ^2 tuples — counting over all the relations in $\mathcal{Q}$ — carry $\boldsymbol{u}(Y_j)$ and z (on attributes Y_j and Z_j, resp.) simultaneously. If there are at least $\lambda \cdot |\mathcal{Q}|$ choices of z, the total number of tuples carrying $\boldsymbol{u}(Y_j)$ must be at least $\frac{n}{\lambda^2}\lambda|\mathcal{Q}| = |\mathcal{Q}|n/\lambda$. But this means that at least n/λ such tuples fall in the same relation in $\mathcal{Q}$, contradicting the fact that $\boldsymbol{u}(Y_j)$ is light.

Denote by c_1 the number of i values satisfying (1), and by c_2, c_3, and c_4 the number of j values satisfying (2), (3), and (4), respectively. The above discussion indicates that the number of distinct $(\mathcal{H}, \boldsymbol{h})$ meeting all the four conditions is (applying $|\mathcal{Q}| = O(1)$)

$$O(\lambda^{c_1 + c_2 + c_3 + 2c_4})$$

Notice that $c_1 + c_2 + c_3 + 2c_4$ is at most the number of attributes in $\mathcal{H} \setminus \mathcal{U}$. This completes the proof. □

Total input size of the residual queries. Lemma 5.3 has an important corollary:

Corollary 5.4. *For any plan $\boldsymbol{P}$ of a unary-free query $\mathcal{Q}$, the residual queries (i.e., one for each full configuration of $\boldsymbol{P}$) together have a total input size $O(n \cdot \lambda^{k-2})$. If $\mathcal{Q}$ is α-uniform, the bound can be reduced to $O(n \cdot \lambda^{k-\alpha})$.*

Proof. Consider any edge $e \in \mathscr{E}$. A tuple $\boldsymbol{u} \in \mathcal{R}_e$ participates in a residual query if and only if the full configuration producing the residual query completes the e-configuration $(e, \boldsymbol{u})$. By Lemma 5.3, $(e, \boldsymbol{u})$ can be completed by $O(\lambda^{k-|e|})$ full configurations. For a unary-free $\mathcal{Q}$, $k - |e| \le k - 2$. Hence, the total input size of all residual queries is $O(n \cdot \lambda^{k-2})$. The claim on α-uniform queries follows from the fact that $k - |e| = k - \alpha$. □

We close the section by pointing out that $\mathcal{Q}$ has only a constant number of plans. Therefore, all the residual queries on the right hand side of (13) can be larger than what is stated in Corollary 5.4 by a constant factor.

6 SIMPLIFYING A RESIDUAL QUERY

This section will concentrate on an arbitrary full configuration $(\mathcal{H}, \boldsymbol{h})$ of $\boldsymbol{P}$. We will explain how to simplify the residual query $\mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$ in (12). As before, let $\mathcal{G} = (\mathit{attset}(\mathcal{Q}), \mathscr{E})$ be the hypergraph of $\mathcal{Q}$. Set $\mathcal{L} = \mathit{attset}(\mathcal{Q}) \setminus \mathcal{H}$.

The residual graph. Define $\mathcal{G}' = (\mathcal{L}, \mathscr{E}')$ as the subgraph of $\mathcal{G}$ induced (Section 3.1) by $\mathcal{L}$; we will refer to $\mathcal{G}'$ as the *residual graph* of $\mathcal{H}$. Note that $\mathcal{G}'$ depends only on $\mathcal{H}$ (not on $\boldsymbol{h}$).

A vertex (a.k.a. attribute) $A \in \mathcal{L}$ is *orphaned* if it appears in a unary edge in $\mathscr{E}'$ (that is, $\{A\} \in \mathscr{E}'$). Such a vertex is further said to be *isolated* if it appears in no non-unary edges in $\mathscr{E}'$. Denote by $\mathcal{I}$ the set of isolated vertices.

Example. For the $\mathcal{G}$ in Figure 1(a), Figure 1(b) shows the residual graph $\mathcal{G}'$ for $\mathcal{H} = \{\mathsf{D}, \mathsf{G}, \mathsf{H}\}$. The isolated set is $\mathcal{I} = \{\mathsf{F}, \mathsf{J}, \mathsf{K}\}$. Every other vertex in $\mathcal{L} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{E}, \mathsf{F}, \mathsf{I}, \mathsf{J}, \mathsf{K}\}$ is orphaned. For example, A is orphaned because the edge $\{\mathsf{A}, \mathsf{G}\}$ in $\mathcal{G}$ shrinks into a unary edge A in $\mathcal{G}'$; however, A is not isolated because it also appears in the edge $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ of $\mathcal{G}'$. □

Unary intersection on an orphaned attribute. Consider an orphaned attribute $A \in \mathcal{L}$. Given an edge e in $\mathcal{G}$, we call e an *orphaning edge* of A if $e \setminus \mathcal{H} = \{A\}$. Define a unary relation for A:

$$\mathcal{R}''_A(\mathcal{H}, \boldsymbol{h}) \quad = \quad \bigcap_{\text{orphaning edge } e \text{ of } A} \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h}) \tag{14}$$

where $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ is the residual relation of e (Section 5). Only the values in $\mathcal{R}''_A(\mathcal{H}, \boldsymbol{h})$ can possibly contribute to the result of $\mathcal{Q}$.

Example (cont.). The orphaned attribute C in Figure 1(b) has two orphaning edges: {C, G} and {C, H}. $\mathcal{R}''_\mathsf{C}(\mathcal{H}, \boldsymbol{h})$ is the intersection of the residual relations of those two edges. Equivalently, $\mathcal{R}''_\mathsf{C}(\mathcal{H}, \boldsymbol{h})$ contains all the values x such that tuples (x, g) and (x, h) belong to relations $\mathcal{R}_{\{\mathsf{C},\mathsf{G}\}}$ and $\mathcal{R}_{\{\mathsf{C},\mathsf{H}\}}$, respectively. Similarly, the isolated vertex K has three orphaning edges: {K, D}, {K, G}, and {K, H}. $\mathcal{R}''_\mathsf{K}(\mathcal{H}, \boldsymbol{h})$ contains all the values x such that tuples (x, d), (x, g), and (x, h) belong to relations $\mathcal{R}_{\{\mathsf{K},\mathsf{G}\}}$, $\mathcal{R}_{\{\mathsf{K},\mathsf{G}\}}$, and $\mathcal{R}_{\{\mathsf{K},\mathsf{H}\}}$, respectively. □

Semi-join reduction. Let e be an edge in $\mathcal{G}$ such that $e' = e \setminus \mathcal{H}$ is not unary. Define the *semi-join reduced relation* of e as:

$$\mathcal{R}''_e(\mathcal{H}, \boldsymbol{h}) \quad = \quad \text{the join result of } \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h}) \text{ and all } \mathcal{R}''_A(\mathcal{H}, \boldsymbol{h}) \text{ where } A \in e' \text{ is an orphaned attribute.} \tag{15}$$

To see the rationale behind, consider a tuple $\boldsymbol{u} \in \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ such that $\boldsymbol{u}(A)$ does not appear in $\mathcal{R}''_A(\mathcal{H}, \boldsymbol{h})$; clearly, $\boldsymbol{u}$ cannot contribute to $\mathcal{J}oin(\mathcal{Q})$. $\mathcal{R}''_e(\mathcal{H}, \boldsymbol{h})$ contains what remains in $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ after eliminating all such tuples $\boldsymbol{u}$.

Example (cont.). In Figure 1(b), consider $e = \{\mathsf{C}, \mathsf{D}, \mathsf{E}\}$, and hence, $e' = \{\mathsf{C}, \mathsf{E}\}$. Recall that $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ includes all tuples (x, y) such that (x, d, y) in $\mathcal{R}_e$. $\mathcal{R}''_e(\mathcal{H}, \boldsymbol{h})$ shrinks $\mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ by keeping only the tuples (x, y) where $x \in \mathcal{R}''_\mathsf{C}(\mathcal{H}, \boldsymbol{h})$ and $y \in \mathcal{R}''_\mathsf{E}(\mathcal{H}, \boldsymbol{h})$. □

The simplified residual query. We are now ready to formulate:

$$\mathcal{Q}''_{light}(\mathcal{H}, \boldsymbol{h}) \quad = \quad \left\{\mathcal{R}''_e(\mathcal{H}, \boldsymbol{h}) \,\middle|\, e \setminus \mathcal{H} \text{ is non-unary}\right\} \tag{16}$$

$$\mathcal{Q}''_{\mathcal{I}}(\mathcal{H}, \boldsymbol{h}) \quad = \quad \left\{\mathcal{R}''_A(\mathcal{H}, \boldsymbol{h}) \,\middle|\, A \in \mathcal{L} \text{ is isolated}\right\} \tag{17}$$

$$\mathcal{Q}''(\mathcal{H}, \boldsymbol{h}) \quad = \quad \mathcal{Q}''_{light}(\mathcal{H}, \boldsymbol{h}) \cup \mathcal{Q}''_{\mathcal{I}}(\mathcal{H}, \boldsymbol{h}). \tag{18}$$

Every relation in $\mathcal{Q}''_{\mathcal{I}}(\mathcal{H}, \boldsymbol{h})$ is unary and is on an isolated vertex, whereas every relation in $\mathcal{Q}''_{light}(\mathcal{H}, \boldsymbol{h})$ has at least two attributes.

Example. In Figure 1(b), $\mathcal{Q}''_{light}(\mathcal{H}, \boldsymbol{h})$ joins the semi-join reduced relations of {A, B, C}, {C, E}, and {E, I}. $\mathcal{Q}''_{\mathcal{I}}(\mathcal{H}, \boldsymbol{h})$ computes the CP (cartesian product) of $\mathcal{R}''_\mathsf{F}(\mathcal{H}, \boldsymbol{h})$, $\mathcal{R}''_\mathsf{J}(\mathcal{H}, \boldsymbol{h})$, and $\mathcal{R}''_\mathsf{K}(\mathcal{H}, \boldsymbol{h})$. Finally, $\mathcal{Q}''(\mathcal{H}, \boldsymbol{h})$ returns the CP of the previous two queries' results. □

We prove in Appendix C:

Proposition 6.1. *$\mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$ and $\mathcal{Q}''(\mathcal{H}, \boldsymbol{h})$ have the same result.*

7 AN ISOLATED CP THEOREM

In this section, we will focus on an arbitrary plan $\boldsymbol{P}$. Every full configuration $(\mathcal{H}, \boldsymbol{h})$ of $\boldsymbol{P}$ has a simplified residual query $\mathcal{Q}''(\mathcal{H}, \boldsymbol{h})$ that computes $CP(\mathcal{Q}''_{\mathcal{I}}(\mathcal{H}, \boldsymbol{h})) \times \mathcal{J}oin(\mathcal{Q}''_{light}(\mathcal{H}, \boldsymbol{h}))$. As shown in Lemma 3.3, the cost of $CP(\mathcal{Q}''_{\mathcal{I}}(\mathcal{H}, \boldsymbol{h}))$ depends on the cartesian product size of every *non-empty* subset of the relations in $CP(\mathcal{Q}''_{\mathcal{I}}(\mathcal{H}, \boldsymbol{h}))$. Next, we will prove that the sum of those cartesian product sizes is small.

For every non-empty subset $\mathcal{J}$ of $\mathcal{I}$, define:

$$\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h}) \quad = \quad \{\mathcal{R}''_A(\mathcal{H}, \boldsymbol{h}) \mid A \in \mathcal{J}\}. \tag{19}$$

where $\mathcal{R}''_A(\mathcal{H}, \boldsymbol{h})$ is given in (14). Although the size of $CP(\mathcal{Q}''_{\mathcal{J}}(H, \boldsymbol{h}))$ may vary significantly on different $(H, \boldsymbol{h})$, we are able to bound the total size (lumping over all $(\mathcal{H}, \boldsymbol{h})$):

Theorem 7.1 (Isolated cartesian product theorem). *Let $\mathcal{Q}$ be a unary-free clean query. For each plan $\boldsymbol{P}$ of $\mathcal{Q}$ and any non-empty $\mathcal{J} \subseteq \mathcal{I}$, it holds that*

$$\sum_{\substack{\text{full config.}\\ (\mathcal{H},\, \boldsymbol{h}) \text{ of } \boldsymbol{P}}} \left|CP\left(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})\right)\right| \quad \leq \quad \lambda^{\alpha(\phi - |\mathcal{J}|) - |\mathcal{L} \setminus \mathcal{J}|} \cdot n^{|\mathcal{J}|}$$

where ϕ the generalized vertex packing number of $\mathcal{Q}$ (Section 4).

Recall that $\mathcal{L} = attset(\mathcal{Q}) \setminus \mathcal{H}$, and α is the maximum arity defined in (2). The rest of the section serves as a proof of Theorem 7.1.

7.1 Three Properties of the Isolated Vertices

Let $\mathcal{G} = (attset(\mathcal{Q}), \mathcal{E})$ be the hypergraph defined by $\mathcal{Q}$. The next lemma states several properties of $\mathcal{J}$:

Lemma 7.2. *If an edge $e \in \mathcal{E}$ satisfies $e \cap \mathcal{J} \neq \emptyset$, then*

1. $|e \cap \mathcal{J}| = 1$,
2. $e \subseteq (\mathcal{J} \cup \mathcal{H})$*, and*
3. $|e| = |e \cap \mathcal{J}| + |e \cap \mathcal{H}| = |e \cap \mathcal{H}| + 1$.

Proof. By definition of isolated vertex (Section 6), no edge $e \in \mathcal{E}$ can contain two vertices in $\mathcal{J}$, which yields Property (1). To prove (2), let A be the only attribute in $e \cap \mathcal{J}$. If (2) does not hold, then because of (1), e must have another attribute $B \notin \mathcal{H}$ which, however, contradicts the fact that A is isolated. (3) follows from (1), (2) and the fact that $\mathcal{J}$ and $\mathcal{H}$ are disjoint. □

7.2 Utilizing the Characterizing Program

Henceforth, we will denote by $\{x_e \mid e \in \mathcal{E}\}$ an optimal assignment for the characterizing program of $\mathcal{Q}$ (Section 4). Thus, $\overline{\phi}(\mathcal{Q}) = \sum_e x_e(|e| - 1)$; we will abbreviate $\overline{\phi}(\mathcal{Q})$ as $\overline{\phi}$.

Define

$$\mathcal{E}^* \quad = \quad \{e \in \mathcal{E} \mid e \cap \mathcal{J} \neq \emptyset\}. \tag{20}$$

We prove in Appendix D (recall that $k = |attset(\mathcal{Q})|$):

Lemma 7.3.

$$k - |\mathcal{J}| - \sum_{e \in \mathcal{E}^*} x_e \cdot (|e| - 1) \quad \leq \quad \alpha\,(\phi - |\mathcal{J}|)\,.$$

In the next subsection, we will prove:

Lemma 7.4.

$$\sum_{\substack{\text{full config.}\\ (\mathcal{H},\, \boldsymbol{h}) \text{ of } \boldsymbol{P}}} \left|CP\left(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})\right)\right| \leq \lambda^{|\mathcal{H}| - \sum_{e \in \mathcal{E}^*} x_e(|e|-1)} \cdot n^{|\mathcal{J}|}.$$

This will complete the whole proof of Theorem 7.1 because

$$\begin{aligned}
|\mathcal{H}| - \sum_{e \in \mathcal{E}^*} |x_e| \cdot (|e| - 1) \quad &\leq \quad |\mathcal{H}| - k + |\mathcal{J}| + \alpha\,(\phi - |\mathcal{J}|) \\
&= \quad \alpha\,(\phi - |\mathcal{J}|) - |\mathcal{L}| + |\mathcal{J}| \\
&= \quad \alpha\,(\phi - |\mathcal{J}|) - |\mathcal{L} \setminus \mathcal{J}|
\end{aligned}$$

where the first inequality applied Lemma 7.3.

7.3 Proof of Lemma 7.4

Recall from (9) that plan $\boldsymbol{P}$ specifies $a \geq 0$ attributes $X_1, ..., X_a$ and $b \geq 0$ attribute pairs $(Y_1, Z_1), ..., (Y_b, Z_b)$. We create $a + b$ special relations as follows:

- for each $i \in [a]$, create a relation $\mathcal{S}_i$ with *scheme*$(\mathcal{S}_i) = \{X_i\}$ which contains all the values heavy on X_i;
- for each $j \in [b]$, create a relation $\mathcal{D}_j$ with *scheme*$(\mathcal{D}_j) = \{Y_j, Z_j\}$ which contains all the value pairs (y, z) such that
  - (y, z) is heavy on $\{Y_j, Z_j\}$;
  - y and z are both light values.

Clearly, $|\mathcal{S}_i| \leq \lambda$ and $|\mathcal{D}_j| \leq \lambda^2$ for all i, j. Let $\mathcal{Q}_{heavy}$ be the query that includes the above $a+b$ relations. Note that $attset(\mathcal{Q}_{heavy}) = \mathcal{H}$. As the relations in $\mathcal{Q}_{heavy}$ have disjoint schemes, the result of $\mathcal{Q}_{heavy}$ is $CP(\mathcal{Q}_{heavy})$.

Define:

$$\mathcal{Q}^* \quad = \quad \{\mathcal{R}_e \mid e \in \mathcal{E}^*\}.$$

where $\mathcal{E}^*$ is defined in (20). Recall that $\mathcal{R}_e$ is the input relation in $\mathcal{Q}$ with scheme e. By Property (2) in Lemma 7.2, $attset(\mathcal{Q}^*) \subseteq \mathcal{J} \cup \mathcal{H}$. We prove in Appendix E:

Proposition 7.5.

$$\sum_{\substack{\text{full config.} \\ (\mathcal{H}, \boldsymbol{h}) \text{ of } \boldsymbol{P}}} \left|CP\left(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})\right)\right| \quad \leq \quad \left|CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}^*)\right|.$$

Our goal, therefore, is to prove that the size of $CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}^*)$ is sufficiently small.

Strategy overview. Towards the above purpose, we will construct a sequence of queries $\mathcal{Q}_0, \mathcal{Q}_1, ..., \mathcal{Q}_\ell$ for some $\ell \geq 0$ such that:

- $\mathcal{Q}_0 = \mathcal{Q}^*$;
- $attset(\mathcal{Q}_s) \subseteq \mathcal{J} \cup \mathcal{H}$ for each $s \in [0, \ell]$;
- $CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}_s) = CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}^*)$ for each $s \in [0, \ell]$.

Eventually, we will prove that $|CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}_\ell)|$ is low enough. This, together with Proposition 7.5, will allow us to establish Lemma 7.4.

Constructing $\mathcal{Q}_{s+1}$ for $s \geq 0$. Suppose that we have already obtained $\mathcal{Q}_s$ for some $s \geq 0$. Let $\mathcal{G}_s = (attset(\mathcal{Q}_s), \mathcal{E}_s)$ be the hypergraph defined by $\mathcal{Q}_s$. For each edge $e \in \mathcal{E}_s$, denote by $\mathcal{R}_{e,s}$ the relation in $\mathcal{Q}_s$ whose scheme is e.

We assume that each edge $e \in \mathcal{E}_s$ is assigned a real value $x_{e,s} \geq 0$ such that $\{x_{e,s} \mid e \in \mathcal{E}_s\}$ is a feasible assignment of the characterizing program of $\mathcal{G}_s$ (Section 4). At $s = 0$, this assumption is fulfilled by simply setting $x_{e,0} = x_e$ for each $e \in \mathcal{E}_s$. Recall (Section 7.2) that $\{x_e \mid e \in \mathcal{E}\}$ is a feasible assignment for the characterizing program of $\mathcal{G}$.

For each attribute $A \in \mathcal{J} \cup \mathcal{H}$, define:

$$\mathcal{F}_s(A) \quad = \quad \begin{cases} \sum_{e \in \mathcal{E}_s : A \in e} x_{e,s} & \text{if } A \in attset(\mathcal{Q}_s) \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

Note that $\mathcal{F}_s(A) \leq 1$ for every A. We build $\mathcal{Q}_{s+1}$ only if there exists some $j \in [b]$ such that $\mathcal{F}_s(Y_j) \neq \mathcal{F}_s(Z_j)$; otherwise, $\mathcal{Q}_s$ is the last query constructed (i.e., $\ell = s$). We refer to j as the *triggering index* of $\mathcal{Q}_{s+1}$. Without loss of generality, our discussion below assumes $\mathcal{F}_s(Y_j) > \mathcal{F}_s(Z_j)$. The opposite case is symmetric and can be handled analogously.

By the fact $\mathcal{F}_s(Y_j) > \mathcal{F}_s(Z_j)$, there must exist a *triggering edge* $e^* \in \mathcal{E}_s$ which includes Y_j but not Z_j. Based on e^*, we will produce a feasible assignment $\{x_{e,s+1} \mid e \in \mathcal{E}_{s+1}\}$ for the characterizing program of the hypergraph $\mathcal{G}_{s+1} = (attset(\mathcal{Q}_{s+1}), \mathcal{E}_{s+1})$ defined by $\mathcal{Q}_{s+1}$. After that, the construction process is iteratively invoked.

Define:

$$e^+ \quad = \quad e^* \cup \{Z_j\} \tag{22}$$

$$\mathcal{R}^+ \quad = \quad \begin{cases} \mathcal{R}_{e^*,s} \bowtie \mathcal{D}_j \bowtie \mathcal{R}_{e^+,s} & \text{if } e^+ \in \mathcal{E}_s \\ \mathcal{R}_{e^*,s} \bowtie \mathcal{D}_j & \text{otherwise} \end{cases} \tag{23}$$

Note that $scheme(\mathcal{R}^+) = e^+$. Set

$$\Delta_s \quad = \quad \min\{x_{e^*,s}, \mathcal{F}_s(Y_j) - \mathcal{F}_s(Z_j)\}.$$

Depending on Δ_s, we generate $\mathcal{Q}_{s+1}$ differently:

- **Case $\Delta_s < x_{e^*,s}$:**
  - If $e^+ \in \mathcal{E}_s$, then $\mathcal{Q}_{s+1} = (\mathcal{Q}_s \setminus \{\mathcal{R}_{e^+,s}\}) \cup \{\mathcal{R}^+\}$;
  - otherwise, $\mathcal{Q}_{s+1} = \mathcal{Q}_s \cup \{\mathcal{R}^+\}$.
- **Case $\Delta_s = x_{e^*,s}$:**
  - If $e^+ \in \mathcal{E}_s$, then $\mathcal{Q}_{s+1} = (\mathcal{Q}_s \setminus \{\mathcal{R}_{e^*,s}, \mathcal{R}_{e^+,s}\}) \cup \{\mathcal{R}^+\}$;
  - otherwise, $\mathcal{Q}_{s+1} = (\mathcal{Q}_s \setminus \{\mathcal{R}_{e^*,s}\}) \cup \{\mathcal{R}^+\}$.

Remember that we also need to produce a feasible assignment $\{x_{e^*,s+1} \mid e^* \in \mathcal{G}_{s+1}\}$ for the characterizing program of $\mathcal{G}_{s+1}$. This is done as follows:

- First, for every edge e in $\mathcal{G}_s$ other than e^* and e^+, retain its assigned value, namely, $x_{e,s+1} = x_{e,s}$.
- Then, we proceed differently depending on Δ_s:
  - **Case $\Delta_s < x_{e^*,s}$:**
    * Set $x_{e^*,s+1} = x_{e^*,s} - \Delta_s$.
    * Set $x_{e^+,s+1}$ to $\Delta_s + x_{e^+,s}$ if $e^+ \in \mathcal{E}_s$, or to Δ_s otherwise.
  - **Case $\Delta_s = x_{e^*,s}$:**
    * Set $x_{e^+,s+1}$ to $\Delta_s + x_{e^+,s}$ if $e^+ \in \mathcal{E}_s$, or to Δ_s otherwise.

We now prove that the above generation fulfills our purposes:

Lemma 7.6. *Both statements below are true:*

- $\{x_{e,s+1} \mid e \in \mathcal{G}_{s+1}\}$ *is a feasible assignment for the characterizing program of* $\mathcal{G}_{s+1}$.
- $CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}_{s+1}) = CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}^*)$.

Proof. For the first statement, we need to prove that $\mathcal{F}_{s+1}(A) \leq 1$ for every $A \in \mathcal{J} \cup \mathcal{H}$, where $\mathcal{F}_{s+1}(A)$ is as defined in (21).[4] Our design makes sure $\mathcal{F}_{s+1}(A) = \mathcal{F}_s(A)$ for every $A \in \mathcal{J} \cup \mathcal{H}$ with $A \neq Z_j$; thus, $\mathcal{F}_{s+1}(A) \leq 1$ follows from $\mathcal{F}_s(A) \leq 1$. Regarding Z_j, our design guarantees $\mathcal{F}_{s+1}(Z_j) = \mathcal{F}_s(Z_j) + \Delta_s$. By definition $\Delta_s \leq \mathcal{F}_s(Y_j) - \mathcal{F}_s(Z_j)$. Therefore, $\mathcal{F}_{s+1}(Z_j) \leq \mathcal{F}_s(Y_j) \leq 1$.

For the second statement, it suffices to show

$$CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}_{s+1}) = CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{J}oin(\mathcal{Q}_s). \tag{24}$$

[4]Obviously, by replacing s with $s + 1$.

Consider first the case where $e^+ \in \mathscr{E}_s$. Recall that $\mathcal{D}_j \in \mathscr{Q}_{heavy}$. Thus, $\mathcal{R}_{e^*,s} \bowtie \mathcal{R}_{e^+,s} \bowtie CP(\mathscr{Q}_{heavy})$ equals $\mathcal{R}_{e^*,s} \bowtie \mathcal{D}_j \bowtie \mathcal{R}_{e^+,s} \bowtie CP(\mathscr{Q}_{heavy})$, which is simply $\mathcal{R}^+ \bowtie CP(\mathscr{Q}_{heavy})$. This proves the correctness of (24). The case where $e^+ \notin \mathscr{E}_s$ is similar. □

The next lemma shows that the above inductive generative process will terminate eventually:

LEMMA 7.7. *The sequence of queries $\mathscr{Q}_0, \mathscr{Q}_1, ..., \mathscr{Q}_\ell$ is finite.*

PROOF. Recall that the generation proceeds differently in two cases depending on Δ_s. It suffices to show that each case can occur only a finite number of times.

A new query is generated only if we can find $j \in [b]$ such that $\mathscr{F}_s(Y_j) \neq \mathscr{F}_s(Z_j)$. Every time the case $\Delta_s < x_{e^*,s}$ happens, $\Delta_s = \mathscr{F}_s(Y_j) - \mathscr{F}_s(Z_j)$. Therefore, $\mathscr{F}_{s+1}(Y_j) = \mathscr{F}_{s+1}(Z_j)$, by the reasoning in the proof of Lemma 7.6. This means that $\mathscr{F}_{s'}$ will remain the same as $\mathscr{F}_{s'}(Z_j)$ for all $s' \geq s+1$. Therefore, this case can occur at most b times.

For each relation $\mathcal{R} \in \mathscr{Q}_s$, we define its *imbalance count* as the number of $j \in [b]$ values satisfying the condition that *scheme*$(\mathcal{R})$ contains exactly one attribute in $\{Y_j, Z_j\}$. The *imbalance count* of $\mathscr{Q}_s$ is the sum of imbalanced counts of all the relations therein.

- After an occurrence of the case $\Delta_s < x_{e^*,s}$, the imbalance count of $\mathscr{Q}_{s+1}$ can increase by at most b compared to that of $\mathscr{Q}_s$, due to the inclusion of $\mathcal{R}^+$.
- After an occurrence of the case $\Delta_s = x_{e^*,s}$, the imbalance count of $\mathscr{Q}_{s+1}$ must decrease by 1 compared to that of $\mathscr{Q}_s$, due to the eviction of $\mathcal{R}_{e^*,s}$.

Given that the $\Delta_s < x_{e^*,s}$ case happens at most b times, we conclude that $\Delta_s = x_{e^*,s}$ can occur at most $b(|\mathscr{Q}| + b)$ times. □

LEMMA 7.8. *In the last query $\mathscr{Q}_\ell$, it must hold for each $j \in [b]$ that*

$$\mathscr{F}_\ell(Y_j) = \mathscr{F}_\ell(Z_j) = \max\{\mathscr{F}_0(Y_j), \mathscr{F}_0(Z_j)\}.$$

For any $A \notin \{Y_1, Z_1, ..., Y_b, Z_b\}$, it holds that $\mathscr{F}_0(A) = \mathscr{F}_1(A) = ... = \mathscr{F}_\ell(A)$.

PROOF. The generative process must continue if $\mathscr{F}_\ell(X_j) \neq \mathscr{F}_\ell(Y_j)$; this proves $\mathscr{F}_\ell(Y_j) = \mathscr{F}_\ell(Z_j)$. Assume, without loss of generality, that $\mathscr{F}_0(Y_j) > \mathscr{F}_0(Z_j)$. By the argument in the proof of Lemma 7.6, we know $\mathscr{F}_0(Y_j) = \mathscr{F}_1(Y_j) = ... = \mathscr{F}_\ell(Y_j)$. This proves the correctness of the first sentence of the lemma.

The second sentence follows directly from the way we design $x_{e,s}$. □

Bounding the size of $CP(\mathscr{Q}_{heavy}) \bowtie Join(\mathscr{Q}_\ell)$**.** For each $s \in [0, \ell]$, define:

$$\mathscr{B}_s = \prod_{e \in \mathscr{E}_s} |\mathcal{R}_{e,s}|^{x_{e,s}}$$
$$\Delta = \sum_{j \in [b]} |\mathscr{F}_0(Y_j) - \mathscr{F}_0(Z_j)|.$$

LEMMA 7.9.

$$\mathscr{B}_\ell \leq \mathscr{B}_0 \cdot \lambda^{\Delta}.$$

PROOF. Let us start by observing:

$$\Delta = \sum_{s \in [0, \ell-1]} \Delta_s. \quad (25)$$

To explain why, consider an arbitrary $s \in [0, \ell-1]$. If j is the triggering index of $\mathscr{Q}_{s+1}$, $|\mathscr{F}_{s+1}(Y_j) - \mathscr{F}_{s+1}(Z_j)|$ must decrease by Δ_s compared to $|\mathscr{F}_s(Y_j) - \mathscr{F}_s(Z_j)|$. The correctness of (25) then follows from Lemma 7.8.

Equipped with (25), to prove the lemma it suffices to show:

$$\mathscr{B}_{s+1} \leq \mathscr{B}_s \cdot \lambda^{\Delta_s} \quad (26)$$

holds for every $s \in [0, \ell-1]$. For this purpose, let us scrutinize once again the two cases that happen in generating $\mathscr{Q}_{s+1}$. As before, let j be the triggering index and e^* be the triggering edge. We will first consider the scenario where $e^+ \notin \mathscr{E}_s$.

- **Case** $\Delta_s < x_{e^*,s}$**:** We have

$$\mathscr{B}_{s+1} = \mathscr{B}_s \cdot \frac{|\mathcal{R}_{e^*,s}|^{x_{e^*,s}-\Delta_s} \cdot |\mathcal{R}^+|^{\Delta_s}}{|\mathcal{R}_{e^*,s}|^{x_{e^*,s}}} \quad (27)$$

  where e^+ and $\mathcal{R}^+$ are defined in (22) and (23), respectively. In general, it holds that

$$|\mathcal{R}^+| \leq |\mathcal{R}_{e^*,s}| \cdot \lambda \quad (28)$$

  To understand why, first notice that less than λ tuples in $\mathcal{D}_j$ can share the same value on Y_j.[5] This, in turn, indicates that a tuple $\mathcal{R}_{e^*,s}$ can join with less than λ tuples in $\mathcal{D}_j$, which yields (28). Putting together (27) and (28) validates (26).
- **Case** $\Delta_s = x_{e^*,s}$**:**

$$\mathscr{B}_{s+1} = \mathscr{B}_s \cdot \frac{|\mathcal{R}^+|^{\Delta_s}}{|\mathcal{R}_{e^*,s}|^{\Delta_s}}$$

  which together with (28) validates (26).

The scenario where $e^+ \in \mathscr{E}_s$ is similar:

- **Case** $\Delta_s < x_{e^*,s}$**:**

$$\mathscr{B}_{s+1} = \mathscr{B}_s \cdot \frac{|\mathcal{R}_{e^*,s}|^{x_{e^*,s}-\Delta_s} \cdot |\mathcal{R}^+|^{\Delta_s + x_{e^+,s}}}{|\mathcal{R}_{e^*,s}|^{x_{e^*,s}} \cdot |\mathcal{R}_{e^+,s}|^{x_{e^+,s}}} \quad (29)$$

- **Case** $\Delta_s = x_{e^*,s}$**:**

$$\mathscr{B}_{s+1} = \mathscr{B}_s \cdot \frac{|\mathcal{R}^+|^{\Delta_s + x_{e^+,s}}}{|\mathcal{R}_{e^*,s}|^{\Delta_s} \cdot |\mathcal{R}_{e^+,s}|^{x_{e^+,s}}} \quad (30)$$

Both (29) and (30) give (26), using (28) and the fact that $|\mathcal{R}^+| \leq |\mathcal{R}_{e^+,s}|$. The completes the proof of Lemma 7.9. □

We are ready to bound the size of $CP(\mathscr{Q}_{heavy}) \bowtie Join(\mathscr{Q}_\ell)$. Towards this purpose, for each attribute $A \in \mathcal{J}$, let us create a "domain" relation $\mathcal{U}_A$ which includes all the A-values that appear in the input relations of $\mathscr{Q}$. Clearly, $|\mathcal{U}_A| \leq n$. Now, define a clean query:

$$\mathscr{Q}_{final} = \mathscr{Q}_{heavy} \cup \mathscr{Q}_\ell \cup \left(\bigcup_{A \in \mathcal{J}} \{\mathcal{U}_A\} \right).$$

We prove in Appendix F:

PROPOSITION 7.10. *$Join(\mathscr{Q}_{final}) = CP(\mathscr{Q}_{heavy}) \bowtie Join(\mathscr{Q}_\ell)$.*

[5]Recall that each tuple (y, z) in $\mathcal{D}_j$ must appear in at least n/λ^2 tuples of the input relations of $\mathscr{Q}$. Hence, if y can pair up with at least λ distinct z, y appears in at least $\frac{n}{\lambda^2}\lambda = n/\lambda$ tuples. This contradicts the fact that y is a light value.

Hence, instead of the size of $CP(\mathcal{Q}_{heavy}) \bowtie \mathcal{Join}(\mathcal{Q}_\ell)$, we can focus on bounding $|\mathcal{Join}(\mathcal{Q}_{final})|$:

Lemma 7.11.
$$|\mathcal{Join}(\mathcal{Q}_{final})| \le n^{|\mathcal{J}|} \cdot \lambda^{|\mathcal{H}| - \sum_{e \in \mathcal{E}^*} x_e(|e|-1)}. \tag{31}$$

Proof. Let $\mathcal{G}_{final}$ be the hypergraph defined by $\mathcal{Q}_{final}$. We will construct a fractional edge covering $\mathcal{W}$ of $\mathcal{G}_{final}$, which by the AGM bound in Lemma 3.2 yields an upper bound on $|\mathcal{Join}(\mathcal{Q}_{final})|$ which will turn out to be the right hand side of (31).

The construction of $\mathcal{W}$ is based on the feasible assignment $\{x_{e,\ell} \mid e \in \mathcal{E}_\ell\}$ of the characterizing program of $\mathcal{G}_\ell$:

- For each edge e of $\mathcal{G}_{final}$, we set $\mathcal{W}(e) = x_{e,\ell}$.
- For the unary edge $\{A\}$ where $A \in \mathcal{J}$, set $\mathcal{W}(\{A\}) = 1 - \mathcal{F}_\ell(A)$. By the feasibility of $\{x_{e,\ell} \mid e \in \mathcal{E}_\ell\}$, $\mathcal{F}_\ell(A) \le 1$; and hence, $\mathcal{W}(\{A\}) \ge 0$.
- For the unary edge $\{X_i\}$ where $i \in [a]$, set $\mathcal{W}(\{X_i\}) = 1 - \mathcal{F}_\ell(X_i)$.
- For the binary edge $\{Y_j, Z_j\}$ where $j \in [b]$, set $\mathcal{W}(\{Y_j, Z_j\}) = 1 - \mathcal{F}_\ell(Y_j)$, which must be equal to $1 - \mathcal{F}_\ell(Z_j)$ by Lemma 7.8.

The weight of A under $\mathcal{W}$ equals exactly 1 for every $A \in attset(\mathcal{Q}_{final}) = \mathcal{J} \cup \mathcal{H}$, namely, $\mathcal{W}$ is an edge covering for $\mathcal{G}_{final}$.

By Lemma 3.2, $|\mathcal{Join}(\mathcal{Q}_{final})|$ is bounded by

$$\prod_{A \in \mathcal{J}} n^{1-\mathcal{F}_l(A)} \prod_{e \in \mathcal{G}_\ell} |\mathcal{R}_{e,\ell}|^{x_{e,\ell}} \prod_{i=1}^{a} |\mathcal{S}_i|^{1-\mathcal{F}_l(X_i)} \prod_{j=1}^{b} |\mathcal{D}_j|^{1-\mathcal{F}_l(Y_j)}$$
$$= n^{|\mathcal{J}| - \sum_{A \in \mathcal{J}} \mathcal{F}_l(A)} \cdot \mathcal{B}_\ell \cdot \lambda^{a - \sum_{i=1}^a \mathcal{F}_\ell(X_i)} \cdot \lambda^{2\sum_{j=1}^b (1-\mathcal{F}_\ell(Y_j))}. \tag{32}$$

Utilizing Lemmas 7.8 and 7.9, we can derive:

$$\begin{aligned}
& \mathcal{B}_\ell \cdot \lambda^{2\sum_{j=1}^b (1-\mathcal{F}_\ell(Y_j))} \\
\le\ & \mathcal{B}_0 \cdot \lambda^\Delta \cdot \lambda^{\sum_{j=1}^b 2 - 2\max\{\mathcal{F}_0(Y_j), \mathcal{F}_0(Z_j)\}} \\
=\ & \mathcal{B}_0 \cdot \lambda^{\sum_{j=1}^b 2 - 2\max\{\mathcal{F}_0(Y_j), \mathcal{F}_0(Z_j)\} + |\mathcal{F}_0(Y_j) - \mathcal{F}_0(Z_j)|} \\
=\ & \mathcal{B}_0 \cdot \lambda^{\sum_{j=1}^b (1-\mathcal{F}_0(Y_j)) + (1-\mathcal{F}_0(Z_j))} \\
\le\ & \left(\prod_{e \in \mathcal{E}^*} n^{x_e}\right) \cdot \lambda^{2b - \sum_{j=1}^b (\mathcal{F}_0(Y_j) + \mathcal{F}_0(Z_j))}.
\end{aligned}$$

Plugging the above into (32) and applying $\mathcal{F}_\ell(X_i) = \mathcal{F}_0(X_i)$ for each $i \in [a]$ (Lemma 7.8) yields:

$$(32) \le n^{|\mathcal{J}| - \sum_{A \in \mathcal{J}} \mathcal{F}_l(A)} \cdot \left(\prod_{e \in \mathcal{E}^*} n^{x_e}\right) \cdot \lambda^{|\mathcal{H}| - \sum_{A \in \mathcal{H}} \mathcal{F}_0(A)}. \tag{33}$$

By Property (1) of Lemma 7.2, every $e \in \mathcal{E}^*$ covers exactly one attribute in $\mathcal{J}$. Thus:

$$\sum_{e \in \mathcal{E}^*} x_e = \sum_{A \in \mathcal{J}} \mathcal{F}_0(A) = \sum_{A \in \mathcal{J}} \mathcal{F}_\ell(A)$$

where the last equality used Lemma 7.8. Furthermore:

$$\sum_{A \in \mathcal{H}} \mathcal{F}_0(A) \ge \sum_{A \in \mathcal{H}} \sum_{e \in \mathcal{E}^* : A \in e} x_e = \sum_{e \in \mathcal{E}^*} x_e(|e|-1)$$

where the last equality used Property (3) of Lemma 7.2. Therefore, from (33), we get

$$(32) \quad \le \quad n^{|\mathcal{J}|} \cdot \lambda^{|\mathcal{H}| - \sum_{e \in \mathcal{E}^*} x_e(|e|-1)}$$

which completes the proof. □

By combining Lemmas 7.11, 7.6 and Propositions 7.10 and 7.5, we conclude the proof of Lemma 7.2.

8 AN MPC JOIN ALGORITHM

Next, we will describe our MPC algorithm for answering an arbitrary query $\mathcal{Q}$. It suffices to consider that $\mathcal{Q}$ is clean (Section 3.1) because otherwise $\mathcal{Q}$ can be converted to a clean query with the same result in load $\tilde{O}(n/p)$ [14].

Specifically, we will fix an arbitrary plan $\boldsymbol{P}$ of $\mathcal{Q}$, and explain how to compute

$$\bigcup_{\text{full config. } (\mathcal{H},\, \boldsymbol{h}) \text{ of } \boldsymbol{P}} \mathcal{Q}''(\mathcal{H}, \boldsymbol{h}) \quad = \quad \bigcup_{\text{full config. } (\mathcal{H},\, \boldsymbol{h}) \text{ of } \boldsymbol{P}} \mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$$

where the equality is due to Proposition 6.1. By taking the union of the above for every $\boldsymbol{P}$, we obtain the final result $\mathcal{Join}(\mathcal{Q})$ (Lemma 5.2). We will prove an identical upper bound on the load for all $\boldsymbol{P}$. Since $\mathcal{Q}$ has $O(1)$ plans, processing all of them concurrently can increase the load only by a constant factor.

We will first discuss the scenario where $\mathcal{Q}$ is unary-free. Extending the algorithm to allow unary relations is easy, as will be shown in Appendix G.

From now, we will fix

$$\lambda \quad = \quad p^{1/(\alpha\phi)}. \tag{34}$$

In general, it holds that[6]

$$k \le \alpha\rho \le \alpha\phi \tag{35}$$

where ρ is the fractional edge covering number of $\mathcal{Q}$. By Proposition 5.1, the number of full configurations of $\boldsymbol{P}$ is at most

$$\lambda^{|\mathcal{H}|} \le \lambda^k = p^{k/(\alpha\phi)} \le p.$$

Without loss of generality, we assume that, given a tuple $\boldsymbol{u}$ in any input relation of $\mathcal{Q}$, the machine can identify all heavy values and value-pairs contained in $\boldsymbol{u}$. This can be achieved with the techniques of [11] which essentially sort the input relations a constant number of times, incurring an extra load of $\tilde{O}(n/p)$.

Step 1: Generating the input relations of the residual queries. Recall (from Section 5) that a residual query $\mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$ is defined for every full configuration $(\mathcal{H}, \boldsymbol{h})$. Denote by $n_{\mathcal{H},\boldsymbol{h}}$ the input size of $\mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$. All the $n_{\mathcal{H},\boldsymbol{h}}$ values can be obtained with sorting and broadcast to all machines in load $\tilde{O}(p + n/p) = \tilde{O}(n/p)$.

We allocate

$$p'_{\mathcal{H},\boldsymbol{h}} \quad = \quad p \cdot \frac{n_{\mathcal{H},\boldsymbol{h}}}{\Theta(n \cdot \lambda^{k-2})}$$

machines to store the relations of $\mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$. By Corollary 5.4, the total number of machines needed is at most p. The number of tuples received by each machine is bounded by

$$O\left(\frac{n_{\mathcal{H},\boldsymbol{h}}}{p'_{\mathcal{H},\boldsymbol{h}}}\right) = O\left(\frac{n \cdot \lambda^{k-2}}{p}\right) = O\left(\frac{n \cdot p^{\frac{k-2}{\alpha\phi}}}{p}\right) = O\left(\frac{n}{p^{\frac{2}{\alpha\phi}}}\right)$$

[6]The first inequality is by Lemma 3.1, and the second is due to fact $\rho \le \phi$ (shown in the proof of Lemma 4.3).

where the last equality used (35).

Step 2: Simplifying the residual queries. In this step, for each $\mathcal{Q}'(\mathcal{H},\boldsymbol{h})$, its $p'_{\mathcal{H},\boldsymbol{h}}$ assigned machines work together to produce the simplified residual query $\mathcal{Q}''(\mathcal{H},\boldsymbol{h})$ as given in (18). This requires only set intersection (for (14)) and semi-join reduction (for (15)), both of which can be performed with load $O(n_{\mathcal{H},\boldsymbol{h}}/p'_{\mathcal{H},\boldsymbol{h}}) = O(n/p^{2/(\alpha\phi)})$ [14]. After this, the size of $CP(\mathcal{Q}''_{\mathcal{I}}(\mathcal{H},\boldsymbol{h}))$ is known for each $(\mathcal{H},\boldsymbol{h})$; those at most p CP sizes are broadcast to all p machines with load $O(p) = O(n/p)$.

Step 3: Computing the simplified residual queries. For each $(\mathcal{H},\boldsymbol{h})$, we allocate

$$p''_{\mathcal{H},\boldsymbol{h}} = \Theta\left(\lambda^{|\mathcal{L}|} + p \sum_{\text{non-empty } \mathcal{J} \subseteq \mathcal{I}} \frac{|CP(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H},\boldsymbol{h}))|}{\Theta(\lambda^{\alpha(\phi-|\mathcal{J}|)-|\mathcal{L}\setminus\mathcal{J}|} \cdot n^{|\mathcal{J}|})}\right) \quad (36)$$

machines to process $\mathcal{Q}''(\mathcal{H},\boldsymbol{h})$, where $\mathcal{Q}''_{\mathcal{J}}(\mathcal{H},\boldsymbol{h})$ is defined in (19). By Theorem 7.1 and the fact $\sum_{(\mathcal{H},\boldsymbol{h})} \lambda^{|\mathcal{L}|} \leq \lambda^{|\mathcal{H}|+|\mathcal{L}|} \leq \lambda^k \leq p$, we can adjust the hidden constants to make sure $\sum_{(\mathcal{H},\boldsymbol{h})} p''_{\mathcal{H},\boldsymbol{h}} \leq p$.

Lemma 8.1. *$\mathcal{Q}''(\mathcal{H},\boldsymbol{h})$ can be answered with load $O(n/p^{2/(\alpha\phi)})$ using $p''_{\mathcal{H},\boldsymbol{h}}$ machines.*

Proof. *Join*$(\mathcal{Q}''(\mathcal{H},\boldsymbol{h}))$ is the cartesian product of $CP(\mathcal{Q}''_{\mathcal{I}}(\mathcal{H},\boldsymbol{h}))$ and *Join*$(\mathcal{Q}''_{light}(\mathcal{H},\boldsymbol{h}))$. If $\mathcal{I} \neq \emptyset$, use $p''_{\mathcal{H},\boldsymbol{h}}/\lambda^{|\mathcal{L}\setminus\mathcal{I}|}$ machines to compute $CP(\mathcal{Q}''_{\mathcal{I}}(\mathcal{H},\boldsymbol{h}))$. By Lemma 3.3, the load is

$$\tilde{O}\left(\frac{|CP(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H},\boldsymbol{h}))|^{1/|\mathcal{J}|}}{\left(\frac{p''_{\mathcal{H},\boldsymbol{h}}}{\lambda^{|\mathcal{L}\setminus\mathcal{I}|}}\right)^{1/|\mathcal{J}|}}\right) \quad (37)$$

for some non-empty $\mathcal{J} \subseteq \mathcal{I}$. (36) guarantees that

$$p''_{\mathcal{H},\boldsymbol{h}} = \Omega\left(p \cdot \frac{|CP(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H},\boldsymbol{h}))|}{\Theta(\lambda^{\alpha(\phi-|\mathcal{J}|)-|\mathcal{L}\setminus\mathcal{J}|} \cdot n^{|\mathcal{J}|})}\right)$$

with which we can derive

$$(37) = \tilde{O}\left(\frac{n \cdot \lambda^{\frac{\alpha(\phi-|\mathcal{J}|)-|\mathcal{I}|+|\mathcal{J}|}{|\mathcal{J}|}}}{p^{1/|\mathcal{J}|}}\right) = \tilde{O}\left(\frac{n \cdot p^{\frac{\alpha(\phi-|\mathcal{J}|)}{\alpha\phi|\mathcal{J}|}}}{p^{1/|\mathcal{J}|}}\right) = \tilde{O}\left(\frac{n}{p^{1/\phi}}\right)$$

which is $O(n/p^{2/(\alpha\phi)})$ because $\alpha \geq 2$. Regarding $\mathcal{Q}''_{light}(\mathcal{H},\boldsymbol{h})$, all of its relations are two-attribute skew free if a share of λ is assigned to each attribute in $\mathcal{L} \setminus \mathcal{I}$. By Lemma 3.5, $\mathcal{Q}''_{light}(\mathcal{H},\boldsymbol{h})$ can be solved with load $\tilde{O}(n/\lambda^2) = \tilde{O}(n/p^{2/(\alpha\phi)})$ using $\lambda^{|\mathcal{L}\setminus\mathcal{I}|}$ machines.

By combining the above with Lemma 3.4, we conclude that *Join*$(\mathcal{Q}''(\mathcal{H},\boldsymbol{h}))$ can be computed with load $\tilde{O}(n/p^{2/(\alpha\phi)})$ using $(p''_{\mathcal{H},\boldsymbol{h}}/\lambda^{|\mathcal{L}\setminus\mathcal{I}|}) \cdot \lambda^{|\mathcal{L}\setminus\mathcal{I}|} = p''_{\mathcal{H},\boldsymbol{h}}$ machines. □

Combining the above with Appendix G for handling queries with unary relations, we have established:

Theorem 8.2. *There exists an MPC algorithm that answers a query $\mathcal{Q}$ with load $\tilde{O}(n/p^{2/(\alpha\phi)})$, where n is the input size of $\mathcal{Q}$, p is the number of machines, α is given in (2), and ϕ is the generalized vertex-covering number of $\mathcal{Q}$.*

When $\alpha = 2$, the fractional edge covering number ρ of $\mathcal{Q}$ equals ϕ (Lemma 4.2); therefore, our algorithm achieves the optimal load $\tilde{O}(n/p^{1/\rho})$.

9 UNIFORM QUERIES

We can strengthen Theorem 8.2 when $\mathcal{Q}$ is α-uniform:

Theorem 9.1. *There exists an MPC algorithm that answers an α-uniform query with $\alpha \geq 2$ using load $\tilde{O}(n/p^{2/(\alpha\phi-\alpha+2)})$, where the meanings of n, p, α, and ϕ are the same as in Theorem 8.2.*

We will prove the theorem by adapting the algorithm in the previous section. The first change is to set λ higher:

$$\lambda \quad = \quad p^{\frac{1}{\alpha\phi-\alpha+2}}. \quad (38)$$

In Step 1, we set

$$p'_{\mathcal{H},\boldsymbol{h}} \quad = \quad p \cdot \frac{n_{\mathcal{H},\boldsymbol{h}}}{\Theta(n \cdot \lambda^{k-\alpha})}$$

Corollary 5.4 ensures that still at most p machines are necessary. The load becomes:

$$O\left(\frac{n_{\mathcal{H},\boldsymbol{h}}}{p'_{\mathcal{H},\boldsymbol{h}}}\right) = O\left(\frac{n \cdot \lambda^{k-\alpha}}{p}\right) \quad (39)$$

Proposition 9.2. *(39) is bounded by $\tilde{O}(n/p^{2/(\alpha\phi-\alpha+2)})$.*

Proof. With the λ in (38), we only need to prove:

$$\frac{k-\alpha+2}{\alpha\phi-\alpha+2} \quad \leq \quad 1$$

which is true because of (35). □

Step 2 requires no changes and entails a load of $O(\frac{n_{\mathcal{H},\boldsymbol{h}}}{p'_{\mathcal{H},\boldsymbol{h}}} + \frac{n}{p})$, which is bounded by (39). Step 3 also proceeds in the same manner as in Section 8, but Lemma 8.1 can be improved to:

Lemma 9.3. *We can answer $\mathcal{Q}''(\mathcal{H},\boldsymbol{h})$ in load $\tilde{O}(n/p^{2/(\alpha\phi-\alpha+2)})$ using $p''_{\mathcal{H},\boldsymbol{h}}$ machines, where $p''_{\mathcal{H},\boldsymbol{h}}$ is given in (36).*

Proof. If $|\mathcal{I}| > 0$, use $p''_{\mathcal{H},\boldsymbol{h}}/\lambda^{|\mathcal{L}\setminus\mathcal{I}|}$ machines to compute $CP(\mathcal{Q}''_{\mathcal{I}}(\mathcal{H},\boldsymbol{h}))$. The load is now

$$\tilde{O}\left(\frac{n \cdot \lambda^{\frac{\alpha(\phi-|\mathcal{J}|)}{|\mathcal{J}|}}}{p^{1/|\mathcal{J}|}}\right) = \tilde{O}\left(\frac{n \cdot p^{\frac{\alpha(\phi-|\mathcal{J}|)}{(\alpha\phi-\alpha+2)\cdot|\mathcal{J}|}}}{p^{1/|\mathcal{J}|}}\right) = \tilde{O}\left(\frac{n}{p^{\frac{1}{|\mathcal{J}|}-\frac{\alpha(\phi-|\mathcal{J}|)}{(\alpha\phi-\alpha+2)\cdot|\mathcal{J}|}}}\right)$$

for some non-empty $\mathcal{J} \subseteq \mathcal{I}$. To bound the above with $\tilde{O}(n/p^{2/(\alpha\phi-\alpha+2)})$, it suffices to show:

$$\begin{aligned} \frac{1}{|\mathcal{J}|} - \frac{\alpha(\phi-|\mathcal{J}|)}{(\alpha\phi-\alpha+2)\cdot|\mathcal{J}|} \quad &\geq \quad \frac{2}{\alpha\phi-\alpha+2} \\ \Leftrightarrow \alpha\phi-\alpha+2-\alpha(\phi-|\mathcal{J}|) \quad &\geq \quad 2|\mathcal{J}| \\ \Leftrightarrow (|\mathcal{J}|-1)(\alpha-2) \quad &\geq \quad 0 \end{aligned}$$

which is true.

The rest of the proof proceeds as in Lemma 8.1. Given $\lambda^{|\mathcal{L}\setminus\mathcal{I}|}$ machines, $\mathcal{Q}''_{light}(\mathcal{H},\boldsymbol{h})$ incurs load $\tilde{O}(n/\lambda^2) = \tilde{O}(n/p^{2/(\alpha\phi-\alpha+2)})$. Lemma 9.3 then follows from an application of Lemma 3.4. □

The proof of Theorem 9.1 is now complete. By combining the theorem with Lemma 4.3, we have:

Corollary 9.4. *There exists an MPC algorithm that answers a symmetric query $\mathcal{Q}$ using load $\tilde{O}(n/p^{2/(k-\alpha+2)})$, where the meanings of n, p, $\alpha \geq 2$ are the same as in Theorem 9.1, and $k = |attset(\mathcal{Q})|$.*

ACKNOWLEDGMENTS

Miao Qiao was patially supported by Marsden Fund UOA1732 from Royal Society of New Zealand, and the Catalyst: Strategic Fund NZ-Singapore Data Science Research Programme UOAX2001, from the Ministry of Business Innovation and Employment, New Zealand. Yufei Tao was partially supported by GRF grant 14207820 from HKRGC and a research grant from Alibaba Group.

REFERENCES

[1] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. 2009. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *Proceedings of the VLDB Endowment (PVLDB)* 2, 1 (2009), 922–933.

[2] Foto N. Afrati, Manas R. Joglekar, Christopher Ré, Semih Salihoglu, and Jeffrey D. Ullman. 2017. GYM: A Multiround Distributed Join Algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*. 4:1–4:18.

[3] Foto N. Afrati and Jeffrey D. Ullman. 2011. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 23, 9 (2011), 1282–1298.

[4] Albert Atserias, Martin Grohe, and Daniel Marx. 2013. Size Bounds and Query Plans for Relational Joins. *SIAM Journal of Computing* 42, 4 (2013), 1737–1767.

[5] Paul Beame, Paraschos Koutris, and Dan Suciu. 2014. Skew in parallel query processing. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 212–223.

[6] Paul Beame, Paraschos Koutris, and Dan Suciu. 2017. Communication Steps for Parallel Query Processing. *Journal of the ACM (JACM)* 64, 6 (2017), 40:1–40:58.

[7] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 137–150.

[8] Xiao Hu. 2021. Cover or Pack: New Upper and Lower Bounds for Massively Parallel Joins. *Accepted to appear in PODS* (2021).

[9] Xiaocheng Hu, Miao Qiao, and Yufei Tao. 2016. I/O-efficient join dependency testing, Loomis-Whitney join, and triangle enumeration. *Journal of Computer and System Sciences (JCSS)* 82, 8 (2016), 1300–1315.

[10] Xiao Hu and Ke Yi. 2019. Instance and Output Optimal Parallel Algorithms for Acyclic Joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 450–463.

[11] Xiao Hu, Ke Yi, and Yufei Tao. 2019. Output-Optimal Massively Parallel Algorithms for Similarity Joins. *ACM Transactions on Database Systems (TODS)* 44, 2 (2019), 6:1–6:36.

[12] Bas Ketsman and Dan Suciu. 2017. A Worst-Case Optimal Multi-Round Algorithm for Parallel Computation of Conjunctive Queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 417–428.

[13] Bas Ketsman, Dan Suciu, and Yufei Tao. 2020. A Near-Optimal Parallel Algorithm for Joining Binary Relations. *CoRR* abs/2011.14482 (2020).

[14] Paraschos Koutris, Paul Beame, and Dan Suciu. 2016. Worst-Case Optimal Algorithms for Parallel Query Processing. In *Proceedings of International Conference on Database Theory (ICDT)*. 8:1–8:18.

[15] Paraschos Koutris and Dan Suciu. 2011. Parallel evaluation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 223–234.

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

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

[18] Rasmus Pagh and Francesco Silvestri. 2014. The input/output complexity of triangle enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 224–233.

[19] Edward R. Scheinerman and Daniel H. Ullman. 1997. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Wiley, New York.

[20] Yufei Tao. 2020. A Simple Parallel Algorithm for Natural Joins on Binary Relations. In *Proceedings of International Conference on Database Theory (ICDT)*. 25:1–25:18.

[21] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*. 96–106.

APPENDIX

A PROOF OF LEMMA 3.5

Denote by **actdom** the set of values that appear in the relations of $\mathcal{Q}$. Let $\mathcal{R}$ be an arbitrary relation in $\mathcal{Q}$. Assume, without loss of generality, that $scheme(\mathcal{R}) = \{A_1, ..., A_r\}$ where $r = arity(\mathcal{R})$. Define p_i as the assigned share of A_i, for each $i \in [r]$. Choose independent and perfectly random hash functions $h_1, ..., h_r$ such that h_i maps **actdom** to $[p_i]$. Allocate each tuple $\boldsymbol{u} \in \mathcal{R}$ to the *bin* $(h_1(a_1), ..., h_r(a_r))$ where $(a_1, ..., a_r) = (\boldsymbol{u}(A_1), ..., \boldsymbol{u}(A_r))$.

Lemma A.1 (Theorem 3.2 of [6]). *If $\mathcal{R}$ is skew free, the probability that every bin is allocated $\tilde{O}(n/\prod_{i=1}^{r} p_i)$ tuples of $\mathcal{R}$ is at least $1-1/p^c$ where the constant c can be arbitrarily large.*

Now, consider $r \geq 2$ and that $\mathcal{R}$ is not skew free, but:

- $\mathcal{R}$ has at most n/p_1 tuples that agree on A_1;
- $\mathcal{R}$ has at most n/p_2 tuples that agree on A_2;
- $\mathcal{R}$ has at most $n/(p_1p_2)$ tuples that agree on A_1 and A_2 simultaneously.

Lemma A.2. *Subject to the above conditions, the probability that every bin is allocated $\tilde{O}(n/(p_1p_2))$ tuples of $\mathcal{R}$ is at least $1 - 1/p^c$ where the constant c can be arbitrarily large.*

Proof. Define $p'_1 = p_1$, $p'_2 = p_2$, and $p'_i = 1$ for all $i \in [3, r]$. Let us re-assign attribute A_i a share of p'_i for each $i \in [r]$. The stated conditions indicate that $\mathcal{R}$ is skew free under the shares $p'_1, ..., p'_r$. Allocate each tuple $(a_1, ..., a_r)$ in $\mathcal{R}$ to the *two-attribute bin* $(h_1(a_1), h_2(a_2))$. By Lemma A.1, with probability at least $1-1/p^c$, each two-attribute bin is allocated $\tilde{O}(n/(p_1p_2))$ tuples.

Lemma A.2 then follows from the fact that each bin $(x_1, x_2, ..., x_r)$ is allocated a subset of the tuples allocated to the two-attribute bin (x_1, x_2). □

Lemma A.2 implies:

Corollary A.3. *If $\mathcal{R}$ has arity $r \geq 2$ and is two-attribute free, the probability that every bin is allocated*

$$\tilde{O}\Big(\min_{\substack{i,j \in [r] \\ i \neq j}} \frac{n}{p_i p_j}\Big)$$

tuples of $\mathcal{R}$ is at least $1 - 1/p^c$ where the constant c can be arbitrarily large.

The BinHC algorithm answers $\mathcal{Q}$ as follows. For every $A \in attset(\mathcal{Q})$, it chooses an independent and perfectly random hash function h_A that maps **actdom** to $[p_A]$. A *bucket* is a function $\boldsymbol{b}$: $attset(\mathcal{Q}) \rightarrow [p]$ subject to the constraint that $\boldsymbol{b}(A) \in [p_A]$ for each $A \in attset(\mathcal{Q})$. Due to (5), the number of distinct buckets is at most p. Assigning a machine to each possible bucket, BinHC solves $\mathcal{Q}$ in two steps:

1. For every relation $\mathcal{R} \in \mathcal{Q}$, send each tuple $\boldsymbol{u} \in \mathcal{R}$ to every machine responsible for a bucket $\boldsymbol{b}$ satisfying the condition that $\boldsymbol{b}(A) = h_A(\boldsymbol{u}(A))$ for all $A \in scheme(\mathcal{R})$.
2. Each machine generates the maximum subset of $Join(\mathcal{Q})$ that can be produced from the data received.

By Corollary A.3 (for non-unary relations) and Lemma A.1 (for unary relations), the load is at most (8).

B PROOF OF LEMMA 5.2

It is obvious that the right hand side of (13) is a subset of the left hand side. Next, we will prove the opposite: any tuple $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q})$ must be produced by the right hand side.

Given $\boldsymbol{u}$, we construct a plan $\boldsymbol{P}$ and its corresponding $\mathcal{H}$ as follows:

```
1. $S_1 = \emptyset, S_2 = \emptyset$
2. $S = attset(\mathcal{Q})$
3. while $\exists X \in S$ such that $\boldsymbol{u}(X)$ is heavy do
4.     add $X$ to $S_1$, and remove $X$ from $S$
5. while $\exists$ distinct $Y, Z \in S$ s.t. $(\boldsymbol{u}(Y), \boldsymbol{u}(Z))$ is heavy do
6.     add $(Y, Z)$ to $S_2$ (assuming $Y \prec Z$), and remove $Y, Z$ from $S$
7. return $\boldsymbol{P} = (S_1, S_2)$ and $\mathcal{H} = scheme(\mathcal{Q}) \setminus S$
```

Set $\boldsymbol{h} = \boldsymbol{u}[\mathcal{H}]$. We will show that $\boldsymbol{u}[attset(\mathcal{Q}) \setminus \mathcal{H}] \in \mathit{Join}(\mathcal{Q}'(\mathcal{H}, \boldsymbol{h}))$, which will complete the proof.

Consider an arbitrary edge $e \in \mathcal{E}$ active on $\boldsymbol{P}$. As $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q})$, we know $\boldsymbol{u}[e] \in \mathcal{R}_e$. As before, set $e' = e \setminus \mathcal{H}$. To prove $\boldsymbol{u}[attset(\mathcal{Q}) \setminus \mathcal{H}] \in \mathit{Join}(\mathcal{Q}'(\mathcal{H}, \boldsymbol{h}))$, it suffices to show that $\boldsymbol{u}[e'] \in \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$. In turn, to prove $\boldsymbol{u}[e'] \in \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$, we must establish two facts:

- for any attribute $A \in e'$, $\boldsymbol{u}(A)$ is light;
- for any distinct attributes $A, B \in e'$, $(\boldsymbol{u}(A), \boldsymbol{u}(B))$ is light.

The first bullet holds because otherwise A would have been added to S_1 at Line 4. The second bullet also holds because otherwise (A, B) (assuming $A \prec B$) would have been added to S_2 at Line 6.

C PROOF OF PROPOSITION 6.1

Let $\boldsymbol{u}$ be a tuple output by $\mathcal{Q}'(\mathcal{H}, \boldsymbol{h})$. We will prove $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q}''(\mathcal{H}, \boldsymbol{h}))$. Consider an arbitrary orphaned vertex A. For each orphaning edge e of A, we must have $\boldsymbol{u}[e] \in \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$; therefore, $\boldsymbol{u}(A) \in \mathcal{R}''_A(\mathcal{H}, \boldsymbol{h})$. This further implies that, for every edge e of $\mathcal{G}$ such that $e \setminus \mathcal{H}$ is non-unary, $\boldsymbol{u}[e \setminus \mathcal{H}] \in \mathcal{R}''_e(\mathcal{H}, \boldsymbol{h})$. It thus follows that $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q}''(\mathcal{H}, \boldsymbol{h}))$.

Conversely, let $\boldsymbol{u}$ be a tuple output by $\mathcal{Q}''(\mathcal{H}, \boldsymbol{h})$. We will prove $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q}'(\mathcal{H}, \boldsymbol{h}))$. This means that, for every orphaned vertex A, we must have $\boldsymbol{u}(A) \in \mathcal{R}''_A(\mathcal{H}, \boldsymbol{h})$. Thus, for each orphaning edge e of A, it holds that $\boldsymbol{u}[e] \in \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$. Consider an arbitrary edge e of $\mathcal{G}$ such that $e \setminus \mathcal{H}$ is non-unary. Clearly, $\boldsymbol{u}[e \setminus \mathcal{H}]$ appears in $\mathcal{R}''_e(\mathcal{H}, \boldsymbol{h})$, which ensures $\boldsymbol{u}[e] \in \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$. It thus follows that $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q}'(\mathcal{H}, \boldsymbol{h}))$.

D PROOF OF LEMMA 7.3

Consider an edge $e \in \mathcal{E}$. As $|e| \leq \alpha$, we have

$$1 \quad \geq \quad \frac{|e|-1}{|e|} \frac{\alpha}{\alpha-1}. \qquad (40)$$

As $\{x_e \mid e \in \mathcal{E}\}$ is a feasible assignment for the characterizing program, $\sum_{e \in \mathcal{E}: A \in e} x_e \leq 1$ holds for every $A \in \mathcal{V}$. Hence:

$$\begin{aligned} k - |\mathcal{J}| &= \sum_{A \notin \mathcal{J}} 1 \\ &\geq \sum_{A \notin \mathcal{J}} \sum_{e \in \mathcal{E}: A \in e} x_e = \sum_{e \in \mathcal{E}} |e \setminus \mathcal{J}| \cdot x_e \\ &= \sum_{e \in \mathcal{E}: e \cap \mathcal{J} = \emptyset} |e| \cdot x_e + \sum_{e \in \mathcal{E}: e \cap \mathcal{J} \neq \emptyset} (|e|-1) \cdot x_e \end{aligned}$$

where the last equality used Property (1) of Lemma 7.2. With the above, we can derive

$$\begin{aligned} & k - |\mathcal{J}| + \frac{1}{\alpha-1} \sum_{e \in \mathcal{E}: e \cap \mathcal{J} \neq \emptyset} (|e|-1) \cdot x_e \\ \geq & \sum_{e \in \mathcal{E}: e \cap \mathcal{J} = \emptyset} x_e |e| + \sum_{e \in \mathcal{E}: e \cap \mathcal{J} \neq \emptyset} (|e|-1) x_e \left(1 + \frac{1}{\alpha-1}\right) \\ \geq & \sum_{e \in \mathcal{E}: e \cap \mathcal{J} = \emptyset} x_e |e| \cdot \frac{|e|-1}{|e|} \frac{\alpha}{\alpha-1} + \sum_{e \in \mathcal{E}: e \cap \mathcal{J} \neq \emptyset} x_e (|e|-1) \frac{\alpha}{\alpha-1} \\ & \text{(applied (40))} \\ = & \sum_{e \in \mathcal{E}} x_e (|e|-1) \cdot \frac{\alpha}{\alpha-1} = \overline{\phi} \cdot \frac{\alpha}{\alpha-1}. \end{aligned}$$

Multiplying both sides by $\alpha - 1$, we have:

$$\begin{aligned} (\alpha-1)(k - |\mathcal{J}|) + \sum_{e \in \mathcal{E}: e \cap \mathcal{J} \neq \emptyset} (|e|-1) \cdot x_e \quad & \geq \quad \overline{\phi} \alpha \\ & = \quad (k - \phi) \alpha \end{aligned}$$

where the last equality used Lemma 4.1. Therefore:

$$k\alpha - k - |\mathcal{J}|\alpha + |\mathcal{J}| + \sum_{e \in \mathcal{E}^*} (|e|-1) x_e \geq k\alpha - \phi\alpha.$$

Re-arranging the terms proves the lemma.

E PROOF OF PROPOSITION 7.5

Clearly:

$$\sum_{\substack{\text{full config.} \\ (\mathcal{H}, \boldsymbol{h}) \text{ of } \boldsymbol{P}}} \left| CP\left(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})\right) \right| = \sum_{\substack{\text{full config.} \\ (\mathcal{H}, \boldsymbol{h}) \text{ of } \boldsymbol{P}}} \left| CP\left(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})\right) \times \{\boldsymbol{h}\} \right|.$$

Thus, we only need to prove:

$$\bigcup_{\substack{\text{full config.} \\ (\mathcal{H}, \boldsymbol{h}) \text{ of } \boldsymbol{P}}} CP\left(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})\right) \times \{\boldsymbol{h}\} \subseteq CP(\mathcal{Q}_{heavy}) \bowtie \mathit{Join}(\mathcal{Q}^*). \quad (41)$$

Fix any $(\mathcal{H}, \boldsymbol{h})$ of $\boldsymbol{P}$. Let $\boldsymbol{u}$ be an arbitrary tuple in $CP(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})) \times \{\boldsymbol{h}\}$. We will show that $\boldsymbol{u} \in CP(\mathcal{Q}_{heavy}) \bowtie \mathit{Join}(\mathcal{Q}^*)$, which will complete the proof.

Consider attribute X_i for any $i \in [a]$. By definition of $\boldsymbol{h}$, $\boldsymbol{u}(X_i) = \boldsymbol{h}(X_i)$ must be heavy. Hence, $\boldsymbol{u}(X_i) \in \mathcal{S}_i$. Likewise, consider attributes Y_j and Z_j for any $j \in [b]$. By definition of $\boldsymbol{h}$, $(\boldsymbol{h}(Y_j), \boldsymbol{h}(Z_j))$ must be heavy while both $\boldsymbol{h}(Y_j)$ and $\boldsymbol{h}(Z_j))$ must be light. Therefore, $(\boldsymbol{u}(Y_j), \boldsymbol{u}(Z_j)) = (\boldsymbol{h}(Y_j), \boldsymbol{h}(Z_j)) \in \mathcal{D}_j$.

Consider an arbitrary edge $e \in \mathcal{E}^*$. Let A be the (only) isolated vertex in e (Property (1) of Lemma 7.2). Note that $e \setminus \{A\} \subseteq \mathcal{H}$ (Property (2) of Lemma 7.2), and that e is an orphaning edge of A (Section 6). The fact $\boldsymbol{u} \in CP(\mathcal{Q}''_{\mathcal{J}}(\mathcal{H}, \boldsymbol{h})) \times \{\boldsymbol{h}\}$ tells us $\boldsymbol{u}(A) \in \mathcal{R}''_A(\mathcal{H}, \boldsymbol{h})$ (Section 6). By (15), this indicates $\boldsymbol{u}[e] \in \mathcal{R}'_e(\mathcal{H}, \boldsymbol{h})$ (Section 5), and hence, $\boldsymbol{u}[e] \in \mathcal{R}_e$.

We have shown that $\boldsymbol{u}[scheme(\mathcal{R})] \in \mathcal{R}$ for every relation $\mathcal{R}$ involved on the right hand side of (41). It thus follows that $\boldsymbol{u} \in CP(\mathcal{Q}_{heavy}) \bowtie \mathit{Join}(\mathcal{Q}^*)$.

F PROOF OF PROPOSITION 7.10

It is obvious that $\mathit{Join}(\mathcal{Q}_{\mathit{final}}) \subseteq \mathit{CP}(\mathcal{Q}_{\mathit{heavy}}) \bowtie \mathit{Join}(\mathcal{Q}_{\ell})$. Next, we will prove that the opposite is also true. Let $\boldsymbol{u}$ be a tuple in $\mathit{CP}(\mathcal{Q}_{\mathit{heavy}}) \bowtie \mathit{Join}(\mathcal{Q}_{\ell})$. For each $A \in \mathcal{I}$, $\boldsymbol{u}(A)$ must appear in some input relation of $\mathcal{Q}$, which means $\boldsymbol{u}(A) \in \mathcal{U}_A$. It thus follows that $\boldsymbol{u}(A) \in \mathit{Join}(\mathcal{Q}_{\mathit{final}})$.

G QUERIES WITH UNARY RELATIONS

A unary relation $\mathcal{R} \in \mathcal{Q}$ can be of two types:

- **non-isolated:** $\mathcal{Q}$ contains another relation $\mathcal{R}'$ such that $\mathit{scheme}(\mathcal{R}) \subseteq \mathit{scheme}(\mathcal{R}')$;
- **isolated:** no such $\mathcal{R}'$ exists.

As shown in [11, 14], all the non-isolated unary relations can be eliminated with load $\tilde{O}(n/p)$. Henceforth, we will assume that all unary relations are isolated.

Let g be the number of (isolated) unary relations which are denoted as $\mathcal{R}_1, ..., \mathcal{R}_g$, respectively. Define $\mathcal{Q}' = \mathcal{Q} \setminus \{\mathcal{R}_1, ..., \mathcal{R}_g\}$; $\mathcal{Q}'$ is a query without isolated relations. We have:

$$\mathit{Join}(\mathcal{Q}) = \mathit{Join}(\mathcal{Q}') \times (\mathcal{R}_1 \times ... \times \mathcal{R}_g).$$

Let ϕ and ϕ' be the generalized vertex packing numbers of $\mathcal{Q}$ and $\mathcal{Q}'$, respectively. It is easy to verify by definition that

$$\phi = \phi' + g.$$

Given p_1 machines, our algorithm in Section 8 computes $\mathit{Join}(\mathcal{Q}')$ with load $\tilde{O}(n/p_1^{2/(\alpha\phi')})$. By Lemma 3.3, $\mathcal{R}_1 \times ... \times \mathcal{R}_g$ can be computed with load $O(n/p_2^{1/g})$ using p_2 machines. Setting

$$p_1 = p^{\phi'/\phi}$$
$$p_2 = p^{g/\phi}$$

we can apply Lemma 3.4 to obtain $\mathit{Join}(\mathcal{Q})$ with load

$$\tilde{O}\left(\frac{n}{\min\{p^{\frac{2}{\alpha\phi}}, p^{\frac{1}{\phi}}\}}\right)$$

using $p_1 p_2 = p$ machines. The above is bounded by $\tilde{O}(n/p^{2/(\alpha\phi)})$ because $\alpha \geq 2$.

H EDGE QUASI-PACKING NUMBER

Consider a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ without exposed vertices (Section 3.1). Given a subset $\mathcal{U}$ of $\mathcal{V}$, denote by $\mathcal{G}_{-\mathcal{U}}$ the graph obtained by removing $\mathcal{U}$ from $\mathcal{G}$, or formally: $\mathcal{G}_{-\mathcal{U}} = (\mathcal{V} \setminus \mathcal{U}, \mathcal{E}_{-\mathcal{U}})$ where $\mathcal{E}_{-\mathcal{U}} = \{e \setminus \mathcal{U} \mid e \in \mathcal{E} \text{ and } e \setminus \mathcal{U} \neq \emptyset\}$. The *edge quasi-packing number* $\psi(\mathcal{G})$ of $\mathcal{G}$ equals

$$\max_{\mathcal{U} \subseteq \mathcal{V}} \tau(\mathcal{G}_{-\mathcal{U}})$$

where $\tau(\mathcal{G}_{-\mathcal{U}})$ is the fractional edge-packing number of $\mathcal{G}_{-\mathcal{U}}$ (Section 3.1).

Example. Let $\mathcal{G}$ be the graph in Figure 1(a). Consider a set $\mathcal{U}$ which includes all attributes except D, G, and H. $\mathcal{G}_{\mathcal{U}}$ contains 8 unary edges: {A}, {B}, {C}, {E}, {F}, {I}, {J}, and {K}. Consider the fractional edge packing $\mathcal{W}$ that maps these edges to 1, and the other edges of $\mathcal{G}_{\mathcal{U}}$ to 0. $\mathcal{W}$ has a weight of 8. We can therefore conclude that $\psi(\mathcal{G}) \geq 8$. □

We now echo the claim in Section 1.3 that if $\mathcal{Q}$ is a k-choose-α join, then $\psi(\mathcal{Q}) \geq k - \alpha + 1$. Let $A_1, ..., A_k$ be the attributes in $\mathit{attset}(\mathcal{Q})$, and $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the hypergraph of $\mathcal{Q}$. Consider a set $\mathcal{U} = \{A_1, ..., A_{k-\alpha+1}\}$. For each $i \in [k-\alpha+1]$, $\mathcal{G}_{\mathcal{U}}$ contains a unary edge $\{A_i\}$ (shrunk from the edge $\{A_i, A_{k-\alpha+2}, A_{k-\alpha+3}, ..., A_k\}$ in $\mathcal{G}$). Thus, $\mathcal{G}_{\mathcal{U}}$ admits a fractional edge packing $\mathcal{W}$ that maps only the $k - \alpha + 1$ edges $\{A_1\}$, $\{A_2\}$, ..., $\{A_{k-\alpha+1}\}$ to 1, and the other edges to 0. It thus follows that $\psi(\mathcal{Q}) \geq \tau(\mathcal{G}_{\mathcal{U}}) = k - \alpha + 1$.