Cover or Pack: New Upper and Lower Bounds for Massively Parallel Joins*

Xiao Hu
Duke University
Durham, NC, USA
xh102@cs.duke.edu

ABSTRACT

This paper considers the worst-case complexity of multi-round join evaluation in the Massively Parallel Computation (MPC) model. Unlike the sequential RAM model, in which there is a unified optimal algorithm based on the AGM bound for all join queries, worst-case optimal algorithms have been achieved on a very restrictive class of joins in the MPC model. The only known lower bound is still derived from the AGM bound, in terms of the optimal fractional edge covering number of the query.

In this work, we make efforts towards bridging this gap. We design an instance-dependent algorithm for the class of α-acyclic join queries. In particular, when the maximum size of input relations is bounded, this complexity has a closed form in terms of the optimal fractional edge covering number of the query, which is worst-case optimal. Beyond acyclic joins, we surprisingly find that the optimal fractional edge covering number does not lead to a tight lower bound. More specifically, we prove for a class of cyclic joins a higher lower bound in terms of the optimal fractional edge packing number of the query, which is matched by existing algorithms, thus optimal. This new result displays a significant distinction for join evaluation, not only between acyclic and cyclic joins, but also between the fine-grained RAM and coarse-grained MPC model.

CCS CONCEPTS

• **Theory of computation → Massively parallel algorithms**; **Database query processing and optimization (theory)**.

KEYWORDS

query processing, massively parallel algorithms, worst-case optimal

ACM Reference Format:
Xiao Hu. 2021. Cover or Pack: New Upper and Lower Bounds for Massively Parallel Joins. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3452021.3458319

*This work has been supported in part by NSF awards IIS-18-14493 and CCF-20-07556.

PODS '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8381-3/21/06…$15.00
https://doi.org/10.1145/3452021.3458319

1 INTRODUCTION

Evaluating join queries is one of the central problems in relational databases, both in theory and practice. The worst-case complexity of join evaluation started to be unraveled, largely thanks to the work of Atserials, Grohe, and Marx [5], who gave a worst-case bound on the join size, known as AGM bound. More specifically, the maximum possible join size is always bounded by $O(N^{\rho^*})$, where N is the maximum size of input relations and ρ^* is the optimal fractional edge covering number of the join query, which is also tight with an instance outputting $\Theta(N^{\rho^*})$ join results. This then led to the worst-case optimal join algorithms [22, 26] in the RAM model, where all joins display a unified form of the worst-case time complexity of $O(N^{\rho^*})$. Ngo, Ré, and Rudra [23] presented a nice survey of these results, and also gave a simpler and unified proof for both the AGM bound and the running time of the algorithm.

Meanwhile, massively parallel algorithms have received much more attention in recent years, due to the rapid development of massively parallel systems such as MapReduce [10] and Spark [28]. Join evaluation in massively parallel computational model is quite different from the RAM model, where an efficient algorithm should make the best use of *data locality*, i.e., it should strive to send as many tuples that can be joined as possible to one machine so that it can produce their join results. Some intriguing questions then arise: Is there a unified worst-case optimal join algorithm in the massively parallel computational model? Is the worst-case optimal complexity only related to the optimal fractional edge covering number of join query? If not, what other query-dependent quantities? This work will answer these questions.

1.1 Join Query

A (natural) *join* is defined as a hypergraph $\mathcal{Q} = (\mathcal{V}, \mathcal{E})$, where the vertices $\mathcal{V} = \{x_1, \ldots, x_n\}$ model the *attributes* and the hyperedges $\mathcal{E} = \{e_1, \ldots, e_m\} \subseteq 2^{\mathcal{V}}$ model the *relations* [1]. Let $\mathrm{dom}(x)$ be the *domain* of attribute $x \in \mathcal{V}$. An *instance* of $\mathcal{Q}$ is a set of relations $\mathcal{R} = \{R(e) : e \in \mathcal{E}\}$, where $R(e)$ is a set of *tuples*, where each tuple is an assignment that assigns a value from $\mathrm{dom}(x)$ to x for every $x \in e$. We use $N = \max_{e \in \mathcal{E}} |R(e)|$ to denote the maximum size of input relations. The *join results* of $\mathcal{Q}$ on $\mathcal{R}$, denoted as $\mathcal{Q}(\mathcal{R})$, consist of all combinations of tuples, one from each $R(e)$, such that they share common values on their common attributes. We study the *data complexity* of join algorithms, i.e., assume that the query size, namely n and m, are constants. Hence, the total number of input tuples, denoted as *input size*, is always $O(N)$.

For a join query $\mathcal{Q} = (\mathcal{V}, \mathcal{E})$, two query-related quantities noted as *edge covering* and *edge packing* will be commonly used throughout this paper. Let f be a mapping from $\mathcal{E}$ to $[0, +\infty)$. Note that f

is a fractional edge covering if

$$\sum_{e \in \mathcal{E}: v \in e} f(e) \geq 1, \text{ for all } v \in \mathcal{V}$$

and a fractional edge packing if

$$\sum_{e \in \mathcal{E}: v \in e} f(e) \leq 1, \text{ for all } v \in \mathcal{V}$$

The quantity $\sum_e f(e)$ is noted as the *number* of f, where the optimal fractional edge covering is the one with minimum number, denoted as ρ^*, and the optimal fractional edge packing is the one with maximum number, denoted as τ^*. Generally, there is no clear relation between τ^* and ρ^*, except for some specific joins.

1.2 The model of computation

We use the Massively Parallel Computation (MPC) model [3, 4, 7, 8, 18–20], which has become the standard model of computation for studying massively parallel algorithms, especially for join algorithms. In the MPC model, data is initially distributed evenly over p servers with each server holding $O(\frac{N}{p})$ tuples. Computation proceeds in rounds. In each round, each server first sends messages to other servers, receives messages from other servers, and then does some local computation. The complexity of an MPC algorithm is measured by the number of rounds and the *load*, denoted as L, which is the maximum size of messages received by any server in any round. A *linear load* $L = O(\frac{N}{p})$ is the ideal case (since the initial load is already $\frac{N}{p}$), while if $L = O(N)$, all problems can be solved trivially in one round by simply sending all data to one server. Initial efforts were mostly spent on what can be done in a single round of computation [4, 7, 8, 19, 20], but recently, more interests have been given to multi-round (but still a constant) algorithms [3, 18, 19], since new main memory based systems, such as Spark and Flink, have much lower overhead per round than previous generations like Hadoop.

We confine ourselves to *tuple-based* algorithms, i.e., tuples are atomic elements that must be processed and communicated in their entirety. The only way to create a tuple is by making a copy, from either the original tuple or one of its copies. We say that an MPC algorithm computes the join query Q on instance $\mathcal{R}$ if the following is achieved: For any join result $(t_1, t_2, \ldots, t_m) \in Q(\mathcal{R})$ where $t_i \in R(e_i)$, $i = 1, 2, \ldots, m$, these m tuples (or their copies) must all be present on the same server at some point. Then the server will call a zero-cost function $emit(t_1, t_2, \ldots, t_m)$ to report the join result. Note that since we only consider constant-round algorithms, whether a server is allowed to keep the tuples it has received from previous rounds is irrelevant: if not, it can just keep sending all these tuples to itself over the rounds, increasing the load by a constant factor. All known join algorithms in the MPC model are tuple-based and obey these requirements. Our lower bounds are combinatorial in nature: we only count the number of tuples that must be communicated in order to emit all join results, while all other information can be communicated for free. The upper bounds include all messages, with a tuple and an integer of $O(\log N)$ bits both counted as 1 unit of communication.

Joins	One-round	Multi-round
α-acyclic	$\widetilde{O}\left(\frac{N}{p^{1/\psi^*}}\right)$ [19]	$O\left(\frac{N}{p^{1/\rho^*}}\right)$ is achieved for r-hierarchical join [16] and all α-acyclic joins [Theorem 16]
Cyclic		$\widetilde{O}\left(\frac{N}{p^{1/\rho^*}}\right)$ is achieved for binary-relation join [18, 19, 25] and the Loomis-Whitney join [19]
		$\Omega\left(\frac{N}{p^{1/\tau^*}}\right)$ for $\boxminus$-join [Theorem 17] and some degree-two joins [Theorem 18]

Table 1: Worst-case complexity of join evaluation in the MPC model. N is the maximum size of input relations. p is the number of servers. ψ^* is the optimal fractional edge quasi-packing number. ρ^* is the optimal fractional edge covering number. τ^* is the optimal fractional edge packing number. It is known that $\psi^* \geq \max\{\rho^*, \tau^*\}$ [19].

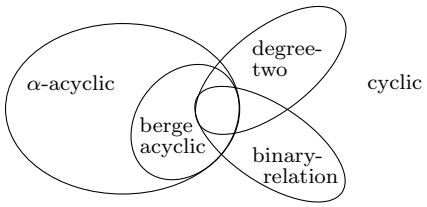

Figure 1: Classification of join queries.

1.3 Worst-case optimal join algorithms

In this work, we will focus on *worst-case optimality* for algorithm design, which is the most commonly-used measurement and provides theoretical guarantees in the worst case. More specifically, the entire space of input instances is divided into classes, where instances in the same class share the same input size N. An algorithm is *worst-case optimal* if its complexity is optimal on the worst instance for each class. Further subdividing the instance space leads to more refined analyses, for example, output-optimality takes both input size N and output size OUT as parameters to divide the instance space, and instance-optimality pushes this idea to the extreme case that each class contains just one instance. Note that by definition, an instance-optimal algorithm must be output-optimal, and an output-optimal algorithm must be worst-case optimal, but the reserve direction may not be true. We refer interested readers to [16] for the fine-grained join algorithms in the MPC model. All results with respect to the worst-case optimality in the MPC model are put into Table 1 and the relationships between join queries mentioned in this work are clarified in Figure 1.

Most of previous efforts have been put to understand what can be done in a single round in the MPC model. Initially, a one-round hashing-based algorithm [4, 7], named as *hypercube*, was proposed for computing all joins on *non-skewed* input instances with load $\widetilde{O}(\frac{N}{p^{1/\tau^*}})$[1]. Later, an improved algorithm built upon hypercube has

[1]The $\widetilde{O}$ notation suppresses polylog factors.

been proposed for tackling arbitrary input instances [19], but it incurs a higher load of $\widetilde{O}(\frac{N}{p^{1/\psi^*}})$, where ψ^* is the optimal fractional edge quasi-packing number[2] of the query. This result has been proved to be optimal (up to a polylog factor) for single-round computation with arbitrary input instances. Note that $\psi^* \geq \tau^*$ [19].

However, people found that even allowing a constant number of rounds may bring a significant (polynomially) reduction in the overall cost. Consider an example join query $Q = R_1(A) \bowtie R_2(A, B) \bowtie R_3(B)$, which has $\psi^* = \tau^* = 2$ by choosing R_1, R_3 in the fractional edge packing and $\rho^* = 1$ by choosing R_2 in the fractional edge cover. If targeting a single round, it can be computed with load $\widetilde{O}(\frac{N}{\sqrt{p}})$ [19]. However, if just allowing one more round, it can be computed through two steps of semi-joins with linear load $\widetilde{O}(\frac{N}{p})$ (see Section 2). The $\sqrt{p}$-gap can be further enlarged to $p^{\frac{n-1}{n}}$ on the *star-dual join* $Q = R_0(x_1, x_2, \cdots, x_n) \bowtie R_1(x_1) \bowtie R_2(x_2) \bowtie \cdots \bowtie R_n(x_n)$. This opens up new door for join evaluation in the MPC model.

The goal of a multi-round worst-case optimal algorithm in the MPC model is believed to achieve a load of $O(\frac{N}{p^{1/\rho^*}})$. The reason why $\Omega(\frac{N}{p^{1/\rho^*}})$ is a lower bound can be argued by the following counting argument: Each server can only produce $O(L^{\rho^*})$ join results in $O(1)$ rounds with its load limited to L (also implied by the AGM bound [5]), so all the p servers can produce $O(p \cdot L^{\rho^*})$ join results. Then, producing $\Theta(N^{\rho^*})$ join results requires $L = \Omega(\frac{N}{p^{1/\rho^*}})$. So far, this bound (up to polylog factors) has been achieved on some specific classes of joins [18, 19, 25], such as binary-relation join where each relation has at most two attributes, and Loomis-Whitney join[3]. All these algorithms resort to the heavy-light decomposition technique for tackling data skew, and then invoke the hypercube algorithm as primitives for handling non-skewed instances. However, whether this bound can be achieved for arbitrary joins, or even just α-acyclic joins, is still open.

Besides, several output-optimal (or output-sensitive) algorithms have been proposed for join queries in the MPC model. For example, an output-optimal algorithm has been proposed for r-hierarchical joins [16], which is also worst-case optimal. Note that r-hierarchical join is a very restrictive class of join queries; for example, the simplest line-3 join query $R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$ is not r-hierarchical. Meanwhile, the classical Yannakakis algorithm [27] can be easily parallelized for computing α-acyclic joins with load complexity $O(\frac{N}{p} + \frac{\text{OUT}}{p})$. Very recently, it has been improved to $O(\frac{N}{p} + \frac{\sqrt{N \cdot \text{OUT}}}{p})$ [16], which is output-optimal if $\text{OUT} = O(p \cdot N)$. However, in the worst-case when OUT approaches the AGM bound $\Theta(N^{\rho^*})$, this complexity would degenerate to $O(\frac{N^{(\rho^*+1)/2}}{p})$, which is rather far away from our target $\Omega(\frac{N}{p^{1/\rho^*}})$.

It is remarkable that a reduction from the MPC model to the *external memory* (EM) model has been established in [19] in a cost-preserving way, such that any MPC algorithm running in r rounds with load $L(N, p)$ can be converted to an EM algorithm incurring $\widetilde{O}(\frac{N}{B} + rp^* \frac{M}{B})$ I/Os[4], where $p^* = \min_p\{L(N, p) \leq M/r\}$. Implied by this reduction, worst-case optimal algorithms can be automatically obtained for LW join[5] and binary-relation join in the EM model. It is worth mentioning that a worst-case optimal algorithm has been proposed [15] for Berge-acyclic join in the EM model using $\widetilde{O}((\frac{N}{M})^{\rho^*} \cdot \frac{M}{B})$ I/Os, without a counterpart in the MPC model. However, Berge-acyclic join is a very restrictive sub-class of α-acyclic join; for example, a simple join query $R_0(A, B, C) \bowtie R_1(A, B, D) \bowtie R_2(B, C, E) \bowtie R_3(A, C, F)$ is α-acyclic but not Berge-cyclic, thus cannot be handled by the algorithm in [15]. On the other hand, there is no result showing any conversion from the sequential EM model to the parallel MPC model. We won't pursue this dimension further.

1.4 Our Results

Our main results are also summarized in Table 1, which can be split into two parts: new upper bound for α-acyclic joins and new lower bound for some cyclic joins. We also include a brief connection of these results from both sides.

New Upper Bound. The primary class of join queries we target in this work is the α-acyclic join [9], which is the most-commonly studied class of acyclic[6] joins in database theory. Formally, a join query $Q = (\mathcal{V}, \mathcal{E})$ is α-acyclic if there exists an undirected tree $\mathcal{T}$ whose nodes are in one-to-one correspondence with the edges in $\mathcal{E}$ such that for any vertex $v \in \mathcal{V}$, all nodes containing v form a connected subtree. Such a tree $\mathcal{T}$ is called the join tree of Q. An example of an acyclic[7] join is illustrated in Figure 4.

We propose a generic MPC algorithm for computing any acyclic join, whose load complexity is related to the choices made by this non-deterministic algorithm while running. We give a characterization of "good" choices for this algorithm, such that its load complexity can be bounded with $O(\frac{N}{p^{1/\rho^*}})$, achieving the worst-case optimality, as long as it always makes a good choice in each step.

This result has reduced the complexity for acyclic join evaluation from $O(\frac{N}{p^{1/\psi^*}})$ to $O(\frac{N}{p^{1/\rho^*}})$ since $\psi^* \geq \rho^*$ [19], only increasing the number of rounds from 1 to a constant. This improvement could be significant because of the possibly huge gap between ψ^* and ρ^*, as we have seen on the example in Section 1.3. In general, we notice several important sub-classes of acyclic joins on which this gap can be as large as $\Theta(m + n)$, in terms of the query size, including path join[8], star-dual join and some tree joins[9]. We refer interested reader to [19] for details. Moreover, by the MPC-EM reduction, this result automatically implies an EM algorithm for computing acyclic joins with $\widetilde{O}(\frac{N^{\rho^*}}{M^{\rho^*-1}B})$ I/Os, shadowing the previous work [12].

[2] For a join query $Q = (\mathcal{V}, \mathcal{E})$, the edge quasi-packing number is defined as follows. Let $x \subseteq \mathcal{V}$ be any subset of vertices of $\mathcal{V}$. Define the residual hypergraph after removing attributes x as $Q_x = (\mathcal{V}_x, \mathcal{E}_x)$, where $\mathcal{V}_x = \mathcal{V} - x$ and $\mathcal{E}_x = \{e - x : e \in E\}$. The edge quasi-packing number of Q is the maximum optimal fractional edge packing number over all Q_x's, i.e., $\psi^* = \max_{x \subseteq \mathcal{V}} \tau^*(Q_x)$.

[3] A join query $Q = (\mathcal{V}, \mathcal{E})$ is a Loomis-Whitney (LW) join if $\mathcal{E} = \{\mathcal{V} - \{x\} : \forall x \in \mathcal{V}\}$. Moreover, it has $\rho^* = \tau^* = n/(n-1)$, where $n = |\mathcal{V}|$. As it is a very restrict class of joins with highly symmetric structures, we omit it in the following discussion.

[4] The EM model has main memory of size M with disk block size B.

[5] A worst-case I/O-optimal algorithm for LW join was proposed in [13] independently.

[6] Other notions of acyclicity have been proposed, including berge-acyclicity, γ-acyclicity and β-acyclicity. Moreover, berge-acyclicity implies γ-acyclicity which implies β-acyclicity which implies α-acyclicity.

[7] In the following of this work, "acyclic" always means "α-acyclic" if not specified.

[8] A *path join* $Q = (\mathcal{V}, \mathcal{E})$ is defined as $\mathcal{V} = \{x_1, x_2, \cdots, x_n\}$ and $\mathcal{E} = \{e_i = \{x_i, x_{i+1}\} : i \in \{1, 2, \cdots, n-1\}\}$.

[9] A join query Q is a tree join if it is acyclic and each relation contains at most two relations. A tree join can be decomposed into a set of vertex-disjoint path joins.

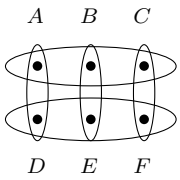

Figure 2: The hypergraph of ⊟-join.

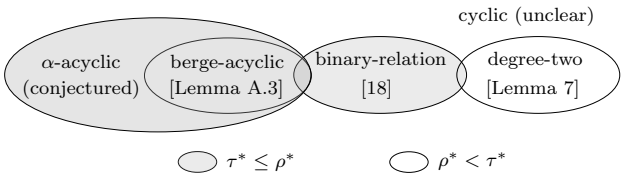

Figure 3: Relationship between optimal fractional edge covering number ρ^* and optimal fractional edge packing number τ^* of reduced join queries.

New Lower Bound. Finally, we turn to cyclic joins. Surprisingly, we find that $O(\frac{N}{p^{1/\rho^*}})$ is not necessarily a correct target for multi-round worst-case optimal join algorithms, since the existing lower bound $\Omega(\frac{N}{p^{1/\rho^*}})$ is not tight any more. We start by answering an open question posed in [18]: On the ⊟-join $Q_{\boxminus} = R_1(A,B,C) \bowtie R_2(D,E,F) \bowtie R_3(A,D) \bowtie R_4(B,E) \bowtie R_5(C,F)$, does there exist a better upper bound than $\widetilde{O}(\frac{N}{p^{1/3}})$, or a better lower bound than $\Omega(\frac{N}{p^{1/2}})$? As shown in Figure 2, $Q_{\boxminus}$ has $\rho^* = 2$ by choosing $\{R_1, R_2\}$ in the fractional edge cover and $\tau^* = 3$ by choosing $\{R_3, R_4, R_5\}$ in the fractional edge packing. We show a probabilistic hard instance on which any MPC algorithm computing it in $O(1)$ rounds must incur a load of $\Omega(\frac{N}{p^{1/3}})$. The intuition is that such an instance has "dense" join results, which is indeed as large as the AGM bound, but each server cannot achieve high efficiency in emitting the join results, no matter which combinations of input tuples it receives. Any attempts in lowering this bound further would break the counting argument that every join result must be emitted at least once, thus violating the correctness of join algorithms. Meanwhile, the existing algorithm [19] can compute it in a single round with load $\widetilde{O}(\frac{N}{p^{1/3}})$[10], which is already optimal implied by our new lower bound.

This framework of lower bound proof for $Q_{\boxminus}$ can be extended to a larger class of cyclic join queries, noted as *degree-two joins*, in which each attribute appears in two relations. Observe that the dual[11] of a degree-two join is a binary-relation join, hence enjoying very nice properties [24]. More specifically, we characterize the *edge-packing-provable condition*, under which there exists a better (at least not worse) lower bound in terms of $\Omega(\frac{N}{p^{1/\tau^*}})$ for computing any degree-two join in the MPC model.

Cover or Pack. From the lower bound side, we know that the worst-case complexity of join evaluation is $\Omega(\frac{N}{p^{1/\rho^*}})$ for binary-relation joins, LW join and acyclic joins, and $\Omega(\frac{N}{p^{1/\tau^*}})$ for some class of cyclic joins, including the ⊟-join. However, there is no clear distinction on the relative ordering between ρ^* and τ^*, at least for acyclic joins and binary-relation joins. A natural question arises: cover or pack, which one is the correct quantity in determining the worst-case complexity of join evaluation?

To clarify this question, we first introduce the notion of *reduced join*. A *reduce* procedure on a hypergraph $(\mathcal{V}, \mathcal{E})$ is to remove an edge $e \in \mathcal{E}$ if there exists another edge $e' \in \mathcal{E}$ such that $e \subseteq e'$. We can repeatedly apply the reduce procedure until no more edge can be removed, and the resulting hypergraph is said to be *reduced*.

From the upper bound side, a join query can be reduced in $O(1)$ rounds with linear load (see Section 2), thus the hardness of multi-round computation comes from the reduced join.

Now, we can draw some distinction for the reduced join queries: (1) $\tau^* \le \rho^*$ holds for reduced berge-acyclic joins (see Lemma A.3) and reduced binary-relation joins [18][12]; and (2) $\rho^* \le \tau^*$ holds for reduced degree-two joins (see Lemma 21), as shown in Figure 3. Moreover, we conjecture that $\tau^* \le \rho^*$ holds for reduced α-acyclic joins, but the formal proof is currently open. At last, we come to the following conjecture. The worst-case complexity of computing a join query Q in $O(1)$ rounds is conjectured to be $\Theta(\frac{N}{p^{1/\max\{\rho^*,\tau^*\}}})$, where ρ^*, τ^* are the optimal fractional edge covering and packing numbers of the reduced join of Q. This conjecture can be verified by existing results, but is still open for general cyclic joins.

1.5 Outline

This paper is organized as follows. In Section 2, we review some basic primitives that will be commonly used in our MPC algorithm. In Section 3 and 4, we present the new results from upper bound side. More specifically, we introduce a generic algorithm for acyclic joins in Section 3.1, analyze its complexity in Section 3.2, and identify the worst-case optimal run in Section 4. In Section 5, we move to the lower bound for cyclic joins. We first prove an edge-packing-based lower bound for the ⊟-join in Section 5.1, and then extend it to degree-two joins in Section 5.2.

2 MPC PRELIMINARIES

We first mention the following deterministic primitives in the MPC model, which can be computed with load $O(\frac{N}{p})$ in $O(1)$ rounds. Assume $N > p^{1+\epsilon}$ where $\epsilon > 0$ is any small constant.

Reduce-by-key [14]. Given N pairs in terms of (key, value), compute the "sum" of values for each key, where the "sum" is defined by any associative operator. This primitive will also be frequently used to compute data statistics, for example the degree. The *degree* of value $a \in \mathrm{dom}(x)$ in relation $R(e)$ is defined as the number of tuples in $R(e)$ having this value in attribute x, i.e., $|\sigma_{x=a}R(e)|$. Each tuple $t \in R(e)$ is considered to have "key" $\pi_x t$ and "value" 1.

Semi-Join [16]. Given two relations R_1 and R_2 with a common attribute v, the semi-join $R_1 \ltimes R_2$ returns all the tuples in R_1 whose value on v matches that of at least one tuple in R_2. For any acyclic join, all dangling tuples, i.e., those that will not participate in the full join results, can be removed by a series of semi-joins [27].

[10] The $Q_{\boxminus}$ has optimal fractional edge quasi-packing number $\psi^* = 3$.

[11] The dual of a join query $Q = (\mathcal{V}, \mathcal{E})$ is define as $Q' = (\mathcal{V}', \mathcal{E})$, where each vertex $v \in \mathcal{V}$ is an hyperedge in $\mathcal{E}'$ and each hyperedge $e \in \mathcal{E}$ in a vertex in $\mathcal{V}'$. Vertex $e \in \mathcal{V}'$ is included by edge $v \in \mathcal{E}'$ if $v \in e$ in Q.

[12] In [18], the term "simple" is used as equivalent to "reduced".

Parallel-packing [16]. Given N numbers $x_1, x_2, \cdots, x_N$ where $0 < x_i \leq 1$ for $i \in [N]$, group them into m sets $Y_1, Y_2, \cdots, Y_m$ such that $\sum_{i \in Y_j} x_i \leq 1$ for all j, and $\sum_{i \in Y_j} x_i \geq \frac{1}{2}$ for all but one j. Initially, the N numbers are distributed arbitrarily across all servers, and the algorithm should produce all pairs (i, j) if $i \in Y_j$ when done. Note that $m \leq 1 + 2\sum_i x_i$.

3 UPPER BOUND FOR ACYCLIC JOINS

In this section, we study how to compute the class of acyclic joins efficiently in the MPC model. Before dividing into the algorithmic details, we first define the following notions to simplify our description (notations used by the algorithm are in Table 2).

In a join query $Q = (\mathcal{V}, \mathcal{E})$, denote $\mathcal{E}^{\mathbf{x}} = \{e \in \mathcal{E} : \mathbf{x} \in e\}$ as the set of relations containing attribute $\mathbf{x} \in \mathcal{V}$. Recall that in an acyclic join, its relations can be organized into a join tree $\mathcal{T}$ such that for each attribute, the nodes containing this attribute form a connected subtree in $\mathcal{T}$. An example is illustrated in Figure 4. To be more general, $\mathcal{T}$ could be a forest consisting a set of node-disjoint connected subtrees, such that each one is a valid single join tree, and the union of nodes over all subtrees is exactly the set of relations $\mathcal{E}$. In this way, if there exists some relation $e \in \mathcal{E}$ which doesn't share any common attributes with other relations, then we just treat it as a single connected component of $\mathcal{T}$. In a join tree $\mathcal{T}$, an attribute is *unique* if it only appears in one node.

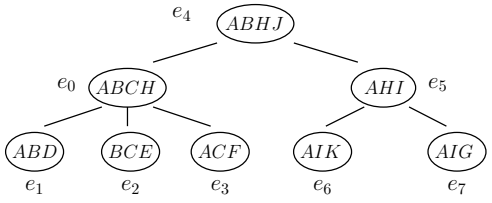

Figure 4: A join tree $\mathcal{T}$ of join query $Q = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{A, B, C, D, E, F, G, H, I, J, K\}$ and $\mathcal{E} = \{e_0(ABCH), e_1(ABD), e_2(BCE), e_3(ACF), e_4(ABHJ), e_5(AHI), e_6(AIK), e_7(AIG)\}$.

3.1 Generic Join Algorithm

We describe our generic algorithm for computing the result $Q(\mathcal{R})$ on the input join tree $\mathcal{T}$. The high-level idea is to recursively decompose the join into multiple subqueries based on data statistics, and apply a different join strategy for each subquery. For a clean presentation, we only focus on the algorithmic description now and delay its analysis to Section 3.2 and Section 4.2.

Our algorithm chooses a fixed threshold L, whose value will be determined later. Moreover, we introduce $\mathcal{S}(\mathcal{E}) \subseteq 2^{\mathcal{E}}$ as a set of subsets of relations, and a quantity $\Psi(\mathcal{T}, \mathcal{R}, S, L)$ for each subset of relations $S \subseteq \mathcal{E}$, which together determine the complexity of our algorithm. We give more details in Section 3.2 and Section 4.2. Note that $\mathcal{S}(\mathcal{E}) \subseteq 2^{\mathcal{E}}$ can be computed locally since the query has constant size. Intuitively, $\Psi(\mathcal{T}, \mathcal{R}, S, L)$ is the number of servers required for computing the join query induced by relations in S with load complexity $O(L)$.

Base Case. When there is only one relation in Q, say $\mathcal{E} = \{e\}$, we just let all servers emit tuples in $R(e)$ directly.

$Q(\mathcal{V}, \mathcal{E})$	join query Q with attributes $\mathcal{V}$ and relations $\mathcal{E}$
$\mathcal{R}$	instance
$Q(\mathcal{R})$	join results of $\mathcal{R}$ for query Q
e, e', e_i	relations
$R(e)$	relation defined over attributes e
$\mathbf{x}, A, B, C$	attributes
$\text{dom}(\mathbf{x})$	domain of attribute $\mathbf{x}$
$\mathcal{E}^{\mathbf{x}}$	the set of relations in $\mathcal{E}$ containing attribute $\mathbf{x}$
$S^{\mathbf{x}}$	a subset of relations containing attribute $\mathbf{x}$ defined by the generic join algorithm
$H(\mathbf{x}, S^{\mathbf{x}})$	a set of heavy values over attribute $\mathbf{x}$ from relation(s) in $S^{\mathbf{x}}$
$\mathcal{T}$	join tree of an acyclic join
$\mathcal{S}(\mathcal{E})$	a set of subsets of relations in $\mathcal{E}$
$Q_{\mathbf{x}}(\mathcal{V}_{\mathbf{x}}, \mathcal{E}_{\mathbf{x}})$	residual join after removing attribute $\mathbf{x}$
$Q_{\mathbf{y}}(\mathcal{V}_{\mathbf{y}}, \mathcal{E}_{\mathbf{y}})$	residual join after removing relations in $\mathcal{E}^{\mathbf{x}}$ and their unique attributes
$Q_i(\mathcal{V}_i, \mathcal{E}_i)$	the i-th connected subquery of $Q = (\mathcal{V}, \mathcal{E})$
$\mathcal{R}_{\mathbf{a}}$	instance induced by heavy value $\mathbf{a}$
$\mathcal{R}_{I_j}$	instance induced by light values in group I_j
$\mathcal{R}_i$	instance induced for $Q_i(\mathcal{V}_i, \mathcal{E}_i)$
N	input size of instance
p	number of servers
L	load complexity of an MPC algorithm

Table 2: Notations used in Section 3.1

General Cases. In general, we distinguish an input acyclic join Q with its join tree $\mathcal{T}$ into two cases.

Case I: $\mathcal{T}$ is a single join tree. We first remove dangling tuples. If there is a pair of nodes $e, e' \in \mathcal{E}$ such that $e \subseteq e'$, we just apply the semi-join $R(e') \ltimes R(e)$ and remove e from the join query. We recursively apply this procedure until the join is reduced.

On a reduced join, we start with an arbitrary leaf node e_1 in $\mathcal{T}$. Denote its parent as e_0. Let $\mathbf{x} \in e_1 \cap e_0$ be an arbitrary join attribute between e_1 and e_0. The algorithm chooses a subset of relations $S^{\mathbf{x}} \subseteq \mathcal{E}^{\mathbf{x}}$ with $e_1 \in S^{\mathbf{x}}$ to tackle in this case.

EXAMPLE 1. *Consider the reduced join query in Figure 4. Assume e_1 is the chosen leaf node, with e_0 as its parent. If $\mathbf{x} = A$, then $\mathcal{E}^{\mathbf{x}} = \{e_1, e_0, e_3, e_4, e_5, e_6, e_7\}$. If $\mathbf{x} = B$, then $\mathcal{E}^{\mathbf{x}} = \{e_1, e_2, e_0, e_4\}$.*

Step (1): Compute data statistics. For each value $\mathbf{a}$ over attribute $\mathbf{x}$, we compute its degree in every relation $R(e)$ for $e \in S^{\mathbf{x}}$, using the reduce-by-key primitive. A value $\mathbf{a}$ over attribute $\mathbf{x}$ is *heavy* if its degree in $R(e)$ is greater than L for any $e \in S^{\mathbf{x}}$, and *light* otherwise. Denote the set of heavy value as

$$H(\mathbf{x}, S^{\mathbf{x}}) = \{\mathbf{a} \in \text{dom}(\mathbf{x}) : \exists e \in S^{\mathbf{x}}, |\sigma_{\mathbf{x}=\mathbf{a}} R(e)| \geq L\}.$$

Note that $|H(\mathbf{x}, S^{\mathbf{x}})| \leq \sum_{e \in S^{\mathbf{x}}} \frac{|R(e)|}{L}$. Moreover, for all light values over attribute $\mathbf{x}$, we run the parallel-packing primitive to put them into $k = O(\sum_{e \in S^{\mathbf{x}}} \frac{|R(e)|}{L})$ groups $I_1, I_2, \cdots, I_k$, where the values in each group have a total degree of $O(L)$ in $\cup_{e \in S^{\mathbf{x}}} R(e)$.

Step (2): Decompose the join query. In this way, we can decompose the original join query into multiple subqueries:

$$Q_?(\mathcal{R}) = \bowtie_{e \in \mathcal{E}} \sigma_? R(e)$$

where ? is either $\mathbf{x} = \mathbf{a}$ for some $\mathbf{a} \in H(\mathbf{x}, S^{\mathbf{x}})$ or $\mathbf{x} \in I_j$ for some $j \in \{1, 2, \cdots, k\}$. There are $O(\sum_{e \in S^{\mathbf{x}}} \frac{|R(e)|}{L})$ subqueries in total.

We introduce a residual join query $Q_{\mathbf{x}} = (\mathcal{V}_{\mathbf{x}}, \mathcal{E}_{\mathbf{x}})$ by removing $\mathbf{x}$ from all relations, where $\mathcal{V}_{\mathbf{x}} = \mathcal{V} - \{\mathbf{x}\}$ and $\mathcal{E}_{\mathbf{x}} = \{e - \mathbf{x} : e \in \mathcal{E}\}$. Each heavy value $\mathbf{a} \in H(\mathbf{x}, S^{\mathbf{x}})$ derives an instance $\mathcal{R}_{\mathbf{a}} = \{\sigma_{\mathbf{x}=\mathbf{a}} R(e) : e \in \mathcal{E}\}$ for the subquery $Q_{\mathbf{x}}$. Similarly, each light group I_j derives an instance $\mathcal{R}_{I_j} = \{\sigma_{\mathbf{x} \in I_j} R(e) : e \in \mathcal{E}\}$ for the join query Q. Note that all subqueries have disjoint results and their union is exactly the result of original join, i.e.,

$$Q(\mathcal{R}) = \left(\cup_{\mathbf{a} \in H(\mathbf{x}, S^{\mathbf{x}})} Q_{\mathbf{x}}(\mathcal{R}_{\mathbf{a}})\right) \cup \left(\cup_{j \in \{1,2,\cdots,k\}} Q(\mathcal{R}_{I_j})\right)$$

thus, the completeness is guaranteed.

Step (3): Compute all subqueries in parallel. The next step is to allocate appropriate number of servers to each subquery and compute them in parallel. For each subquery $Q_{\mathbf{x}}$ with input instance $\mathcal{R}_{\mathbf{a}}$, we allocate $p_{\mathbf{a}} = \max_{S \subseteq \mathcal{S}(\mathcal{E}_{\mathbf{x}})} \lceil \Psi(\mathcal{T}, \mathcal{R}_{\mathbf{a}}, S, L) \rceil$ servers, and invoke the whole algorithm for computing $Q_{\mathbf{x}}(\mathcal{R}_{\mathbf{a}})$ recursively.

Let $\mathbf{y}$ be the set of unique attributes contained by any relation in $S^{\mathbf{x}}$. We introduce a residual query $Q_{\mathbf{y}} = (\mathcal{V}_{\mathbf{y}}, \mathcal{E}_{\mathbf{y}})$ by removing all attributes in $\mathbf{y}$ and relations in $S^{\mathbf{x}}$, where $\mathcal{V}_{\mathbf{y}} = \mathcal{V} - \mathbf{y}$ and $\mathcal{E}_{\mathbf{y}} = \mathcal{E} - S^{\mathbf{x}}$. Let $\mathcal{T}'$ be the resulting join tree by removing nodes in $S^{\mathbf{x}}$ from $\mathcal{T}$, which may contain multiple connected subtrees.

For each subquery Q with input instance $\mathcal{R}_{I_j}$, we allocate $p_j = \max_{S \subseteq \mathcal{S}(\mathcal{E}_{\mathbf{y}})} \left\lceil \Psi(\mathcal{T}', \mathcal{R}_{I_j}, S, L) \right\rceil$ servers. To compute $Q(\mathcal{R}_{I_j})$, we first broadcast all tuples in $\cup_{e \in S^{\mathbf{x}}} \sigma_{\mathbf{x} \in I_j} R(e)$ to the p_j servers and then compute $Q_{\mathbf{y}}(\mathcal{R}_{I_j})$ by running the whole algorithm recursively. At last, each server just emits the combination (t_1, t_2) for each join result $t_1 \in \bowtie_{e \in S^{\mathbf{x}}} \sigma_{\mathbf{x} \in I_j} R(e)$ and each join result $t_2 \in Q_{\mathbf{y}}(\mathcal{R}_{I_j})$ if they can be joined by local computation.

EXAMPLE 2. *Continue with the example in Figure 4. Assume e_1 is the leaf node with e_0 as its parent. If $\mathbf{x} = A$ and $S^{\mathbf{x}} = \{e_1, e_0\}$, Figure 5 shows the join trees for residual join queries in step (2).*

Case II: $\mathcal{T}$ consists of multiple connected subtrees. Let $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ be the connected subtrees in $\mathcal{T}$ and $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_k$ be the corresponding set of relations in each subtree. Define $\mathcal{R}_i = \{R(e) : e \in \mathcal{E}_i\}$, and $Q_i = (\mathcal{V}_i, \mathcal{E}_i)$, where $\mathcal{V}_i = \bigcup_{e \in \mathcal{E}_i} e$. In this case, it computes a Cartesian product $Q_1(\mathcal{R}_1) \times \cdots \times Q_k(\mathcal{R}_k)$, where computing each $Q_i(\mathcal{R}_i)$ is captured by Case I.

We arrange servers into a $p_1 \times p_2 \times \cdots \times p_k$ hypercube, where

$$p_i = \max_{S \in \mathcal{S}(\mathcal{E}_i)} \lceil \Psi(\mathcal{T}_i, \mathcal{R}_i, S, L) \rceil.$$

Each server is identified with coordinates $(c_1, c_2, \cdots, c_k)$, where $c_i \in [p_i]$. For every combination $c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_k$, the p_i servers with coordinates $(c_1, \cdots, c_{i-1}, *, c_{i+1}, \cdots, c_k)$ form a group to compute $Q_i(\mathcal{R}_i)$ (using the algorithm under Case I). Consider a particular server $(c_1, c_2, \ldots, c_k)$. It participates in k groups, one for each $Q_i(\mathcal{R}_i), i = 1, \ldots, k$. For each $Q_i(\mathcal{R}_i)$, it emits a subset of its join results, denoted $Q_i(\mathcal{R}_i, c_1 \ldots, c_k)$. Then the server computes the Cartesian product $Q_1(\mathcal{R}_1, c_1 \ldots, c_k) \times \cdots \times Q_k(\mathcal{R}_k, c_1 \ldots, c_k)$ locally and emit the join results if the participating tuples can be truly joined. Note that for each group of servers computing $Q_i(\mathcal{R}_i)$, the p_i servers in the group emit $Q_i(\mathcal{R}_i)$ with no redundancy, so there is no redundancy in emitting the join result.

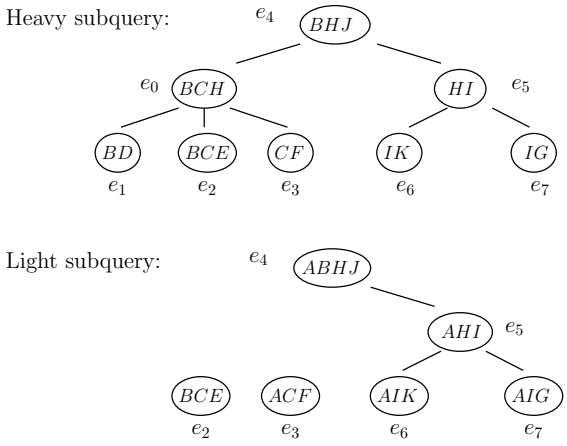

Figure 5: A running example of Case I. Assume $\mathbf{x} = A$ and $S^{\mathbf{x}} = \{e_1, e_0\}$. Then, $\mathbf{y} = \{D\}$. In step (2), $Q_{\mathbf{x}} = (\mathcal{V}_{\mathbf{x}}, \mathcal{E}_{\mathbf{x}})$ where $\mathcal{V}_{\mathbf{x}} = \{B, C, D, E, F, G, H, I, J, K\}$ and $\mathcal{E}_{\mathbf{x}} = \{e_0(BCH), e_1(BD), e_2(BCE), e_3(CF), e_4(BHJ), e_5(HI), e_6(IK), e_7(IG)\}$, and $Q_{\mathbf{y}} = (\mathcal{V}_{\mathbf{y}}, \mathcal{E}_{\mathbf{y}})$ where $\mathcal{V}_{\mathbf{y}} = \{A, B, C, E, F, G, H, I, J, K\}$ and $\mathcal{E}_{\mathbf{y}} = \{e_2(BCE), e_3(ACF), e_4(ABHJ), e_5(AHI), e_6(AIK), e_7(AIG)\}$.

3.2 Analysis

In this part, we analyze the complexity of the generic algorithm. To illustrate the key idea, we present a detailed analysis for one simple class of runs. The analysis for other runs follows the same framework, but users can choose more complicated functions of $\Psi(\mathcal{T}, \mathcal{R}, S, L)$ and $\mathcal{S}(\mathcal{E})$, as long as they satisfy the recurrence formulas implied by the generic algorithm.

As an example, we will focus on the most conservative run if the generic algorithm always chooses $S^{\mathbf{x}} = \{e_1\}$ in Case I. Before diving into the details, we introduce the notion of *subjoin* first.

DEFINITION 1 (SUBJOIN). *For a join query $Q = (\mathcal{V}, \mathcal{E})$ with join tree $\mathcal{T}$, and instance $\mathcal{R}$, the subjoin of a subset of relations $S \subseteq \mathcal{E}$ is*

$$\oplus(\mathcal{T}, \mathcal{R}, S) = \times_{S_i \in \mathcal{T}[S]} \bowtie_{e \in S_i} R(e)$$

where $\mathcal{T}[S]$ is the set of maximally connected components of S on $\mathcal{T}$.

EXAMPLE 3. *Let's take two examples in Figure 4 for illustration. $S_1 = \{e_1, e_3, e_7\}$ is a single connected component since all share the common attribute A; however, they are not directly connected on the join tree $\mathcal{T}$, so $\mathcal{T}[S_1] = \{\{e_1\}, \{e_3\}, \{e_7\}\}$. Adding one more edge e_0 to S_1 yields $S_2 = \{e_0, e_1, e_3, e_7\}$. Note that S_2 is still a single connected component, but e_0, e_1, e_3 form a connected component on the join tree $\mathcal{T}$, so $\mathcal{T}[S_2] = \{\{e_0, e_1, e_3\}, \{e_7\}\}$. The subjoin of S_1 is defined as $\oplus(\mathcal{T}, \mathcal{R}, S_1) = R(e_1) \times R(e_3) \times R(e_7)$ and that of S_2 is defined as $\oplus(\mathcal{T}, \mathcal{R}, S_2) = (R(e_0) \bowtie R(e_1) \bowtie R(e_3)) \times R(e_7)$.*

From Example 3, we can see the difference between subjoin of S and the join result of relations in S, or even the projection of final join results on attributes appearing in any relation of S, i.e.,

$$\pi_S Q(\mathcal{R}) \subseteq \bowtie_{e \in S} R(e) \subseteq \oplus(\mathcal{T}, \mathcal{R}, S)$$

Moreover, we mention an important observation for acyclic join as follows, which will be used in our analysis.

Lemma 2. *For an acyclic join $Q = (\mathcal{V}, \mathcal{E})$ with its join tree $\mathcal{T}$, consider an arbitrary leaf node e_1 and its parent e_0. For any $e \in \mathcal{E} - \{e_0, e_1\}$, $(e \cap e_1) - e_0 = \emptyset$.*

Proof. By contradiction, assume $v \in (e \cap e_1) - e_0$. Implied by the definition of join tree $\mathcal{T}$, all edges containing v form a connected subtree of $\mathcal{T}$. As e_1 is only connected to e_0, $v \in e_0$ if $v \in e \cap e_1$, coming to a contradiction. □

Theorem 3. *For a join query $Q = (\mathcal{V}, \mathcal{E})$ with join tree $\mathcal{T}$, an instance $\mathcal{R}$ and a parameter L, the $Q(\mathcal{R})$ can be computed using $O\left(\max_{S \subseteq \mathcal{S}(\mathcal{E})} \Psi(\mathcal{T}, \mathcal{R}, S, L)\right)$ servers in $O(1)$ rounds with load complexity $O(L)$, where $\Psi(\mathcal{T}, \mathcal{R}, S, L) = \frac{|\oplus(\mathcal{T}, \mathcal{R}, S)|}{L^{|S|}}$ and $\mathcal{S}(\mathcal{E}) = 2^{\mathcal{E}}$.*

Proof. We first prove the complexity of the generic algorithm in Theorem 3 by induction on the size of Q, and then show how to compute $\Psi(\mathcal{T}, \mathcal{R}, S, L)$ efficiently at last.

In the base case when there is only one relation, say $\mathcal{E} = \{e\}$, emitting all tuples in $R(e)$ with $\frac{|R(e)|}{L}$ servers achieves a load of $O(L)$, matching the bound in Theorem 3, since $\Psi(\mathcal{T}, \mathcal{R}, \{e\}, L) = \frac{|R(e)|}{L}$. In general, we first point out that all primitives can be computed using $O(\max_{e \in \mathcal{E}} \frac{|R(e)|}{L})$ servers in $O(1)$ rounds with load complexity $O(L)$. As $\frac{|R(e)|}{L} = \Psi(\mathcal{T}, \mathcal{R}, \{e\}, L)$, the complexity of these primitives can be bounded by Theorem 3. Next we will analyze the complexity for two cases separately.

Case I. Recall that the algorithm only peels a leaf e_1 from its parent node e_0, where $e_1 \cap e_0 \neq \emptyset$. In this case, a single join tree won't be broken except at e_1. We have the following hypotheses for handling the subqueries $Q_{\mathbf{x}}$ and $Q_{\mathbf{y}}$ separately, which will be used later.

Hypothesis 1. For a join query $Q_{\mathbf{x}}$ with join tree $\mathcal{T}$, an input instance $\mathcal{R}_{\mathbf{a}}$, and a pre-determined parameter L, the result can be computed using $O(\sum_{S \subseteq \mathcal{E}_{\mathbf{x}}} \Psi(\mathcal{T}, \mathcal{R}_{\mathbf{a}}, S, L))$ servers in $O(1)$ rounds with load complexity $O(L)$.

Hypothesis 2. For a join query $Q_{\mathbf{y}}$ with join tree $\mathcal{T}$, an instance $\mathcal{R}_{I_j}$ and a pre-determined parameter L, the result $Q_{\mathbf{y}}(\mathcal{R}_{I_j})$ can be computed using $O(\sum_{S \subseteq \mathcal{E}_{\mathbf{y}}} \Psi(\mathcal{T}, \mathcal{R}_{I_j}, S, L))$ servers in $O(1)$ rounds with load complexity $O(L)$.

Complexity of computing heavy subqueries. Implied by hypothesis 2, it remains to bound the number of servers allocated over all heavy values as follows:

$$\begin{aligned}\sum_{\mathbf{a} \in H(\mathbf{x}, \{e_1\})} p_{\mathbf{a}} &= \sum_{\mathbf{a} \in H(\mathbf{x}, \{e_1\})} \max_{S \subseteq \mathcal{E}_{\mathbf{x}}} \lceil \Psi(\mathcal{T}, \mathcal{R}_{\mathbf{a}}, S, L) \rceil \\ &\leq \sum_{S \subseteq \mathcal{E}_{\mathbf{x}}} \sum_{\mathbf{a} \in H(\mathbf{x}, \{e_1\})} \Psi(\mathcal{T}, \mathcal{R}_{\mathbf{a}}, S, L) \\ &\quad + 2^{|\mathcal{E}|} \cdot \Psi(\mathcal{T}, \mathcal{R}, \{e_1\}, L)\end{aligned}$$

where the second inequality is implied by the fact that each heavy value in $H(\mathbf{x}, \{e_1\})$ has degree more than L in relation $R(e_1)$. We distinguish each $S \subseteq \mathcal{E}_{\mathbf{x}}$ into two cases. If $e_0 \notin S$ and $e_1 \notin S$, the term induced on S can be bounded by

$$\begin{aligned}\sum_{\mathbf{a} \in H(\mathbf{x}, \{e_1\})} \Psi(\mathcal{T}, \mathcal{R}_{\mathbf{a}}, S, L) &\leq \Psi(\mathcal{T}, \mathcal{R}, \{e_1\}, L) \cdot \Psi(\mathcal{T}, \mathcal{R}, S, L) \\ &\leq \Psi(\mathcal{T}, \mathcal{R}, S \cup \{e_1\}, L)\end{aligned}$$

where the first inequality is implied by the fact that there are at most $O(\frac{|R(e_1)|}{L})$ heavy values and the second one is implied by the fact that e_1 forms a single connected component in $\mathcal{T}[S \cup \{e_1\}]$ from Lemma 2. Otherwise, $e_0 \notin S$ and $e_1 \notin S$. This term induced on S can be directly bounded by $\sum_{\mathbf{a}} \Psi(\mathcal{T}, \mathcal{R}_{\mathbf{a}}, S, L) \leq \Psi(\mathcal{T}, \mathcal{R}, S, L)$.

Complexity of computing light subqueries. In Step 3 of computing $Q(\mathcal{R}_{I_j})$, each server receives at most L input tuples from $\sigma_{\mathbf{x} \in I_j} R(e_1)$ and $O(L)$ tuples in computing $Q_{\mathbf{y}}(\mathcal{R}_{I_j})$ by hypothesis. So this step has a load of $O(L)$. It remains to bound that the total number of servers allocated to all light groups as follows:

$$\begin{aligned}\sum_j p_j &\leq \sum_j \sum_{S \subseteq \mathcal{E}_{\mathbf{y}}} \left\lceil \Psi(\mathcal{T}, \mathcal{R}_{I_j}, S, L) \right\rceil \\ &\leq 2^{|\mathcal{E}|} \cdot \Psi(\mathcal{T}, \mathcal{R}, \{e_1\}, L) + \sum_{S \subseteq \mathcal{E}_{\mathbf{y}}} \sum_j \Psi(\mathcal{T}, \mathcal{R}_{I_j}, S, L)\end{aligned}$$

Recall that $e_1 \notin \mathcal{E}_{\mathbf{y}}$ by definition. We distinguish each $S \subseteq \mathcal{E}_{\mathbf{y}}$ into two cases: $e_0 \in S$ and $e_0 \notin S$. If $e_0 \notin S$, this term induced on S can be bounded by

$$\begin{aligned}\sum_j \Psi(\mathcal{T}, \mathcal{R}_{I_j}, S, L) &\leq \Psi(\mathcal{T}, \mathcal{R}, \{e_1\}, L) \cdot \Psi(\mathcal{T}, \mathcal{R}, S, L) \\ &= \Psi(\mathcal{T}, \mathcal{R}, S \cup \{e_1\}, L)\end{aligned}$$

where the first inequality is implied by the fact that there are at most $O(\frac{|R(e_1)|}{L})$ light groups and the second one is implied by the fact that e_1 forms a single connected component in $\mathcal{T}[S \cup \{e_1\}]$ from Lemma 2. when $e_0 \notin S$. Otherwise, $e_0 \in S$. This term induced on S can be directly bounded by $\sum_j \Psi(\mathcal{T}, R_j, S, L) \leq \Psi(\mathcal{T}, \mathcal{R}, S, L)$.

Over all subqueries, the total number of servers allocated in total can be bounded by (big-Oh of)

$$\begin{aligned}&\sum_{S \subseteq \mathcal{E}_{\mathbf{x}} : e_1, e_0 \notin S} \Psi(\mathcal{T}, \mathcal{R}, S \cup \{e_1\}, L) + \sum_{S \subseteq \mathcal{E}_{\mathbf{x}} : e_1 \in S, \text{or } e_0 \in S} \Psi(\mathcal{T}, \mathcal{R}, S, L) \\ &+ \sum_{S \subseteq \mathcal{E}_{\mathbf{y}} : e_0 \notin S} \Psi(\mathcal{T}, \mathcal{R}, S \cup \{e_1\}, L) + \sum_{S \subseteq \mathcal{E}_{\mathbf{y}} : e_0 \in S} \Psi(\mathcal{T}, \mathcal{R}, S, L) \\ &\leq 4 \cdot \sum_{S \subseteq \mathcal{E}} \Psi(\mathcal{T}, \mathcal{R}, S, L)\end{aligned}$$

where the last inequality is implied by the following facts:

- $S \cup \{e_1\} \subseteq \mathcal{E}$ for each $S \subseteq \mathcal{E}_{\mathbf{x}}$ but $e_0, e_1 \notin S$;
- $S \cup \{e_1\} \subseteq \mathcal{E}$ for each $S \subseteq \mathcal{E}_{\mathbf{y}}$ but $e_0 \notin S$;
- $S \subseteq \mathcal{E}$ for each $S \subseteq \mathcal{E}_{\mathbf{x}}$ and $S \subseteq \mathcal{E}_{\mathbf{y}}$,

thus completing the induction proof for Case I.

Case II. Recall that the algorithm computes the Cartesian product of $Q_1(\mathcal{R}_1) \times \cdots \times Q_k(\mathcal{R}_k)$ over all connected subtrees of Q.

Hypothesis 3. For a join query Q_i with join tree $\mathcal{T}$, an input instance $\mathcal{R}_i$, and a pre-determined parameter L, the result $Q_i(\mathcal{R}_i)$ can be computed using $O(\sum_{S \subseteq \mathcal{E}_i} \Psi(\mathcal{T}, \mathcal{R}_i, S, L))$ servers in $O(1)$ rounds with load complexity $O(L)$.

In computing the Cartesian product, each server receives at most $O(L)$ tuples from each Q_i by hypothesis. So, each server has a load of $O(L)$. It remains to bound the total number of servers used in this step. Note that the number of servers allocated is (big-Oh of)

$$\prod_i p_i \leq \prod_i \sum_{S \subseteq \mathcal{E}_i} \lceil \Psi(\mathcal{T}_i, \mathcal{R}_i, S_i, L) \rceil$$

$$= \sum_{(S_1,S_2,\cdots,S_k)\subseteq \mathcal{E}_1\times\mathcal{E}_2\times\cdots\mathcal{E}_k} \prod_i \Psi(\mathcal{T}_i,\mathcal{R}_i,S_i,L)$$
$$\leq \sum_{S\subseteq \mathcal{E}_1\times\mathcal{E}_2\times\cdots\mathcal{E}_k} \Psi(\mathcal{T},\mathcal{R},S,L) \leq \sum_{S\subseteq\mathcal{E}} \Psi(\mathcal{T},\mathcal{R},S,L)$$

where the second last inequality is implied by the definition of $\oplus$ and the last inequality is implied by $\mathcal{E}_1 \times \mathcal{E}_2 \times \cdots \times \mathcal{E}_k \subseteq \mathcal{E}$, thus completing the induction proof for Case II.

At last, we show how to compute $\Psi(\mathcal{T},\mathcal{R},S,L)$ efficiently. Given the value of L, it boils down to computing a set of subjoins. In Case I, for each $S \subseteq \mathcal{E}_{e_1}$ or $S \subseteq \mathcal{E}_{\mathbf{y}}$, we use $O(\max_{S\in\mathcal{E}}\lceil\frac{|R(e)|}{L}\rceil)$ servers to compute $|\oplus(\mathcal{T},\mathcal{R}_{\mathbf{a}},S)|$'s over all heavy values or $|\oplus(\mathcal{T},\mathcal{R}_{I_j},S)|$'s over all light groups. Computing these statistics can be captured by a free-connex join-aggregate query[13]

$$\sum_{\mathcal{V}-\mathbf{x}} \times_{S_i\in\mathcal{T}[S]} \bowtie_{e\in S_i} R(e),$$

where each tuple has weight/annotation as 1. We invoke the algorithm in [16] to compute this query in $O(1)$ rounds, whose result is in forms of $(t, w(t))$ for each value $t \in \mathrm{dom}(\mathbf{x})$, with size bounded by $O(|R(e_1)|)$. If $\mathbf{a} \in H(\mathbf{x},\{e_1\})$, $|\oplus(\mathcal{T},\mathcal{R}_{\mathbf{a}},S)| = w(\mathbf{a})$; otherwise, we run the reduce-by-key primitive to compute $|\oplus(\mathcal{T},\mathcal{R}_{I_j},S)| = \sum_{\mathbf{a}\in I_j} w(\mathbf{a})$ for all light groups. As there are $O(1)$ subsets of relations, this step can be done in $O(1)$ rounds. In Case II, the values of p_i's can be computed similarly. Together, this step can be done using $O(\sum_{e\in\mathcal{E}}\Psi(\mathcal{T},\mathcal{R},\{e\},L))$ servers, thus completing the whole proof for Theorem 3. □

3.3 Choosing L

Theorem 3 displays a full trade-off between the number of servers available and the load complexity. A natural question arises, if we are only given p servers, what's the smallest load complexity that can be achieved for computing an acyclic join query in $O(1)$ rounds. We choose the value of L as below:

$$L = \max_{S\subseteq\mathcal{E}} \left(\frac{|\oplus(\mathcal{T},\mathcal{R},S)|}{p}\right)^{\frac{1}{|S|}},$$

where S is taken over all subsets of $\mathcal{E}$. It can be easily checked that for each $S \subseteq \mathcal{E}$, $\Psi(\mathcal{T},\mathcal{R},S,L) \leq p$ holds, thus this is feasible. Moreover, the value of L can also be computed through a join-aggregate query similarly using p servers in $O(1)$ rounds with load complexity $O(\sum_{e\in\mathcal{E}}\frac{|R(e)|}{p})$, which is also bounded by $O(L)$. Together, we come to the following result directly.

Theorem 4. *For an acyclic join query $\mathcal{Q} = (\mathcal{V},\mathcal{E})$ with a join tree $\mathcal{T}$, and an instance $\mathcal{R}$, the join result $\mathcal{Q}(\mathcal{R})$ can be computed using p servers in $O(1)$ rounds with load $O\left(\max_{S\subseteq\mathcal{E}}(\frac{|\oplus(\mathcal{T},\mathcal{R},S)|}{p})^{\frac{1}{|S|}}\right)$.*

So far we have obtained an acyclic join algorithm whose load complexity is in terms of a set of subjoins. However, we observe a gap between Theorem 4 and our target $O(\frac{N}{p^{1/\rho^*}})$. Next, we use two examples to discuss the reasons behind this intrinsic gap.

[13]The definitions of join-aggregate query and free-connex query are provided in Appendix A.3. In short, for join-aggregate query $\sum_{\mathbf{z}} Q$, if Q is acyclic and $\mathcal{V}-\mathbf{z}$ is contained by one relation, this query is free-connex.

Figure 6: A join tree of join query $Q = (\mathcal{V},\mathcal{E})$, where $\mathcal{V} = \{A,B,C,D,E,F,G,H\}$ and $\mathcal{E} = \{e_1(ABCD), e_2(BCDE), e_3(CDEF), e_4(DEFG), e_5(EFGH)\}$.

example 4. *Let's consider the example query in Figure 4, which has $\rho^* = 6$ by choosing $\{e_1,e_2,e_3,e_4,e_6,e_7\}$. Consider a hard instance constructed as below. There are N distinct values in the domain of attributes D,E,F,H,J,K,G and a single value in the domain of remaining attributes. Relation $R_4(ABHJ)$ is a one-to-one mapping over attributes H,J, and every remaining relation is a Cartesian product over its all attributes, containing N tuples in total. Also, this instance has its join size matching the AGM bound as $O(N^6)$. On $S_3 = \{e_1,e_2,e_3,e_0,e_5,e_6,e_7\}$, its subjoin has size as large as N^7 since $|R(e_1) \bowtie R(e_2) \bowtie R(e_3) \bowtie R(e_0)| = N^4$ and $|R(e_5) \bowtie R(e_6) \bowtie R(e_7)| = N^3$. Thus, our (conservative) generic algorithm computes this hard instance with load complexity $\Theta(\frac{N}{p^{1/7}})$, which is worse than the optimal target by a factor of $O(p^{7/6})$.*

Careful inspection reveals that not every subset $S \subseteq \mathcal{E}$ appears in the cost formula of Theorem 4, depending on which choices the algorithm makes while running. A key observation is that join query is always reduced before going into recursion. For example, after peeling e_1,e_2,e_3 off by choosing attribute $\mathbf{x} = A,B,C$ sequentially, relation e_0 could be reduced since $e_0 - e_1 \cup e_2 \cup e_3 \subseteq e_4$. In this way, e_0 won't appear together with any of e_5,e_6,e_7, and S_3 does not contributes to the cost formula of Theorem 3. Thus, this example implies that the gap partly comes from our non-tight analysis.

example 5. *Let's consider another join query in Figure 6. This query has $\rho^* = 2$ by choosing $\{e_1,e_5\}$. Consider a hard instance constructed as below. There are N distinct values in the domain of all attributes and each relation is a one-to-one mapping over its attributes. This instance only has $O(N)$ join results, but our (conservative) generic algorithm computes it with load complexity $\Theta(\frac{N}{p^{1/3}})$ since $\oplus(\mathcal{T},\mathcal{R},\{e_1,e_3,e_5\}) = N^3$, which is worse than our target by a factor of $O(p^{3/2})$. More specifically, this is tight when our (conservative) algorithm first (1) chooses e_1 as the leaf node with its parent e_2, and $\mathbf{x} = B$ with $S^{\mathbf{x}} = \{e_1\}$; and then (2) chooses e_5 as the leaf node with its parent e_4, and $\mathbf{x} = G$ with $S^{\mathbf{x}} = \{e_5\}$.*

This example motivates us to seek for more aggressive choices for this generic algorithm in the next section.

4 WORST-CASE OPTIMALITY FOR ACYCLIC JOINS

In this section, we take a further investigation of the generic algorithm presented in Section 3 and focus on the worst-case complexity. We identify a characterization of "good" choices that the algorithm can follow in each step and show that such a supervised run can achieve worst-case optimality on acyclic joins.

4.1 Characterization of A Good Run

Assume the input join query is reduced. Now, we describe how to tackle a reduced acyclic join with its join tree through two steps: (1)

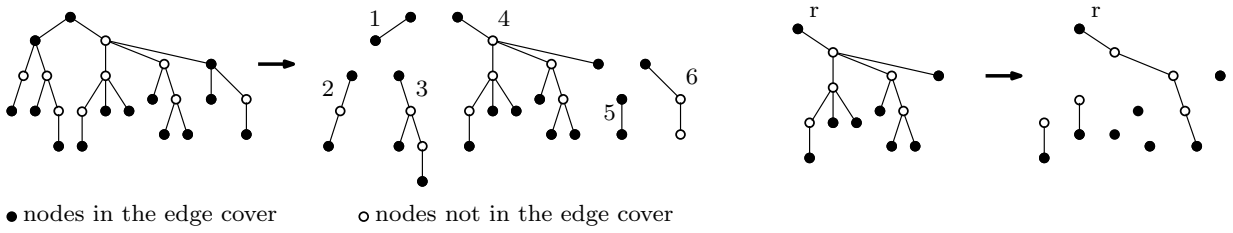

Figure 7: The left is a decomposition of a join tree $\mathcal{T}$ into 6 twigs and a possible valid ordering of these twigs is $(2, 3, 5, 6, 4, 1)$. The right is an example of linear covering of twig 4 rooted at r.

decompose the join tree into a set of subtree and define an ordering on these connected subtrees; (2) tackle subtrees one by one following the ordering defined in (1), and invoke the generic algorithm in Section 3 for each one, following our specified algorithmic choices. We next present each step in more detail.

Step 1: Decompose a join tree. We start with an important property on the optimal fractional edge covering for acyclic joins.

Lemma 5. *An acyclic join admits an integral solution for the optimal fractional edge covering number.*

The proof of Lemma 5 is left to Appendix A.1. Intuitively, given a join tree $\mathcal{T}$ for an acyclic join Q, such an optimal covering can be obtained through a greedy strategy by always choosing leaves (containing unique attributes) in $\mathcal{T}$. Denote $\rho : \mathcal{E} \to \{0, 1\}$ as the optimal edge covering for Q and $\mathcal{E}_\rho = \{e \in \mathcal{E} : \rho(e) = 1\}$ as the set of relations chosen by ρ. Moreover, $\rho^* = |\mathcal{E}_\rho|$. As the query has constant size, ρ as well as $\mathcal{E}_\rho$ can be computed locally. Based on ρ, we first identify an important subclass of acyclic joins as below.

Definition 6 (Twig Join). *a reduced join Q is a twig if it has a join tree $\mathcal{T}$ such that the set of leaves is an edge covering for Q.*

For an acyclic join Q with a join tree $\mathcal{T}$ and an optimal integral edge covering ρ, we next break the join tree at every internal node $e \in S_\rho$. This decomposes the join tree into a number of *twigs*, denoted as $\mathcal{G}$. Observe that in each twig, all the leaves are exactly the relations in S_ρ (see Figure 7). Moreover, each pair of twigs share at most one common node in $\mathcal{E}$, and they are *incident* if sharing exactly one node. Later, we will see that relations in the same twig will be handled as a whole. Moreover, twigs in $\mathcal{G}$ will be tackled following some specific ordering, defined as below.

Definition 7 (Ordering of Twigs in A Join Tree). *An ordering of a set of twigs $\mathcal{G}$ is defined as follows: (1) it always chooses a twig $\mathcal{G}_i$ if it is incident to at most one twig in $\mathcal{G} - \{\mathcal{G}_i\}$; (2) it removes $\mathcal{G}_i$ from $\mathcal{G}$ and apply this procedure recursively.*

Note that there could be multiple valid orderings. An example is given in Figure 7. The correctness of the recursion in Definition 7 is guaranteed by the underlying tree hierarchy across twigs in $\mathcal{G}$. We further define the root for each twig of $\mathcal{G}$ in this ordering.

Definition 8 (Root of A Twig). *For an acyclic join with the join tree $\mathcal{T}$ and a set of twigs $\mathcal{G}$:*

- *If $|\mathcal{G}| = 1$, $\mathcal{T}$ is a twig and an arbitrary leaf of $\mathcal{T}$ is the root;*
- *Otherwise, for each twig $\mathcal{G}_i \in \mathcal{G}$, its root is the unique non-leaf node in $\mathcal{E}_\rho$ when it is chosen by (1) in Definition 7.*

Step 2: Decompose a twig. Now, we focus on handling a single twig join Q with its join tree $\mathcal{T}$ inherited from last step, which enjoy very nice properties: (1) Q is reduced; (2) the set of leaves of $\mathcal{T}$ is a valid edge covering for Q, which is also optimal. Let r be the root of Q. Note that $r \in \mathcal{E}_\rho$ only has one child in $\mathcal{T}$; otherwise, this twig will be further decomposed, coming to a contradiction. Let c be the child of r. To abuse notations, we also use r, c to denote the sets of attributes contained by the corresponding relations separately.

We next show how the generic algorithm in Section 3 proceeds on a twig. As the join tree of a twig join is a single connected component, the algorithm goes into Case I directly. Then, the question comes: Which attribute should be chosen to tackle first?

We introduce the notion of *first attribute* in Definition 9. As Q is reduced, $c - r \neq \emptyset$. The algorithm chooses an arbitrary *first attribute* of $\mathcal{T}$ as $\mathbf{x}$ and a leaf node e containing $\mathbf{x}$. Note that such a leaf node always exists; otherwise, $\mathbf{x}$ will only appear in nodes of $\mathcal{T} - \mathcal{E}_\rho$, contradicting the fact that ρ is a valid edge covering. Let $\text{path}(c, e)$ be the set of nodes lying on the path from c to e. The algorithm then chooses $S^{\mathbf{x}} = \text{path}(c, e)$.

Definition 9 (First Attribute in A Twig). *In a twig join $\mathcal{T}$ rooted at r with the child c, the first attribute is defined as an arbitrary one in $c - r$.*

In step 2, the generic algorithm defines a set of heavy subqueries $Q_{\mathbf{x}}$ and light subqueries $Q_{\mathbf{y}}$. Note that $Q_{\mathbf{x}}$ is also a twig join rooted at r, thus can be handled by invoking the whole algorithm recursively. For $Q_{\mathbf{y}}$, the join tree will be decomposed into a set of subtrees after removing all nodes in $\text{path}(c, e)$. Consider an arbitrary node $e' \in \text{path}(c, e)$. For each children c' of e' but $c' \notin \text{path}(c, e)$, the subtree rooted at c' together with e' form a twig join, with its root as e', and can be handled by invoking the whole algorithm recursively. Computing the join results of these subtrees fall into Case II, which is done by enumerating the Cartesian product of their individual join results first and then emitting true join results after checking join conditions locally.

example 6. *Consider the example in Figure 8, which is an instantiation of twig 4 in Figure 7. The root is (EFG) with its child as (CEF). The first attribute is C, leaf node e is (ABC), and $\text{path}(c, e) = \{(ABC), (BCD), (CDE), (CEF)\}$. $Q_{\mathbf{x}}$ is the residual join by replacing (ABC) by (AB), (BCD) by (BD), (CDE) by (DE) and (CEF) by (EF) in $\mathcal{T}$, which is a twig join. $Q_{\mathbf{y}}$ is the residual join by removing (ABC), (BCD), (CDE) and (CEF) from $\mathcal{T}$. Computing $Q_{\mathbf{y}}$ degenerates to computing the Cartesian product of (1) $\{(EFG)\}$; (2) $\{(FM)\}$; (3) $\{(DU)\}$; (4) $\{(BK)\}$; and (5) $\{(FHJ), (FHZ), (WZ), (IJX), (HY)\}$. Note that*

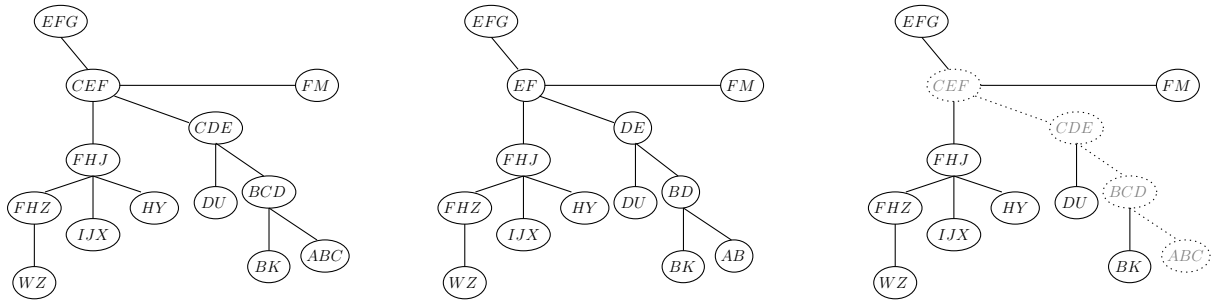

Figure 8: A running example of Step 2 on join query $Q = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{A, B, C, D, E, F, G, H, I, J, K, M, U, W, X, Y, Z\}$ and $\mathcal{E} = \{(ABC), (BCD), (CDE), (CEF), (EFG), (FHJ), (FM), (BK), (DU), (FHZ), (WZ), (IJX), (HY)\}$. The left is the join tree of Q (left), the middle is Q_x, and the right is Q_y.

(5) together with (CEF) is a twig join rooted at (CEF), which can be recursively decomposed.

4.2 Analysis

Next, we show the complexity for the generic algorithm following the choice in Section 4.1. Before diving into details, we first define

$$\Psi(\mathcal{T}, \mathcal{R}, S, L) = \prod_{e \in S} \frac{|R(e)|}{L}.$$

Obviously, the value of $\Psi(\mathcal{T}, \mathcal{R}, S, L)$ can be computed locally, since the query has constant size. As we tackle an acyclic join by decomposing it into a set of twig joins, we will start with the complexity for twig join and present the general result for acyclic join at last.

Complexity of Linear join. We first identify a special case of twig join, which only contains two leaves.

Definition 10 (Linear join). *A reduced join $Q = (\mathcal{V}, \mathcal{E})$ is linear if it has a join tree $\mathcal{T}$ with relations being arranged in a line staring at e_1 and ending at e_k, such that $e \in e_1 \cup e_k$ for any $e \in \mathcal{E} - \{e_1, e_k\}$.*

Moreover, it can be easily shown that $(e_i \cap e_k) \subseteq (e_{i+1} \cap e_k)$ and $(e_{i+1} \cap e_1) \subseteq (e_i \cap e_1)$ for any $i \in \{1, 2, \cdots, k-1\}$. Applying the rule in Definition 9 recursively will yield two valid orderings of attributes for a linear join, corresponding to the root being e_1 and e_k separately. Surprisingly, a linear join can be computed very efficiently following the first-attribute-based orderings, as stated in Lemma 11.

Lemma 11. *For a linear join $Q = (\mathcal{V}, \mathcal{E})$ with its join tree $\mathcal{T}$, an instance $\mathcal{R}$ and a parameter L, the join result $Q(\mathcal{R})$ can be computed using $O\left(\sum_{S \in \mathcal{S}(\mathcal{E})} \Psi(\mathcal{T}, \mathcal{R}, S, L)\right)$ servers in $O(1)$ rounds with load $O(L)$, where*

$$\mathcal{S}(\mathcal{E}) = \{\{e, r\} : e \in \mathcal{E} - \{r\}\} \cup \{\{e\} : e \in \mathcal{E}\}$$

for $r \in \{e_1, e_k\}$ and e_1, e_k being the two leaves in $\mathcal{T}$.

example 7. *An example of linear join $Q = (\mathcal{V}, \mathcal{E})$ is in Figure 6, where $\mathcal{V} = \{A, B, C, D, E, F, G, H\}$ and $\mathcal{E} = \{e_1(ABCD), e_2(BCDE), e_3(CDEF), e_4(DEFG), e_5(EFGH)\}$. Two orderings of attributes are D, C, B, A if root is e_5 and E, F, G, H if root is e_1. Let $N_i = |R(e_i)|$ for $i \in \{1, 2, 3, 4, 5\}$. If e_5 is the root, we have $S(\mathcal{E}) = \{\{e_1, e_5\}, \{e_2, e_5\}, \{e_3, e_5\}, \{e_4, e_5\}, \{e_1\}, \{e_2\}, \{e_3\}, \{e_4\}, \{e_5\}\}$. Implied by Lemma 11, it can be computed using $O(\frac{1}{L^2} \cdot N_5 \cdot (N_1 + N_2 + N_3 + N_4) + \frac{1}{L}(N_1 + N_2 + N_3 + N_4 + N_5))$ servers in $O(1)$ rounds with load $O(L)$.*

Complexity of Twig Join. Next, we move to a general twig join. To better describe the result, we introduce the notion of *linear cover* for a twig join first.

Definition 12 (Linear-covering of A Twig). *For a twig join Q with its join tree $\mathcal{T}$, a linear-covering for $\mathcal{T}$ denoted as $\mathcal{P}(\mathcal{T})$, is defined as follows:*

- *If $\mathcal{T}$ has at most two leaves (i.e., a linear join), $\mathcal{P}(\mathcal{T}) = \{\mathcal{T}\}$;*
- *Otherwise, $\mathcal{P}(\mathcal{T}) = \{\mathcal{L}\} \cup \left(\bigcup_{i=1}^{\ell} \mathcal{P}(\mathcal{T}_i)\right)$ where $\mathcal{L}$ is an arbitrary root-to-leaf path of $\mathcal{T}$ and $\{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_\ell\}$ is the set of connected subtrees by removing $\mathcal{L}$ from $\mathcal{T}$.*

An example of linear-covering of a twig join is shown in Figure 7. In Lemma 13, we use the linear cover to define the complexity of the first-attribute-based generic algorithm. Note that computing the cartesian product of node/relation-disjoint paths in the linear-covering has the similar complexity form as that in Lemma 13, but each path in the linear-covering may not be a linear join, thus may lead to much higher complexity. Our first-attribute-based generic algorithm is totally different from it.

Lemma 13. *For a twig join $Q = (\mathcal{V}, \mathcal{E})$ with its join tree $\mathcal{T}$ rooted at r, an instance $\mathcal{R}$ and a parameter L, the join result $Q(\mathcal{R})$ can be computed using $O\left(\sum_{\mathcal{P}(\mathcal{T})} \sum_{S \in \mathcal{S}(\mathcal{E})} \Psi(\mathcal{T}, \mathcal{R}, S, L)\right)$ servers in $O(1)$ rounds with load $O(L)$, where $\mathcal{P}(\mathcal{T})$ is over all linear coverings of $\mathcal{T}$, and*

$$\begin{aligned}\mathcal{S}(\mathcal{E}) = &\{\{S\} : S \subseteq (\mathcal{L}_1 - \{r\}) \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_\ell \times \{r\}\} \\ &\cup \{\{S\} : S \subseteq \{r\} \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_\ell\}\end{aligned}$$

for $\mathcal{P}(\mathcal{T}) = \{\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_\ell\}$.

example 8. *Continue the example of twig join query in Figure 8. One of its linear coverings is $\{\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4, \mathcal{L}_5, \mathcal{L}_6, \mathcal{L}_7\}$, where $\mathcal{L}_1 = \{(EFG), (CEF), (CDE), (BCD), (ABC)\}$, $\mathcal{L}_2 = \{(FHJ), (IJX)\}$, $\mathcal{L}_3 = \{(FHZ), (WZ)\}$, $\mathcal{L}_4 = \{(HY)\}$, $\mathcal{L}_5 = \{(DU)\}$, $\mathcal{L}_6 = \{(BK)\}$ and $\mathcal{L}_7 = \{(FM)\}$. By the definition of $\mathcal{S}(\mathcal{E})$, then $S = \{(ABC), (BK), (DU), (HY), (IJX), (WZ), (FM), (EFG)\}$ belongs to $\mathcal{S}(\mathcal{E})$, but any superset of S doesn't belongs to $\mathcal{S}(\mathcal{E})$.*

Complexity of Acyclic Join. By reassembling a set of twigs into the original join tree, we obtain the complexity result in Theorem 14.

Theorem 14. *For an acyclic join $Q = (\mathcal{V}, \mathcal{E})$ with join tree $\mathcal{T}$, an instance $\mathcal{R}$ and a parameter L, the result $Q(\mathcal{R})$ can be computed using $O\left(\sum_{S\in\mathcal{S}(\mathcal{E})} \Psi(\mathcal{T}, \mathcal{R}, S, L)\right)$ servers in $O(1)$ rounds with load $O(L)$, where*

$$\Psi(\mathcal{T}, \mathcal{R}, S, L) = \prod_{e\in S} \frac{|R(e)|}{L}$$

and $\mathcal{S}(\mathcal{E})$ is recursively defined on $\mathcal{T}$ as follows:

- **(1):** *If there is a pair of nodes e_1, e_0 such that $e_1 \subseteq e_0$, then $\mathcal{S}(\mathcal{E}) = \mathcal{S}(\mathcal{E}) \cup \{\{e_1\}\}$. It should be noted that $\mathcal{T}$ is updated by removing e_1 from $\mathcal{T}$ and putting every child of e_1 as a new child of e_1 if e_1 is not a leaf in $\mathcal{T}$.*
- **(2):** *$\mathcal{T}$ is decomposed into a set of twigs $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_\ell$ as defined in Definition 7,*

  $$\mathcal{S}(\mathcal{E}) = \{\{S\} : S \subseteq \mathcal{S}(\mathcal{E}_1) \times \mathcal{S}(\mathcal{E}_2) \times \cdots \times \mathcal{S}(\mathcal{E}_\ell)\}$$

  where $\mathcal{E}_i = \mathcal{G}_i - \mathcal{G}_j$ if there exists $j > i$ such that $\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset$ and $\mathcal{E}_i = \mathcal{G}_i$ otherwise.
- **(3):** *$\mathcal{T}$ is a twig join with a linear covering $\mathcal{P} = \{\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_\ell\}$. If $\mathcal{T}$ includes the root r,*

  $$\mathcal{S}(\mathcal{E}) = \{\{S\} : S \subseteq \mathcal{L}_1 \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_\ell\}$$

  Otherwise,

  $$\begin{aligned}\mathcal{S}(\mathcal{E}) =& \{\{S\} : S \subseteq (\mathcal{L}_1 - \{r\}) \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_\ell \times \{r\}\} \\ & \cup \{\{S\} : S \subseteq \{r\} \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_\ell\}.\end{aligned}$$

example 9. *With respect to rule (2) in Theorem 14, we use the example in Figure 7 for clarification. Assume the join tree is decomposed into 6 twigs following the ordering of $(\mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_5, \mathcal{G}_6, \mathcal{G}_4, \mathcal{G}_1)$. Under rule (2), twigs $\mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_5, \mathcal{G}_6, \mathcal{G}_4$ exclude their root and twig $\mathcal{G}_1$ includes its root in the computation, since $\mathcal{G}_2 \cap \mathcal{G}_1 \neq \emptyset$, $\mathcal{G}_3 \cap \mathcal{G}_1 \neq \emptyset$, $\mathcal{G}_4 \cap \mathcal{G}_1 \neq \emptyset$, $\mathcal{G}_5 \cap \mathcal{G}_4 \neq \emptyset$, $\mathcal{G}_6 \cap \mathcal{G}_4 \neq \emptyset$.*

The proof of Theorem 14, together with that of Lemma 11 and Lemma 13 are given in Appendix A.2.

4.3 Worst-case Optimality

We run the generic algorithm following choices according to Section 4.1, but using a different value of L defined as below:

$$L = \max_{S\in\mathcal{S}(\mathcal{E})} \left(\frac{\prod_{e\in S} |R(e)|}{p}\right)^{\frac{1}{|S|}}.$$

Since the join query has constant complexity, $\mathcal{S}(\mathcal{E})$ as well as the value of L can be computed locally.

Theorem 15. *For an α-acyclic join Q with a join tree $\mathcal{T}$, and an instance $\mathcal{R}$, the join result $Q(\mathcal{R})$ can be computed using p servers in $O(1)$ rounds with load $O\left(\sum_{S\in\mathcal{S}(\mathcal{E})} (\frac{\prod_{e\in S} |R(e)|}{p})^{\frac{1}{|S|}}\right)$.*

Moreover, when each relation has at most N input tuples, such a complicated bound has a clean form as stated in Theorem 16. A more fine-grained analysis of the optimality of Theorem 15 in terms of arbitrary input sizes $N(e)$'s would be an interesting and challenging open question. We won't go into this direction further.

Theorem 16. *For an α-acyclic join Q and an instance $\mathcal{R}$ where each relation contains at most N tuples, there is an algorithm computing $Q(\mathcal{R})$ in $O(1)$ rounds with load $O\left(\frac{N}{p^{1/\rho^*}}\right)$, where ρ^* is the optimal fractional edge covering number of Q, which is worst-optimal.*

Note that the optimality of Theorem 16 is shown on a hard instance implied by the AGM bound [5], where each relation contains $O(N)$ input tuples and the join size is as large as $\Theta(N^{\rho^*})$. Here we give a sketch proof of Theorem 16.

Proof of sketch. To prove $\Psi(\mathcal{T}, \mathcal{R}, S, L) \leq (\frac{N}{L})^{\rho^*}$ for any $S \in \mathcal{S}(\mathcal{E})$, it suffices to show that $|S| \leq \rho^*$ for any $S \in \mathcal{S}(\mathcal{E})$. Recall that the join tree $\mathcal{T}$ is decomposed into a set of twigs as $\mathcal{G}$. Let $(\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_\ell)$ be a valid ordering of twigs in $\mathcal{G}$. Observe that we also obtain a partition of S as $S_1, S_2, \cdots, S_\ell$ such that (1) $S_i = (\mathcal{G}_i \cap S) - \mathcal{G}_j$ if there exists $j > i$ such that $\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset$; (2) $S_i = \mathcal{G}_i \cap S$ otherwise. It can be easily checked that $S = S_1 \cup S_2 \cup \cdots \cup S_\ell$ and $S_i \cap S_j = \emptyset$ for any pair of $i \neq j$. Implied by the definition of $\mathcal{S}(\mathcal{E})$, the number of relations in S_i is at most the number of leaves in $\mathcal{G}_i$ minus 1, i.e., $|S_i| \leq |\mathcal{G}_i \cap \mathcal{E}_\rho| - 1$. In this way, we can bound the number of relations in S as

$$|S| = \sum_{i=1}^{\ell} |S_i| \leq \sum_{i=1}^{\ell} |\mathcal{G}_i \cap \mathcal{E}_\rho| - 1 = |\mathcal{E}_\rho| = \rho^*,$$

where the second last equality is implied by Definition 7. □

5 LOWER BOUNDS FOR CYCLIC JOINS

In this section, we first prove a lower bound $\Omega(\frac{N}{p^{1/3}})$ for the $Q_\boxminus$ query, as stated in Theorem 17, which is matched by the existing upper bound $O(\frac{N}{p^{1/3}})$ [19], thus being optimal. We then identify the class of degree-two joins, as well as the edge-packing-provable conditions, such that the edge-packing-dependent lower bound $\Omega(\frac{N}{p^{1/\tau^*}})$ can be proved for any degree-two join, as long as it satisfies the edge-packing-provable conditions, as stated in Theorem 18.

Theorem 17. *For any $N/\log^6 N \geq p^6$, there exists an instance $\mathcal{R}$ for $Q_\boxminus$ where each relation has N input tuples such that any tuple-based algorithm computing $Q_\boxminus(\mathcal{R})$ in $O(1)$ rounds must incur a load of $\Omega\left(\frac{N}{p^{1/3}}\right)$, which is optimal.*

Theorem 18. *On any edge-packing-provable degree-two join Q, for any $N/\log^c N \geq p^c$ with some constant c, there exists an instance $\mathcal{R}$ for Q where each relation has N input tuples such that any tuple-based algorithm computing $Q(\mathcal{R})$ in $O(1)$ rounds must incur a load of $\Omega\left(\frac{N}{p^{1/\tau^*}}\right)$, where τ^* is the optimal fractional edge packing number.*

The high-level idea of our lower bound proof is to show that with positive probability, an instance $\mathcal{R}$ for Q can be constructed with bounded $J(L)$, the maximum number of join results a server can produce, if it loads at most L tuples from each relation. We again resort to the counting argument that each join result must be emitted by at least one server. Setting $p \cdot J(L) = \Omega(|Q(\mathcal{R})|)$ yields a lower bound on L.

5.1 ⊟-Join

We now prove Theorem 17. Assume $N \geq p^3$. Note that $L \geq N/p \geq N^{2/3}$ in this case. Our hard instance $\mathcal{R}$ is constructed as follows.

Hard Instance. Each one of the attributes A, B, C has $N^{1/3}$ distinct values and each of the attributes D, E, F has $N^{2/3}$ distinct values. Relations $R_1(A, B, C)$, $R_3(A, D)$, $R_4(B, E)$ and $R_5(C, F)$ are Cartesian products, each with exactly N tuples. Relation $R_2(D, E, F)$ is constructed in a probabilistic way. For $R_2(D, E, F)$, each combination $(d, e, f) \in \text{dom}(D) \times \text{dom}(E) \times \text{dom}(F)$ has a probability of $1/N$ to form a tuple in $R_2(D, E, F)$. In this way, relation $R_2(D, E, F)$ have N tuples in expectation. The join result of this instance can be represented as the Cartesian product of $R_1(A, B, C) \times R_2(D, E, F)$. So, this instance has input size as $5N$ and output size as N^2 in expectation. By the Chernoff bound, the probability that the input size and output size deviate from their expectation by more than a constant factor is at most $\exp(-\Omega(N))$.

Step 1: Make a reasonable restriction on loading tuples.

To bound $J(L)$, we first argue that on any instance constructed as above, we can limit the choice of the L tuples loaded from $R_1(A, B, C)$, $R_3(A, D)$, $R_4(B, E)$ and $R_5(C, F)$ by any server in the form of $L_A \times L_B \times L_C$, $L_A \times L_D$, $L_B \times L_D$, and $L_C \times L_F$ for $L_A \subseteq \text{dom}(A), L_B \subseteq \text{dom}(B), L_C \subseteq \text{dom}(C), L_D \subseteq \text{dom}(D), L_E \subseteq \text{dom}(E)$ and $L_F \subseteq \text{dom}(F)$, i.e., the algorithm should load tuples from $R_1(A, B, C)$, $R_3(A, D)$, $R_4(B, E)$ and $R_5(C, F)$ in the form of Cartesian product. More precisely, we first prove this result for attribute A as stated in Lemma 19. The similar argument can be applied for attributes B and C.

Lemma 19. *Restricting loading tuples from $R_3(A, D)$ in a form of $L_A \times L_D$ and those from $R_1(A, B, C)$ in a form of $L_A \times L_{BC}$ where L_{BC} are the assignments over attributes B, C, will not make $J(L)$ smaller by more than a constant factor.*

Proof. Suppose a server has loaded L tuples from $R_2(D, E, F)$, $R_4(B, E)$ and $R_5(C, F)$. Then the server needs to decide which L tuples from $R_1(A, B, C)$ and $R_3(A, D)$ to load to maximize the number of join results produced. This is a combinatorial optimization problem that can be formulated as an integer program (IP). Introduce a variable x_{abc} for each triple $(a, b, c) \in \text{dom}(A) \times \text{dom}(B) \times \text{dom}(C)$, y_{ad} for each pair $(a, d) \in \text{dom}(A) \times \text{dom}(D)$. Let $I_{def} = 1$ if tuple $(d, e, f) \in R_2(D, E, F)$ is loaded by the server, and 0 otherwise. The similar definition applies for I_{be} and I_{cf}. Then IP_1 below defines this optimization problem, where a always ranges over $\text{dom}(A)$, b over $\text{dom}(B)$, c over $\text{dom}(C)$, d over $\text{dom}(D)$, e over $\text{dom}(E)$, f over $\text{dom}(F)$ unless specified otherwise.

$$
\begin{aligned}
(IP_1) \quad \max \quad & \sum_{a,b,c,d,e,f} I_{def} \cdot I_{be} \cdot I_{cf} \cdot x_{abc} \cdot y_{ad} \\
\text{s.t.} \quad & \max\{\sum_{abc} x_{abc}, \sum_{ad} y_{ad}\} \le L \\
& I_{def}, I_{be}, I_{cf}, x_{abc}, y_{ad} \in \{0, 1\}, \forall a, b, c, d, e, f
\end{aligned}
$$

However, it seems very difficult to dig out any structural property of the optimal solution of IP_1. Instead, we introduce a relaxed version of IP_1, shown as IP_3 below.

$$
\begin{aligned}
(IP_3) \quad \max \quad & \sum_{a} \Delta(w_a) \\
\text{s.t.} \quad & \sum_{a} w_a \le 2L \\
& w_a \in \{1, 2, \cdots, L\}, \forall a
\end{aligned}
$$

Note that IP_3 uses a function $\Delta(w)$, which denotes the optimal solution of IP_2 defined as below:

$$
\begin{aligned}
(IP_2) \quad \max \quad & \sum_{b,c,d,e,f} I_{def} \cdot I_{be} \cdot I_{cf} \cdot x_{abc} \cdot y_{ad} \\
\text{s.t.} \quad & \max\{\sum_{bc} x_{abc}, \sum_{d} y_{ad}\} \le w \\
& I_{def}, I_{be}, I_{cf}, x_{abc}, y_{ad} \in \{0, 1\}, \forall a, b, c, d, e, f
\end{aligned}
$$

IP_2 is parameterized by w and a, which finds the maximum number of join results that can be formed by tuples loaded from $R_2(D, E, F)$, $R_4(B, E)$ and $R_5(C, F)$, subject to the constraint that at most w tuples containing value a are loaded from $R_1(A, B, C)$ and $R_3(A, D)$.

Since all values in the domain of attribute A are structurally equivalent, the optimal solution of IP_2 does not depend on the particular choice of a, which is why we write the optimal solution of IP_2 as $\Delta(w)$. Also, it is obvious that $\Delta(.)$ is a non-decreasing function. Then, IP_3 tries to find the optimal allocation of the L tuples to different values $a \in \text{dom}(A)$ so as to maximize the total number of join results formed. Let the optimal solutions of IP_1, IP_3 be OPT_1, OPT_3, respectively. Because IP_3 only restricts the server to load at most $2L$ tuples from $R_1(A, B, C)$ and $R_3(A, D)$ in total, any feasible solution to IP_1 is also a feasible solution to IP_3, so $OPT_1 \le OPT_3$. Next we construct a feasible solution of IP_3 with the Cartesian product restriction above, and show that it is within a constant factor from OPT_3, hence OPT_1.

Regarding to the function $\Delta(.)$, we define

$$
w^* = \arg \max_{L/N^{1/3} \le w \le L} \frac{L}{w} \cdot \Delta(w).
$$

We choose $\frac{L}{w^*}$ distinct values arbitrarily from $\text{dom}(A)$ and allocate w^* tuples to each value $a \in A$. For each a, we use the optimal solution of IP_2 to find the w^* tuples to load from $R_1(A, B, C)$ and $R_3(A, D)$. Note that the optimal solution is the same for every a, so each a will choose the same sets of (b, c)'s and d's. Thus, this feasible solution loads tuples from $R_1(A, B, C)$ and $R_3(A, D)$ in the form of Cartesian products. The number of join results that can be produced is $W = \frac{L}{w^*} \cdot \Delta(w^*)$. We show that W is a constant-factor approximation of OPT_3, as stated in Lemma 20, thus completing the whole proof. □

Lemma 20. $W \ge \frac{1}{3} OPT_3 \ge \frac{1}{3} OPT_1$.

Proof. Suppose OPT_3 chooses a set of values $A^* \subseteq A$, and each $a \in A^*$ has w_a tuples loaded from $R_1(A, B, C)$ and $R_3(A, D)$. A value $a \in A^*$ is *efficient* if $\frac{\Delta(w_a)}{w_a} \ge \frac{\Delta(w^*)}{w^*}$, and *inefficient* otherwise. Let A_1^*, A_2^* be the set of efficient, inefficient values separately. Note that for every efficient value a, $w_a \le \frac{L}{N^{1/3}}$ by the definition of w^*.

We relate W and OPT_3 by showing how to cover all the join results reported by OPT_3 with the feasible solution constructed above. First, we use $\frac{\sum_{a \in A_2^*} w_a}{3w^*}$ values of A each with w^* tuples from $R_1(A, B, C)$ and $R_3(A, D)$ to cover the join results reported by A_2^*. The total number of tuples needed is at most $\frac{2}{3} \sum_{a \in A_2^*} w_a \le \frac{4}{3} L$. The number of join results that can be reported is

$$
\frac{\sum_{a \in A_2^*} w_a}{3w^*} \cdot \Delta(w^*) \ge \frac{1}{3} \sum_{a \in A_2^*} w_a \cdot \frac{\Delta(w_a)}{w_a} = \frac{1}{3} \sum_{a \in A_2^*} \Delta(w_a).
$$

Next, we use $\frac{L}{3w^*}$ values each with w^* tuples from $R_1(A,B,C)$ and $R_3(A,D)$ to cover the join results reported by A_1^*. The total number of tuples needed is $\frac{2}{3}L$. Recall that $w_a \le \frac{L}{N^{1/3}}$ for each $a \in A_1^*$. The number of join results that can be reported is

$$\frac{L}{3w^*}\cdot\Delta(w^*) \ge \frac{L}{3}\cdot\frac{\Delta(\frac{L}{N^{1/3}})}{\frac{L}{N^{1/3}}} = \frac{N^{1/3}}{3}\cdot\Delta(\frac{L}{N^{1/3}}) \ge \frac{1}{3}\sum_{a\in A_1^*}\Delta(w_a),$$

where the rationale behinds the last inequality is that there are at most $N^{1/3}$ values in A_1^* and there is $\Delta\left(\frac{L}{N^{1/3}}\right) \ge \Delta(w_a)$ for each $a \in A_1^*$ by the non-decreasing property of $\Delta(.)$.

Combining the two parts for the optimal solution A^*, our alternative solution loads at most $2L$ tuples from $R_1(A,B,C)$ and $R_3(A,D)$, and can report at least $\frac{1}{3}\cdot OPT_3$ join results. □

Note that Lemma 19 implies that $L_{AB} = L_A \times L_B$ and $L_{AC} = L_A \times L_C$ in relation $R_1(A,B,C)$. Applying a similar argument to attribute B, we get $L_{BC} = L_B \times L_C$. Together, we come to $L_{ABC} = L_A \times L_B \times L_C$, i.e., tuples in relation $R_1(A,B,C)$ should be loaded in form of Cartesian product over all attributes.

Step 2: Prove a upper bound on $J(L)$.

Next, we show that with positive probability (actually high probability), we obtain an instance on which $J(L)$ is bounded, no matter which L tuples are loaded. By the analysis above, we only need to consider the case where tuples from $R_1(A,B,C)$, $R_3(A,D)$, $R_4(B,E)$, $R_5(C,F)$ are loaded in the form of Cartesian products. Denote the number of distinct values in dom(A), dom(B) loaded by the server as α, β respectively. The number of distinct values in dom(C), dom(D), dom(E), dom(F) loaded by the server are $\frac{L}{\alpha\beta}, \frac{L}{\alpha}, \frac{L}{\beta}, \alpha\beta$. Moreover, $\frac{L}{N^{2/3}} \le \alpha, \beta \le N^{1/3}$.

There are L^3 combinations in terms of (a,b,c,d,e,f) in total. Each one is a valid join result if and only if $(d,e,f) \in R_2(D,E,F)$, which happens with probability $\frac{1}{N}$. By the linearity of expectation, the expected number of join results can be produced by the L tuples is $\frac{L^3}{N}$. More careful inspection reveals that the L^3 combinations are not independent; instead we can divide them into L^2 independent groups where each one is associated to one distinct triple (d,e,f). Implied by the Chernoff bound, the probability that this server produces more than $2\cdot\frac{L^3}{N}$ join results is at most $\exp(-\Omega(\frac{L^2}{N}))$.

For A, there are $\binom{N^{1/3}}{\alpha} = O(N^{\frac{\alpha}{3}})$ choices of loading α distinct values from dom(A). Similar argument can be applied to B,C,D,E,F. Over all values of α, β, the number of choices in total is

$$\sum_{\alpha=\frac{L}{N^{2/3}}}^{N^{1/3}}\sum_{\beta=\frac{L}{N^{2/3}}}^{N^{1/3}} N^{\frac{1}{3}\cdot(\alpha+\beta+\frac{L}{\alpha\beta})}\cdot N^{\frac{2}{3}\cdot(\frac{L}{\alpha}+\frac{L}{\beta}+\alpha\beta)}$$
$$= \exp\left(\widetilde{O}(\alpha\beta+\frac{L}{\alpha}+\frac{L}{\beta})\right) = \exp\left(\widetilde{O}(N^{\frac{2}{3}})\right)$$

By the union bound, the probability that any of the choice produces more than $\frac{2L^3}{N}$ join results is at most

$$\exp(-\Omega(\frac{L^2}{N}) + \widetilde{O}(N^{\frac{2}{3}})),$$

which is exponentially small if

$$\frac{L^2}{N} \ge c_1\cdot N^{\frac{2}{3}}\cdot\log N$$

for some sufficiently large constant c_1. Rearranging, we get

$$L^2 \ge c_1\cdot N^{\frac{5}{3}}\cdot\log N.$$

We know that $L = \Omega(\frac{N}{p})$, so this is true as long as

$$(\frac{N}{p})^2 \ge c_2\cdot N^{\frac{5}{3}}\cdot\log N,$$

for some sufficiently large constant c_2, or $N/\log^6 N \ge c_2\cdot p^6$.

Step 3: Apply counting argument.

So far, we have shown that with exponentially high probability each server produces no more than $2\cdot\frac{L^3}{N}$ join results in each round. Over p servers, the total number of join results that can be produced in $O(1)$ rounds is $O(\frac{L^3}{N})$. Each of the N^2 join results must be emitted at least once, so we will have $p\cdot\frac{L^3}{N} \ge N^2$, i.e., $L \ge N/p^{1/3}$.

We have completed the whole proof for Theorem 17.

5.2 Degree-two Joins

Our lower bound proof for the ⊟-join can be extended to a larger class of join queries, named as *degree-two join*, where every vertex appears in exactly two edges. As mentioned, degree-two joins enjoy several nice properties, as stated in Lemma 21. The proof of Lemma 21 is given in Appendix A.4.

Lemma 21. *For any reduced degree-two join $Q = (\mathcal{V},\mathcal{E})$, the following holds: (1) $\tau^* \ge \frac{|\mathcal{E}|}{2} \ge \rho^*$; (2) $\tau^* + \rho^* = |\mathcal{E}|$; (3) The optimal fractional edge packing/covering admits half-integral solution; (4) if there exists no odd-length cycle[14], the optimal fractional edge packing/covering admits an integral solution.*

However, not all degree-two joins fit for the lower bound framework, two additional conditions are captured in Definition 22. Before describing the conditions, we introduce some notions first. In a hypergraph $Q = (\mathcal{V},\mathcal{E})$, let $\Gamma(e)$ be the set of neighbors of edge $e \in \mathcal{E}$, i.e., $\Gamma(e) = \{e' \in \mathcal{E} : e \cap e' \ne \emptyset\}$. A *fractional vertex covering* for $Q = (\mathcal{V},\mathcal{E})$ is a mapping x from $\mathcal{V}$ $[0,+\infty)$ such that $\sum_{v\in\mathcal{V}:v\in e} x_v \ge 1$ holds for each edge $e \in \mathcal{E}$; and the optimal solution is to minimize the quantity $\sum_{v\in\mathcal{V}} x_v$. In addition, a vertex covering x is *constant-small* if $\max_v x_v \le 1-\epsilon$ for some constant $0 < \epsilon < 1$.

Definition 22 (Edge-packing-provable Degree Two Join). *A degree-two join $Q = (\mathcal{V},\mathcal{E})$ is* edge-packing-provable *if (1) it is reduced; (2) there is no odd-length cycle; (3) there exists an optimal fractional constant-small vertex covering x, such that $|\Gamma(e)\cap\mathcal{E}'| \le 1$ for every $e \in \mathcal{E}$, where $\mathcal{E}' = \{e \in \mathcal{E} : \sum_{v:v\in e} x_v > 1\}$.*

Note that $Q_{\boxminus}$ is an edge-packing-provable degree-two join. Obviously, $Q_{\boxminus}$ is reduced, there is no odd-length cycle in $Q_{\boxminus}$, and a valid vertex covering x with $x_A = x_B = x_C = \frac{1}{3}$ and $x_D = x_E = x_F = \frac{2}{3}$ is constant-small, which is also used in the lower bound proof of Theorem 17. Some other examples of edge-packing-provable degree-two joins are given in Figure 9.

[14] A cycle $(\mathcal{V},\mathcal{E})$ is defined as $\mathcal{V} = \{v_1, v_2, \cdots, v_n\} \subseteq \mathcal{V}$ and $\mathcal{E} = \{e_i = \{v_i, v_{(i+1) \bmod n}\} : i \in \{1,2,\cdot,n\}$. The length of a cycle $(\mathcal{V},\mathcal{E})$ is defined as $|\mathcal{E}|$.

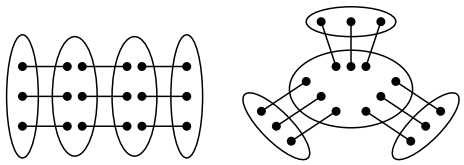

Figure 9: Examples of edge-packing-provable joins.

The detailed proof of Theorem 18 is deferred to Appendix A.5. Here, we only give some intuition why the three conditions can be put together for generalizing this framework to degree-two joins.

In Definition 22, if there is no odd-length cycle in Q, it admits integral optimal edge packing τ^* and covering ρ^*, implied by Lemma 21. More specifically, there exists a partition $(\mathcal{E}_\alpha, \mathcal{E}_\beta)$ of $\mathcal{E}$: $\mathcal{E}_\alpha = \{e \in \mathcal{E} : \rho^*(e) = 1, \tau^*(e) = 0\}$ and $\mathcal{E}_\beta = \{e \in \mathcal{E} : \rho^*(e) = 0, \tau^*(e) = 1\}$, for example, $\mathcal{E}_\alpha = \{e_1, e_2\}$ and $\mathcal{E}_\beta = \{e_3, e_4, e_5\}$ in $Q_\boxminus$. Moreover, all edges in $\mathcal{E}_\alpha$ are vertex-disjoint, as well as edges in $\mathcal{E}_\beta$, due to the fact that each vertex appears in at most two edges.

Consider any optimal fractional vertex covering x satisfying (2) in Definition 22. Note that x defines a partition $(\mathcal{E}', \mathcal{E}'')$ of $\mathcal{E}$:

$$\mathcal{E}' = \{e \in \mathcal{E} : \sum_{v \in \mathcal{V}: v \in e} x_v > 1\},$$
$$\mathcal{E}'' = \{e \in \mathcal{E} : \sum_{v \in \mathcal{V}: v \in e} x_v = 1\}$$

for example, $\mathcal{E}' = \{e_2\}$ and $\mathcal{E}'' = \{e_1, e_3, e_4, e_5\}$ in $Q_\boxminus$. Note that the fractional edge packing and vertex covering are prime-dual. Implied by the slackness theorem, $\mathcal{E}' \subseteq \mathcal{E}_\alpha$. Edges in $\mathcal{E}'$ are also vertex-disjoint. The hard instance is constructed by x. More specifically, the domain of each attribute v contains N^{x_v} distinct values. Relations in $\mathcal{E}''$ are deterministically constructed as Cartesian products, containing N tuples exactly, while those in $\mathcal{E}'$ are probabilistically constructed. As $|\Gamma(e) \cap \mathcal{E}'| \leq 1$ holds for every $e \in \mathcal{E}$, each edge in $e \in \mathcal{E}'$ derives a connected components $C(e) = \{e' \in \mathcal{E}'' : e' \cap e \neq \emptyset\}$. More importantly, $C(e_1) \cap C(e_2) = \emptyset$ for any pair of $e_1, e_2 \in \mathcal{E}'$. We then apply a similar argument for $Q_\boxminus$ to each such component.

At last, the notion of "constant-small" is used to prove a upper bound on $J(L)$ with exponentially high probability; and more details can be found in Appendix A.5.

Remark. We only give a sufficient condition in Theorem 18. Several questions remain to be answered, for example, (1) what is a complete characterization of cyclic queries on which our framework can be applied? and (2) is there any matching upper bound on the degree-two joins? It is still unclear whether $\psi^* = \tau^*$ holds for a degree-two join satisfying the edge-packing-provable conditions. If this is the case, then the lower bound $\Omega\left(\frac{N}{p^{1/\tau^*}}\right)$ will be matched by the existing one-round algorithm [19].

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases: the logical level.* Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] M. Abo Khamis, H. Q. Ngo, and A. Rudra. Faq: questions asked frequently. In *PODS*, pages 13–28, 2016.

[3] F. Afrati, M. Joglekar, C. Ré, S. Salihoglu, and J. D. Ullman. GYM: A multiround join algorithm in MapReduce. In *ICDT*, 2017.

[4] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *TKDE*, 23(9):1282–1298, 2011.

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

[6] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, pages 208–222. Springer, 2007.

[7] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *PODS*, 2013.

[8] P. Beame, P. Koutris, and D. Suciu. Skew in parallel query processing. In *PODS*, 2014.

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

[10] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *CACM*, 51(1):107–113, 2008.

[11] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[12] X. Hu, P. Koutris, and K. Yi. The relationships among coarse-grained parallel models. Technical report, HKUST, 2016.

[13] X. Hu, M. Qiao, and Y. Tao. Join Dependency Testing, Loomis-Whitney Join, and Triangle Enumeration. In *PODS*, 2015.

[14] X. Hu, K. Yi., and Y. Tao. Output-optimal massively parallel algorithms for similarity joins. *TODS*, 2019.

[15] X. Hu and K. Yi. Towards a worst-case I/O-optimal algorithm for acyclic joins. In *PODS*, 2016.

[16] X. Hu and K. Yi. Instance and output optimal parallel algorithms for acyclic joins. In *PODS*, pages 450–463, 2019.

[17] M. R. Joglekar, R. Puttagunta, and C. Ré. AJAR: Aggregations and joins over annotated relations. In *PODS*, 2016.

[18] B. Ketsman and D. Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *PODS*, 2017.

[19] P. Koutris, P. Beame, and D. Suciu. Worst-case optimal algorithms for parallel query processing. In *ICDT*, 2016.

[20] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, 2011.

[21] G. L. Nemhauser and L. E. Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical programming*, 6(1):48–61, 1974.

[22] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In *PODS*, pages 37–48, 2012.

[23] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. *ACM SIGMOD Record*, 42(4):5–16, 2014.

[24] E. R. Scheinerman and D. H. Ullman. *Fractional graph theory: a rational approach to the theory of graphs.* Courier Corporation, 2011.

[25] Y. Tao. A simple parallel algorithm for natural joins on binary relations. In *ICDT*, 2020.

[26] T. Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, 2014.

[27] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

[28] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

A OMITTED PROOFS

A.1 Acyclic joins

An equivalent definition for α-acyclicity is based on the GYO reduction [1]: (1) if there is a vertex $v \in \mathcal{V}$ only appearing in edge e, then remove v from e; (2) if there is a pair of edges $e, e' \in \mathcal{E}$ such that $e \subseteq e'$, then remove e from $\mathcal{E}$. A join query $Q = (\mathcal{V}, \mathcal{E})$ is α-acyclic if the GYO reduction results in an empty hypergraph.

A join tree can be built by the GYO reduction recursively. If some unique attribute is removed from e, we add e as a leaf node if it does not exist. If e is removed by (2), we put e as a child of e'. We show some nice properties for acyclic join in Lemma A.1, A.2, and A.3. For short, we use "acyclic" to denote "α-acyclic" below.

Lemma A.1. *For any acyclic join $Q = (\mathcal{V}, \mathcal{E})$ and an attribute $\mathbf{x} \in \mathcal{V}$, the residual join $Q_{\mathbf{x}} = (\mathcal{V}_{\mathbf{x}}, \mathcal{E}_{\mathbf{x}})$ is also acyclic.*

Proof. Recall that $Q_{\mathbf{x}}$ is the residual query by removing attribute $\mathbf{x}$ from all relations in $\mathcal{E}$. Let $\mathcal{T}$ be the join tree of Q, such that (1) there is a one-to-one correspondence between edges in $\mathcal{E}$ and nodes in $\mathcal{T}$; (2) for any attribute $\mathbf{x} \in \mathcal{V}$, all nodes containing $\mathbf{x}$ form a connected subtree. We derive another tree $\mathcal{T}'$ by removing attributes $\mathbf{x}$ from each node in $\mathcal{T}$. It can be easily checked that $\mathcal{T}'$ is a valid join tree for $Q_{\mathbf{x}}$, thus $Q_{\mathbf{x}}$ is acyclic. □

Lemma A.2. *Acyclic join has integral optimal edge covering.*

Proof. Let $\rho^*(Q)$ be the optimal edge covering for hypergraph $Q = (\mathcal{V}, \mathcal{E})$. An edge cover of Q can be obtained along with its GYO reduction. More specifically, if Q is emptyset, we set $\rho^* = 0$. In general, we apply the following two procedures: (1) If attribute $\mathbf{x} \in \mathcal{V}$ only appears in e, then assign e with weight 1 and remove all attributes in e from $\mathcal{V}$; (2) if $e, e' \in \mathcal{E}$ are distinct edges such that $e \subseteq e'$, then assign e with weight 0 and remove e from $\mathcal{E}$.

Next we will prove its optimality. The base case is trivial. In general, we prove it for these two cases separately.

If Q is reduced through (1), let $Q_e = (\mathcal{V} - e, \mathcal{E}_e)$ be the residual join by removing attributes in e from all relations in $\mathcal{E}$. By hypothesis, let $\rho^*(Q_e)$ be the integral optimal edge covering of Q_e. Note that $\rho^*(Q) = \rho^*(Q_e) + 1$, which is optimal since any edge cover require to assign $\rho^*(e) = 1$ to cover attribute v. Moreover, $\rho^*(Q)$ also admits integral optimal edge covering, since $\rho^*(Q_e)$ admits integral optimal edge covering.

If Q is reduced through step (2), let $Q' = (\mathcal{V}, \mathcal{E} - \{e\})$ be the residual join by removing edge e. Obviously, Q' is also acyclic. By hypothesis, let $\rho^*(Q_e)$ be the integral optimal edge covering of Q_e. Note that $\rho^*(Q) = \rho^*(Q')$, which is optimal since we can always shift any weight assigned to e to e' while maintaining its optimality. Thus, $\rho^*(Q)$ is an optimal integral edge covering for Q. □

We next introduce the berge-acyclic joins. For a join query $Q = (\mathcal{V}, \mathcal{E})$, consider the bipartite graph G, in which $\mathcal{V}$ corresponds to vertices on one side and $\mathcal{E}$ to vertices on the other side. There is an edge between $v \in \mathcal{V}$ and $e \in \mathcal{E}$ if $v \in e$. Then the hypergraph $(\mathcal{V}, \mathcal{E})$ is *berge-acyclic* if this bipartite graph is acyclic. This notion of acyclicity preserves many natural properties in ordinary acyclic graphs. For example, there is only one path between any two vertices $u, v \in \mathcal{V}$, and any subgraph of $(\mathcal{V}, \mathcal{E})$ is still acyclic. Note that this definition of berge-acyclicity does not allow two relations to have two or more common attributes. But if these attributes always appear together in any relation, then they can be simply considered as one "combined" attribute. In a berge-acyclic join $Q = (\mathcal{V}, \mathcal{E})$, a relation e is called a *leaf* if it contains at least one unique attribute and exactly one join attribute, and *non-leaf* otherwise.

Lemma A.3. *For any reduced berge-acyclic join Q, $\tau^* \le \rho^*$ where τ^*, ρ^* are the optimal fractional edge packing and covering number of Q respectively.*

Proof. We will prove this by induction for a berge-acyclic join $Q = (\mathcal{V}, \mathcal{E})$. The base case is trivial that when $\mathcal{E}$ contains a single relation, with $\rho^* = \tau^* = 1$. In general, we consider two more cases:

Case 1. If Q is disconnected, let $Q_1, Q_2, \cdots, Q_k$ be its connected components. As Q_i is also berge-acyclic for any $i \in \{1, 2, \cdots, k\}$, by hypothesis we have $\tau^*(Q_i) \le \rho^*(Q_i)$. Observe that $\tau^*(Q) = \sum_i \tau^*(Q_i)$ and $\rho^*(Q) = \sum_i \rho^*(Q_i)$. Thus, $\tau^*(Q) \le \rho^*(Q)$.

Case 2. Otherwise, Q is connected. First, we can always find a non-leaf e_0 such that removing all leaves in Q would turn it into a new leaf. Let $\Gamma(e_0)$ be the set of leaves connected with e_0. Note that each relation in $\Gamma(e_0)$ include one attribute in e_0. Let $\mathcal{E}_x = \{e \in \mathcal{E} : x \in e\}$ be the set of relations containing attribute x. Define

$$\mathcal{V}' = \{v \in e_0 : \mathcal{E}_v \subseteq \{e_0\} \cup \Gamma(e_0)\}$$

Note that $|\mathcal{V}'| = 1$; otherwise, e_0 is not a new leaf after removing all leaves in $\Gamma(e_0)$. Define

$$S = \{e \in \Gamma(e_0) : e \cap e_0 \in \mathcal{V}'\}$$

We further distinguish it into two more cases:

Case 2.1. If e_0 contains unique attributes, let Q' be the residual query by removing all edges in S. By hypothesis, $\rho^*(Q') \ge \tau^*(Q')$ since Q' is also a reduced berge-acyclic join. Note that $\rho^*(Q) = \rho^*(Q') + |S|$ and $\tau^*(Q) \le \tau^*(Q') + |S|$. Thus, $\rho^*(Q) \ge \tau^*(Q)$.

Case 2.2. Otherwise, let Q' be the residual query by removing all edges of $\{e_0\} \cup S$. By hypothesis, $\rho^*(Q') \ge \tau^*(Q')$ since Q' is also a reduced berge-acyclic join. Note that $\rho^*(Q) = \rho^*(Q') + |S|$ and $\tau^*(Q) \le \tau^*(Q') + |S|$. Thus, $\rho^*(Q) \ge \tau^*(Q)$. □

Lemma A.4. *For a join query Q, the following properties hold:*

(1) If Q is both binary-relation and α-acyclic, then Q is berge-acyclic.

(2) If Q is degree-two and α-acyclic but not binary-relation, it may or may not be berge-acyclic.

Proof. For (1), if a join query Q is both binary-relation and α-acyclic, the hypergraph of Q is a tree, which is berge-acyclic. For (2), $R_1(A, B, C) \bowtie R_2(A, B, D) \bowtie R_3(C, E)$ is not berge-acyclic, and $R_1(A, B, C) \bowtie R_2(A, D) \bowtie R_3(B, E) \bowtie R_4(C, F)$ is berge-acyclic. □

A.2 Missing Proofs in Section 4

Proof of Lemma 11. Without loss of generality, assume $r = e_k$. The base case with $k = 1$ always holds. Let $\mathbf{x} \in e_{k-1} - e_k$ be the first attribute. Note that $S^{\mathbf{x}} = \{e_1, e_2, \cdots, e_{k-1}\}$.

The residual join $Q_{\mathbf{x}}$ will be reduced first and then tackled by invoking the whole algorithm recursively. By hypothesis, each heavy instance $\mathcal{R}_{\mathbf{a}}$ can be computed using $O(\sum_{S\in\mathcal{S}(\mathcal{E}_{\mathbf{x}})}\lceil\Psi(\mathcal{T},\mathcal{R}_{\mathbf{a}},S,L)\rceil)$ servers in $O(1)$ rounds with load $O(L)$. The total number of servers allocated for all heavy assignments is

$$\begin{aligned}
&\sum_{\mathbf{a}}\sum_{S\in\mathcal{S}(\mathcal{E}_{\mathbf{x}})}\lceil\Psi(\mathcal{T},\mathcal{R}_{\mathbf{a}},S,L)\rceil\\
\le&\sum_{S\in\mathcal{S}(\mathcal{E}_{\mathbf{x}})}\sum_{\mathbf{a}}\Psi(\mathcal{T},\mathcal{R}_{\mathbf{a}},S,L)+2^{|\mathcal{E}|}\cdot\sum_{i\in[k-1]}\frac{|R(e_i)|}{L}\\
\le&\sum_{i\in[k-1]}\frac{|R(e_k)|}{L}\cdot\frac{|R(e_i)|}{L}+(2^{|\mathcal{E}|}+1)\sum_{i\in[k-1]}\frac{|R(e_i)|}{L}\\
\le&\sum_{S\in\mathcal{S}(\mathcal{E})}\Psi(\mathcal{T},\mathcal{R},S,L)
\end{aligned}$$

where the first inequality is implied by the fact that there are $O(\sum_{i\in[k-1]}\frac{|R(e_i)|}{L})$ heavy assignments, the second inequality is implied by distinguishing $S\in\mathcal{S}(\mathcal{E}_{\mathbf{x}})$ into two cases, depending on whether $S\cap\{e_1,e_2,\cdots,e_{k-1}\}=\emptyset$, and the last inequality is implied by the definition of $\mathcal{S}(\mathcal{E})$.

The residual join $Q_{\mathbf{y}}$ degenerates to the base case with one relation e_k. Then, for each light group I_j, $Q_{\mathbf{y}}(\mathcal{R}_j)$ can be computed using $O(\lceil\frac{|R(e_k)|}{L}\rceil)$ servers in $O(1)$ rounds with load $O(L)$. The total number of servers allocated for all light groups is

$$\begin{aligned}
\sum_j\lceil\frac{|R(e_k)|}{L}\rceil&\le\sum_j\left(\frac{|R(e_k)|}{L}+1\right)\\
&\le\left(\sum_{i\in[k-1]}(\frac{|R(e_i)|}{L}+1)\right)\cdot\left(\frac{|R(e_k)|}{L}+1\right)\\
&\le\sum_{S\in\mathcal{S}(\mathcal{E})}\Psi(\mathcal{T},\mathcal{R},S,L)
\end{aligned}$$

where the second last inequality is implied by the fact that there are $O(\sum_{i\in[k-1]}\lceil\frac{|R(e_i)|}{L}\rceil)$ light groups ad the last inequality is implied by the definition of $\mathcal{S}(\mathcal{E})$.

Combining the analysis for heavy and light subqueries, we complete the whole proof for Lemma 11. □

Proof of Lemma 13. We will prove the following complexity for a twig join excluding its root r:

$$\mathcal{S}(\mathcal{E})=\{\{S\}:S\subseteq(\mathcal{L}_1-\{r\})\times\mathcal{L}_2\times\cdots\times\mathcal{L}_\ell\}$$

Let c be the child of r. Let $\mathbf{x}\in c-r$ be the first attribute, and e be the leaf node with $\mathbf{x}\in e$. Let path(c,e) as the set of nodes lying on the path from c to e. The residual join $Q_{\mathbf{x}}$ is reduced first and then tackled by invoking the algorithm recursively. By hypothesis, each heavy instance $\mathcal{R}_{\mathbf{a}}$ induced by heavy value $\mathbf{a}\in H(\mathbf{x},S^{\mathbf{x}})$ can be computed using $O(\sum_{S\in\mathcal{S}(\mathcal{E}_{\mathbf{x}})}\lceil\Psi(\mathcal{T},\mathcal{R}_{\mathbf{a}},S,L)\rceil)$ servers in $O(1)$ rounds with load $O(L)$. The total number of servers allocated for all heavy assignments is

$$\begin{aligned}
&\sum_{\mathbf{a}}\sum_{S\in\mathcal{S}(\mathcal{E}_{\mathbf{x}})}\lceil\Psi(\mathcal{T},\mathcal{R}_{\mathbf{a}},S,L)\rceil\\
\le&\sum_{S\in\mathcal{S}(\mathcal{E}_{\mathbf{x}})}\sum_{\mathbf{a}}\Psi(\mathcal{T},\mathcal{R}_{\mathbf{a}},S,L)+2^{|\mathcal{E}|}\cdot\sum_{e'\in\text{path}(c,e)}\frac{|R(e')|}{L}\\
\le&\sum_{\mathcal{P}=\{\mathcal{L}_1,\cdots,\mathcal{L}_\ell\}}\sum_{S_1}\sum_{e'\in\text{path}(c,e)}\Psi(\mathcal{T},\mathcal{R},S_1\cup\{e'\},L)\\
+&\sum_{\mathcal{P}=\{\mathcal{L}_1,\cdots,\mathcal{L}_\ell\}}\sum_{S_2}\Psi(\mathcal{T},\mathcal{R},S_2,L)\\
+2^{|\mathcal{E}|}\cdot&\sum_{e'\in\text{path}(c,e)}\frac{|R(e')|}{L}\le\sum_{\mathcal{P}'}\sum_{S\in\mathcal{S}(\mathcal{E})}\Psi(\mathcal{T},\mathcal{R},S,L)
\end{aligned}$$

for all linear coverings $\mathcal{P},\mathcal{P}'$ over $\mathcal{T}$, $S_1\subseteq(\mathcal{L}_1-\{r\})\times\mathcal{L}_2\times\cdots\times\mathcal{L}_\ell$ with $S_1\cap\text{path}(c,e)=\emptyset$, and $S_2\subseteq(\mathcal{L}_1-\{r\})\times\mathcal{L}_2\times\cdots\times\mathcal{L}_\ell$ with $S_2\cap\text{path}(c,e)\neq\emptyset$. Note that the first inequality is implied by the fact that there are $O(\sum_{e'\in\text{path}(c,e)}\frac{|R(e')|}{L})$ heavy assignments, the second inequality is implied by distinguishing S into two cases, depending on whether $S\cap\text{path}(c,e)=\emptyset$ or not, and the last inequality is implied by Lemma A.5.

Lemma A.5. *For a twig join Q with its join tree $\mathcal{T}$, consider an arbitrary linear covering $\mathcal{P}=\{\mathcal{L}_1,\mathcal{L}_2,\cdots,\mathcal{L}_\ell\}$, and an arbitrary root-to-leaf path P of $\mathcal{T}$, where P could be different from $\mathcal{L}_1$. Then, $\{\mathcal{L}_1-P,\mathcal{L}_2-P,\cdots,\mathcal{L}_\ell-P\}$ is a linear covering of $\mathcal{T}-P$.*

Proof. We first observe that if $\mathcal{L}_1=P$, $\{\mathcal{L}_1-P,\mathcal{L}_2-P,\cdots,\mathcal{L}_\ell-P\}=\{\mathcal{L}_2,\mathcal{L}_3,\cdots,\mathcal{L}_\ell\}$ is a linear covering of $\mathcal{T}-\mathcal{L}_1$, by the definition of $\mathcal{P}$. It remains to consider the case with $\mathcal{L}_1\neq P$. Note that $\mathcal{L}_1-P$ is a root-to-leaf path of $\mathcal{T}-P$. By induction, assume $\{\mathcal{L}_2-P,\mathcal{L}_3-P,\cdots,\mathcal{L}_\ell-P\}$ is a valid linear covering of $\mathcal{T}-P-\mathcal{L}_1$. As $(\mathcal{L}_i-P)\cap(\mathcal{L}_1-P)=\emptyset$ for any $i\in\{2,3,\cdots,\ell\}$, then $\mathcal{L}_1-P$ together with $\{\mathcal{L}_2-P,\mathcal{L}_3-P,\cdots,\mathcal{L}_\ell-P\}$ is a valid linear covering of $\mathcal{T}-P$. □

Let $\{\mathcal{T}_1,\mathcal{T}_2,\cdots,\mathcal{T}_h\}$ be the set of connected subtrees by removing path(c,e) from $\mathcal{T}$. Let $Q_i=(\mathcal{V}_i,\mathcal{E}_i)$ be the twig join defined on the subtree $\mathcal{T}_i$, excluding its root in $\mathcal{L}_1$. For each light group I_j, computing $Q_{\mathbf{y}}(\mathcal{R}_j)$ degenerates to computing the cartesian products over $Q_i(\mathcal{R}_j)$'s. By hypothesis, $Q_i(\mathcal{R}_j)$ can be computed using $O(\sum_{S\in\mathcal{S}(\mathcal{E}_i)}\lceil\Psi(\mathcal{T},\mathcal{R}_j,S,L)\rceil)$ servers in $O(1)$ rounds with load $O(L)$. The total number of servers allocated for all light groups is

$$\begin{aligned}
&\sum_j\sum_i\sum_{S\in\mathcal{S}(\mathcal{E}_i)}\lceil\Psi(\mathcal{T},\mathcal{R}_j,S,L)\rceil\\
\le&\sum_i\sum_{S\in\mathcal{S}(\mathcal{E}_i)}\sum_j\Psi(\mathcal{T},\mathcal{R}_j,S,L)+\sum_{e'\in\text{path}(c,e)}\frac{|R(e')|}{L}\\
\le&\sum_{e'\in\text{path}(c,e)}\sum_{S\in\mathcal{S}(\mathcal{E}_1)\times\mathcal{S}(\mathcal{E}_2)\times\cdots\times\mathcal{S}(\mathcal{E}_h)}\Psi(\mathcal{T},\mathcal{R},S\cup\{e'\},L)\\
+&\sum_{e'\in\text{path}(c,e)}\frac{|R(e')|}{L}\le\sum_{S\in\mathcal{S}(\mathcal{E})}\Psi(\mathcal{T},\mathcal{R},S,L)
\end{aligned}$$

where the first inequality is implied by the fact that there are $O(\sum_{e'\in\text{path}(c,e)}\frac{|R(e')|}{L})$ light groups and the last inequality is implied by the fact that the union of linear-covers of $\mathcal{T}_1,\mathcal{T}_2,\cdots,\mathcal{T}_h$ together with path(c,e) is still a valid linear-cover of $\mathcal{T}$.

When this twig join Q includes root r, it degenerates to compute the Cartesian product between r and the residual twig join excluding r. Thus, the complexity in Lemma 13 follows. □

The proof of Theorem 14 follows directly from the primitive of reducing a join, the Case II of the generic algorithm in Section 3, and Lemma 13 sequentially.

A.3 Join-Aggregate Query

We consider join-aggregate queries over *annotated relations* [2, 11, 17] with one semiring. Let $(\mathbb{R}, \oplus, \otimes)$ be a commutative semiring. We assume that every tuple t is associated with an *annotation* $w(t) \in \mathbb{R}$. The annotation of a join result $t \in Q(\mathcal{R})$ is

$$w(t) := \otimes_{t_e \in R(e), \pi_e t = t_e, e \in \mathcal{E}} w(t_e).$$

Let $\mathbf{y} \subseteq \mathcal{V}$ be a set of *output attributes* (a.k.a. *free variables*) and $\bar{\mathbf{y}} = \mathcal{V} - \mathbf{y}$ the non-output attributes (a.k.a. *bound variables*). A *join-aggregate query* $Q_{\mathbf{y}}(\mathcal{R})$ asks us to compute $\oplus_{\bar{\mathbf{y}}} Q(\mathcal{R}) =$

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

In plain language, a join-aggregate query first computes the join $Q(\mathcal{R})$ and the annotation of each join result, which is the $\otimes$-aggregate of the tuples comprising the join result. Then it partitions $Q(\mathcal{R})$ into groups by their projection on $\mathbf{y}$. Finally, for each group, it computes the $\oplus$-aggregate of the annotations of the join results.

Many queries can be formulated as special join-aggregate queries. For example, if we take $\mathbb{R}$ to be the domain of integers, $\oplus$ to be addition, $\otimes$ to be multiplication, and set $w(t) = 1$ for all t, then it becomes the `COUNT(*) GROUP BY` $\mathbf{y}$ query; in particular, if $\mathbf{y} = \emptyset$, the query computes $|Q(\mathcal{R})|$. The join-project query $\pi_{\mathbf{y}} Q(\mathcal{R})$, also known as a *conjunctive query*, is a special join-aggregate query by discarding the annotations.

With respect to join-aggregate queries, *free-connex* queries [6] are an important subclass. To define a free-connex query, we introduce the notion of a *width-1 GHD*, which can be considered as a generalized join tree. A *width-1 GHD* of a hypergraph $Q = (\mathcal{V}, \mathcal{E})$ is a tree $\mathcal{T}$, where each node $u \in \mathcal{T}$ is a subset of $\mathcal{V}$, such that (1) for each attribute $x \in \mathcal{V}$, the nodes containing x are connected in $\mathcal{T}$; (2) for each hyperedge $e \in \mathcal{E}$, there exists a node $u \in \mathcal{T}$ such that $e \subseteq u$; and (3) for each node $u \in \mathcal{T}$, there exists a hyperedge $e \in \mathcal{E}$ such that $u \subseteq e$.

Given a set of output attributes $\mathbf{y}$, $\mathcal{T}$ is free-connex if there is a subset of connected nodes of $\mathcal{T}$, denoted as $\mathcal{T}'$ (such a $\mathcal{T}'$ is said to be a connex subset), such that $\mathbf{y} = \bigcup_{u \in \mathcal{T}'} u$. A join-aggregate query $Q_{\mathbf{y}}$ is free-connex if it has a free-connex width-1 GHD.

A.4 Proof of Lemma 21

The proof directly follows the fact that the dual of any reduced degree-two join is a reduced binary-relation join [18, 24]. We can show more details for each property in Lemma 21.

For (1), let f be the mapping from $\mathcal{E}$ to $[0, +\infty)$. Let $f(e) = \frac{1}{2}$ for every $e \in \mathcal{E}$. In the reduced degree-two join Q, each vertex appears in exactly two hyperedges. In this way, f is both a valid fractional edge packing and edge covering. Implied by the maximization of fractional edge packing and minimization of fractional edge covering, we get $\tau^* \geq \frac{|\mathcal{E}|}{2} \geq \rho^*$.

For (2), let f be a valid fractional edge packing for Q. It can be easily checked that $g = \{1 - f(e) : e \in \mathcal{E}\}$ is a valid fractional edge covering for Q. Thus, we get $\rho^* = |\mathcal{E}| - \tau^*$.

A similar result has been proved for optimal fractional vertex covering for an ordinary graph [21], that the vertex packing for an ordinary graph admits half-integral solution, and the set of vertices with value $\frac{1}{2}$ form a set of vertex-disjoint odd-length cycles. Thus, (3) and (4) follow.

A.5 Proof of Theorem 18

We first point out several important properties for degree-two joins satisfying edge-packing-provable conditions.

If there is no odd-length cycle in Q, it admits integral optimal edge packing τ^* and covering ρ^*, implied by Lemma 21. More specifically, there exists a partition $(\mathcal{E}_\alpha, \mathcal{E}_\beta)$ of $\mathcal{E}$: $\mathcal{E}_\alpha = \{e \in \mathcal{E} : \rho^*(e) = 1, \tau^*(e) = 0\}$ and $\mathcal{E}_\beta = \{e \in \mathcal{E} : \rho^*(e) = 0, \tau^*(e) = 1\}$. Consider any vertex v incident to two edges e, e'. There must be $e \in \mathcal{E}_\alpha, e' \in \mathcal{E}_\beta$, or $e \in \mathcal{E}_\beta, e' \in \mathcal{E}_\alpha$. This also implies that all edges in $\mathcal{E}_\alpha$ are vertex-disjoint, as well as edges in $\mathcal{E}_\beta$.

Let x be an optimal fractional vertex covering for Q. An edge e is denoted as *deterministic* if $\sum_{v: v \in e} x_v = 1$, and *probabilistic* otherwise. Let $\mathcal{E}'$ be the set of probabilistic edges, i.e. $\{e \in \mathcal{E} : \sum_{v: v \in e} x_v > 1\}$. Note that vertex covering and edge packing are prime-dual problems. The following result is directly implied by the complementary slackness.

Lemma A.6. *Let $\mathcal{E}' = \{e \in \mathcal{E} : \sum_{v: v \in e} x_v > 1\}$. For any $e \in \mathcal{E}'$, $\tau^*(e) = 0$ and $\rho^*(e) = 1$.*

For each edge $e \in \mathcal{E}$, let $\Upsilon(e)$ be the set of vertices appearing in the neighbor of e, i.e., $\Upsilon(e) = \cup_{e' \in \Gamma(e)} e' - e$. Two nice properties on the edges in $\mathcal{E}'$ are stated in Lemma A.7 and Lemma A.8.

Lemma A.7. *Let $\mathcal{E}' = \{e \in \mathcal{E} : \sum_{v: v \in e} x_v > 1\}$. For any edge $e \in \mathcal{E}'$, $\sum_{v: v \in e} x_v + \sum_{v: v \in \Upsilon(e)} x_v = |\Gamma(e)|$.*

Proof. As $\tau^*(e) = 0$ and $\rho^*(e) = 1$, we have $\tau^*(e') = 1$ and $\rho^*(e) = 0$ for every edge $e' \in \Gamma(e)$. This also implies that for any pair of edges $e_1, e_2 \in \Gamma(e), e_1 \cap e_2 = \emptyset$. Moreover, each edge $e' \in \Gamma(e)$ is deterministic implied by Lemma A.6, thus $\sum_{v: v \in e'} x_v = 1$. We also observe that $e \cup \left(\cup_{e' \in \Gamma(e)} e' - e\right) = \cup_{e' \in \Gamma(e)} e'$. Thus, the left-hand-side of the target equation can be rewritten as

$$\sum_{v: v \in \cup_{e' \in \Gamma(e)} e'} x_v = \sum_{e' \in \Gamma(e)} \sum_{v: v \in e} x_v = |\Gamma(e)|$$

thus yielding the desired result. □

Lemma A.8. *Let $\mathcal{E}' = \{e \in \mathcal{E} : \sum_{v: v \in e} x_v > 1\}$. $\rho^* - \tau^* = |\mathcal{E}'| - \sum_{e: e \in \mathcal{E}'} \sum_{v: v \in e} x_v$.*

Proof. By the duality theorem, $\tau^* = \sum_{v: v \in \mathcal{V}} x_v$. In this way, we can rewrite $\sum_{e \in \mathcal{E}'} \sum_{v: v \in e} x_v$ as

$$\sum_{e \in \mathcal{E}'} \sum_{v: v \in e} x_v = \sum_{v \in \mathcal{V}} x_v - \sum_{e \in \mathcal{E}_\alpha - \mathcal{E}'} \sum_{v: v \in e} x_v = \tau^* - (\rho^* - |\mathcal{E}'|)$$

thus yielding the desired result. □

Now, we are able to prove Theorem 18. As it follows the same framework as Section 5.1, we will focus on addressing the difference in this non-trivial extension. Similarly, we will show that with positive probability, an instance constructed this way will have a bounded $J(L)$, the maximum number of join results a server can produce, if it loads at most L tuples from each relation. Then setting $p \cdot J(L) = \Omega(|Q(\mathcal{R})|)$ yields a lower bound on L.

Hard instance construction. There are N^{x_v} distinct values in the domain of attribute v. Namely, a deterministic relation $R(e)$ is a Cartesian product over all attributes in e, with $\prod_{v:v\in e} N^{x_v} = N^{\sum_{v:v\in e} x_v} = N$ tuples in total; and a probabilistic relation $R(e)$ is constructed in a probabilistic way, such that each combination $t \in \times_{v:v\in e}\text{dom}(v)$ has a probability of $p(e) = 1/N^{\sum_{v:v\in e} x_v - 1}$ to form a tuples in $R(e)$, with $\prod_{v:v\in e} N^{x_v} \cdot p(e) = N$ tuples in expectation. Moreover, each relation has its input size deviates from its expectation by a constant factor is at most $\exp(-\Omega(N))$. Taking all relations in the edge covering, they together form N^{ρ^*} results while remaining relations are deterministic Cartesian product. So this instance has its output size deviating from its expectation by a constant factor is at most $\exp(-\Omega(N))$.

Step 1: Making a reasonale restriction on loading tuples.

Lemma A.9. *For any deterministic edge e, if $|\Gamma(e) \cap \mathcal{E}'| \le 1$, then assuming that tuples loaded from $R(e)$ should be in forms of Cartesian products over all attributes, doesn't decrease the maximum number of join results that can be produced per server by a constant factor.*

Proof. Note that for any vertex $v \in \mathcal{V}$, if the two edges incident to it are deterministic, the same argument in Lemma 19 can be applied to v, i.e., loading tuples in $R(e)$ for $v \in e$ in terms of $L_v \times L_{e-\{v\}}$ will not decrease the optimal solution by a constant factor. Moreover, if $L_e = L_v \times L_{e-\{v\}}$, then $L_{uv} = L_v \times L_u$ for any $u \in e - \{v\}$. To prove this result, it suffices to show that $L_{uv} = L_v \times L_u$ for every pair of vertices $u, v \in e$. We distinguish e into two cases.

If $\Gamma(e) \cap \mathcal{E}' = \emptyset$, Lemma 19 can be applied to all vertices to e, thus $L_{uv} = L_v \times L_u$ for every pair of vertices $u, v \in e$. Otherwise, $|\Gamma(e) \cap \mathcal{E}'| = 1$, say $\Gamma(e) \cap \mathcal{E}' = \{e''\}$. Applying Lemma 19 can be applied to every attribute $v \in e - e''$, we have $L_{uv} = L_v \times L_u$ for every $u \in e, v \in e - e''$. Note that when $|e'' \cap e| = 1$, we are done. The remaining case is when $|e'' \cap e| \ge 2$, we can assume that the optimal edge covering x could shift all weights on vertices in $e - e''$ to one specific vertex in $e - e''$, and assign 0 for remaining vertices in $e - e''$, without changing its optimality and property in Definition 22. In this way, the condition is also satisfied. □

Step 2: Prove an upper bound on $J(L)$.

Assume the number of distinct values from attribute v loaded by the server is z_v. Observe that $1 \le z_v \le N^{x_v}$ for each vertex $v \in V$. Moreover, $\prod_{v:v\in e} z_v = L$ for each deterministic relation e.

Recall that relations in $\mathcal{E}_\beta$ are vertex-disjoint. After loading L tuples from all deterministic relations, there are L^{τ^*} combinations of results in total, where each of them has a probability of

$$\prod_{e\in\mathcal{E}'} p(e) = \prod_{e\in\mathcal{E}'} \frac{1}{N^{\sum_{v:v\in e} x_v - 1}} = N^{|\mathcal{E}'| - \sum_{e:e\in\mathcal{E}'}\sum_{v:v\in e} x_v} = N^{\rho^* - \tau^*}$$

to form a valid join result, implied by Lemma A.8. The expected number of join results that can be produced by one server is $L^{\tau^*} \cdot N^{\rho^*-\tau^*}$. Next, we will show that this number of join results deviates from its expectation by a constant factor is exponentially small.

For each relation $e \in \mathcal{E}'$, we introduce a random variable Y_t for each combination $t \in \times_{v:v\in e} L_v$, which follows the bernoulli distribution with parameter $p(e)$. Denote $Y(e) = \sum_t Y_t$. Observe that there are $L^{|\Gamma(e)| - \sum_{v\in\Upsilon(e)} x_v}$ independent random variables in the space $\times_{v:v\in e} L_v$. So,

$$E[Y(e)] = L^{|\Gamma(e)| - \sum_{v\in\Upsilon(e)} x_v} \cdot p(e) = N \cdot (\frac{L}{N})^{|\Gamma(e)| - \sum_{v\in\Upsilon(e)} x_v}$$

where the last inequality is implied by Lemma A.7. By Chernoff bound, the probability that $Y(e)$ deviates from its expectation by a constant factor is at most $\exp(-\Omega(E[Y(e)]))$.

Let $Y = \prod_{e\in\mathcal{E}_2} Y(e)$. As mentioned, these L^{τ^*} combinations are not fully independent, as long as they share any same variable Y_t. In fact, the number of independent combinations is $L^{\tau^*-\rho^*+|\mathcal{E}'|}$, since all combinations can be put into disjoint groups by the random variables shared and each group has exactly $L^{\rho^*-|\mathcal{E}'|}$ combinations. Thus, the probability that the server produces more than $2^{|\mathcal{E}'|} \cdot \frac{L^{\tau^*-\rho^*}}{N^{\tau^*-\rho^*-|\mathcal{E}'|}}$ join results is at most

$$\Pr\left(Y \ge 2^{|\mathcal{E}'|} \cdot \frac{L^{\tau^*-\rho^*+|\mathcal{E}'|}}{N^{\tau^*-\rho^*}}\right) \le \sum_{e\in\mathcal{E}'} \Pr\left(Y(e) \ge 2 \cdot E[Y(e)]\right) \le \exp\left(-\Omega(\min_{e\in\mathcal{E}'} E[Y(e)])\right)$$

Consider an arbitrary attribute $v \in V$. Note that there are $\binom{N^{x_v}}{z_v} = O(N^{x_v \cdot z_v}))$ choices of loading z_v distinct values from the domain of attribute v. Over all possible values of z_v, the number of choices over all attributes in total is

$$\prod_{v:v\in V} \sum_{L_v=1}^{N^{x_v}} \exp\left(O(N^{x_v \cdot L_v})\right) = \exp\left(\widetilde{O}(\max_v N^{x_v})\right)$$

By the union bound, the probability that any of the choices produces more than $2^{|\mathcal{E}'|} \cdot \frac{L^{\tau^*-\rho^*}}{N^{\tau^*-\rho^*-|\mathcal{E}'|}}$ join results is at most

$$\exp\left(-\Omega(\min_{e\in\mathcal{E}'} E[Y(e)]) + \widetilde{O}(\max_{v\in\mathcal{V}} N^{x_v})\right)$$

which is exponentially small if

$$\min_{e\in\mathcal{E}'} E[Y(e)] \ge c_1 \cdot \max_{v\in\mathcal{V}} N^{x_v} \cdot \log N$$

for some sufficiently large constant c_1, or

$$N \cdot (\frac{L}{N})^{\lambda(Q)} \ge c_1 \cdot \max_{v\in\mathcal{V}} N^{x_v} \cdot \log N$$

where $\lambda(Q) = \min_{e\in\mathcal{E}'} |\Gamma(e)| - \sum_{v\in\Upsilon(e)} x_v \le |\mathcal{E}|$. Rearranging it,

$$L^{\lambda(Q)} \ge c_1 \cdot N^{\lambda(Q)-1} \cdot \max_{v\in\mathcal{V}} N^{x_v} \cdot \log N$$

We know that $L \ge \Omega(\frac{N}{p})$, so this is true as long as

$$(\frac{N}{p})^{\lambda(Q)} \ge c_2 \cdot N^{\lambda(Q)-1} \cdot \max_{v\in\mathcal{V}} N^{x_v} \cdot \log N$$

for some sufficiently large constant c_2, or

$$N \ge c_3 \cdot p^{\frac{\lambda(Q)}{1-\max_{v:v\in\mathcal{V}} x_v}} \cdot (\log N)^{\frac{1}{1-\max_{v:v\in\mathcal{V}}}}$$

for some sufficiently large constant c_3. Note that x is constant-small, so $\frac{\lambda(Q)}{1-\max_v x_v} = O(|\mathcal{E}|)$ is still a constant.

Step 3: Apply counting argument.

So far, we have shown that with exponentially high probability each server produces no more than $O(L^{\tau^*} \cdot N^{\rho^*-\tau^*})$ join results in each round. Over p servers, the number of join results produced in total is $O(pL^{\tau^*} \cdot N^{\rho^*-\tau^*})$ with high probability. As there are $\Theta(N^{\rho^*})$ join results, we must have $pL^{\tau^*} \cdot N^{\rho^*-\tau^*} \ge N^{\rho^*}$, which leads to a lower bound $L \ge N/p^{1/\tau^*}$.