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

Xiao Hu
Duke University
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 optimal fractional edge covering number of the query.

In this work, we make efforts towards bridging this gap. We first design an instance-dependent algorithm for the class of 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, which is worst-case optimal. Beyond acyclic joins, we surprisingly find that optimal fractional edge covering number doesn’t lead to a tight lower bound any more. More specifically, we prove for a class of cyclic joins a better lower bound in terms of optimal fractional edge packing number of the query, which is also tight as matched by existing algorithms. This new result displays a significant distinction not only between acyclic and cyclic joins, but also between the fine-grained RAM and coarse-grained MPC model.

ACM Reference Format:

1 INTRODUCTION
Evaluating join queries is one of the most 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 Atserias, Grolhe, and Marx [4], 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 $\rho^*$ 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 worst-case optimal join algorithms [22, 26] in the RAM model. All joins display a unified form of worst-case 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 algorithm has received much more attention in recent years due to the rapid development of massively parallel systems such as MapReduce [9] and Spark [28]. Join evaluation in the massively computational model are quite different from the RAM model, where an efficient algorithm should make best use of data locality, i.e., the algorithm tries to send as many tuples that can be joined as possible to one machine so that it can produce their join results. Intriguing questions arise, can we also achieve a unified worst-case optimal join algorithm in the MPC model? Is the worst-case complexity only related to the optimal fractional edge covering number? If not, what other query-dependent quantities? This work will answer these two questions.

1.1 Join Query
A (natural) join is defined as a hypergraph $Q = (V, E)$, where the vertices $V = \{x_1, \ldots, x_n\}$ model the attributes and the hyperedges $E = \{e_1, \ldots, e_m\} \subseteq 2^V$ model the relations [1]. Let $\text{dom}(x)$ be the domain of attribute $x \in V$. An instance of $Q$ is a set of relations $R = \{R(e) : e \in E\}$, where $R(e)$ is a set of tuples, where each tuple is an assignment that assigns a value from $\text{dom}(x)$ to $x$ for every $x \in e$. We use $N = \max_{e \in E}|R(e)|$ to denote the maximum size of input relations. The join results of $Q$ on $R$, denoted as $Q(R)$, consist of all combinations of tuples, one from each $R(e)$, such that they share common values on their common attributes. People usually study the data complexity of join algorithms, i.e., we 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 $Q = (V, E)$, two query-related quantities as edge covering and edge packing will be commonly used throughout this paper. Let $f$ be a mapping from $E$ to $[0, +\infty)$. Note that $f$ is a fractional edge covering if

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

and a fractional edge packing if

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

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

1.2 The model of computation
We consider the MPC model [2, 3, 6, 7, 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 the algorithm is measured by the number of rounds.
and the load, denoted as $L$, which is the maximum message size received by any server in any round. A linear load $L = O\left(\frac{N}{p}\right)$ 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 [3, 6, 7, 19, 20], but recently, more interests have been given to multi-round (but still constant) algorithms [2, 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., the 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 $t_i \in \mathcal{R}(e_i), i = 1, 2, \ldots, m$, 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.

### 1.3 Worst-case optimal join algorithms

In this work, we will focus on worst-case optimality, which is the most commonly used measurement for algorithm design, providing theoretical guarantees for the performance of algorithms 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-optimal algorithms take both input size $N$ and output size $OUT$ as parameters to divide the instance space, and display optimality on the worst instance for each class. In extreme case when each class contains just one instance, we obtain instance-optimal algorithms. Note that by definition, an instance-optimal algorithm must be output-optimal, and an output-optimal algorithm must be worst-case optimal, but the relative direction may not be true. We refer interested readers to [15] for a summary of fine-grained join algorithms in the MPC model.

In the RAM model, there is a unified worst-case optimal algorithm [23] for computing all joins in $O(N\rho^e)$ time. The situation becomes much more interesting in the MPC model. All results to be reviewed for the MPC model are put into Table 1 and the relationship between join queries mentioned is clarified in Figure 1.

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 [3, 6], named as hypercube, was proposed for computing all joins on non-skewed input instances with load $\tilde{O}\left(\frac{N}{p^{1/\gamma}}\right)$.

Later, an improved algorithm built upon hypercube has been proposed by the same researchers [19], for computing all joins with arbitrary input instances. This algorithm still runs in a single round, but incurs a higher load of $\tilde{O}\left(\frac{N}{p^{1/\gamma} \rho^{e}}\right)$, where $\psi^e$ is the optimal fractional edge quasi-packing number\(^1\) of the query. This result has been proved to be optimal (up to a polylog factor) for single-round computation with arbitrary input instances. It should be noted that $\psi^e \geq \tau^e$ [19].

Meanwhile, 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 $Q = R_1(A) \bowtie R_2(A, B)$, which has $\psi^e = \tau^e = 2$ by choosing $R_1, R_3$ in the fractional edge packing and $\rho^e = 1$ by choosing $R_2$ in the fractional edge cover. If targeting a single round, it can be computed with load $\tilde{O}\left(\frac{N}{p^{1/\gamma}}\right)$ implied by [19]. However, if just allowing one more round, it can be computed through two steps of semi-joins with linear load.

---

\(^1\)The $\tilde{O}$ notation suppresses polylog factors.

\(^2\)For a join query $Q = (V, E)$, the edge quasi-packing number is defined as follows. Let $x \subseteq V$ be any subset of vertices of $V$. Define the residual hypergraph after removing attributes $x$ as $Q_x = (\mathcal{V}_x, \mathcal{E}_x)$, where $\mathcal{V}_x = V - x$ 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., $\psi^e = \max_{e \subseteq V} \tau^e(Q_x)$.

---

<table>
<thead>
<tr>
<th>Joins</th>
<th>one-round</th>
<th>multi-round</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$-acyclic</td>
<td>$O\left(\frac{N}{p^{1/\gamma}}\right)$ is achieved for $\tau$-hierarchical join [15] and all $\alpha$-acyclic joins [Theorem 5]</td>
<td>$\tilde{O}\left(\frac{N}{p^{1/\gamma}}\right)$ is achieved for binary-relation join [18, 19, 25] and the Loomis-Witney join [19]</td>
</tr>
<tr>
<td>cyclic</td>
<td>$\tilde{O}\left(\frac{N}{p^{1/\gamma}}\right)$</td>
<td>$\Omega\left(\frac{N}{p^{1/\gamma}}\right)$ for $\boxdot$-join [Theorem 6] and some degree-two joins [Theorem 7]</td>
</tr>
</tbody>
</table>

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, $\psi^e$ is the optimal fractional edge quasi-packing number, $\rho^e$ is the optimal fractional edge covering number, $\tau^e$ is the optimal fractional edge packing number. It is known that $\psi^e \geq \max\{\rho^e, \tau^e\}$ [19].

---

**Figure 1:** Classification of join queries.
\(\tilde{O}(N^\frac{2}{p})\) (see Section 2). The \(\sqrt{p}\)-gap can be further enlarged to \(p^{\frac{m+1}{m}}\), on the star-dual join \(Q = R_0(x_1, x_2, \ldots, x_q) \|^* R_1(x_1) \|^* R_2(x_2) \|^* \cdots \|^* R_n(x_n)\). This opens up new opportunities for how to compute multi-way join 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 \(O\left(\frac{N}{p^{\frac{m+1}{m}}}\right)\). The reason why this is a reasonable lower bound can be argued by the counting argument: Each server can only produce \(O(L^p)\) join results in a constant number rounds with the load limited to \(L\) (also implied by the AGM bound [4]), so all the \(p\) servers can produce at most \(O(p \cdot L^p)\) join results. Thus, producing \(N^p\) join results needs at least a load of \(L = \Omega\left(\frac{N}{p^{\frac{m+1}{m}}}\right)\). 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 \(\alpha\)-acyclic joins, is still open.

Besides, several output-optimal (or output-sensitive) algorithms have been proposed for join queries in the MPC model. For the class of \(r\)-hierarchical joins, an output-optimal algorithm has been proposed in [15], which must also be worst-case optimal. Note that the \(r\)-hierarchical join is a very restrictive class of join, for example, the simplest line-3 join query \(R_1(A, B) \equiv R_2(B, C) \odot R_3(C, D)\) is not \(r\)-hierarchical. Meanwhile, the classical Yannakakis algorithm designed for \(\alpha\)-acyclic join can be easily parallelized with load complexity \(O\left(\frac{N}{p} + \frac{\text{OUT}}{p}\right)\). Very recently, it has been improved to \(O\left(\frac{N}{p} + \frac{\text{OUT}}{p}\right)\) [15], which is output-optimal if \(\text{OUT} = O(p \cdot N)\). However, in the worst-case when OUT approaches the AGM bound \(O(N^\rho)\), this complexity degenerates to \(O\left(\frac{N^{(\alpha+1)/2}}{p}\right)\), which is very far away from our target \(\Omega\left(\frac{N}{p^{\frac{m+1}{m}}}\right)\).

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 external memory algorithm incurring \(\tilde{O}\left(\frac{N}{p} + p^{\frac{m}{m}}\right)\) I/Os, where \(p^\alpha = \min\{L(N, p) \leq M/r\}\). Implied by this reduction, worst-case optimal algorithms can be automatically obtained for LW join\(^4\) and binary-relation join in the EM model. It is worth mentioning that a worst-case optimal algorithm has been proposed [14] for berge-acyclic join in the EM model using \(O\left(\left(\frac{N}{M^\rho}, \frac{M}{p}\right)\right)\) I/Os, without a counterpart in the MPC model. However, berge-acyclic join is a very restrictive sub-class of \(\alpha\)-acyclic join; for example, a simple join query \(R_0(A, B, C) \equiv R_1(A, B, D) \equiv R_2(B, C, E) \equiv R_3(A, C, F)\) is \(\alpha\)-acyclic but not berge-acyclic, thus cannot be handled by the algorithm in [14]. 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.

\(^1\)A join query \(Q = (V, E)\) is a loomis-whitney (LW) join if \(E = \{V - \{x\} : x \in V\}\).
\(^2\)More generally, let \(\rho^* = \rho^* = n/(n - 1)\), where \(n = |V|\). As it is a very restrictive class of joins with highly symmetric structures, we omit it in the following discussion.
\(^3\)Other notions of acyclicity have been proposed, including berge-acyclicity, \(y\)-acyclicity and \(\beta\)-acyclicity. Moreover, berge-acyclicity implies \(y\)-acyclicity and \(\beta\)-acyclicity.
\(^4\)In the following of this work, “acyclic” always means “\(\alpha\)-acyclic” if not specified.

### 1.4 Our Results

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

**New Upper Bound.** The primary class of join queries we target in this work is the \(\alpha\)-acyclic join [8], which is the most commonly studied class of acyclic\(^5\) joins in database theory. Formally, a join query \(Q = (V, E)\) is \(\alpha\)-acyclic if there exists an undirected tree \(T\) whose nodes are in one-to-one correspondence with the edges in \(E\) such that for any vertex \(v \in V\), all nodes containing \(v\) form a connected subtree. Such a tree \(T\) is called the join tree of \(Q\). An example of an acyclic\(^5\) join is illustrated in Figure 4.

We propose a generic algorithm for computing any acyclic join \(Q\) in the MPC model, whose load complexity is closely related to the choices made by this non-deterministic algorithm while running. We give a characterization of “good” choices for this algorithm, and show its complexity in terms of a decomposition of the join tree. When all relations contain at most \(N\) tuples, this complexity has a closed form of \(O\left(\frac{N}{p^{\frac{m+1}{m}}}\right)\), which is worst-case optimal.

This result has reduced the complexity of acyclic joins evaluation from \(O\left(\frac{N}{p^\alpha}\right)\) to \(O\left(\frac{N}{p^{\frac{m+1}{m}}}\right)\) 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 \(\psi^*\) and \(\rho^*\), 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\(^7\), star-dual join and some tree joins\(^6\). We refer interested reader to [19] for details. Moreover, by the MPC-EM reduction, this result implies an EM algorithm for computing all acyclic joins with \(O\left(\frac{N^{\rho^*}}{M^{\rho^* - 1}B}\right)\) I/Os, shadowing the previous work [11].

**New Lower Bound.** Finally, we turn to cyclic joins. Surprisingly, we find that \(O\left(\frac{N}{p^\rho}\right)\) is not necessarily a correct target for multi-round worst-case optimal join algorithms, since the existing lower bound \(\Omega\left(\frac{N}{p^{\frac{m+1}{m}}}\right)\) is not tight any more. We start by answering an open question posed in [18]: On the edge-join \(Q_{\square} = R_1(A, B, C) \equiv R_2(D, E, F) \equiv R_3(A, D) \equiv R_4(E, B, C) \equiv R_5(C, D)\), whether there exists a better upper bound than \(\tilde{O}\left(\frac{N}{p^\rho}\right)\), or a better lower bound than \(\Omega\left(\frac{N}{p^\rho}\right)\)? As shown in Figure 2, \(Q_{\square}\) has \(\rho^* = 2\) by choosing \(\{R_1, R_2\}\) in the fractional edge cover and \(\rho = 3\) 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 \(\Omega\left(\frac{N}{p^\rho}\right)\). 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

---

\(^5\)A join query \(Q = (V, E)\) is a loomis-whitney (LW) join if \(E = \{V - \{x\} : x \in V\}\). Moreover, it has \(\rho^* = \rho^* = n/(n - 1)\), where \(n = |V|\). As it is a very restrictive class of joins with highly symmetric structures, we omit it in the following discussion.

\(^6\)A worst-case I/O-optimal algorithm for LW join was proposed in [12] independently.

\(^7\)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 \(\Box\)-join.

Figure 3: Relationship between optimal fractional edge covering number \(\rho^*\) and optimal fractional edge packing number \(\tau^*\) of reduced join queries.

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. We first prove an edge-packing-based lower bound for the \(\Box\)-join in Section 5.1, and then extend it to degree-two joins in Section 5.2.

2 MPC PRELIMINARIES

We mention the following deterministic primitives in the MPC model, which can be computed with load \(O(N^2)\) in \(O(1)\) rounds.

Reduce-by-key [13]. 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 information. The degree of value \(a \in \text{dom}(v)\) in relation \(R(e)\) is defined as the number of tuples in \(R(e)\) having this value in attribute \(a\), i.e., \(|\{e \in R(e) \mid v(e) = a\}|\). Each tuple \(t \in R(e)\) is considered to have “key” \(\pi_t(a)\) and “value” \(v\).

Semi-Join [15]. Given two relations \(R_1\) and \(R_2\) with a common attribute \(a\), the semi-join \(R_1 \times R_2\) returns all the tuples in \(R_1\) whose value on \(a\) 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].

Parallel-packing [15]. Given \(N\) numbers \(x_1, x_2, \ldots, x_N\) where \(0 < x_i \leq 1\) for \(i \in [N]\), group them into \(m\) sets \(Y_1, Y_2, \ldots, Y_m\) such that \(\sum_{j \in Y_j} x_j \leq 1\) for all \(j\), and \(\sum_{j \in Y_j} x_j \geq \frac{1}{m}\) 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 \(m \leq 1 + 2 \sum_{j} x_j\).

3 GENERIC ACYCLIC JOIN ALGORITHM

In this section, we study how to compute the class of acyclic joins efficiently in the MPC model. Before diving into the algorithmic details, we first define the following concepts to simplify our description. In a join query \(Q = (\mathcal{V}, \mathcal{E})\), let \(E_q = \{e \in E : x \in e\}\) be the set of relations containing attribute \(x \in \mathcal{V}\). Recall that in an acyclic join, its relations can be organized into a join tree \(T\) such...
that for each attribute $x$, the nodes containing this attribute, i.e., $E_x$, form a connected subtree in $T$. An example is illustrated in Figure 4. To be more general, $T$ 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 $E$. In this way, if there exists some relation $e \in E$ which doesn’t share any common attributes with other relations, then we just treat it as a single connected component of $T$. In a join tree $T$, an attribute is unique if it only appears in one node.

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

### 3.1 Algorithm

We describe our generic algorithm for computing the result $Q(R)$ on the input join tree $T$. The high-level idea is to recursively decompose the join into multiple subqueries based on the join tree, and apply a different join strategy for each subquery. For a clean presentation, we now only focus on the algorithmic details 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 $S(E) \subseteq 2^E$ as a set of subsets of relations, and a quantity $\Psi(T, R, S, L)$ for each subset of relations $S \subseteq E$. The definitions of $S(E)$ and $\Psi(T, R, S, L)$ determine the complexity of our algorithm, which will be thoroughly discussed in Section 3.2 and Section 4.2. Note that $S(E) \subseteq 2^E$ can be computed locally since the query has constant size. Intuitively, $\Psi(T, R, S, L)$ is the number of servers required by this algorithm for computing the join of relations in $S$ with low complexity $O(L)$.

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

**General Cases.** In general, we distinguish the input join tree $T$ into two cases.

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

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

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

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

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

$$Q(R) = \bigcup_{e \in E} \sigma_{e}(R(e))$$

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

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

$$Q(R) = \left(\bigcup_{a \in H(x, S^x)} Q_x(R_a) \right) \cup \left(\bigcup_{j \in \{1, 2, \ldots, k\}} Q(R_j)\right)$$

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 $Q_x$ with input instance $R_a$, we allocate $p_a = \max_{S \subseteq E} \Psi(T, R_a, S, L)$ servers, and invoke the whole algorithm for computing $Q_x(R_a)$ recursively.

Let $y$ be the set of unique attributes over all relations $R(e)$ for $e \in S^x$. We introduce a residual query $Q_y = (V_y, E_y)$ by removing all attributes in $y$ and relations in $S^x$, where $V_y = V \setminus y$ and $E_y = E \setminus S^x$. Let $T'$ be the resulting join tree by removing nodes in $S^x$ from $T$, which may contain multiple connected subtrees.

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

**Case II: $T$ consists of multiple connected subtrees.** Let $T_1, T_2, \ldots, T_k$ be the connected subtrees in $T$ and $E_1, E_2, \ldots, E_k$ be the corresponding set of relations in each subtree. Define $R_i = \{r \in E : e \in E_i\}$, and $Q_i = (V_i, E_i)$, where $V_i = \cup_{e \in E_i} e$. In this case, it becomes to compute a Cartesian product $Q_1(R_1) \times \cdots \times Q_k(R_k)$, where each $Q_i(R_i)$ is captured by Case I.

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

$$pi = \max_{S \subseteq E_i} \Psi(T_i, R_i, S, L)$$

Each server is identified with coordinates $(c_1, c_2, \ldots, c_k)$, where $c_i \in [p_i]$. For every combination $c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_k$, the $pi$ servers with coordinates $(c_1, \ldots, c_{i-1}, c, c_{i+1}, \ldots, c_k)$ form a group to compute $Q_i(R_i)$ (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.

### 3.2 Analysis

In this part, we analyze the complexity for 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 \(\Psi(T, \mathcal{R}, S, L)\) and \(S(E)\), as long as they satisfy the recurrence formulas implied by the generic algorithm.

As an example, we will focus on the generic algorithm if it always conservatively chooses \(S^0 = \{e_1\}\) in Case 1. Before diving into the details, we introduce the notion of subjoin first. Take an arbitrary subset of relations \(S \subseteq E\). We define \(T[S]\) as the set of maximally connected components of \(S\) on the join tree \(T\), which should be distinguished from the notion of maximally connected components of \(S\).

**Definition 3.1 (Subjoin).** For a join query \(Q = (V, E)\) with join tree \(T\), and an instance \(\mathcal{R}\), the subjoin of \(S \subseteq E\) is defined as
\[
\Theta(T, \mathcal{R}, S) = \bigcup_{S_i \in T[S]} \bigwedge_{e \in S_i} R(e)
\]
i.e., the Cartesian product of the join results over all maximally connected components in \(T[S]\).

**Example 3.2.** Let’s take two examples in Figure 4 for illustration. On \(S_1 = \{e_1, e_3, e_7\}\), \(S_1\) itself is a single connected component since all edges share the common attribute \(A\); however, all edges are not directly connected on the join tree \(T\), so \(T[S_1] = \{\{e_1\}, \{e_3\}, \{e_7\}\}\). Adding one more edge \(e_0\) to \(S_1\), denoted as \(S_2 = \{e_0, e_1, e_3, e_7\}\), does not change the fact that \(S_2\) is a single connected component, but \(e_0, e_1, e_3, e_7\) form a connected component on the join tree \(T\), so \(T[S_2] = \{\{e_0, e_1\}, \{e_3\}\}\). The subjoin\(^{12}\) of \(S_2\) is defined as \(\Theta(T, \mathcal{R}, S_2) = (e_0 \Join e_1 \Join e_3 \Join e_7)R\).

From Example 3.2, 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.,
\[
\pi_S Q(R) \subseteq \bigwedge_{e \in S} R(e) \subseteq \bigwedge_{e \in T} (\Theta(T, \mathcal{R}, S))
\]
Moreover, we mention an important observation for acyclic join as follows, which will be used in our analysis.

**Lemma 3.3.** For an acyclic join \(Q = (V, E)\) with its join tree \(T\), consider an arbitrary leaf node \(e_1\) and its parent \(e_0\). For any \(e \in E - \{e_0, e_1\}, (e \cap e_1) - e_0 = \emptyset\).

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

---

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

**Proof.** We first prove the complexity of the generic algorithm in Theorem 1 by induction on the size of \(Q\), and then show how to compute \(\Psi(T, \mathcal{R}, S, L)\) efficiently at last.

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

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

**Hypothesis 2.** For a join query \(Q_y\) with join tree \(T\), an instance \(\mathcal{R}_y\) and a pre-determined parameter \(L\), the result \(Q_y(\mathcal{R}_y)\) can be computed using \(O(\sum_{S \subseteq E_y} \Psi(T, \mathcal{R}_y, S, L))\) 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 assignments as follows:
\[
\sum_{a \in H(x, \{e_1\})} p_a = \sum_{a \in H(x, \{e_1\})} \max_{S \subseteq E_x} \Psi(T, \mathcal{R}_a, S, L) \\
\leq \sum_{S \subseteq E_x} \max_{a \in H(x, \{e_1\})} \Psi(T, \mathcal{R}_a, S, L) + 2^{|E|} \cdot \Psi(T, \mathcal{R}, \{e_1\}, L)
\]
where the second inequality is implied by the fact that each heavy assignment in \(H(x, \{e_1\})\) has degree more than \(L\) in relation \(R(e_1)\). We distinguish each \(S \subseteq E_x\) into two cases. If \(e_0 \notin S\) and \(e_1 \notin S\), the term induced on \(S\) can be bounded by
\[
\sum_{a \in H(x, \{e_1\})} \Psi(T, \mathcal{R}_a, S, L) \leq \Psi(T, \mathcal{R}, \{e_1\}, L) \cdot \Psi(T, \mathcal{R}, S, L)
\]
where the first inequality is implied by the fact that there are at most \(O(\frac{|R(e_1)|}{L})\) heavy assignments and the second one is implied by the fact that \(e_1\) forms a single connected component in \(T[S \cup \{e_1\}]\) from Lemma 3.3. Otherwise, \(e_0 \notin S\) and \(e_1 \notin S\). This term induced on \(S\) can be directly bounded by \(\sum_{a \in H(x, \{e_1\})} \Psi(T, \mathcal{R}_a, S, L) \leq \Psi(T, \mathcal{R}, S)\).

**Complexity of computing light subqueries.** In Step 3 of computing \(Q(y)_j\), each server receives at most \(L\) input tuples from \(\sigma_{e_1} R(e_1)\) and \(O(L)\) tuples in computing \(Q(y)_j(\mathcal{R}_j)\) 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:

$$
\sum_j p_j \leq \sum_j \sum_{S \subseteq E_y} [\Psi(T, \mathcal{R}_j, S, L)] \\
\leq 2^{|E|} \cdot \Psi(T, \mathcal{R}, \{e_1\}, L) + \sum_{S \subseteq E_y} \Psi(T, \mathcal{R}_j, S, L)
$$

Recall that $e_1 \notin E_y$ by definition. We distinguish each $S \subseteq E_y$ 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

$$
\sum_j \Psi(T, \mathcal{R}_j, S, L) \leq \Psi(T, \mathcal{R}, \{e_1\}, L) \cdot \Psi(T, \mathcal{R}, S, L)
$$

where the first inequality is implied by the fact that there are at most $O\left(\frac{|R(e_1)|}{|E|}\right)$ light groups and the second one is implied by the fact that $e_1$ forms a single connected component in $T[S \cup \{e_1\}]$ from Lemma 3.3. when $e_0 \notin S$. Otherwise, $e_0 \in S$. This term induced on $S$ can be directly bounded by $\sum_j \Psi(T, \mathcal{R}_j, S, L) \leq \Psi(T, \mathcal{R}, S, L)$.

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

$$
\sum_{S \subseteq E_y, e_0 \notin S} \Psi(T, \mathcal{R}, S \cup \{e_1\}, L) + \sum_{S \subseteq E_y, e_0 \notin S, e_1 \in S} \Psi(T, \mathcal{R}, S, L) + \sum_{S \subseteq E_y} \Psi(T, \mathcal{R}, S, L) \leq 4 \cdot \sum_{S \subseteq E} \Psi(T, \mathcal{R}, S, L)
$$

where the last inequality is implied by the following facts:

- $S \subseteq \{e_1\} \subseteq E$ for each $S \subseteq E_x$ but $e_0, e_1 \notin S$;
- $S \subseteq \{e_1\} \subseteq E$ for each $S \subseteq E_y$ but $e_0 \notin S$;
- $S \subseteq E$ for each $S \subseteq E_x$ and $S \subseteq E_y$,

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 $T_i$, an input instance $\mathcal{R}_i$, and a pre-determined parameter $L$, the result $Q_i(\mathcal{R}_i)$ can be computed using $O\left(\sum_{S \subseteq E} \Psi(T_i, \mathcal{R}_i, S, L)\right)$ 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)

$$
\prod_i p_i \leq \prod_i \sum_{S \subseteq E_i} [\Psi(T_i, \mathcal{R}_i, S_i, L)] \\
= \sum_{S_1, S_2, \cdots, S_k \subseteq E_1 \times E_2 \times \cdots \times E_k} \prod_i \Psi(T_i, \mathcal{R}_i, S_i, L) \\
\leq \sum_{S = (S_1, S_2, \cdots, S_k) \subseteq E_1 \times E_2 \times \cdots \times E_k} \Psi(T, \mathcal{R}, S, L) \leq \sum_{S \subseteq E} \Psi(T, \mathcal{R}, S, L)
$$

where the second last inequality is implied by the definition of $\otimes$ and the last inequality is implied by $E_1 \times E_2 \times \cdots \times E_k \subseteq E$, thus completing the induction proof for Case II.

At last, we show how to compute $\Psi(T, \mathcal{R}, S, L)$ efficiently. Given the value of $L$, it boils down to computing a set of subjoins. In Case I, for each $S \subseteq E_0$, or $S \subseteq E_y$, we use $O(\max_{\mathcal{E} \in E} |R(e)|)$ servers to compute $|\otimes(\mathcal{R}, S)|$’s over all heavy assignments or $|\otimes(\mathcal{R}_j, S)|$’s over all light groups. Computing these statistics can be captured by a free-connex join-aggregate query

$$
\sum_{T \in \mathcal{E}} \Psi(T, \mathcal{R}, \{e\}, S) \cdot \Psi(T, \mathcal{R}, S, L)
$$

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 assignment $t \in \text{dom}(x)$, with size bounded by $O(|\mathcal{E}|)$. If $a \in H(x, \{e_1\})$, then $|\otimes(\mathcal{R}_j, S)| = w(a)$; otherwise, we run the reduce-by-key primitive to compute \(\otimes(\mathcal{R}_j, S)\) = $\sum_{a \in \mathcal{E}} w(a)$ 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 $O(\sum_{e \in \mathcal{E}} \Psi(T, \mathcal{R}, \{e\}, S))$ servers, thus completing the whole proof for Theorem 1.

**3.3 Choosing $L$**

Theorem 1 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:

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

where $S$ is taken over all subsets of $E$. It can be easily checked that for each $S \subseteq E$, $\Psi(T, \mathcal{R}, S, L) \leq p$ 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 $O\left(\max_{S \subseteq E} \left(\frac{|\otimes(\mathcal{R}, S)|}{p}\right)^{\frac{1}{|E|}}\right)$, which is also bounded by $O(L)$. Together, we come to the following result directly.

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

So far we have obtained an algorithm for acyclic joins whose load complexity is in terms of subjoins. However, we observe a gap between the result in Theorem 2 and our target $O\left(\frac{N}{p^{|E|}}\right)$.

**Example 3.4.** Let’s use the example query in Figure 4 for illustration, which has $\rho^* = 6$ by choosing $\{e_1, e_2, e_3, e_4, e_5, e_7\}$. 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 $R_6(ABHJ)$ 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 $O(N^5)$. On $S_7 = \{e_1, e_2, e_3, e_5, e_6, e_7\}$, its subjoin has size as large as $N^3$ since $|R_5 \bowtie R_7 \bowtie R_7 \bowtie R_5| = N^3$ and $|R_5 \bowtie R_6 \bowtie R_7| = N^3$.

Thus, our generic algorithm compute the example join query on $\sum_{e \in \mathcal{E}} \Psi(T, \mathcal{R}, \{e\}, S)$ if $Q$ is acyclic and $V - z$ is contained by one relation, this query is free-connex.
this hard instance with load complexity $\Theta(\frac{N}{\rho^{7/6}})$, which is worse than the optimal by a factor of $O(1/\rho^{7/6})$.

Careful inspection reveals that not every subset $S \subseteq E$ appears in the cost formula of Theorem 2, 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 $x = A, B, C$ 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 $S_1$ does not contributes to the cost formula of Theorem 1. Thus, this example implies that the gap partly comes from the non-tight analysis.

Another example on which this conservative choice does not lead to optimality is given in Section 4.2. This motivates us to seek for more aggressive choices for this algorithm in the next section.

4 WORST-CASE OPTIMALITY OF ACYCLIC JOIN ALGORITHM

In this section, we take a further investigation of this generic algorithm and focus on the worst-case complexity. We identify a special case of an acyclic join as below, when there are only two leaves.

We first identify a special case of a join tree directly. Then, the question comes which attribute should be chosen to tackle first. We introduce the notion of first attribute in Definition 4.4. As $Q$ is reduced, $e - r \neq \emptyset$. The algorithm chooses any first attribute of $T$ as $x$ and a leaf node $e$ with $x \in e$. Note that such a leaf node always exists; otherwise, $x$ will only appear in nodes of $T - S_p$, contradicting the fact that $r$ is a valid edge covering for all attributes. Let $P$ be the set of nodes lying on the path from $c$ to $e$. The algorithm then chooses $S^* = P$.

Definition 4.4 (First Attribute in A Twig). In a twig join $T$ rooted at $r$ with child $c$, the first attribute is defined as any one in $e - r$.

In step 2, it defines a set of heavy subqueries $Q_h$ and light subqueries $Q_l$. Note that $Q_h$ is also a twig join, thus can be handled by invoking the whole algorithm recursively. For $Q_h$, the join tree will be decomposed into a set of subqueries after removing all nodes lying on the path from $c$ to $e$. Consider each node $e'$ in $P$. For each child $e'$ of $e'$ but not in $P$, the subtree rooted at $e'$ together with $e'$ form a twig join, and can be handled by invoking the whole algorithm recursively. The join results of these subtree fall into Case II, which is computed by enumerating the Cartesian product of their individual join results first and emitting true join results after checking locally.

4.2 Complexity

Next, we show the complexity for a good run of the generic algorithm. In this case, we first define

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

Obviously, the value of $\Psi(T, \mathcal{R}, S, L)$ can be computed locally, since the query has constant size. As acyclic join is tackled by decomposing it into a set of small queries, we start with the complexity for these primitives and come to that for a whole join tree at last.

Complexity of Linear join. We first identify a special case of twig join as below, when there are only two leaves.

Definition 4.5 (Linear join). An reduced join $Q = (V, E)$ is linear if it has a join tree $T$ 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 $e \in E - \{e_1, e_k\}$. 

Note that there could be multiple valid orderings. An example is given in Figure 5. The correctness of the recursion in Definition 4.3 is guaranteed by the tree hierarchy across twigs in $G$. 

Step 2: Decompose a twig. Now, we focus on handling a single twig join $Q$ with its join tree $T$ inherited from last step, which enjoy very nice properties: (1) $Q$ is reduced; (2) the set of leaves of $T$ is a valid edge covering for $Q$, which is also optimal; (3) $T$ has at most one non-leaf node in $S_p$. If such a node in (3) exists, we define it as the root of $T'$, otherwise, we pick an arbitrary leaf as the root of $T'$, denoted as $r$. Note that $r \in S_p$ only has one child in $T'$; otherwise, this twig will be further decomposed in step 1. Let $c$ be the child of $r$.

We next show how the generic algorithm proceeds on a twig. As the twig join is reduced with a single join tree, the algorithm goes into Case I directly. Then, the question comes which attribute should choose to tackle first. We introduce the notion of first attribute in Definition 4.4. As $Q$ is reduced, $e - r \neq \emptyset$. The algorithm chooses any first attribute of $T$ as $x$ and a leaf node $e$ with $x \in e$. Note that such a leaf node always exists; otherwise, $x$ will only appear in nodes of $T - S_p$, contradicting the fact that $r$ is a valid edge covering for all attributes. Let $P$ be the set of nodes lying on the path from $c$ to $e$. The algorithm then chooses $S^* = P$.

Definition 4.4 (First Attribute in A Twig). In a twig join $T$ rooted at $r$ with child $c$, the first attribute is defined as any one in $e - r$.

In step 2, it defines a set of heavy subqueries $Q_h$ and light subqueries $Q_l$. Note that $Q_h$ is also a twig join, thus can be handled by invoking the whole algorithm recursively. For $Q_h$, the join tree will be decomposed into a set of subqueries after removing all nodes lying on the path from $c$ to $e$. Consider each node $e'$ in $P$. For each children $e'$ of $e'$ but not in $P$, the subtree rooted at $e'$ together with $e'$ form a twig join, and can be handled by invoking the whole algorithm recursively. The join results of these subtree fall into Case II, which is computed by enumerating the Cartesian product of their individual join results first and emitting true join results after checking locally.
nodes in the edge cover

nodes not in the edge cover

The right is an example of linear-covering of twig 4 rooted at r.

An example of linear join is given in Figure 6. It should be noted that this is exactly a class of join queries, on which Theorem 2 fails to achieve optimality. Moreover, it can be easily shown that $e_i \cap e_j \subseteq e_{i+1} \cap e_k$ and $e_{i+1} \cap e_j \subseteq e_i \cap e_k$ for any $i \leq k - 1$. In this way, applying the rule in Definition 4.4 recursively, we can easily obtain two valid orderings of attributes for a linear join, corresponding to the root as $e_1$ and $e_k$ separately. Surprisingly, a linear join can be computed very efficiently following the orderings, as stated in Lemma 4.6.

**Lemma 4.6.** For a linear join $Q$ with its join tree $T$ being arranged in a line as $e_1, e_2, \ldots, e_k$, an instance $R$ and a parameter $L$, the join result $Q(R)$ can be computed using $O\left(\sum_{S \subseteq S(E)} \Psi(T, R, S, L)\right)$ servers in $O(1)$ rounds with load $O(L)$, where $S(E) = \{(e_1, e_2, \ldots, e_k) - (r)\} \times \{(r)\}$ for $r \in \{e_1, e_k\}$.

**Complexity of Twig Join.** A linear join is related to a twig join by the notion of linear-cover, defined as below.

**Definition 4.7 (Linear-cover of a Twig).** For a twig join $Q$ with its join tree $T$, a linear-cover for $T$ denoted as $\mathcal{P}(T)$ is:
- If $Q$ is a linear join, $\mathcal{P}(T) = \{T\}$;
- Otherwise, $\mathcal{P}(T) = \{L\} \cup \bigcup_{i=1}^{L} \mathcal{P}(T_i)$ where $L$ is an arbitrary root-to-leaf path of $T$ and $\{T_1, T_2, \ldots, T_l\}$ is the set of connected subtrees by removing $L$ from $T$.

An example of linear-cover of a twig is shown in Figure 5. Intuitively, a linear-cover defines a way of decomposing a twig join into a set of node/relation-disjoint linear joins. As shown in Lemma 4.8, computing a twig join degenerates to computing the Cartesian product of linear joins in a linear-cover.

**Lemma 4.8.** For a twig join $Q = (V, E)$ with its join tree $T$ rooted at $r$, an instance $R$ and a parameter $L$, the join result $Q(R)$ can be computed using $O\left(\sum_{P(T)} \sum_{S \subseteq S(E)} \Psi(T, R, S, L)\right)$ servers in $O(1)$ rounds with load $O(L)$, where $P(T) = \{L_1, L_2, \ldots, L_l\}$ is over all linear-covers of $T$, and $S(E) = (L_i - \{r\}) \times L_2 \times \cdots \times L_l \times \{r, \emptyset\}$.

By reassembling a set of twigs into the original join tree, we obtain the complexity result in Theorem 3. Abusing notations, we redefine the cross product between two sets of sets as $X \times Y = \{S_1 \cup S_2 : S_1 \in X, S_2 \in Y\}$.

**Theorem 3.** For an $\alpha$-acyclic join $Q = (V, E)$ with join tree $T$, an instance $R$ and a parameter $L$, the result $Q(R)$ can be computed using $O\left(\sum_{S \subseteq S(E)} \Psi(T, R, S, L)\right)$ servers in $O(1)$ rounds with load $O(L)$, where $\Psi(T, R, S, L) = \prod_{e \in S} \frac{|R(e)|}{L^2}$ and $S(E)$ is recursively defined as follows:

1. If there is a pair of nodes $e_1, e_0$ such that $e_1 \subseteq e_0$, then $S(E) = S(E) \cup \{\{e_1\}\}$. It should be noted that $T$ is updated by removing $e_1$ from $T$ and putting every child of $e_1$ as a new child of $e_0$ if $e_1$ is not a leaf in $T$.
2. $T$ is decomposed into a set of twigs $G_1, G_2, \ldots, G_\ell$ in order, $S(E) = \bigtimes_{i=1}^{\ell} S(E_i)$ where $E_i = G_i - G_{i-1}$ if there exists $j > i$ such that $G_i \cap G_j \neq \emptyset$ and $E_i = G_i$ otherwise.
3. $T$ is a twig with an arbitrary linear-cover $\mathcal{P} = \{L_1, L_2, \ldots, L_l\}$. If $T$ includes its root $r$, $S(E) = \bigcup_{r \in L} S(L_1 \cdot \cdots \cdot L_{l-r} \cdot \{r\}) \cup \bigcup_{r \in E} S(L_2) \cdot \cdots \cdot S(L_\ell)$. Otherwise, $S(E) = \bigcup_{r \in E} S(L_1 \cdot \cdots \cdot L_{l-r} \cdot \{r\} \cdot r) \cdot \bigcup_{r \in E} S(L_2) \cdot \cdots \cdot S(L_\ell)$.
4. $T$ is a linear join. If $T$ excluding root $r$, $S(E) = \{\{\} : e \in E\}$. Otherwise, $S(E) = \{\{e : e \in E \setminus \{r\}\} \times \{\{r, \emptyset\}\}$.

The complete proof of Theorem 3, together with that of Lemma 4.6 and Lemma 4.8 is given in Appendix A.3.

**4.3 Worst-case Optimality**

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

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

Since the query has constant complexity, the $S(T)$ as well as the value of $L$ can be computed locally.

**Theorem 4.** For an $\alpha$-acyclic join $Q$ with a join tree $T$, and an instance $R$, the result $Q(R)$ can be computed using $p$ servers in $O(1)$ rounds with load $O\left(\sum_{S \subseteq S(E)} \frac{\prod_{e \in S} |R(e)|}{L^2}\right)$.

Moreover, when each relation has at most $N$ input tuples, such a complicated bound has a clean form as stated in Theorem 5. A more fine-grained analysis of the optimality of Theorem 4 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 5.** For an $\alpha$-acyclic join $Q$ and an instance $R$ where each relation contains at most $N$ tuples, there is an algorithm computing...
We next prove Theorem 6. 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 $SN$ and output size $N^2$ 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 $L_A \times L_B \times L_C \times L_D, L_B \times L_D$, 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 5.1. The similar argument can be applied for attributes $B$ and $C$.

**Lemma 5.1.** 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 $L_A \times L_B \times L_C$ 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 $R_3(D, E, F), R_4(B, E)$ and $R_5(C, F)$, then the server needs to decide which $L$ tuples from $R_1(A, B, C)$ and $R_2(A, D)$ 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 $a \in \text{dom}(A), b \in \text{dom}(B), c \in \text{dom}(C), y_{ad}$ for each pair $a \in \text{dom}(A), d \in \text{dom}(D)$. 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 $I_{b}$. Then IP1 below defines this optimization problem, where $a$ always ranges over dom(A), $b$ over dom(B), $c$ over dom(C), $d$ over dom(D), $e$ over dom(E), $f$ over dom(F) unless specified otherwise.

\[
\begin{align*}
(IP1) \quad \max & \sum_{abc,d,e,f} I_{def} \cdot x_{b} \cdot x_{c} \cdot y_{ad} \\
\text{s.t.} & \max \left(\sum_{abc} x_{abc}, \sum_{ad} y_{ad}\right) \leq L \\
& I_{def} \cdot I_{b} \cdot I_{e} \cdot x_{abc} \cdot y_{ad} \in \{0, 1\}, \forall a, b, c, d, e, f
\end{align*}
\]

However, it seems very difficult to dig out any structural property of the optimal solution of IP1. Instead, we introduce a relaxed version of IP1, shown as IP3 below.

\[
\begin{align*}
(IP3) \quad \max & \sum_{a} \Delta(a_{a}) \\
\text{s.t.} & \sum_{a} a_{a} \leq 2L \\
& a_{a} \in \{1, 2, \ldots, L\}, \forall a
\end{align*}
\]
Note that IP3 uses a function $\Delta(w)$, which denotes the optimal solution of IP2 defined as below:

\[
(IP_2) \quad \max \sum_{b,c,d,e,f} I_{def} \cdot I_{be} \cdot I_{cf} \cdot x_{abc} \cdot y_{ad}
\]

s.t. \hspace{1em} \max\{\sum_{bc} x_{abc} \sum_d y_{ad}\} \leq \omega

\[
I_{def}, I_{be}, I_{cf}, x_{abc}, y_{ad} \in \{0, 1\}, \forall a, b, c, d, e, f
\]

IP2 is parameterized by $\omega$ 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 $R_3(C, F)$, subject to the constraint that at most $\omega$ 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 IP2 does not depend on the particular choice of $a$, which is why we write the optimal solution of IP2 as $\Delta(\omega)$. Also, it is obvious that $\Delta(\cdot)$ is a non-decreasing function. Then, IP3 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 IP1, IP3 be OPT1, OPT3, respectively. Because IP3 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 IP1 is also a feasible solution to IP3, so OPT1 $\leq$ OPT3. Next we construct a feasible solution of IP3 with the Cartesian product restriction above, and show that it is within a constant factor from OPT3, hence OPT1.

Set $w^* = \max \frac{\omega}{\Delta(\omega)} \leq \frac{\omega}{\Delta(\omega)}$. We choose $\frac{\omega}{\Delta(\omega)}$ distinct values arbitrarily from dom(A) and allocate $w^*$ tuples to each such $a$. For each $a$, we use the optimal solution of IP2 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 $W = \frac{\omega}{\Delta(\omega)} \cdot \Delta(w^*)$. We show that $W$ is a constant-factor approximation of OPT3, as below.

**Lemma 5.2.** $W \geq \frac{1}{\alpha} \cdot \text{OPT3} \geq \frac{1}{\alpha} \cdot \text{OPT1}$.

The proof of Lemma 5.2 is deferred to Appendix A.4, thus completing our whole proof.

Note that Lemma 5.1 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 $R_1(A, B, C)$, $R_1(A, D)$, $R_1(B, E)$, $R_3(C, F)$ are loaded in the form of Cartesian products. Denote the number of distinct values in dom(A), dom(B) loaded by the server as $\alpha$, $\beta$ respectively. The number of distinct values in dom(C), dom(D), dom(E), dom(F) loaded by the server are $\frac{1}{\beta^2}$, $\frac{1}{\beta}$, $\frac{1}{\alpha}$, $\frac{1}{\alpha \beta}$. Moreover, $\frac{1}{\beta^2} \leq \alpha, \beta \leq N^{1/3}$.

There are $L^6$ 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 $\frac{L^6}{N}$. More careful inspection reveals that the $L^2$ 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 $\exp(-\Omega(L^2))$.

For $A$, there are $\binom{N^{1/3}}{\alpha}^2 = O(N^{2/3})$ choices of loading $\alpha$ distinct values from dom(A). Similar argument can be applied to $B, C, D, E, F$. Over all values of $\alpha, \beta$, the number of choices in total is

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

By the union bound, the probability that any of the choices produces more than $2 \cdot \frac{L^3}{N}$ join results is at most $\exp(-\Omega(L^2)) + O(\frac{N^{2/3}}{\alpha \beta})$, which is exponentially small if $\frac{L^3}{N} \geq c_1 \cdot N^{1/3} \cdot \log N$ for some sufficiently large constant $c_1$. Rearranging, we get $L^2 \geq c_1 \cdot N^{1/3} \cdot \log N$. We know that $L = \Omega(N^{2/3})$, so this is true as long as $\left(\frac{N^{2/3}}{\alpha \beta}\right)^2 \geq c_2 \cdot N^{1/3} \cdot \log N$, for some sufficiently large constant $c_2$, or $N/\log^6 N \geq c_2 \cdot p^6$.

**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 $p \cdot \frac{L^3}{N} \geq N^2$, i.e., $L \geq N/p^{1/3}$.

We have completed the whole proof for Theorem 6.

### 5.2 Degree-two Joins

Our lower bound proof for the $\oplus$-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 5.3. The proof of Lemma 5.3 is given in Appendix A.6.

**Lemma 5.3.** For any reduced degree-two join $Q = (V, E)$, the following holds: (1) $\tau^* \geq \frac{\sqrt{\Delta}}{\alpha} \geq \rho^* \geq \rho^* = \Delta(e) \geq |E|$; (2) $\tau^* + \rho^* = |E|$; (3) The optimal fractional edge packing/covering admits half-integral solution; (4) if there exists no odd-length cycle\footnote{A cycle $(V, E)$ is defined as $V = \{v_1, v_2, \ldots, v_n\} \subseteq V$ and $E = \{e_i = (v_i, v_{i+1}) \mod n \} \cup \{1, 2, \ldots, n\}$. The length of a cycle $(V, E)$ is defined as $|E|$.}, the optimal fractional edge packing/covering admits integral solution.

However, not all degree-two joins fit for the lower bound framework, two additional conditions are captured in Definition 5.4. Before describing the conditions, we introduce some terminologies and notions first. In a hypergraph $Q = (V, E)$, let $\Gamma(e)$ be the set of neighbors of edge $e \in E$, i.e., $\Gamma(e) = \{e' \in E : e \cap e' \neq \emptyset\}$. A fractional vertex covering for $Q = (V, E)$ is a mapping $x$ from $V \cup \{0, +\infty\}$ such that $\sum_{e \in E} x_e \geq 1$ holds for each edge $e \in E$; and the optimal solution is to minimize the quantity $\sum_{e \in E} x_e$. In addition,
a vertex covering \(x\) is constant-small if \(\max_0 \leq x \leq 1 - \epsilon\) for some constant \(0 < \epsilon < 1\).

**Definition 5.4.** A degree-two join \(Q = (V, E)\) 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 \(|\{e \in E : \sum_{e \in e} x_e > 1\}| \leq 1\) for every \(e \in E\), where \(E' = \{e \in E : \sum_{e \in e} x_e > 1\}\).

Note that \(Q_{s_{}\_}\_3\) is an edge-packing-provable degree-two join. Obviously, there is no odd-length cycle, and a valid vertex covering \(x = x_A = x_B = \frac{1}{2}\) and \(x_D = x_E = x_F = \frac{3}{2}\), which is also used in the lower bound proof of Theorem 6. Some other examples of edge-packing-provable degree-two joins are given in Figure 7.

**Figure 7:** Examples of edge-packing-provable joins.

The detailed proof of Theorem 7 is deferred to Appendix A.7. Here, we only give some intuition why the three conditions can be put together for generalizing this framework to degree-two joins.

In Definition 5.4, if there is no odd-length cycle in \(Q\), it admits integral optimal edge packing \(r^*\) and covering \(p^*\), implied by Lemma 5.3. More specifically, there exists a partition \((E_{\alpha}, E_{\beta})\) of \(E\): \(E_{\alpha} = \{e \in E : p^*(e) = 1, r^*(e) = 0\}\) and \(E_{\beta} = \{e \in E : p^*(e) = 0, r^*(e) = 1\}\), for example, \(E_{\alpha} = \{e_1, e_2\}\) and \(E_{\beta} = \{e_3, e_4, e_5\}\) in \(Q_{s_{}\_}\_3\). Moreover, all edges in \(E_{\alpha}\) are vertex-disjoint, as well as edges in \(E_{\beta}\), due to the fact that each vertex appears in at most two edges.

Consider any optimal fractional vertex covering \(x\) satisfying (2) in Definition 5.4. Note that \(x\) defines another partition \((E', E'')\) of \(E\): \(E' = \{e \in E : \sum_{e \in e} x_e > 1\}\) and \(E'' = \{e \in E : \sum_{e \in e} x_e = 1\}\), for example, \(E' = \{e_1, e_2\}\) and \(E'' = \{e_3, e_4, e_5\}\) in \(Q_{s_{}\_}\_3\). Note that the fractional edge packing and vertex covering are prime-dual. Implied by the slackness theorem, \(E' \subseteq E_{\alpha}\). Edges in \(E'\) are also vertex-disjoint. The hard instance is constructed by \(x\).

More specifically, the domain of each attribute \(\sigma\) contains \(N^\sigma\) distinct values. Relations in \(E''\) are deterministically constructed as Cartesian products, containing \(N^\tau\) tuples exactly, while those in \(E'\) are probabilistically constructed. As \(|\{e \in E : \sum_{e \in e} x_e \leq 1\}| \leq 1\) holds for every \(e \in E\), each edge in \(E'\) derives a connected components \(C(e) = \{e' \in E' : e' \cap e \neq \emptyset\}\). More importantly, \(C(e_1) \cap C(e_2) = \emptyset\) for any pair of \(e_1, e_2 \in E'\). We then apply a similar argument for \(Q_{s_{}\_}\_3\) to each such component.

At last, the rationale behind the notion of “constant-small” is used to prove a upper bound on \(I(L)\) with exponentially high probability; and more details can be found in Appendix A.7.

**Remark.** We only give a sufficient condition in Theorem 7, and several questions remain to be answered, for example, (1) what is a complete characterization of cyclic queries on which our framework can be applied, (2) is there any matching upper bound on the degree-two joins. It is still unclear whether \(\text{\#cycles} = r^*\) holds for a degree-two join satisfying edge-packing-provable conditions. If this is the case, then the lower bound \(\Omega \left(\frac{N^\text{\#cycles}}{r^*}\right)\) will be matched by the existing one-round algorithm [19].

**REFERENCES**

A OMITTED PROOFS

A.1 α-acyclic join

An equivalent definition for α-acyclicity is based on the GYO reduction [1]: (1) if there is a vertex v ∈ V only appearing in edge e, then remove v from e; (2) if there is a pair of edges e, e′ ∈ E such that e ⊆ e′, then remove e from E. A join query Q = (V, 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 Lemma A.2, A.1, and A.3. For short, we use “acyclic” to denote “α-acyclic” below.

Lemma A.1. For any acyclic join Q = (V, E), the residual join Q_αx = (V - x, E_αx) is also acyclic, where E_αx = {e - x : e ∈ E}.

Proof. Let T be the join tree of Q, such that (1) there is a one-to-one correspondence between edges in E and nodes in T; (2) for any attribute v ∈ V, all nodes containing v form a connected subtree. We derive another tree T′ by removing attributes x from each node in T. It can be easily checked that T′ is a valid join tree for Q_αx, thus Q_αx is acyclic. □

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

Proof. Let ρ^* be the optimal edge covering for hypergraph Q = (V, E). An edge cover of Q can be obtained along with its GYO reduction. More specifically, if Q is empty set, we set ρ^* = 0. In general, we apply the following two procedures: (1) If attribute v ∈ V only appears in e, then assign e with weight 1 and remove all attributes in e from ‘V’. (2) If two distinct edges such that e ⊆ e′, then assign e with weight 0 and remove e′ from 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_α = (V - e, E_α) be the residual join by removing attributes in e from all relations in E. By hypothesis, let ρ^*(Q_α) be the integral optimal edge covering of Q_α. Note that ρ^*(Q) = ρ^*(Q_α) + 1, which is optimal since any edge cover require to assign ρ^*(e) = 1 to cover attribute e. Moreover, ρ^*(Q) also admits integral optimal edge covering, since ρ^*(Q_α) admits integral optimal edge covering.

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

A.2 Berge-acyclic join

We next introduce the berge-acyclic joins. For a join query Q = (V, E), consider the bipartite graph G, in which V corresponds to vertices on one side and E to vertices on the other side. There is an edge between v ∈ V and e ∈ E if v ∈ e. Then the hypergraph (V, 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 ∈ V, and any subgraph of (V, 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 = (V, 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_α, r^* ≤ ρ^* where r^*, 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 = (V, E). The base case is trivial that when E contains a single relation, with ρ^* = r^* = 1. In general, we consider two more cases:

Case 1. If Q is disconnected, let Q_1, Q_2, ..., Q_k be its connected components. As Q_α is also berge-acyclic for any i ∈ {1, 2, ..., k}, by hypothesis we have r^*(Q_i) ≤ ρ^*(Q_i). Observe that r^*(Q) = Σ_i r^*(Q_i) and ρ^*(Q) = Σ_i ρ^*(Q_i). Thus, r^*(Q) ≤ ρ^*(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 Γ(e_0) be the set of leaves connected with e_0. Note that each relation in Γ(e_0) include one attribute in e_0. Let E_α = {e ∈ E : e ∈ e_0} be the set of relations containing attribute x. Define

V′ = {v ∈ e_0 : e_0 ⊆ {e_0} ∪ Γ(e_0)}

Note that |V′| = 1; otherwise, e_0 is not a new leaf after removing all leaves in Γ(e_0). Define

S = {e ∈ Γ(e_0) : e ∩ e_0 ∈ 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, ρ^*(Q′) ≥ r^*(Q′) since Q′ is also a reduced berge-acyclic join. Note that ρ^*(Q) = ρ^*(Q′) + |S| and r^*(Q) ≤ r^*(Q′) + |S|. Thus, ρ^*(Q) ≤ r^*(Q).

Case 2.2. Otherwise, let Q′ be the residual query by removing all attributes of e_0 ⊆ S. By hypothesis, ρ^*(Q′) ≥ r^*(Q′) since Q′ is also a reduced berge-acyclic join. Note that ρ^*(Q) = ρ^*(Q′) + |S| and r^*(Q) ≤ r^*(Q′) + |S|. Thus, ρ^*(Q) ≥ r^*(Q). □

A.3 Missing Proofs in Section 4

Proof of Lemma 4.6. Without loss of generality, assume r = e_k. The base case with k = 1 always holds. Let x ∈ e_k+1 be the first attribute. Note that S^x = {e_1, e_2, ..., e_k-1}.

The residual join Q_k will be reduced first and then tackled by invoking the whole algorithm recursively. By hypothesis, each heavy instance R_α can be computed using O(∑S∈S(E_α)[Ψ(R, S, L)]) servers in O(1) rounds with load O(L). The total number of servers allocated for all heavy assignments is

\[\sum_{\alpha} \sum_{S \in S(E_\alpha)} [\Psi(T, R_\alpha, S, L)] \leq \sum_{\alpha} \sum_{S \in S(E_\alpha)} [\Psi(T, R_\alpha, S, L) + 2|E_\alpha| \sum_{i \in [k-1]} |R(e_i)|] \leq \sum_{i \in [k-1]} \frac{|R(e_i)|}{L} \leq \sum_{S \in S(E_\alpha) \not\supset \{e_k\}} \Psi(T, R, S, L)\]
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 \(S \in S(E_x)\) into two cases (\(S \cap \{e_1, e_2, \ldots, e_{k-1}\} = \emptyset\) or not), and the last inequality is implied by the definition of \(S(E)\).

The residual join \(Q_x\) degenerates to the base case with one relation \(e_k\). Then, for each light group \(I_j, Q_x(R_j)\) can be computed using \(O(\frac{2|E|}{L})\) servers in \(O(1)\) rounds with load \(O(L)\). The total number of servers allocated for all light groups is

\[
\sum_{j} \frac{|R(e_j)|}{L} \leq \sum_{i \in [k-1]} \frac{|R(e_i)|}{L^2} + \sum_{i \in [k-1]} \Psi(T, R, S, L)
\]

where the first inequality is implied by the fact that there are \(O(\sum_{i \in [k-1]} \frac{|R(e_i)|}{L})\) light groups.

Combining the analysis for heavy and light subqueries, we complete the whole proof for Lemma 4.6. \(\Box\)

**Proof of Lemma 4.8.** We will prove the complexity for a twig join excluding its root \(r\), with

\[
S(E) = (L_1 - \{r\}) \times L_2 \times \cdots \times L_{\ell}
\]

Let \(x\) be the first attribute appearing in the leaf \(e\). Denote \(P\) as the set of nodes lying on the path from \(r\) to \(e\).

The residual join \(Q_x\) will be reduced first and then tackled by invoking the whole algorithm recursively. By hypothesis, each heavy instance \(R_x\) can be computed using \(O(\sum_{S \in S(E_x)} |\Psi(T, R_x, S, L)|)\) servers in \(O(1)\) rounds with load \(O(L)\). The total number of servers allocated for all heavy assignments is

\[
\sum_{a} \sum_{S \in S(E_x)} |\Psi(T, R_x, S, L)| \leq \sum_{S \in S(E_x)} \sum_{a} |\Psi(T, R_x, S, L)| + 2|E| \sum_{i \in P} \frac{|R(e)|}{L}
\]

\[
\leq \sum_{\{L_1, \ldots, L_{\ell}\} \subseteq (L_1 - \{r\}) \times L_2 \times \cdots \times L_{\ell}} \sum_{S \cap \emptyset \neq \emptyset} \sum_{e \in P - \{r\}} \Psi(T, R, S \cup e, L)
\]

\[
+ \sum_{\{L_1, \ldots, L_{\ell}\} \subseteq (L_1 - \{r\}) \times L_2 \times \cdots \times L_{\ell}} \sum_{S \cap \emptyset = \emptyset} \Psi(T, R, S, L)
\]

\[
\leq 2|E| \sum_{i \in P} \frac{|R(e)|}{L} \leq \sum_{S \in S(E)} \Psi(T, R, S, L)
\]

where the first inequality is implied by the fact that there are \(O(\sum_{a} \sum_{S \in S(E_x)} |\Psi(T, R_x, S, L)|)\) heavy assignments, the second inequality is implied by distinguishing \(S\) into two cases (\(S \cap P = \emptyset\) or not), and the last inequality is implied by the definition of linear-cover.

Let \(\{T_1, T_2, \ldots, T_{\ell}\}\) be the set of connected subtrees by deleting nodes in \(P\) from \(T\). Let \(Q_y = \{V_y, E_y\}\) be the twig join defined on the subtree \(T_i\), excluding their individual roots in \(L_i\). For each light group \(I_j\), the \(Q_y(R_j)\) degenerates to the Cartesian products over \(Q_y(R_j)\)’s. By hypothesis, \(Q_y(R_j)\) can be computed using \(O(\sum_{S \in S(E_y)} |\Psi(T, R, S, L)|)\) servers in \(O(1)\) rounds with load \(O(L)\). The total number of servers allocated for all light groups is

\[
\sum_{j} \sum_{S \in S(E_y)} \sum_{t} |\Psi(T, R_j, S, L)|
\]

\[
\leq \sum_{t} \sum_{S \in S(E_y)} \sum_{j} |\Psi(T, R_j, S, L)| + \sum_{e \in P - \{r\}} \frac{|R(e)|}{L}
\]

\[
\leq \sum_{e \in P - \{r\}} \sum_{S \in S(E_y) \times S(E_1) \times \cdots \times S(E_\ell)} \frac{|R(e)|}{L} \leq \sum_{S \in S(E)} \Psi(T, R, S \cup \{e\}, L)
\]

where the first inequality is implied by the fact that there are \(O(\sum_{a} \sum_{S \in S(E_x)} |\Psi(T, R_x, S, L)|)\) light groups and the last inequality is implied by the fact that the union of linear-covers of \(T_1, T_2, \ldots, T_{\ell}\) together with \(P\) is still a valid linear-cover of \(T\).

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 4.8 follows. \(\Box\)

The proof of Theorem 3 follows directly from the primitive of reducing a join, the Case II of the generic algorithm in Section 3, Lemma 4.8 and Lemma 4.6 sequentially.

### A.4 Proof of Lemma 5.2

**Proof.** Suppose \(OPT_3\) chooses a set of values \(A^*\) from \(A\), 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 \(A(a, w_a) \geq A(w)\), otherwise inefficient. Denote the set of efficient values as \(A_1\) and inefficient values as \(A_2\). Note that for every efficient value \(a, w_a \leq \frac{L}{3N^{1/3}}\) 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 \(\sum_{a \in A_1} \frac{w_a}{3w^*} \cdot \Delta(w)\) 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 \(A^*_1\). The total number of tuples needed is at most \(\frac{1}{2} \sum_{a \in A_1} \frac{w_a}{3w^*} \leq \frac{L}{2N^{1/3}}\). The number of join results that can be reported is

\[
\sum_{a \in A_1} \frac{w_a}{3w^*} \cdot \Delta(w) = \frac{3}{3} \sum_{a \in A_1} w_a \frac{\Delta(w_a)}{w_a} = \frac{1}{3} \sum_{a \in A_1} \Delta(w_a)
\]

Next, we use \(\frac{L}{3w^*} \cdot \Delta(w)\) values each with \(x^*\) 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 \(\frac{L}{2N^{1/3}}\). Recall that \(w_a \leq \frac{L}{N^{1/3}}\) for each \(a \in A_2\). The number of join results that can be reported is

\[
\frac{L}{3w^*} \cdot \Delta(w) \geq \frac{3}{3} \cdot \Delta\left(\frac{L}{N^{1/3}}\right) = \frac{1}{3} \cdot \Delta\left(\frac{L}{N^{1/3}}\right) \geq \frac{1}{3} \sum_{a \in A_1} \Delta(w_a)
\]

where the rationale behinds the last inequality is that there are at most \(N^{1/3}\) values in \(A_1\) and there is \(\Delta\left(\frac{L}{N^{1/3}}\right) \geq \Delta(w)\) 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. \(\Box\)
A.5 Background knowledge for Join-Aggregate Query and Free-connex Query

We consider join-aggregate queries over annotated relations [10, 17] with one semiring. Let \((E, \circ, \oplus)\) 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 \(t \in Q(R)\) is:

\[
w(t) := \Theta_{t \in E} (e) \pi_{t \in E} w(t).
\]

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

\[
\left\{ (f_y, w(f_y)) : t \in \pi_{y} Q(R), w(t) = \Theta_{t \in E} (e) \pi_{t \in E} w(t) \right\}.
\]

In plain language, a join-aggregate query first computes the join \(Q(R)\) and the annotation of each join result, which is the \(\oplus\)-aggregate of the tuples comprising the join result. Then it partitions \(Q(R)\) into groups by their projection on \(y\); 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, \(\circ\) to be multiplication, and set \(w(t) = 1\) for all \(t\), then it becomes the COUNT(\(*\)) GROUP BY \(y\) query; in particular, if \(y = \emptyset\), the query computes \(|Q(R)|\). The join-project query \(\pi_y Q(R)\), 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 [5] 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 \(Q = (V, E)\) is a tree \(T\), where each node \(u \in T\) is a subset of \(V\), such that (1) for each attribute \(x \in V\), the nodes containing \(x\) are connected in \(T\); (2) for each hyperedge \(e \in E\), there exists a node \(u \in T\) such that \(e \subseteq u\); and (3) for each node \(u \in T\), there exists a hyperedge \(e \in E\) such that \(e \subseteq u\).

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

A.6 Proof of Lemma 5.3

The proof directly follows the fact that the dual of any reduced degree-two join is an ordinary binary-relation join [18, 24]. We can show more details for each property in Lemma 5.3.

For (1), assume \(f\) be the mapping from \(E\) to \([0, +\infty)\). Let \(f(e) = \frac{1}{2}\) for every \(e \in E\). 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 maximization of fractional edge packing and minimization of fractional edge covering, we get \(r^* = \frac{|E|}{2} \geq \rho^*\).

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

A result similar to 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.7 Proof of Theorem 7

Before diving into the proof of Theorem 7, we point out several important properties for degree-two joins satisfying edge-packable conditions.

If there is no odd-length cycle in \(Q\), it admits integral optimal edge packing \(\tau^*\) and covering \(\rho^*\), implied by Lemma 5.3. More specifically, there exists a partition \((E_a, E_b)\) of \(E\): \(E_a = \{e \in E : \rho^*(e) = 1, \tau^*(e) = 0\}\) and \(E_b = \{e \in E : \rho^*(e) = 0, \tau^*(e) = 1\}\). Consider any vertex \(v\) incident to two edges, \(e, e'\). There must be \(e \in E_a, e' \in E_b\) or \(e \in E_b, e' \in E_a\). This also implies that all edges in \(E_a\) are vertex-disjoint, as well as edges in \(E_b\).

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

**Lemma A.4.** Let \(E' = \{e \in E : \sum_{v \in E} x_v > 1\}\). For any edge \(e \in E'\), \(r^*(e) = 0\) and \(\rho^*(e) = 1\).

For each edge \(e \in E\), let \(\Upsilon(e)\) be the set of vertices appearing in the neighbor of \(e\), i.e., \(\Upsilon(e) = \bigcup_{v \in E} e' - v\). Two nice properties on the edges in \(E'\) are stated in Lemma A.5 and Lemma A.6.

**Lemma A.5.** Let \(E' = \{e \in E : \sum_{v \in E} x_v > 1\}\). For any edge \(e \in E'\), \(\sum_{v \in E} x_v + \sum_{v \in E} x_v = \gamma(e)\).

**Proof.** As \(r^*(e) = 0\) and \(\rho^*(e) = 1\), we have \(\gamma(e') = 1\) and \(\gamma(e') = 0\) 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.4, thus \(\sum_{v \in E} x_v = 1\). We also observe that \(\sum_{v \in E} x_v = \gamma(e)\). Thus, the left-hand-side of the target equation can be rewritten as:

\[
\sum_{v \in E} x_v = \sum_{e' \in \Gamma(e)} \sum_{v \in E} x_v = |\Gamma(e)|
\]

thus yielding the desired result. \(\square\)

**Lemma A.6.** Let \(E' = \{e \in E : \sum_{v \in E} x_v > 1\}\). \(\rho^* - r^* = |E'| - \sum_{e \in E' \setminus \sum_{v \in E} x_v}\).

**Proof.** By the duality theorem, \(r^* = \sum_{v \in E} x_v\). In this way, we can rewrite \(\sum_{e \in E} x_v = \sum_{e \in E} x_v\) as:

\[
\sum_{e \in E} \sum_{v \in E} x_v = \sum_{e \in E} \sum_{v \in E} x_v - \sum_{e \in E} \sum_{v \in E} x_v = \tau^* - (\rho^* - |E'|)
\]

thus yielding the desired result. \(\square\)

Now, we are able to prove Theorem 7. As it follows the same framework as Section 5.1, we will focus on addressing the difference in this non-trivial extension.

**Proof.** Similarly, we will show that with positive probability, an instance constructed this way will have a bounded \(L(L)\), the maximum number of join results a server can produce, if it loads at
most $L$ tuples from each relation. Then setting $p \cdot J(L) = \Omega((Q(R)))$ 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 $\prod_{e \in e} N^{x_v} = N^{\sum_{e \in e} x_v} = N$ tuples in total; and a probabilistic relation $R(e)$ is constructed in a probabilistic way, such that each combination $t \in \times_{e \in e} \text{dom}(e)$ has a probability of $p(e) = 1/N^{\sum_{e \in e} x_v}$ to form a tuples in $R(e)$, with $\prod_{e \in e} N^{x_v} \cdot p(e) = N$ 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 $N^{\rho L}$ 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 reasonable restriction on loading tuples.**

**Lemma A.7.** For any deterministic edge $e$, if $|\Gamma(e) \cap \mathcal{E}'| \leq 1$, then making the assumption 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 5.1 can be applied to $v$, i.e., loading tuples in $R(e)$ for $v \in e$ in terms of $L_v \times L_{v- \{v\}}$ will not decrease the optimal solution by a constant factor. Moreover, if $L_v = L_v \times L_{v- \{v\}}$, then $L_{v- \{v\}} = L_v \times L_u$ for any $u \in e - \{v\}$. To prove this result, it suffices to show that $L_{v- \{v\}} = L_v \times L_u$ for every pair of vertices $u, v \in e$. We distinguish any deterministic edge $e$ into two cases.

1. If $\Gamma(e) \cap \mathcal{E}' = \emptyset$, Lemma 5.1 can be applied to all vertices to $e$, thus $L_{v- \{v\}} = L_v \times L_u$ 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 5.1 can be applied to every attribute $v \in e - e''$, we have $L_{v- \{v\}} = L_v \times L_u$ for every $u \in e, v \in e - e''$. Now that when $|e'' \cap e| = 1$, we are done. The remaining case is when $|e'' \cap e| \geq 2$, we can assume that the optimal edge covering $\Gamma$ 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 5.1. In this way, the condition is also satisfied.

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

Assume the number of distinct values from attribute $v$ loaded by the server is $z_v$. Observe that $1 \leq z_v \leq N^{x_v}$ for each vertex $v \in \mathcal{V}$. Moreover, $\prod_{e \in e} z_v = L$ 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 $L^{\gamma \beta}$ combinations of results in total, where each tuple has a probability of

$$\prod_{e \in \mathcal{E}_\beta} p(e) = \prod_{e \in \mathcal{E}_\beta} \frac{1}{N^{\sum_{e \in e} x_v}} = N^{|\mathcal{E}_\beta| - \sum_{e \in e} x_v} = N^{\rho L - e'}$$

to form a valid join result, implied by Lemma A.6. The expected number of join results that can be produced by one server is $L^{\gamma \beta} \cdot N^{\rho L - e'}$. 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 $Y_e$ for each combination $t \in \times_{e \in e} L_e$, which follows the bernoulli distribution with parameter $p(e)$. Denote $Y(e) = \sum_{e \in e} Y_e$. Observe that there are $L^{|\mathcal{E}'| - \sum_{e \in e} x_v}$ independent random variables in the space $\times_{e \in e} L_e$. So,

$$E[Y(e)] = L^{|\mathcal{E}'| - \sum_{e \in e} x_v} \cdot p(e) = N \cdot \left( \frac{L}{N} \right)^{|\mathcal{E}'| - \sum_{e \in e} x_v}$$

where the last inequality is implied by Lemma A.5. 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 $Y = \prod_{e \in \mathcal{E}} Y(e)$. As mentioned, these $L^{\gamma \beta}$ combinations are not fully independent, as long as they share any same variable $y$. In fact, the number of independent combinations is $L^{\gamma \beta - |\mathcal{E}'|}$, since all combinations can be put into disjoint groups by the random variables shared and each group has exactly $L^{\gamma \beta - |\mathcal{E}'|}$ combinations. Thus, the probability that the server produces more than $2L^{\gamma \beta} \cdot L^{\gamma \beta - |\mathcal{E}'|}$ join results is at most

$$\Pr\left( Y \geq 2L^{\gamma \beta} \cdot \frac{L^{\gamma \beta - |\mathcal{E}'|}}{N^{\rho L - e'}} \right) \leq \sum_{e \in \mathcal{E}'} \Pr(\{Y(e) \geq 2 \cdot E[Y(e)]\}) \leq \exp(-\Omega(\min_{e \in \mathcal{E}'} E[Y(e)]))$$

Consider an arbitrary attribute $v \in \mathcal{V}$, there are $N^{x_v}$ 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

$$\prod_{v \in \mathcal{V}} \sum_{z_v = 1}^{N^{x_v}} \exp(\Omega(N^{z_v}))$$

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

$$\exp(-\Omega(\min_{e \in \mathcal{E}'} E[Y(e)])) + \Omega(\max_{e \in \mathcal{E}'} N^{x_v})$$

which is exponentially small if $\min_{e \in \mathcal{E}'} E[Y(e)] \geq c_1 \cdot \max_{e \in \mathcal{E}'} N^{x_v} \cdot \log N$ for some sufficiently large constant $c_1$, or

$$N \cdot \left( \frac{L}{N} \right)^{|\mathcal{E}'|} \geq c_1 \cdot \max_{e \in \mathcal{E}'} N^{x_v} \cdot \log N$$

where $\lambda(Q) = \min_{e \in \mathcal{E}'} E[Y(e)] - \sum_{e \in \mathcal{E}'} X_v \leq |\mathcal{E}'|$. Rearranging it,

$$L^{\lambda(Q)} \geq c_1 \cdot N^{\lambda(Q) - 1} \cdot \max_{e \in \mathcal{E}'} N^{x_v} \cdot \log N$$

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

$$\left( \frac{N}{p} \right)^{\lambda(Q)} \geq c_2 \cdot N^{\lambda(Q) - 1} \cdot \max_{e \in \mathcal{E}'} N^{x_v} \cdot \log N$$

for some sufficiently large constant $c_2$, or $N \geq c_3 \cdot p^{-\min_{e \in \mathcal{E}'} \lambda(Q)} \cdot (\log N)^{-\min_{e \in \mathcal{E}'} \lambda(Q)}$ for some sufficiently large constant $c_3$. Note that $x$ is constant-small, so $\lambda(Q) = O(|\mathcal{E}'|)$ 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 $O(L^{\gamma \beta} \cdot N^{\rho L - e'})$ 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 $N^{\rho^*}$ join results, we must have $pL^\tau \cdot N^{\rho^* - \tau^*} \geq N^{\rho^*}$, thus $L \geq N/p^{1/\tau^*}$. □