Parallel Acyclic Joins: Optimal Algorithms and Cyclicity Separation

XIAO HU, University of Waterloo, Waterloo, Canada
YUFEI TAO, Chinese University of Hong Kong, Hong Kong, China

We study equi-join computation in the massively parallel computation (MPC) model. Currently, a main open question under this topic is whether it is possible to design an algorithm that can process any join with load $O(N \text{ polylog } N/p^{1/\rho^*})$ — measured in the number of words communicated per machine — where N is the total number of tuples in the input relations, ρ^* is the join's fractional edge covering number, and p is the number of machines. We settle the question in the *negative* for the class of tuple-based algorithms (all the known MPC join algorithms fall in this class) by proving the existence of a join query with $\rho^* = 2$ that requires a load of $\Omega(N/p^{1/3})$ to evaluate. Our lower bound provides solid evidence that the "AGM bound" alone is not sufficient for characterizing the hardness of join evaluation in MPC (a phenomenon that does not exist in RAM). The hard join instance identified in our argument is cyclic, which leaves the question of whether $O(N \text{ polylog } N/p^{1/\rho^*})$ is still possible for acyclic joins. We answer this question in the *affirmative* by showing that any acyclic join can be evaluated with load $O(N/p^{1/\rho^*})$, which is asymptotically optimal (there are no polylogarithmic factors in our bound). The separation between cyclic and acyclic joins is yet another phenomenon that is absent in RAM. Our algorithm owes to the discovery of a new mathematical structure — we call "canonical edge cover" — of acyclic hypergraphs, which has numerous non-trivial properties and makes an elegant addition to database theory.

CCS Concepts: • **Theory of computation** → **Massively parallel algorithms**; • **Information systems** → **Join algorithms;**

Additional Key Words and Phrases: Joins, conjunctive queries, massively parallel computation, I/O-efficient, lower bounds

ACM Reference format:
Xiao Hu and Yufei Tao. 2024. Parallel Acyclic Joins: Optimal Algorithms and Cyclicity Separation. *J. ACM* 71, 1, Article 6 (February 2024), 44 pages.
https://doi.org/10.1145/3633512

1 INTRODUCTION

Join evaluation is an important problem at the core of database theory. The past decade has witnessed significant progress towards understanding the problem's complexity in the **random access machine (RAM)** model. For a join involving a constant number of attributes, Atserials, Grohe,

Preliminary versions of this article appeared in PODS'21 [11] and ICDT'22 [31]. This work was supported in part by GRF projects 14207820, 14203421, and 14222822 from HKRGC.
Authors' addresses: X. Hu, University of Waterloo, 200 University Ave W, Waterloo, ON N2L 3G1, Canada; Y. Tao, Chinese University of Hong Kong, Shatin, New Territories, Hong Kong, China.

0004-5411/2024/02-ART6 $15.00
https://doi.org/10.1145/3633512

and Marx proved in their seminal work [6] that the join result can include only $O(N^{\rho^*})$ tuples, where N is the number of tuples in the input relations, and ρ^* is the join's fractional edge covering number.[1] The bound—commonly known as the *AGM bound*— is tight in the sense that a join can indeed return $\Omega(N^{\rho^*})$ tuples in the worst case. An algorithm, therefore, is *worst-case optimal* if it can process any join in $O(N^{\rho^*})$ time. Many algorithms whose running time matches this bound—sometimes up to an $\tilde{O}(1)$ factor, where $\tilde{O}(.)$ hides a polylog N term—have been discovered [5, 18, 21–25, 33].

In big-data analysis, the input relations may not fit in one machine's memory, and therefore, joins are often processed with multiple machines on a massively parallel system like MapReduce [9], Spark [35], Hive [32], Dremmel [20], and the like. CPU calculation is no longer the performance bottleneck in those environments. The new bottleneck, instead, is network communication, because of which the design of "massive join" algorithms has focused on the **massively parallel computation (MPC)** [8] model (to be formally defined in Section 1.1). Unraveling the worst-case complexity of join evaluation in MPC, however, has turned out to be an intriguing challenge. On the one hand, the AGM bound implies [19] a lower bound of $\Omega(N/p^{1/\rho^*})$ on the cost of any MPC algorithm—measured in the number of words communicated per machine—where p is the number of machines. On the other hand, despite significant efforts [2, 3, 8, 13, 14, 16, 17, 19, 27, 30], no known algorithms have been able to match this bound.

In this article, we will *disprove* the possibility of any MPC algorithm that can ensure cost $\tilde{O}(N/p^{1/\rho^*})$ for arbitrary joins. We will establish a new, higher, lower bound, thereby revealing the somewhat surprising fact that the join problem (unlike in RAM) cannot be characterized by the AGM bound alone in MPC. Furthermore, we will contrast the lower bound by developing an optimal algorithm with cost $O(N/p^{1/\rho^*})$ for "acyclic joins", which form a class of joins with profound importance in database systems [1, 8, 13, 15, 34]. The separation between acyclic and cyclic joins is another characteristic of the join problem that does not exist in RAM.

1.1 Problem Definitions and Complexity Parameters

Natural Joins. Let **att** be a set where each element is called an *attribute*, and **dom** be another set where each element is called a *value*. The concrete choice of **dom** is unimportant, although each value in **dom** should occupy only a constant number of words. We assume a total order on **dom** (if necessary, manually impose one by ordering the values arbitrarily). A *tuple* over a set $U \subseteq$ **att** is a function $\boldsymbol{u} : U \rightarrow$ **dom**. For each attribute $X \in U$, we refer to $\boldsymbol{u}(X)$ as the *value* of $\boldsymbol{u}$ on X. Given a subset $U' \subseteq U$, define $\boldsymbol{u}[U']$ as the tuple $\boldsymbol{u'}$ over U' such that $\boldsymbol{u'}(X) = \boldsymbol{u}(X)$ for every $X \in U'$. A *relation* is a set R of tuples over the same set U of attributes; we call U the *scheme* of R, a fact denoted as $scheme(R) = U$. Given a subset $U \subseteq scheme(R)$, the *projection of R on U* — denoted as $\pi_U(R)$ — is a relation with scheme U defined as $\pi_U(R) = \{\text{tuple } \boldsymbol{u} \text{ over } U \mid \exists \text{ tuple } \boldsymbol{v} \in R \text{ s.t. } \boldsymbol{u}[U] = \boldsymbol{v}[U]\}$.

We represent a *join query* (henceforth, simply a *join* or *query*) as a set Q of relations. Define $attset(Q) = \bigcup_{R \in Q} scheme(R)$. The query result is the following relation over $attset(Q)$:

$$\mathcal{J}oin(Q) \quad = \quad \{\text{tuple } \boldsymbol{u} \text{ over } attset(Q) \mid \forall R \in Q, \boldsymbol{u}[scheme(R)] \in R\}\,. \tag{1}$$

We refer to $|\mathcal{J}oin(Q)|$ as the *output size* of Q. If the relations in Q are $R_1, R_2, \ldots, R_{|Q|}$, we may also represent $\mathcal{J}oin(Q)$ as $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_{|Q|}$.

The query Q can be characterized by a *schema graph* $G = (V, E)$, which is a hypergraph where each vertex in V is a distinct attribute in $attset(Q)$, and each edge in E is the scheme of a distinct

[1] A formal definition of ρ^* will appear in Section 1.1. For our discussion here, it suffices to understand ρ^* as a value at least 1.

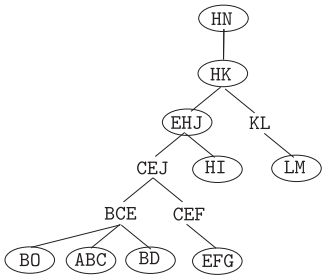


Fig. 1. An edge tree example.

relation in Q. The set E may contain identical edges because two (or more) relations in Q can have the same scheme. The term "hyper" suggests that an edge can have more than two attributes.

A query Q is *acyclic* if its schema graph is acyclic. Specifically, a hypergraph $G = (V, E)$ is *acyclic* if we can create a tree T where

- every node in T stores—hence, "corresponds to"—a distinct edge in E;
- *(the connectedness requirement)* for every attribute $X \in V$, the set of nodes whose corresponding edges contain X forms a connected subtree in T.

We will call T an *edge tree* of G. We say that Q is *cyclic* if it does not satisfy the above conditions.

Example 1.1. Consider the hypergraph $G = (V, E)$ where $V = \{\mathsf{A}, \mathsf{B}, \ldots, \mathsf{O}\}$ and $E = \{\mathsf{ABC}, \mathsf{BD}, \mathsf{BO}, \mathsf{EFG}, \mathsf{BCE}, \mathsf{CEF}, \mathsf{CEJ}, \mathsf{HI}, \mathsf{LM}, \mathsf{EHJ}, \mathsf{KL}, \mathsf{HK}, \mathsf{HN}\}$. Figure 1 shows an edge tree T of G (which is therefore acyclic). To understand the connectedness requirement, observe the connected subtree formed by the five edges involving E.

We use

$$N = \sum_{R \in Q} |R| \tag{2}$$

to denote the *input size* of Q, namely, the total number of tuples in the relations participating in the join. Our discussion focuses on *data complexities*, that is, we are interested in the influence of N on the algorithm performance. For that reason, we assume that the schema graph G of Q has $O(1)$ vertices, i.e., $|attset(Q)| = O(1)$.

Computation Model. The **MPC (massively parallel computation)** model [8] has been widely deployed to design parallel algorithms on large-scaled data [2, 3, 8, 13, 14, 16, 17, 19, 27, 29, 30]. In this model, we have p share-nothing machines that are interconnected in a network. In the beginning, each machine stores $O(N/p)$ tuples from the relations of a query Q. An algorithm starts by having each machine perform some *initial computation* on its local data and then executes in *rounds*, each having two phases:

- in the first phase, the machines exchange messages (every message should have been prepared either in the initial computation or the second phase of the previous round);
- in the second phase, each machine performs local computation.

An algorithm is required to finish in a constant number of rounds, and when it does, every tuple in $\mathcal{J}oin(Q)$ is required to have been produced on at least one machine. The *load of a round* is the largest number of words received by a machine in that round. The *load of an algorithm* is the maximum load of all the rounds. We consider $p < N^{1-\epsilon}$, where $\epsilon > 0$ can be an arbitrarily small constant; this is a standard assumption behind all the previous work on MPC. With load N, any problem can be solved trivially in one round by simply sending all data to one machine. The

crux of designing a load-efficient MPC algorithm is to limit the "intermediate results" that need to be transmitted across machines.

We will confine our attention to the class of *tuple-based* algorithms, which treat tuples in the relations of Q as "atoms" that must always be transmitted in their entirety. Atoms are allowed to be copied, but each copy must again be sent in its entirety. To report a result tuple $\boldsymbol{u} \in \mathcal{J}oin(Q)$, a machine must have received all the atom tuples $\boldsymbol{u}[scheme(R)]$ for every $R \in Q$. While the tuple-based class of algorithms does not encompass all possible approaches, it does include the existing MPC join algorithms that we are aware of, which will be discussed in Section 1.2. Therefore, analyzing the optimal communication complexity achievable by this class can provide valuable insights into the problem's characteristics.

Results are reported by invoking a special zero-cost function $emit(.)$. In particular, the machine, which has received all the necessary atom tuples for a result tuple $\boldsymbol{u} \in \mathcal{J}oin(Q)$, outputs $\boldsymbol{u}$ by employing $emit(\boldsymbol{u})$, with the stipulation that $\boldsymbol{u}$ can be output only once (across all machines) throughout the algorithm's execution. This reporting "style" reflects how join results are usually consumed in database systems: they could be (i) transmitted to a remote server via the network, (ii) written to a certain type of persistent storage, or (iii) directly supplied to a downstream process, such as an aggregate function like counting or a user-defined utility function. From the MPC model's perspective, incorporating the $emit(.)$ function effectively absolves the algorithm from the responsibility of storing the tuples of the join result. Generally, the size of $\mathcal{J}oin(Q)$ could be polynomial in N (and exponential in $|Q|$, which is regarded as a constant in this article), making $|\mathcal{J}oin(Q)|/p$ potentially much larger than a machine's memory capacity. In contrast, an MPC algorithm should utilize far less than N memory on each machine: ideally, the memory usage on each machine should be at the same order as the algorithm's load.

Our lower bounds are combinatorial in nature. We count only how many atom tuples must be communicated in order to emit all the tuples in the join result, while any other information can be communicated for free.

Fractional Edge Coverings and Packings. Consider a query Q — which may or may not be acyclic — with schema graph $G = (V, E)$. Let W be a function that associates every edge $e \in E$ with a real-valued *weight* $W(e)$ between 0 and 1. The function is called a *fractional edge covering* of G if

$$\sum_{e \in E: X \in e} W(e) \geq 1$$

holds for every attribute $X \in V$, namely, the total weight of all the edges covering X is *at least* 1. Similarly, W is a called a *fractional edge packing* of G if

$$\sum_{e \in E: X \in e} W(e) \leq 1$$

holds for every attribute $X \in V$, namely, the total weight of all the edges covering X is *at most* 1. In any case, we refer to $\sum_{e \in E} W(e)$ as the *total weight* of W.

The *fractional edge covering number* of G (also of Q) — denoted as ρ^* — is the minimum total weight of all possible fractional edge coverings of G. The *fractional edge packing number* of G (also of Q)—denoted as τ^* —is the maximum total weight of all possible fractional edge packings of G. A fractional edge covering (respectively, packing) is *optimal* if its total weight equals ρ^* (respectively, τ^*).

1.2 Previous Results

AGM Bound and Join Algorithms in RAM. Consider an arbitrary join query Q whose schema graph $G = (V, E)$ admits a fractional edge covering W. For each edge $e \in E$, let R_e be the (only) relation

in Q whose scheme corresponds to e. The AGM bound [6] states that $\mathcal{J}oin(Q)$ can contain no more than $\prod_{e\in E}|R_e|^{W(e)}$ tuples. Applying the trivial fact $|R_e| \leq N$ (where N is the input size of Q) and supplying an optimal fractional edge covering W, we obtain $|\mathcal{J}oin(Q)| \leq N^{\rho^*}$. This inequality is asymptotically tight because, for any hypergraph $G = (V, E)$ where V has a constant size, there exists a join query Q with schema graph G whose $\mathcal{J}oin(Q)$ has $\Omega(N^{\rho^*})$ tuples [6].

An algorithm able to answer Q using $O(N^{\rho^*})$ time in the RAM model is considered worst-case optimal because when $|\mathcal{J}oin(Q)| = \Omega(N^{\rho^*})$, we need $\Theta(N^{\rho^*})$ time even just to output $\mathcal{J}oin(Q)$. Ngo et al. [23] designed the first algorithm that guarantees a running time of $O(N^{\rho^*})$ for all queries.[2] Since then, the community has discovered more algorithms [5, 18, 21–25, 33] that are all worst-case optimal (sometimes up to an $\tilde{O}(1)$ factor) but differ in their own features. When Q is acyclic, optimal efficiency can be achieved using a simpler algorithm due to Yannakakis [34].

Join Algorithms in MPC (and the Quest for Load $\tilde{O}(N/p^{1/\rho^*})$). Via a reduction from the set-disjointness problem in communication complexity, Hu et al. [14] showed that $\Omega(N/p)$ is a lower bound on the load of join evaluation in MPC.[3] Separately, Koutris et al. [19] observed that, the AGM bound implies another lower bound of $\Omega(N/p^{1/\rho^*})$ on the load. To understand why, suppose that each machine sees at most L atom tuples (i.e., tuples in the input relations of Q) during the entire algorithm. By the AGM bound, the machine can produce at most L^{ρ^*} tuples in the join result. Thus, when $|\mathcal{J}oin(Q)| = \Omega(N^{\rho^*})$, we must have $p \cdot L^{\rho^*} = \Omega(N^{\rho^*})$, which yields $L = \Omega(N/p^{1/\rho^*})$. For $\rho^* > 1$ (the case of $\rho^* = 1$ has been captured by the lower bound of [14]), $N/p^{1/\rho^*} \gg N/p$ such that at least $\Omega(N/p^{1/\rho^*})$ of the tuples seen by a machine need to come from other machines, suggesting that the algorithm's load must be $\Omega(N/p^{1/\rho^*})$.

The above negative results have motivated considerable research looking for MPC algorithms whose loads are bounded by $\tilde{O}(N/p^{1/\rho^*})$; such algorithms are worst-case optimal up to an $\tilde{O}(1)$ factor. The goal has been realized only on four query classes. The first consists of all the Cartesian-product joins where the relations in Q have disjoint schemes; see [3], [7], and [17] for several optimal algorithms on such queries. The second is the so-called *Loomis-Whitney join*, where E consists of all the $|V|$ possible edges of $|V| - 1$ attributes; see [19] for an optimal algorithm for such queries. The third class includes every join where each relation has at most two attributes; see [16], [17], [27], and [30] for optimal algorithms for these queries. The fourth class comprises all the so-called *r-hierarchical joins*, which constitute a subset of the acyclic queries considered in this article; see [13] for an optimal r-hierarchical algorithm.

We refer the reader to (i) [7] and [19] for join algorithms that perform only a single round, and (ii) [2], [13], and [14] for algorithms whose loads are sensitive to the join size $|\mathcal{J}oin(Q)|$ and hence can be even lower than $\Omega(N/p^{1/\rho^*})$ when the join result is small.

1.3 Contributions

New Results. Our first result eliminates the possibility of answering an arbitrary join query with load $\tilde{O}(N/p^{1/\rho^*})$ in the MPC model. Specifically, we prove (in Theorem 1) the existence of a cyclic query Q with fractional edge covering number $\rho^* = 2$ and fractional edge packing number $\tau^* = 3$, such that any algorithm solving the query must incur a load of $\Omega(N/p^{1/3})$ when $p = O(N^{1/3})$. This offers solid evidence that, unlike in RAM, the AGM bound alone is insufficient to characterize the performance of join queries in MPC.

[2]The algorithm proposed in [23] achieves this time complexity by using a preprocessing step that creates perfect-hashing data structures on the input relations. This step takes $O(N)$ expected time. Without preprocessing, the algorithm has an expected time complexity of $O(N^{\rho^*})$ or a worst-case time complexity of $O(N^{\rho^*}\log N)$.
[3]The lower bound of [14] holds even on algorithms that are not tuple-based.

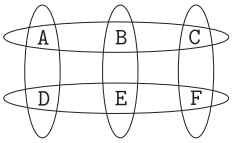


Fig. 2. The schema graph of boat joins.

Given this new finding, a natural question is whether cyclicity is the "culprit" for the above, somewhat bizarre, MPC characteristic. We answer the question in the affirmative. Specifically, we prove (in Theorem 15) that every acyclic query can be evaluated with load $O(N/p^{1/\rho^*})$ in MPC, which is asymptotically optimal (note that the load complexity does not hide any polylogarithmic factors). This officially separates the class of acyclic queries from the class of cyclic queries in MPC (recall that no such separation exists in RAM). Our algorithm uses $O(N/p^{1/\rho^*})$ memory on every machine.

Our Techniques. The cyclic query behind our lower bound has a schema graph illustrated in Figure 2 (every letter is a vertex and every ellipse is an edge). A join having this schema graph—which we will refer to as a *boat join* —has five relations with schemes ABC, DEF, AD, BE, and CF, respectively (the reader should take a moment to verify its fractional edge covering number 2 and fractional edge packing number 3). The crux of our proof is to construct a boat join Q with a special property: *for* $L = \Omega(N^{5/6})$, *any* L *"atom tuples" from the input relations can produce* $O(L^3/N)$ *tuples in the join result.* To contrast this property with the AGM bound, we note that any L atom tuples can produce at most L^2 result tuples under the AGM bound. Because $L^3/N = o(L^2)$ for $L = o(N)$, each machine, if permitted to see only L atom tuples, can actually produce fewer result tuples than predicted by the AGM bound. This is the rationale behind our stronger lower bound.

Our construction has a deeper implication. The hard join query Q described earlier produces $\Theta(N^2)$ join tuples, asymptotically the largest possible size asserted by the AGM bound. However, unlike in RAM where (for proving lower bounds) it suffices to look at the size of the *global* join result, our techniques suggest that in MPC it is imperative to look at the maximum size of *local* joins — namely, how many result tuples can be produced by $L \ll N$ tuples only. The AGM bound can be very loose in bounding the local join sizes, which is the core reason why it does not (fully) characterize the join performance in MPC.

To develop our optimal MPC algorithm for acyclic queries, we present a theory of acyclic hypergraphs revolving around a new concept "canonical edge cover". To pave the way for the concept, we first prove that any acyclic hypergraph $G = (V, E)$ admits an *integral* optimal fractional edge covering W, namely, W assigns every edge in E an integer weight: either 0 or 1. This fact allows us to connect W to edge "covers": a subset $S \subseteq E$ is an *edge cover*[4] of G if every attribute of V appears in at least one edge of S. Thus, the fractional edge covering number ρ^* of G is simply the minimum size of all edge covers, namely, the smallest number of edges that we must pick to cover all the attributes.

A hypergraph G can have multiple optimal edge covers (all with size ρ^*), among which we identify one as "canonical". In Figure 1, the nine circled nodes constitute a canonical edge cover $\mathcal{F}$ of G. Let us give an informal explanation on the derivation of $\mathcal{F}$. After rooting the tree in Figure 1 at HN, we add to $\mathcal{F}$ all the leaf nodes: BO, ABC, BD, EFG, HI, and LM. Next, we process the non-leaf nodes bottom up. At BCE, we ask: which attributes will disappear as we ascend further in the tree? The answer is B, which is thus a "disappearing" attribute of BCE. Then, we ask: does $\mathcal{F}$ already

[4]In case the reader is wondering, the literature uses the words "covering" and "cover" exactly the way they are used in our article.

cover B? The answer is yes, due to the existence of BO; we therefore do *not* include BCE in $\mathcal{F}$. We continue to process CEF and CEJ similarly, but neither of them enters $\mathcal{F}$. At EHJ, the disappearing attributes are E and J. In general, as long as one disappearing attribute has not been covered by $\mathcal{F}$, we pick the node; this is why EHJ is in $\mathcal{F}$. The other nodes HK and HN in $\mathcal{F}$ are chosen based on the same reasoning.

We show that a canonical edge cover determined this way has appealing properties, which eventually lead to a recursive strategy for evaluating any acyclic join optimally in MPC. At a high level, our algorithm works by simplifying G into several "residual" hypergraphs, each of which defines a sub-query to be computed recursively. Apart from some trivial modifications (such as removing the attributes and edges that have become irrelevant), *a canonical edge cover of G remains canonical on every residual hypergraph.* We utilize this crucial property to relate the load of the original query to the loads of the sub-queries, which yields an unusual recurrence whose solution proves an overall load of $O(N/p^{1/\rho^*})$. Canonical edge cover is, we believe, an elegant addition to database theory and finds further applications. In fact, by adapting our MPC algorithm to the *external memory* model [4], we can obtain an I/O-efficient algorithm for evaluating any acyclic join in $O(\frac{N^{\rho^*}}{M^{\rho^*-1}B}\log_{M/B}\frac{N}{B})$ I/Os, which improves several existing algorithms [12, 19, 26].

Remark. Gottlob et al. [10] proved that detecting whether an acyclic query has an empty result is LOGCFL-complete, even if the number of relations in the query is not constant. Their result has important implications, such as the ability to solve the problem in a polylogarithmic number of steps on an EREW PRAM with a polynomial number of processors. However, our work on parallel evaluation of acyclic queries focuses on minimizing cross-machine communication, which is different from the goal of [10] to reduce concurrent computation steps. Thus, our findings are not directly comparable to theirs.

2 A LOWER BOUND FOR BOAT JOINS

In this section, we will focus on boat joins, which have the schema graph in Figure 2. Our main result is:

Theorem 1. *For any sufficiently large integers of n and p satisfying $p \leq c \cdot n^{1/3}$, where $c > 0$ is a fixed constant, there is a boat join Q with input size $N = \Theta(n)$ (see (2) for the definition of input size) such that any tuple-based MPC algorithm computing $\mathcal{J}oin(Q)$ must incur a load of $\Omega(N/p^{1/3})$ when the number of machines is p.*

Recall that a boat join has fractional edge covering number $\rho^* = 2$ and fractional edge packing number $\tau^* = 3$. Hence, the theorem indicates that the join's load can exceed $O(N/p^{1/\rho^*})$ and reach $\Omega(N/p^{1/\tau^*})$. The theorem is tight up to an $\tilde{O}(1)$ factor because there are algorithms [19, 27] able to evaluate any boat join with load $\tilde{O}(N/p^{1/3})$.

Given a join Q, (as before) we use the term, *atom tuple,* to refer to a tuple in the input relations of Q. The core of our argument is to prove:

Lemma 2. *For any sufficiently large integers of n and L satisfying $L \geq c' \cdot n^{5/6}$, where $c' > 0$ is a fixed constant, there is a boat join Q with input size $N = \Theta(n)$ such that $|\mathcal{J}oin(Q)| = \Theta(n^2)$ and any L atom tuples can produce $O(L^3/n)$ result tuples in $\mathcal{J}oin(Q)$.*

Theorem 1 is in fact a corollary of Lemma 2 and the standard counting argument reviewed in Section 1.2. Consider the boat join Q given in the lemma. Let L be the maximum number of atom tuples that a machine sees during the entire evaluation of Q. As $\mathcal{J}oin(Q)$ has $\Theta(n^2)$ tuples, we know $L = \Omega(n/p^{1/\rho^*}) = \Omega(n/\sqrt{p})$ from the argument of [19] (see Section 1.2). To prove Theorem 1, we consider $p \leq c \cdot n^{1/3}$ for a sufficiently large constant $c > 0$ to satisfy the requirement $L^2 = \Omega(n^2/p) \geq c' \cdot n^{5/3}$, where c' is the constant stated in Lemma 2. The lemma then tells us that each

machine can generate $O(L^3/n)$ result tuples. To produce all the $\Theta(n^2)$ tuples in $\mathcal{J}oin(Q)$, we need $p \cdot O(L^3/n) = \Theta(n^2)$, which gives $L = \Omega(n/p^{1/3}) = \Omega(N/p^{1/3})$.

The rest of the section serves as a proof of Lemma 2. Given an integer $k \geq 1$, we denote by $[k]$ the set of integers $\{1, 2, \ldots, k\}$. Fix integers n and L satisfying the condition $L^2 \geq c' \cdot n^{5/3}$, where the constant c' will be chosen later in the proof. We consider, w.l.o.g., that $n^{1/3}$ is an integer. A boat join, as shown in Figure 2, has attributes $\mathsf{A}, \mathsf{B}, \ldots,$ and F. We design their domains to be

$$\mathbf{dom}(\mathsf{A}) = \mathbf{dom}(\mathsf{B}) = \mathbf{dom}(\mathsf{C}) = [n^{1/3}]$$
$$\mathbf{dom}(\mathsf{D}) = \mathbf{dom}(\mathsf{E}) = \mathbf{dom}(\mathsf{F}) = [n^{2/3}].$$

Recall that a boat join has five relations: $R_{\mathsf{ABC}}, R_{\mathsf{DEF}}, R_{\mathsf{AD}}, R_{\mathsf{BE}}$, and R_{CF}. Henceforth, for each edge $e \in \{\mathsf{ABC}, \mathsf{DEF}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$ in the schema graph, we use R_e to denote the relation with scheme e. Furthermore, for any $\{X_1, \ldots, X_k\} \subseteq \{\mathsf{A}, \mathsf{B}, \ldots, \mathsf{E}\}$ where $k \geq 2$, we use $\mathbf{dom}(X_1) \times \cdots \times \mathbf{dom}(X_k)$ to denote the "Cartesian product" relation that contains $\prod_{i=1}^{k} |\mathbf{dom}(X_i)|$ tuples such that, for any $(x_1, \ldots, x_k) \in \mathbf{dom}(X_1) \times \cdots \times \mathbf{dom}(X_k)$, the relation has a tuple $\boldsymbol{u}$ with $\boldsymbol{u}(X_i) = x_i$ for every $k \in [i]$.

We will construct a *set* — denoted as $\mathcal{Q}_{\text{boat}}$ — of boat joins. All those joins have precisely the same $R_{\mathsf{ABC}}, R_{\mathsf{AD}}, R_{\mathsf{BE}}$, and R_{CF}, but differ in R_{DEF}. Specifically, $R_{\mathsf{ABC}} = \mathbf{dom}(\mathsf{A}) \times \mathbf{dom}(\mathsf{B}) \times \mathbf{dom}(\mathsf{C})$, $R_{\mathsf{AD}} = \mathbf{dom}(\mathsf{A}) \times \mathbf{dom}(\mathsf{D})$, $R_{\mathsf{BE}} = \mathbf{dom}(\mathsf{B}) \times \mathbf{dom}(\mathsf{E})$, and $R_{\mathsf{CF}} = \mathbf{dom}(\mathsf{C}) \times \mathbf{dom}(\mathsf{F})$. Note that these four relations have exactly n tuples each. It remains to clarify R_{DEF} (the relation that distinguishes different boat joins). In every boat join $Q \in \mathcal{Q}_{\text{boat}}$, the relation $R_{\mathsf{DEF}} \in Q$ is a subset of the Cartesian-product relation $\mathbf{dom}(\mathsf{D}) \times \mathbf{dom}(\mathsf{E}) \times \mathbf{dom}(\mathsf{F})$. The number of possible subsets is 2^{n^2}, which is exactly the number of boat joins in $\mathcal{Q}_{\text{boat}}$, each using a distinct subset as its R_{DEF}. For every join $Q \in \mathcal{Q}_{\text{boat}}$, the result $\mathcal{J}oin(Q)$ is always $R_{\mathsf{ABC}} \times R_{\mathsf{DEF}}$.

We will show that at least one of the joins in $\mathcal{Q}_{\text{boat}}$ possesses the properties in Lemma 2. Our proof will proceed in two steps. First, we will reveal an intrinsic property of the boat joins in $\mathcal{Q}_{\text{boat}}$ regarding how to select a designated number of tuples to maximize the number of result tuples. The second step will then utilize the property to find a hard boat join to establish Lemma 2.

2.1 Maximizing the Size of a Local Join

This subsection will concentrate on an arbitrary boat join $Q \in \mathcal{Q}_{\text{boat}}$ and, therefore, every mention of "R_{DEF}" refers to *the* relation R_{DEF} in Q (remember R_{DEF} can be any subset of $\mathbf{dom}(\mathsf{D}) \times \mathbf{dom}(\mathsf{E}) \times \mathbf{dom}(\mathsf{F})$). We now define a combinatorial optimization problem crucial to our analysis:

> *Local-Join Maximiztion.* Given a boat join $Q \in \mathcal{Q}_{\text{boat}}$ and an arbitrary integer $L \geq 1$, choose $R'_e \subseteq R_e$ for each $e \in \{\mathsf{ABC}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$ to maximize the output size of the *local join* $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ subject to the constraint that each of the relations $R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}$, and R'_{CF} contains at most L tuples. We will represent the above problem as LJM(L).

Note that the size-L constraint concerns only the edges ABC, AD, BE, and CF, while the entire R_{DEF} participates in the local join. Define

$$\text{OPT}(Q, L) \quad = \quad \text{the maximum output size of all possible local joins in LJM}(L). \tag{3}$$

Solving the LJM problem exactly is challenging. However, as will become evident in Section 2.2, it suffices to find a way to approximate $\text{OPT}(Q, L)$ within a constant factor. For that purpose, we can restrict our attention to $R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}$, and R'_{CF} that conform to a special form. In general, a relation R with scheme $\{X_1, X_2, \ldots, X_k\}$ (for some $k \geq 1$) is said to be in the *Cartesian-product form* (CP-form) if $R = \pi_{X_1}(R) \times \pi_{X_2}(R) \times \cdots \times \pi_{X_k}(R)$, namely, R is the Cartesian product of its projections on the k attributes. We now define a variant of the LJM problem:

Local-Join Maximization with Cartesian Products. Given a boat join $Q \in \mathcal{Q}_{\text{boat}}$ and an arbitrary integer $L \geq 1$, choose $R'_e \subseteq R_e$ for each $e \in \{\mathsf{ABC}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$ to maximize the output size of the *local join* $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ subject to the constraint that each of the relations $R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}$, and R'_{CF} (i) contains at most L tuples, and (ii) is in the CP-form. We will represent the above problem as LJM-CP(L).

Define

$$\text{OPT}_{\text{CP}}(Q, L) = \text{the maximum output size of all possible local joins in LJM-CP}(L). \tag{4}$$

The lemma below, whose proof is deferred to Section 2.3, gives a crucial relationship between the functions in Equations (3) and (4).

Lemma 3. *For any boat join $Q \in \mathcal{Q}_{boat}$, $\text{OPT}_{\text{CP}}(Q, 8L) \geq \frac{1}{128} \cdot \text{OPT}(Q, L)$.*

2.2 Identifying a Hard Boat Join

In this subsection, we will prove the existence of a boat join $Q \in \mathcal{Q}_{\text{boat}}$ such that

- Q has an input size $\Theta(n)$,
- $|\mathcal{J}oin(Q)| = \Theta(n^2)$, and
- $\text{OPT}_{\text{CP}}(Q, 8L) = O(L^3/n)$,

as long as n is sufficiently large and $L \geq c' \cdot n^{5/6}$ for some constant c' to be chosen later. It follows from Lemma 3 that $\text{OPT}(Q, L) = O(L^3/n)$. By definition of LJM($L$) and the meaning of $\text{OPT}(Q, L)$ (see Equation (3)), any L atom tuples of Q can produce $O(L^3/n)$ result tuples. This will then complete the proof of Lemma 2.

Recall that all the boat joins in $\mathcal{Q}_{\text{boat}}$ differ only in their R_{DEF}, which can be any subset of $\mathbf{dom}(\mathsf{D}) \times \mathbf{dom}(\mathsf{E}) \times \mathbf{dom}(\mathsf{F})$. Next, we impose a distribution over $\mathcal{Q}_{\text{boat}}$. For this purpose, create R_{DEF} by including each tuple of $\mathbf{dom}(\mathsf{D}) \times \mathbf{dom}(\mathsf{E}) \times \mathbf{dom}(\mathsf{F})$ independently with probability $1/n$. The expected size of R_{DEF} is $(n^{2/3})^3 \cdot \frac{1}{n} = n$. Accordingly, the boat join Q thus obtained—which is now a random variable—has an expected input size of $5n$ and an expected output size of n^2 (recall that $\mathcal{J}oin(Q) = R_{\mathsf{ABC}} \times R_{\mathsf{DEF}}$). Our goal is to prove that, with a positive probability, Q satisfies two conditions simultaneously:

- **C.2.2-1**: R_{DEF} has at most $2n$ tuples;
- **C.2.2-2**: $\text{OPT}_{\text{CP}}(Q, 8L) \leq 2 \cdot (8L)^3/n$.

The positive probability assures us that a boat join Q fulfilling the two conditions definitely exists. Condition **C.2.2-1** implies that Q has an input size $\Theta(n)$ and an output size $\Theta(n^2)$. It thus follows that Q has all the properties promised at the beginning of this subsection.

The satisfaction probability of **C.2.2-1** is easy to analyze: as each tuple in $\mathbf{dom}(\mathsf{A}) \times \mathbf{dom}(\mathsf{B}) \times \mathbf{dom}(\mathsf{C})$ belongs to R_{REF} independently with probability $1/n$, a simple application of Chernoff bound (42) in Appendix A (supplying $\gamma = 1$) shows that the probability for $|R_{\mathsf{DEF}}|$ to be over twice its expectation $\mathbf{E}[|R_{\mathsf{DEF}}|] = n$ is at most $\exp(-\Omega(\mathbf{E}[|R_{\mathsf{DEF}}|])) = \exp(-\Omega(n))$, which is less than $1/4$ for sufficiently large n. The following discussion will focus on Condition **C.2.2-2**.

We refer to $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}\}$ as a *legal CP-form selection* if, for each $e \in \{\mathsf{ABC}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$, the relation R'_e (i) is a subset of R_e, and (ii) is in the CP-form, and (iii) contains at most $8L$ tuples. The next lemma offers a bound on the number of such selections.

Lemma 4. *The number of legal CP-form selections cannot exceed $2^{O(n^{2/3})}$, regardless the value of L.*

Proof. R'_{AD}, which is in the CP-form, equals $\pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{D}}(R'_{\mathsf{AD}})$. There are $2^{n^{1/3}}$ ways to choose $\pi_{\mathsf{A}}(R'_{\mathsf{AD}})$ and $2^{n^{2/3}}$ ways to choose $\pi_{\mathsf{D}}(R'_{\mathsf{AD}})$, because they are subsets of $\mathbf{dom}(\mathsf{A})$ and $\mathbf{dom}(\mathsf{D})$, re-

spectively. Hence, the number of possible choices for R'_{AD} cannot exceed $2^{n^{1/3}+n^{2/3}} = 2^{O(n^{2/3})}$. The same bound also applies to R'_{BE} and R'_{CF}. Regarding R'_{ABC}, the number of choices is $2^{O(n^{1/3})}$ because $\mathbf{dom}(\mathsf{A})$, $\mathbf{dom}(\mathsf{B})$, and $\mathbf{dom}(\mathsf{C})$ all have a size of $n^{1/3}$. Therefore, the number of legal CP-form selections is at most $(2^{O(n^{2/3})})^3 \cdot 2^{O(n^{1/3})} = 2^{O(n^{2/3})}$. □

In the LJM-CP($8L$) problem defined by a boat join Q, each local join is formed by a legal CP-form selection and together with the relation $R_{\mathsf{DEF}} \in Q$. The value $\mathrm{OPT}_{\mathrm{CP}}(Q, 8L)$ is the maximum size of all those local joins. We thus have:

$$\begin{aligned}
&\mathbf{Pr}[\mathrm{OPT}_{\mathrm{CP}}(Q, 8L) > 2 \cdot (8L)^3/n] \\
&\text{(note: the probability is over the distribution of } Q) \\
&= \mathbf{Pr}[\exists \text{ one legal CP-form selection } \{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}\} \text{ s.t. } |\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| > 2 \cdot (8L)^3/n] \\
&\text{(note: the probability is over the distribution of } R_{\mathsf{DEF}}) \\
&\leq \sum_{\text{legal CP-form selection } \{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}\}} \mathbf{Pr}[|\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| > 2 \cdot (8L)^3/n] \qquad (5) \\
&\text{(note: the probability is over the distribution of } R_{\mathsf{DEF}}).
\end{aligned}$$

The lemma below bounds the probability in (5).

Lemma 5. $\mathbf{Pr}[|\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| > 2 \cdot (8L)^3/n] \leq \exp(-\Omega(\frac{L^2}{n}))$, *for any legal CP-form selection* $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}\}$.

Proof. Let us examine the result of $R'_{\mathsf{ABC}} \bowtie R'_{\mathsf{AD}} \bowtie R'_{\mathsf{BE}} \bowtie R'_{\mathsf{CF}}$ first. As $R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}$, and R'_{CF} are all in the CP-form, every tuple $\boldsymbol{u} \in \pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})$ corresponds to exactly the same number — which we denote as λ — of tuples $\boldsymbol{v} \in R'_{\mathsf{ABC}} \bowtie R'_{\mathsf{AD}} \bowtie R'_{\mathsf{BE}} \bowtie R'_{\mathsf{CF}}$, where the term "corresponds to" means $\boldsymbol{u} = \boldsymbol{v}[\mathsf{DEF}]$. To see this, consider another tuple $\boldsymbol{u'} \in \pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})$. Construct a tuple $\boldsymbol{v'}$ such that $\boldsymbol{v'}[\mathsf{ABC}] = \boldsymbol{v}[\mathsf{ABC}]$ and $\boldsymbol{v'}[\mathsf{DEF}] = \boldsymbol{u'}$. It is rudimentary to verify that $\boldsymbol{v'}$ must be a result tuple of $R'_{\mathsf{ABC}} \bowtie R'_{\mathsf{AD}} \bowtie R'_{\mathsf{BE}} \bowtie R'_{\mathsf{CF}}$.

Next, we give two crucial inequalities on λ. For this purpose, fix any tuple $\boldsymbol{u} \in \pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})$, and consider the tuples $\boldsymbol{v} \in R'_{\mathsf{ABC}} \bowtie R'_{\mathsf{AD}} \bowtie R'_{\mathsf{BE}} \bowtie R'_{\mathsf{CF}}$ that $\boldsymbol{u}$ corresponds to.

— As any such $\boldsymbol{v}$ must satisfy $\boldsymbol{v}(\mathsf{A}) \in \pi_{\mathsf{A}}(R'_{\mathsf{ABC}})$, $\boldsymbol{v}(\mathsf{B}) \in \pi_{\mathsf{B}}(R'_{\mathsf{ABC}})$, and $\boldsymbol{v}(\mathsf{C}) \in \pi_{\mathsf{C}}(R'_{\mathsf{ABC}})$, we have:

$$\lambda \leq |\pi_{\mathsf{A}}(R'_{\mathsf{ABC}})||\pi_{\mathsf{B}}(R'_{\mathsf{ABC}})||\pi_{\mathsf{C}}(R'_{\mathsf{ABC}})| = |R'_{\mathsf{ABC}}| \leq 8L. \qquad (6)$$

— As such a tuple $\boldsymbol{v}$ must satisfy $\boldsymbol{v}(\mathsf{A}) \in \pi_{\mathsf{A}}(R'_{\mathsf{AB}})$, $\boldsymbol{v}(\mathsf{B}) \in \pi_{\mathsf{B}}(R'_{\mathsf{BD}})$, and $\boldsymbol{v}(\mathsf{C}) \in \pi_{\mathsf{C}}(R'_{\mathsf{CE}})$, the value λ cannot exceed $|\pi_{\mathsf{A}}(R'_{\mathsf{AD}})||\pi_{\mathsf{B}}(R'_{\mathsf{BE}})||\pi_{\mathsf{C}}(R'_{\mathsf{CF}})|$. Equiped with this fact, we obtain:

$$\begin{aligned}
\lambda \cdot |\pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})| &= \lambda \cdot |\pi_{\mathsf{D}}(R'_{\mathsf{AD}})||\pi_{\mathsf{E}}(R'_{\mathsf{BE}})||\pi_{\mathsf{F}}(R'_{\mathsf{CF}})| \\
&\leq |\pi_{\mathsf{A}}(R'_{\mathsf{AD}})||\pi_{\mathsf{B}}(R'_{\mathsf{BE}})||\pi_{\mathsf{C}}(R'_{\mathsf{CF}})||\pi_{\mathsf{D}}(R'_{\mathsf{AD}})||\pi_{\mathsf{E}}(R'_{\mathsf{BE}})||\pi_{\mathsf{F}}(R'_{\mathsf{CF}})| \\
&= |R'_{\mathsf{AD}}||R'_{\mathsf{BE}}||R'_{\mathsf{CF}}| \leq (8L)^3. \qquad (7)
\end{aligned}$$

Now, generate R_{DEF} (by including each tuple of $\mathbf{dom}(\mathsf{D}) \times \mathbf{dom}(\mathsf{E}) \times \mathbf{dom}(\mathsf{F})$ with probability $1/n$ independently) and consider the local join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$. For each tuple $\boldsymbol{u} \in \pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})$, if $\boldsymbol{u}$ appears in R_{DEF}, then $\boldsymbol{u}$ corresponds to exactly λ tuples in $\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})$; otherwise, it corresponds to none. Therefore:

$$|\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| = \sum_{\boldsymbol{u} \in \pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})} \lambda \cdot \mathbb{1}_{\boldsymbol{u} \in R_{\mathsf{DEF}}} = \lambda \cdot s, \qquad (8)$$

where

$$s = \sum_{\boldsymbol{u} \in \pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})} \mathbb{1}_{\boldsymbol{u} \in R_{\mathsf{DEF}}}$$

and $\mathbb{1}_{\boldsymbol{u}\in R_{\mathsf{DEF}}}$ is an indicator random variable that equals 1 if $\boldsymbol{u} \in R_{\mathsf{DEF}}$ or 0, otherwise. All such indictor variables are mutually independent and $\mathbf{Pr}[\mathbb{1}_{\boldsymbol{u}\in R_{\mathsf{DEF}}} = 1] = 1/n$ for every $\boldsymbol{u}$. Thus, $\mathbf{E}[s] = |\pi_{\mathsf{D}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{E}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{F}}(R'_{\mathsf{CF}})|/n$, which, together with (7), yields $\lambda \cdot \mathbf{E}[s] \leq (8L)^3/n$.

From here, we will proceed differently depending on how large $\mathbf{E}[s]$ is. Consider first the case where $\mathbf{E}[s] \geq (8L)^2/n$. Applying Chernoff bound (42) with $\gamma = 1$, we know that the probability for s to exceed $2 \cdot \mathbf{E}[s]$ is at most $\exp(-\Omega(\mathbf{E}[s])) = \exp(-\Omega(L^2/n))$. It thus follows from (8) that $\mathbf{Pr}[|\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| > 2 \cdot \lambda \cdot \mathbf{E}[s]]$ is at most $\exp(-\Omega(L^2/n))$, and hence, $\mathbf{Pr}[|\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| > 2 \cdot (8L)^3/n] \leq \exp(-\Omega(L^2/n))$.

It remains to discuss the case where $\mathbf{E}[s] < (8L)^2/n$. Set $\gamma = \frac{2\cdot(8L)^2/n}{\mathbf{E}[s]}$. Applying Chernoff bound (43) with this γ, we know that the probability for s to exceed $\gamma \cdot \mathbf{E}[s] = 2 \cdot (8L)^2/n$ is $\exp(-\Omega(\gamma \cdot \mathbf{E}[s])) = \exp(-\Omega(L^2/n))$. It thus follows from (8) that $\mathbf{Pr}[|\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| > \lambda \cdot 2 \cdot (8L)^2/n]$ is at most $\exp(-\Omega(L^2/n))$. Since $\lambda \leq 8L$ (see (6)), we conclude that $\mathbf{Pr}[|\mathit{Join}(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| > 2 \cdot (8L)^3/n] \leq \exp(-\Omega(L^2/n))$. □

We can now combine (5) with Lemmas 4 and 5 to derive:

$$\mathbf{Pr}[\mathrm{OPT}_{\mathrm{CP}}(Q, 8L) > (8L)^3/n] \leq 2^{O(n^{2/3})} \cdot \exp(-\Omega(L^2/n)) = \exp(O(n^{2/3}) - \Omega(L^2/n)), \tag{9}$$

which is at most 1/4 as long as

$$\frac{L^2}{n} > c_0 \cdot n^{2/3} \quad \Leftrightarrow \quad L > \sqrt{c_0} \cdot n^{5/6}$$

for some sufficiently large constant $c_0 > 0$. We can now fix the constant c' in Lemma 2 to $\sqrt{c_0}$.

In conclusion, we have shown that, with probability at least $1 - (1/4) - (1/4) = 1/2$, a boat join Q we generated at the beginning of the subsection satisfies both conditions **C2.2-1** and **C2.2-2**.

2.3 Proof of Lemma 3

This subsection is devoted to proving Lemma 3. Recall that, in the context of this lemma, we concentrate on one (arbitrarily) given boat join $Q \in \boldsymbol{Q}_{\mathrm{boat}}$ (in other words, R_{DEF} has been fixed). Let $R^*_{\mathsf{ABC}}, R^*_{\mathsf{AD}}, R^*_{\mathsf{BE}}$, and R^*_{CF} constitute an optimal solution to LJM(L), i.e., OPT(Q, L) equals the output size of the local join $\{R^*_{\mathsf{ABC}}, R^*_{\mathsf{AD}}, R^*_{\mathsf{BE}}, R^*_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$. We will construct $R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}$, and R'_{CF} such that

— all of them are in the CP-form and have at most $8L$ tuples each;
— the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ has size at least $\frac{1}{128} \cdot \mathrm{OPT}(Q, L)$.

This will then establish Lemma 3.

Our conversion starts by setting $R'_e = R^*_e$ for each $e \in \{\mathsf{AD}, \mathsf{BE}, \mathsf{CF}, \mathsf{ABC}\}$ and proceeds by converting—in this order—R'_{AD},R'_{AD}, R'_{BE}, R'_{CF}, and R'_{ABC} to the CP-form incrementally. After turning each of R'_{AD}, R'_{BE}, R'_{CF} into the CP-form, the output size of the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ can decrease by a factor at most 4, while the size of R'_e at most doubles for each $e \in \{\mathsf{ABC}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$. The last conversion (on R'_{ABC}) can reduce the join output size by another factor of 2, but will not increase the size of any relation further.

Conversion of R'_{AD}. At this moment, $R'_e = R^*_e$ for each $e \in \{\mathsf{ABC}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$ and, hence, $|R'_e| \leq L$. We will produce two new relations R''_{ABC} and R''_{AD} such that

— R''_{AD} is in the CP-form (but R''_{ABC} may not);
— each of R''_{ABC} and R''_{AD} has at most $2L$ tuples;
— the output size of the join $\{R''_{\mathsf{ABC}}, R''_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is at least 1/4 of that of the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$.

The conversion will then finish by replacing R'_{ABC} and R'_{AD} with R''_{ABC} and R''_{AD}.

Our strategy for generating R''_{ABC} and R''_{AD} involves three steps:

- First, obtain a "good" subset $S_{\mathsf{BC}} \subseteq \mathbf{dom}(\mathsf{B}) \times \mathbf{dom}(\mathsf{C})$, and a "good" subset $S_{\mathsf{D}} \subseteq \mathbf{dom}(D)$. The reader can regard S_{BC} as a relation over $\{\mathsf{B}, \mathsf{C}\}$ and S_{D} as a relation over $\{\mathsf{D}\}$.
- Second, choose a subset $S_{\mathsf{A}} \subseteq \mathbf{dom}(\mathsf{A})$, which can be regarded as a relation over $\{\mathsf{A}\}$.
- Third, create $R''_{\mathsf{ABC}} = S_{\mathsf{A}} \times S_{\mathsf{BC}}$ and $R''_{\mathsf{AD}} = S_{\mathsf{A}} \times S_{\mathsf{D}}$.

Given a value $a \in \mathsf{A}$, we denote by $\{a\}$ the special "singleton" relation that contains only one tuple with value a on attribute A. Regardless of S_{BC}, S_{D}, and S_{A}, it always holds that

$$\begin{aligned}
&\left| \mathit{Join}\left(\{R''_{\mathsf{ABC}}, R''_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}\right) \right| \\
&\text{(namely, output size of the join } \{R''_{\mathsf{ABC}}, R''_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}) \\
&= \sum_{a \in S_{\mathsf{A}}} \left| \mathit{Join}\left(\{\{a\} \times S_{\mathsf{BC}}, \{a\} \times S_{\mathsf{D}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}\right) \right| \\
&= |S_{\mathsf{A}}| \cdot \left| \mathit{Join}\left(\{\{1\} \times S_{\mathsf{BC}}, \{1\} \times S_{\mathsf{D}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}\right) \right| \qquad (10)
\end{aligned}$$

where the term "$\{1\}$" in (10) refers to the relation $\{a\}$ with $a = 1 \in \mathbf{dom}(\mathsf{A})$. The equality in (10) holds because, once S_{BC} and S_{D} are fixed, the output size of the join $\{\{a\} \times S_{\mathsf{BC}}, \{a\} \times S_{\mathsf{D}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is identical for any $a \in \mathbf{dom}(\mathsf{A})$.

Motivated by (10), we consider the following refined variant of LJM:

> *Local-Join Maximization by Choosing BC and D.* Fix an arbitrary integer $t \geq 1$. Choose $S_{\mathsf{BC}} \subseteq \mathbf{dom}(\mathsf{B}) \times \mathbf{dom}(\mathsf{C})$ and $S_{\mathsf{D}} \subseteq \mathbf{dom}(D)$ to maximize the size of the *local join* $\{\{1\} \times S_{\mathsf{BC}}, \{1\} \times S_{\mathsf{D}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ subject to the constraint $|S_{\mathsf{BC}}| + |S_{\mathsf{D}}| \leq t$. We will represent the above problem as LJM-choose-BC-D(t).

Define

$$\Delta(t) = \text{the maximum output size of all possible local joins in the problem LJM-choose-BC-D}(t). \qquad (11)$$

How to compute $\Delta(t)$ precisely is of no relevance to us; what matters, instead, is that $\Delta(t)$ exists and is monotonically increasing. Define

$$t^* = \arg\max_{t \in \left[\frac{2L}{N^{1/3}}, 2L\right]} \frac{\Delta(t)}{t}. \qquad (12)$$

Note, importantly, that t^* is selected from the range $[\frac{2L}{N^{1/3}}, 2L]$. We are now ready to explain how to generate S_{BC}, S_{D}, and S_{A} for computing R''_{ABC} and R''_{AD}:

- Set S_{BC} and S_{D} as in an optimal solution to the LJM-choose-BC-D(t^*) problem, i.e., the join $\{\{1\} \times S_{\mathsf{BC}}, \{1\} \times S_{\mathsf{D}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ has size $\Delta(t^*)$;
- Set $S_{\mathsf{A}} = \{1, 2, \ldots, \lfloor 2L/t^* \rfloor\}$.

This ensures

$$|R''_{\mathsf{ABC}}| + |R''_{\mathsf{AD}}| = |S_{\mathsf{A}}| \cdot (|S_{\mathsf{BC}}| + |S_{\mathsf{D}}|) \leq \lfloor 2L/t^* \rfloor \cdot t^* \leq 2L.$$

Hence, each of R''_{ABC} and R''_{AD} has at most $2L$ tuples, as desired. Next, we prove that the join $\{R''_{\mathsf{ABC}}, R''_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ has a sufficiently large result.

Lemma 6. *The output size of the join* $\{R''_{\mathsf{ABC}}, R''_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ *is at least 1/4 of that of* $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$.

Proof. For any $a \in \mathbf{dom}(\mathsf{A})$, denote by $R'_{\mathsf{ABC}}(a)$ the set of tuples $\boldsymbol{u} \in R'_{\mathsf{ABC}}$ with $\boldsymbol{u}(A) = a$, and similarly by $R'_{\mathsf{AD}}(a)$ the set of tuples $\boldsymbol{u} \in R'_{\mathsf{AD}}$ with $\boldsymbol{u}(A) = a$. Define

$$t_a = |R'_{\mathsf{ABC}}(a)| + |R'_{\mathsf{AD}}(a)|.$$

It holds that $\sum_{a \in \mathbf{dom}(\mathsf{A})} t_a \leq 2L$ (because R'_{ABC} and R'_{AD} have L tuples each). We say that a is *dominated* if $\Delta(t_a)/t_a \leq \Delta(t^*)/t^*$, or *undominated* otherwise; function $\Delta(.)$ and value t^* are defined in (11) and (12), respectively. Because t^* is from the range $[\frac{2L}{N^{1/3}}, 2L]$, an undominated value a must satisfy $t_a < \frac{2L}{N^{1/3}}$.

For any $a \in \mathbf{dom}(\mathsf{A})$, $\pi_{\mathsf{BC}}(R'_{\mathsf{ABC}}(a))$ and $\pi_{\mathsf{D}}(R'_{\mathsf{AD}}(a))$ are permissible choices for S_{BC} and S_{D}, respectively, under LJM-choose-BC-D(t_a). Hence:

$$\begin{aligned}
&|\mathit{Join}\,(\{R'_{\mathsf{ABC}}(a), R'_{\mathsf{AD}}(a), R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| \\
&\quad = |\mathit{Join}\,(\{\{1\} \times \pi_{\mathsf{BC}}(R'_{\mathsf{ABC}}(a)), \{1\} \times \pi_{\mathsf{D}}(R'_{\mathsf{AD}}(a)), R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| \\
&\quad \leq \Delta(t_a).
\end{aligned}$$

Therefore:

$$\begin{aligned}
|\mathit{Join}\,(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| &= \sum_{a \in \mathbf{dom}(\mathsf{A})} |\mathit{Join}\,(\{R'_{\mathsf{ABC}}(a), R'_{\mathsf{AD}}(a), R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| \\
&\leq \sum_{a \in \mathbf{dom}(\mathsf{A})} \Delta(t_a). \qquad (13)
\end{aligned}$$

Regarding the dominated values in $\mathbf{dom}(\mathsf{A})$, we have:

$$\begin{aligned}
\sum_{\text{dominated } a \in \mathbf{dom}(\mathsf{A})} \Delta(t_a) &= \sum_{\text{dominated } a \in \mathbf{dom}(\mathsf{A})} t_a \cdot \frac{\Delta(t_a)}{t_a} \leq \sum_{\text{dominated } a \in \mathbf{dom}(\mathsf{A})} t_a \cdot \frac{\Delta(t^*)}{t^*} \\
&= \frac{\Delta(t^*)}{t^*} \sum_{\text{dominated } a \in \mathbf{dom}(\mathsf{A})} t_a \leq \frac{\Delta(t^*)}{t^*} \cdot 2L. \qquad (14)
\end{aligned}$$

Regarding the undominated values in $\mathbf{dom}(\mathsf{A})$, we have:

$$\begin{aligned}
\sum_{\text{undominated } a \in \mathbf{dom}(\mathsf{A})} \Delta(t_a) &\leq \sum_{\text{undominated } a \in \mathbf{dom}(\mathsf{A})} \Delta\left(\frac{2L}{N^{1/3}}\right) \\
&\quad \text{(as } \Delta(.) \text{ is monotonically increasing and } t_a < 2L/N^{1/3}\text{)} \\
&\leq |\mathbf{dom}(\mathsf{A})| \cdot \Delta\left(\frac{2L}{N^{1/3}}\right) = N^{1/3} \cdot \Delta\left(\frac{2L}{N^{1/3}}\right) \\
&= 2L \cdot \frac{\Delta(2L/N^{1/3})}{2L/N^{1/3}} \leq \frac{\Delta(t^*)}{t^*} \cdot 2L. \qquad (15)
\end{aligned}$$

It follows from (13), (14), and (15) that the output size of the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is at most $\frac{\Delta(t^*)}{t^*} \cdot 4L$.

On the other hand, the size of S_{A} selected by our strategy is $\lfloor 2L/t^* \rfloor \geq L/t^*$. By (10) and the definition of t^*, the output size of the join $\{R''_{\mathsf{ABC}}, R''_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is at least $\Delta(t^*) \cdot L/t^*$. This completes the proof of Lemma 6. □

Conversion of R'_{BE}. At this moment, $|R'_e| \leq 2L$ for each $e \in \{\mathsf{ABC}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$. We aim to produce two new relations R''_{ABC} and R''_{BE} such that (i) R''_{BE} is in the CP-form (but R''_{ABC} may not), (ii) each of R''_{ABC} and R''_{BE} has at most $4L$ tuples, and (iii) the output size of the join $\{R''_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R''_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is at least 1/4 of that of the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$. Due to symmetry, we can achieve the

purpose by applying the same argument presented earlier for R'_{AD} and changing L to $2L$ (it would help to "rename" A to B and D to E in applying the argument and then restore the names afterwards). The conversion then finishes by replacing R'_{ABC} and R'_{BE} with R''_{ABC} and R''_{BE}. Note that R'_{AD} is not affected by this conversion and hence remains in the CP-form.

Conversion of R'_{CF}. This should have become straightforward from the previous two conversions. R'_{AD} and R'_{BE} are not affected by this conversion and hence remain in the CP-form.

Conversion of R'_{ABC}. At this moment, $|R'_e| \leq 8L$ for each $e \in \{\mathsf{ABC}, \mathsf{AD}, \mathsf{BE}, \mathsf{CF}\}$. Furthermore, R'_{AD}, R'_{BE}, and R'_{CF} are already in the CP-form. We will produce a new relation R''_{ABC} such that

- R''_{ABC} is in the CP-form and has at most $8L$ tuples;
- the output size of the join $\{R''_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is at least half of that of the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$.

After setting R'_{ABC} to R''_{ABC}, we will have obtained the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ needed to complete the proof of Lemma 3; note that R'_{AD}, R'_{BE}, and R'_{CF} are not affected by this conversion.

Given a tuple $\boldsymbol{u} \in \mathbf{dom}(\mathsf{A}) \times \mathbf{dom}(\mathsf{B}) \times \mathbf{dom}(\mathsf{C})$, we use $\{\boldsymbol{u}\}$ to denote the singleton relation with scheme ABC containing only $\boldsymbol{u}$. Given also a tuple $\boldsymbol{v} \in R_{\mathsf{DEF}}$, we use $\boldsymbol{u} \circ \boldsymbol{v}$ to denote the tuple over scheme ABCDEF that takes value $\boldsymbol{u}(X)$ for every attribute $X \in \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ and value $\boldsymbol{v}(X)$ for every attribute $X \in \{\mathsf{D}, \mathsf{E}, \mathsf{F}\}$. The lemma below explains why we want to make sure that R'_{AD}, R'_{BE}, and R'_{CF} are already in the CP-form.

Lemma 7. *Suppose that R'_{AD}, R'_{BE}, and R'_{CF} are in the CP-form. For any distinct tuples $\boldsymbol{u}, \boldsymbol{u'} \in \pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})$, the output size of the join $\{\{\boldsymbol{u}\}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is the same as the output size of the join $\{\{\boldsymbol{u'}\}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$.*

Proof. Consider an arbitrary tuple $\boldsymbol{v} \in R_{\mathsf{DEF}}$. As R'_{AD}, R'_{BE}, and R'_{CF} are in the CP-form, the tuple $\boldsymbol{u} \circ \boldsymbol{v}$ is in the result of the join $\{\{\boldsymbol{u}\}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ if and only if $\boldsymbol{v}(\mathsf{D}) \in \pi_{\mathsf{D}}(R'_{\mathsf{AD}})$, $\boldsymbol{v}(\mathsf{E}) \in \pi_{\mathsf{E}}(R'_{\mathsf{BE}})$, and $\boldsymbol{v}(\mathsf{F}) \in \pi_{\mathsf{F}}(R'_{\mathsf{CF}})$. The same sentence is also true if we replace $\boldsymbol{u}$ with $\boldsymbol{u'}$. This completes the proof. □

We denote by s_1 the output size of the join $\{\{\boldsymbol{u}\}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ for an arbitrary $\boldsymbol{u} \in \pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})$. It follows from Lemma 7 that

$$|\mathit{Join}\,(\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\})| = s_1 \cdot s_2. \tag{16}$$

where

$$s_2 = |R'_{\mathsf{ABC}} \cap (\pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})|. \tag{17}$$

Next, we will construct an $R''_{\mathsf{ABC}} \subseteq \pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})$ such that R''_{ABC} is in the CP-form and

$$s_2/2 \leq |R''_{\mathsf{ABC}}| \leq 8L. \tag{18}$$

Since (by Lemma 7) the output size of the join $\{R''_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$ is $s_1 \cdot |R''_{\mathsf{ABC}}|$, (16) and (18) together will assure us that the aforementioned join size is at least half of that of the join $\{R'_{\mathsf{ABC}}, R'_{\mathsf{AD}}, R'_{\mathsf{BE}}, R'_{\mathsf{CF}}, R_{\mathsf{DEF}}\}$.

Henceforth, we consider $s_2 > 0$ (otherwise, simply choose R''_{ABC} to be an empty solution). Define

$$\begin{aligned} S'_{\mathsf{A}} &= \pi_{\mathsf{A}}(R'_{\mathsf{ABC}} \cap (\pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})) \\ S'_{\mathsf{B}} &= \pi_{\mathsf{B}}(R'_{\mathsf{ABC}} \cap (\pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})) \\ S'_{\mathsf{C}} &= \pi_{\mathsf{C}}(R'_{\mathsf{ABC}} \cap (\pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})). \end{aligned}$$

It must hold that $|S'_{\mathsf{A}}||S'_{\mathsf{B}}||S'_{\mathsf{C}}| \geq s_2$ (otherwise, $R'_{\mathsf{ABC}} \cap (\pi_{\mathsf{A}}(R'_{\mathsf{AD}}) \times \pi_{\mathsf{B}}(R'_{\mathsf{BE}}) \times \pi_{\mathsf{C}}(R'_{\mathsf{CF}})$ would have a size less than s_2, giving a contradiction).

We will choose subsets $S_A \subseteq S'_A$, $S_B \subseteq S'_B$, $S_C \subseteq S'_C$, and then generate $R''_{ABC} = S_A \times S_B \times S_C$. Specifically, we first set S_A directly to S'_A. Let k_B be the greatest integer in $[1, |S'_B|]$ satisfying $|S'_A| \cdot k_B \leq 8L$; note that, if $k_B < |S'_B|$, then we must have $|S'_A| \cdot k_B > 4L$.[5] Now, create S_B by including k_B arbitrary values in S'_B. Let k_C be the greatest integer in $[1, |S'_C|]$ satisfying $|S'_A| \cdot k_B \cdot k_C \leq 8L$ (if $k_C < |S'_C|$, then $|S'_A| \cdot k_B \cdot k_C > 4L$). Create S_C by including k_C arbitrary values in S'_C.

$|R''_{ABC}|$ has size $|S'_A| \cdot k_B \cdot k_C \leq 8L$. For validating (18), it remains to explain why $|R''_{ABC}| \geq s_2/2$. This is obvious if $k_B = |S'_B|$ and $k_C = |S'_C|$ (in this case, $|R''_{ABC}| = |S'_A||S'_B||S'_C| \geq s_2$). Otherwise, $|S'_A| \cdot k_B \cdot k_C$ must be greater than $4L$, which is at least $s_2/2$ because $s_2 \leq |R'_{ABC}| \leq 8L$. This concludes the proof of Lemma 3.

3 CANONICAL EDGE COVERS FOR ACYCLIC HYPERGRAPHS

Since it is no longer feasible to process all cyclic joins with a load of $\tilde{O}(N/p^{1/\rho^*})$, our focus will shift to acyclic joins, as defined in Section 1.1. In this section, we will concentrate solely on graph theory and introduce the concept of "canonical edge cover" for acyclic hypergraphs, along with several important properties. These properties will then be leveraged in Sections 4 and 5 to design an MPC algorithm for evaluating any acyclic join with a load of $O(N/p^{1/\rho^*})$.

Our discussion throughout the section is based on:

— an acyclic hypergraph $G = (V, E)$ with $|E| \geq 2$, and
— an edge tree T of G.

As G and T both contain "vertices" and "edges", for better clarity we will obey several conventions in our presentation. A vertex in G will always be referred to as an *attribute*, while the term *node* is reserved for the vertices in T. Furthermore, to avoid confusion with the edges in G, we will always refer to an edge in T as a *link*.

An edge $e \in E$ is *subsumed* in G if it is a subset of another edge $e' \in E$, i.e., $e \subseteq e'$. If an attribute X appears in only one edge of E, it is an *exclusive attribute*; otherwise, X is *non-exclusive*. Unless otherwise stated, we allow G to be an arbitrary acyclic hypergraph. This means that E can have two or more edges containing the same set of attributes (nonetheless, they are still different edges) and may even have empty edges. We say that G is *non-empty* if $E \neq \emptyset$ and that G is *reduced* if E has no subsumed edges.

By rooting T at an arbitrary leaf, we can regard T as a *rooted* tree. Make all the links from parent to child; this way, T becomes a directed acyclic graph. We say that the root of T is the *highest* node in T and, in general, a node is *higher* (or *lower*) than any of its proper descendants (or ancestors). For any non-root node e, we denote its parent node in T as $parent(e)$.

Now that there are two views of T (i.e., undirected and directed), we will be careful with tree terminology. By default, we will treat T as a directed tree. Accordingly, a *leaf* of T is a node with out-degree 0, a *path* is a sequence of nodes where each node has a link pointing to the next node, and a *subtree* rooted at a node e is the directed tree induced by the nodes reachable from e in T. Sometimes, we may revert back to the undirected view of T. In that case, we use the term *raw leaf* for a leaf in the undirected T (a raw leaf can be a leaf or the root under the directed view).

3.1 Canonical Edge Cover: Formulation and Basic Properties

For each attribute $X \in V$, we define the *summit* of X as the highest node in T containing X. If node e is the summit of X, we call X a *disappearing* attribute in e. By acyclicity's connectedness

[5] This is because otherwise $|S'_A| \cdot (k_B + 1)$, which is at most $|S'_A| \cdot (2k_B)$, would be at most $8L$, contradicting the definition of k_B.

requirement (Section 1.1), X can appear only in the subtree rooted at e and hence "disappears" as soon as we leave the subtree.

Example 3.1. Let $G = (V, E)$ be the hypergraph in Example 1.1 whose (rooted) edge tree T is shown in Figure 1. The summit of C is node CEJ; thus, C is a disappearing attribute of CEJ. Node EHJ is the summit of E and J; thus, both E and J are disappearing attributes of EHJ.

We say that a subset $S \subseteq E$ *covers* an attribute $X \in V$ if S has an edge containing X. Recall (from Section 1.3) that an optimal edge cover of G is the smallest S covering every attribute in V. Optimal edge covers are not unique; some are of particular importance to us, and we will identify them as "canonical". Towards a procedural definition, consider the following algorithm:

edge-cover (T) /* T is rooted */
1. $F_{\text{tmp}} \leftarrow \emptyset$
2. obtain a reverse topological order $e_1, e_2, \ldots, e_{|E|}$ of the nodes (a.k.a., edges) in T
3. **for** $i \leftarrow 1$ to $|E|$ **do**
4. **if** e_i has a disappearing attribute not covered by F_{tmp} **then** add e_i to F_{tmp}
5. **return** F_{tmp}

As proved shortly, the output of edge-cover is uniquely determined by T, regardless of the reverse topological order used at Line 2. This permits us to define the *canonical edge cover* (CEC) of G induced by T to be the output of edge-cover.

Example 3.2. We now continue the discussion in Example 3.1 (see Figure 1 for the edge tree T). Consider the reverse topological order of T:

ABC, BD, BO, BCE, EFG, CEF, CEJ, HI, EHJ, LM, KL, HK, HN.

When processing ABC, algorithm *edge-cover* adds it to F_{tmp} because ABC has a disappearing attribute A and yet $F_{\text{tmp}} = \emptyset$. When processing BCE, $F_{\text{tmp}} = \{\text{ABC}, \text{BD}, \text{BO}\}$. BCE has a disappearing attribute B, which, however, has been covered by F_{tmp}. Thus, B is not added to F_{tmp}. If $\mathcal{F}$ is the final F_{tmp} returned by the algorithm, then

$$\mathcal{F} = \{\text{ABC}, \text{BD}, \text{BO}, \text{EFG}, \text{HI}, \text{LM}, \text{EHJ}, \text{HK}, \text{HN}\},$$

which is the CEC of G induced by T. The edges in $\mathcal{F}$ are circled in Figure 1.

The lemma below gives three properties of edge-cover that pave the foundation of all the development in the subsequent sections.

Lemma 8. *All the following statements are true about the* edge-cover *algorithm.*

(1) *Its output—denoted as $\mathcal{F}$ —is an edge cover of G containing ρ^* edges, where ρ^* is the fractional edge covering number of G.*
(2) *The output is always the same, no matter which reverse topological order is deployed at Line 2.*
(3) *If G is reduced, $\mathcal{F}$ includes all the raw leaves of T.*

Proof. It is easy to see that $\mathcal{F}$ is an edge cover of G. Each attribute $X \in V$ is a disappearing attribute of some edge $e \in E$. When e is processed at Line 4 of edge-cover, either X is already covered or e itself will be added to F_{tmp} (which will then cover X).

Next, we argue that $\mathcal{F}$ has exactly ρ^* edges. Let W be an arbitrary optimal fractional edge covering of G. As edge-cover runs, we will gradually construct a function $W' : E \rightarrow \mathbb{R}$ such that

— **C3.2-1:** $\sum_{e \in E} W'(e) \leq \sum_{e \in E} W(e)$;

— **C3.2-2:** At the end of edge-cover, for each edge $e \in E$, $W'(e) \geq 1$ if $e \in \mathcal{F}$ or 0 otherwise.

The two conditions imply that $|\mathcal{F}| \leq \sum_{e \in E} W'(e) \leq \sum_{e \in E} W(e) = \rho^*$. Conversely, $|\mathcal{F}| \geq \rho^*$ obviously holds by definition of ρ^*. It will then follow that $|\mathcal{F}| = \rho^*$.

Our modification is carried out as follows. Before running edge-cover, we initialize W' by equating it directly to W. Then, run edge-cover. Whenever Line 4 decides *not* to add the current edge e_i to F_{tmp}, we

— (if e_i is the root of T) set $W'(e_i)$ to 0;
— (otherwise) increase $W'(\mathit{parent}(e_i))$ by $W'(e_i)$ and then set $W'(e_i)$ to 0 (effectively, e_i passes its weight under W' to its parent).

The modification clearly satisfies Condition **C3.2-1**. To explain why it also satisfies **C3.2-2**, consider what happens when Line 4 decides to include the current edge e_i to F_{tmp}. We argue that $W'(e_i)$ must be at least 1 at this moment. Let X be any disappearing attribute of e_i that had not been covered by F_{tmp} prior to executing Line 4 (X must exist because e_i has entered F_{tmp}). Denote by S the set of edges in E containing X. The entire S must be in the subtree of e_i (as X is disappearing at e_i). Due to the connectedness requirement, for every node (a.k.a., edge) $e \in S$, the whole path from e_i to e must also be in S. Note also that, before processing e_i, Line 4 must have already processed all the other nodes in S (due to the reverse topological order at Line 2), yet none of them had been added to F_{tmp} (otherwise, X would have been covered by F_{tmp} before Line 4 processed e_i). Under our modification strategy, if a node escapes being included into F_{tmp}, it passes its weight under W' to its parent. Hence, when Line 4 encountered e_i, $W'(e_i)$ must have accumulated the weight $W(e)$ (NOTE: It is W here, not W') of every edge $e \in S$. Thus, $W'(e_i)$ is at least $\sum_{e \in S} W(e)$, which in turn must be at least 1 because the original W is a fractional edge covering. This proves that our modification satisfies **C3.2-2** and, hence, statement (i).

We now proceed to prove statement (ii). For any node e in T, whether e is added to F_{tmp} is determined by which of the proper descendants of e are included into F_{tmp}. Line 4 processes all those descendants before e (due to the reverse topological order). The observation gives rise to an inductive argument. First, if e is a leaf, e enters F_{tmp} if and only if it has a disappearing attribute (which must be exclusive), independently of the reverse topological order used. For a non-leaf node e, inductively, once we have decided whether e' should be added to F_{tmp} for every proper descendent e' of e, whether e will enter F_{tmp} has also been decided. We thus conclude that the reverse topological order has no influence on the output.

To prove statement (iii), consider any raw leaf e of T. If e is not the root of T, it must have an attribute X absent from $\mathit{parent}(e)$ (otherwise, e is subsumed by its parent and G is not reduced). Similarly, if e is the root of T, it must have an attribute X absent from its child (there is only one child because e is a raw leaf). In both cases, the attribute X is exclusive at e and will force edge-cover to add e to F_{tmp}. □

As a remark, Lemma 8 implies that any acyclic hypergraph G has an integral optimal fractional edge covering that maps every edge of G to either 0 or 1.

3.2 Signature Paths, Clusterings, k-Groups, Anchor Leaves, and Anchor Attributes

Suppose that we have already computed the CEC $\mathcal{F}$ of $G = (V, E)$ induced by an edge tree T of G. This subsection will introduce several concepts derived from $\mathcal{F}$ that are important to our analysis.

Whenever $\mathcal{F}$ includes the root of T, we can define a *signature path* — denoted as $\text{sigpath}(f, T)$ — for each node $f \in \mathcal{F}$. Specifically, $\text{sigpath}(f, T)$ is a set of nodes obtained as follows.

— If f is the root of T, $\text{sigpath}(f, T) = \{f\}$.

— Otherwise, let $\hat{f}$ be the lowest node in $\mathcal{F}$ that is a proper ancestor of f. Then, sigpath(f, T) is the set of nodes on the path from $\hat{f}$ to f, except $\hat{f}$.

Example 3.3. Consider the set $\mathcal{F} = \{\mathsf{ABC}, \mathsf{BD}, \mathsf{BO}, \mathsf{EFG}, \mathsf{HI}, \mathsf{LM}, \mathsf{EHJ}, \mathsf{HK}, \mathsf{HN}\}$ obtained in Example 3.2 (see Figure 1 for the edge tree T). If $f = \mathsf{ABC}$, then $\hat{f} = \mathsf{EHJ}$; and the signature path of f is $\{\mathsf{ABC}, \mathsf{BCE}, \mathsf{CEJ}\}$. If $f = \mathsf{HN}$, then the signature path of f is $\{\mathsf{HN}\}$ ($\hat{f}$ is not defined).

The concepts to be defined in the remainder of this subsection apply only if G is reduced. When G is reduced, $\mathcal{F}$ contains the root and all the leaves of T (Lemma 8). In this case, we define the *T-clustering* of G as

$$\mathcal{C} = \{\text{sigpath}(f, T) \mid f \in \mathcal{F}\}. \tag{19}$$

For each $f \in \mathcal{F}$, we will refer to sigpath(f, T) as a *cluster* of $\mathcal{C}$. Note that every node of T (or equivalently, every edge of E) belongs to at least one—but possibly more than one—cluster. If f is not the root of T, we call sigpath(f, T) a *non-root* cluster. Given an integer $k \geq 1$, we define a *k-group* of $\mathcal{C}$ to be a collection of k edges, each taken from a *distinct* cluster in $\mathcal{C}$.

Example 3.4. In Example 3.2 (see Figure 1 for the edge tree T, where the circled edges constitute the CEC $\mathcal{F}$), the T-clustering of G is

$$\mathcal{C} = \{\{\mathsf{BO}, \mathsf{BCE}, \mathsf{CEJ}\}, \{\mathsf{ABC}, \mathsf{BCE}, \mathsf{CEJ}\}, \{\mathsf{BD}, \mathsf{BCE}, \mathsf{CEJ}\}, \{\mathsf{EFG}, \mathsf{CEF}, \mathsf{CEJ}\}, \{\mathsf{HI}\}, \{\mathsf{EHJ}\}, \{\mathsf{LM}, \mathsf{KL}\}, \{\mathsf{HK}\}, \{\mathsf{HN}\}\}.$$

All the clusters except $\{\mathsf{HN}\}$ in $\mathcal{C}$ are non-root clusters. A 3-group example is $\{\mathsf{ABC}, \mathsf{BD}, \mathsf{EFG}\}$. In general, the edges in a k-group do not need to be distinct. For example, $\{\mathsf{CEJ}, \mathsf{CEJ}, \mathsf{CEJ}\}$ is also a 3-group: the first CEJ is taken from the cluster $\{\mathsf{ABC}, \mathsf{BCE}, \mathsf{CEJ}\}$, the second from $\{\mathsf{BD}, \mathsf{BCE}, \mathsf{CEJ}\}$, and the third from $\{\mathsf{EFG}, \mathsf{CEF}, \mathsf{CEJ}\}$. For a non-example, $\{\mathsf{ABC}, \mathsf{LM}, \mathsf{KL}\}$ is not a 3-group.

Let f_{anc} be a leaf node in $\mathcal{F}$, and $\hat{f}$ be the lowest proper ancestor of f_{anc} in $\mathcal{F}$. We call f_{anc} an *anchor leaf* of T if

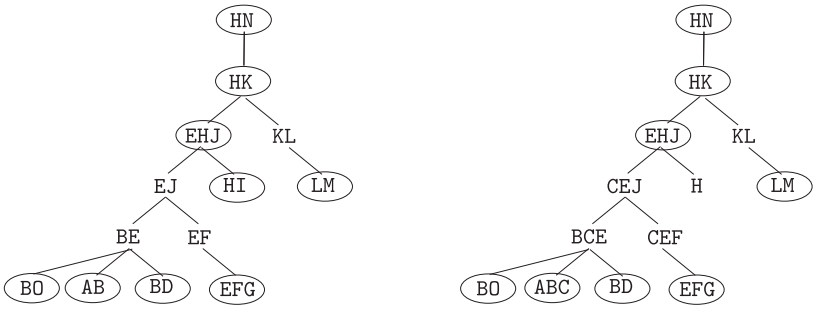
— $\hat{f}$ has no non-leaf proper descendant in $\mathcal{F}$, and
— f_{anc} has an attribute A_{anc} such that
  – $A_{\text{anc}} \notin \hat{f}$;
  – $A_{\text{anc}} \in e$ for every node $e \in \text{sigpath}(f_{\text{anc}}, T)$.

We call A_{anc} an *anchor attribute* of f_{anc}. The above definition does not apply to the case where $\hat{f} = nil$ (i.e., T has only a single node, which is f_{anc}); in that special case, we define the anchor leaf of T to be f_{anc} and call any attribute in f_{anc} an anchor attribute.

Lemma 9. *If G is non-empty and reduced, $\mathcal{F}$ always contains an anchor leaf of T.*

Proof. We discuss only the case where $f_{\text{anc}} \neq nil$ (the opposite case is trivial). Identify an arbitrary non-leaf node $f \in \mathcal{F}$ with the property that no other non-leaf node in $\mathcal{F}$ is lower than f. The existence of f is guaranteed because $\mathcal{F}$ includes the root of T. Consider any child node e of f in T. Since G is reduced, e must have an attribute A_{anc} that does not appear in f. Let f_{anc} be any node in $\mathcal{F}$ that contains A_{anc}. By the connectedness requirement of acyclicity, f_{anc} must be in the subtree of T rooted at e and, therefore, must be a leaf (by definition of f). We argue that f_{anc} is an anchor leaf of T. The signature path of f_{anc} includes all the nodes on the path from e to f_{anc}. Because $A_{\text{anc}} \in e$ and $A_{\text{anc}} \in f_{\text{anc}}$, A_{anc} must appear in all the nodes on the path (due to connectedness) and is thus an anchor attribute of f_{anc}. □

Example 3.5. In Example 3.4 (see Figure 1 for the edge tree T, where the circled edges constitute the CEC $\mathcal{F}$), we have obtained the T-clustering $\mathcal{C} = \{\{\mathsf{BO}, \mathsf{BCE}, \mathsf{CEJ}\}, \{\mathsf{ABC}, \mathsf{BCE}, \mathsf{CEJ}\}, \{\mathsf{BD}, \mathsf{BCE}, \mathsf{CEJ}\}, \{\mathsf{EFG}, \mathsf{CEF}, \mathsf{CEJ}\}, \{\mathsf{HI}\}, \{\mathsf{EHJ}\}, \{\mathsf{LM}, \mathsf{KL}\}, \{\mathsf{HK}\}, \{\mathsf{HN}\}\}$. ABC is an anchor leaf of T with an anchor



(a) T' and $\mathcal{F}'$ after removing attribute C (b) T' and $\mathcal{F}'$ after removing attribute I

Fig. 3. Residual hypergraphs.

attribute C. HI is another anchor leaf with an anchor attribute I. For a non-example, BD is not an anchor leaf because it does not have an attribute that exists in all the nodes in sigpath(BD, T) = {BD, BCE, CEJ}. Furthermore, LM is not an anchor leaf because HK, the lowest proper ancestor of LM in $\mathcal{F}$, has a non-leaf proper descendant in $\mathcal{F}$ (i.e., EHJ).

3.3 CEC Properties after Removing an Anchor Attribute

This subsection assumes that the input hypergraph $G = (V, E)$ is reduced. As before, let T be an edge tree of G and $\mathcal{F}$ be the CEC induced by T. Identify an arbitrary anchor leaf f_{anc} of T and an arbitrary anchor attribute A_{anc} of f_{anc}. As will be clear in Section 4, in join evaluation, we will need to simplify G by removing A_{anc}. CEC has several interesting properties under such simplification as discussed next.

3.3.1 Residual Hypergraph and Its CEC. Removing A_{anc} from G produces a *residual hypergraph* $G' = (V', E')$ where

- $V' = V \setminus \{A_{\text{anc}}\}$, and
- E' is produced by including, for every $e \in E$, an edge $map(e) = e \setminus \{A_{\text{anc}}\}$.

Define map^{-1} as the inverse function of map, namely, for each $e' \in E'$, $map^{-1}(e')$ is the unique edge $e \in E$ satisfying $e' = map(e)$. The functions map and map^{-1} capture the one-one correspondence between E and E'.

Denote by T' the edge tree of G' obtained by discarding A_{anc} from every node in T. The next lemma, whose proof can be found in Appendix B, shows that the CEC of G' induced by T' can be derived directly from $\mathcal{F}$.

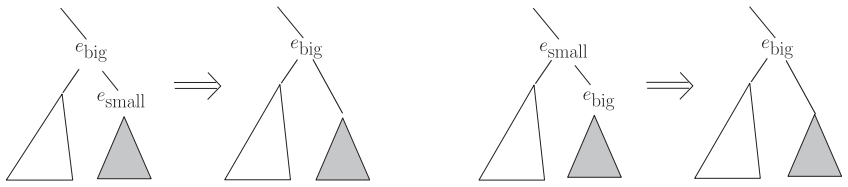
Lemma 10. *If G is reduced, the CEC of G' induced by T' is*

$$\mathcal{F}' = \begin{cases} \mathcal{F} \setminus \{f_{anc}\} & \text{if } map(f_{anc}) \text{ is subsumed in } G' \\ \{map(f) \mid f \in \mathcal{F}\} & \text{otherwise} \end{cases} \tag{20}$$

Furthermore, if an edge $e' \in E'$ is subsumed, then $e' \notin \mathcal{F}'$.

Example 3.6. Continuing on Example 3.5 (see Figure 1 for the edge tree T, where the circled edges constitute $\mathcal{F}$), suppose that we choose f_{anc} = ABC and eliminate A_{anc} = C from the tree T. Figure 3(a) illustrates the edge tree T' obtained, where the circled nodes constitute the set $\mathcal{F}'$. Similarly, if we choose f_{anc} = HI with A_{anc} = I, then T' and $\mathcal{F}'$ are as illustrated in Figure 3(b).

3.3.2 Cleansing the Residual Graph and Preserving the CEC. Even though G is reduced, the residual hypergraph G' may contain subsumed edges. Next, we describe a *cleansing* procedure which



(a) e_{big} parents e_{small} (b) e_{small} parents e_{big}

Fig. 4. Two cases of cleansing.

converts G' into a reduced hypergraph $G^* = (V', E^*)$ (note that G^* has the same vertices as G') and converts T' into an edge tree T^* of G^*.

cleanse (G', T')
1. **if** $map(f_{\text{anc}})$ is subsumed in G' **then**
2. $G^* \leftarrow$ the hypergraph obtained by removing edge $map(f_{\text{anc}})$ from G',
   $T^* \leftarrow$ the tree obtained by removing the leaf $map(f_{\text{anc}})$ from T'
3. **return** G^* and T^*
/* the following assumes that $map(f_{\text{anc}})$ not subsumed */
4. $G^* \leftarrow G', T^* \leftarrow T'$
5. **while** G^* has edges e_{small} and e_{big} s.t. $e_{\text{small}} \subseteq e_{\text{big}}$ and they are connected by a link in T^* **do**
6. remove e_{small} from G^* and T^* /* by Lemma 10, e_{small} cannot belong to $\mathcal{F}'$ */
7. **if** e_{big} was the parent of e_{small} in T^* **then**
8. make e_{big} the new parent for all the child nodes of e_{small}; see Figure 4(a)
   else
9. make e_{big} the new parent for the child nodes of e_{small}, and
   make e_{big} a child of the (original) parent of e_{small} in T^*; see Figure 4(b)
10. **return** G^* and T^*

At the end of cleansing, we always set $\mathcal{F}^* = \mathcal{F}'$ directly. An important property of cleansing is that it does not affect the CEC, as formally stated below.

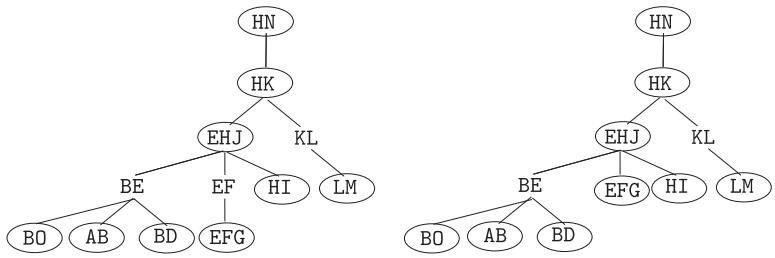
Lemma 11. *After cleansing, $\mathcal{F}^*$ is the CEC of G^* induced by T^*.*

The proof of the lemma can be found in Appendix C.

Example 3.7. In Example 3.6, the residual hypergraph G' in Figure 3(a) has two subsumed edges EJ and EF, each removed by an iteration of cleanse. Suppose that the first iteration sets $e_{\text{small}} = \text{EJ}$ and $e_{\text{big}} = \text{EHJ}$ (this is a case of Figure 4(a)). Figure 5(a) illustrates the T^* after removing EJ. The next iteration sets $e_{\text{small}} = \text{EF}$ and $e_{\text{big}} = \text{EFG}$ (a case of Figure 4(b)). Figure 5(b) illustrates the T^* after removing EF. In both Figures 5(a) and 5(b), the circled nodes constitute the CEC $\mathcal{F}^*$ of G^* induced by T^*.

3.3.3 Preserving k-Groups. The next property concerns the hypergraph $G^* = (V', E^*)$ after cleansing the original hypergraph $G = (V, E)$. Recall that T^* and T are edge trees of G^* and G, respectively. Before proceeding, the reader should recall that every edge $e^* \in E^*$ corresponds to a distinct edge $map^{-1}(e^*) \in E$.

Lemma 12. *For any $k \in [|\mathcal{F}^*|]$, if $\{e_1^*, \ldots, e_k^*\}$ is a k-group of the T^*-clustering $\mathcal{C}^*$ of G^*, then $\{map^{-1}(e_1^*), \ldots, map^{-1}(e_k^*)\}$ must be a k-group of the T-clustering $\mathcal{C}$ of G.*



(a) After removing EJ (b) After removing EF

Fig. 5. Changes to T^* during cleansing.

By definition of k-group (see Section 3.2), $e_1^*, \ldots, e_k^*$ originate from k distinct clusters in C^*. The lemma essentially promises k different clusters in C, each of which contains a distinct edge in $\{map^{-1}(e_1^*), \ldots, map^{-1}(e_k^*)\}$.

Example 3.8. Consider the T^* and $\mathcal{F}^*$ illustrated in Figure 5(b). The T^*-clustering of G^* is $C^* = \{\{\mathtt{BO}, \mathtt{BE}\}, \{\mathtt{AB}, \mathtt{BE}\}, \{\mathtt{BD}, \mathtt{BE}\}, \{\mathtt{EFG}\}, \{\mathtt{EHJ}\}, \{\mathtt{HI}\}, \{\mathtt{LM}, \mathtt{KL}\}, \{\mathtt{HK}\}, \{\mathtt{HN}\}\}$. Because $\{\mathtt{BE}, \mathtt{EFG}, \mathtt{KL}\}$ is a 3-group of C^*, Lemma 12 asserts that $\{map^{-1}(\mathtt{BE}), map^{-1}(\mathtt{EFG}), map^{-1}(\mathtt{KL})\} = \{\mathtt{BCE}, \mathtt{EFG}, \mathtt{KL}\}$ must be a 3-group of the T-clustering of G in Example 3.4 (see Figure 1 for the edge tree T, where the circled edges constitute $\mathcal{F}$).

3.4 CEC Properties after Removing a Signature Path

This subsection will discuss another simplification needed in join evaluation. As before, we have a hypergraph $G = (V, E)$, and denote by T an edge tree of G and by $\mathcal{F}$ the CEC of G induced by T. Identify an arbitrary anchor leaf f_{anc} of T and an arbitrary anchor attribute A_{anc} of f_{anc}. The simplification deletes all the edges in the signature path $\text{sigpath}(f_{\text{anc}}, T)$ from G. Next, we discuss the properties of CEC under such simplification.

3.4.1 Residual Hypergraphs and Their CECs. Removing $\text{sigpath}(f_{\text{anc}}, T)$ decomposes G into multiple components. To explain, define

$$Z = \{\text{node } z \text{ in } T \mid z \notin \text{sigpath}(f_{\text{anc}}, T) \text{ and } parent(z) \in \text{sigpath}(f_{\text{anc}}, T)\}. \tag{21}$$

For each $z \in Z$, define a rooted tree T_z^* as follows:

— The root of T_z^* is the parent of z in T;
— The root of T_z^* has only one child in T_z^*, which is z;
— The subtree rooted at z in T_z^* is the same as the subtree rooted at z in T.

Separately, define $\overline{T^*}$ as the rooted tree obtained by removing from T the subtree rooted at the highest node in $\text{sigpath}(f_{\text{anc}}, T)$.

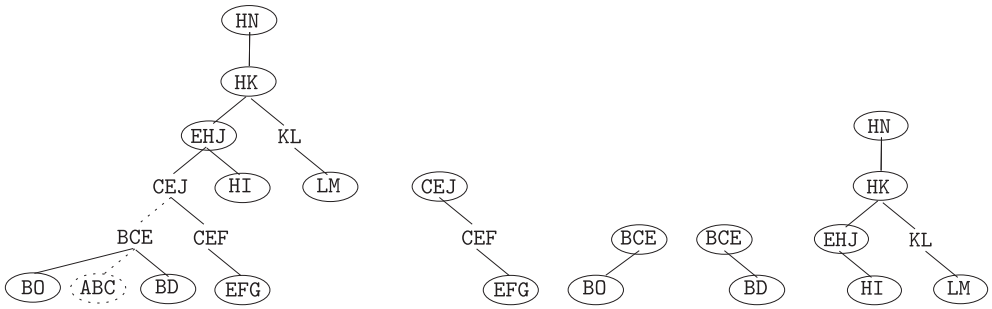
From each T_z^*, generate a *residual hypergraph* $G_z^* = (V_z^*, E_z^*)$ where:

— E_z^* includes all and only the nodes (a.k.a., edges of G) in T_z^*;
— V_z^* is the set of attributes appearing in at least one edge in E_z^*.

Likewise, from $\overline{T^*}$, generate a *residual hypergraph* $\overline{G^*} = (\overline{V^*}, \overline{E^*})$ where

— $\overline{E^*}$ includes all and only the nodes in $\overline{T^*}$;
— $\overline{V^*}$ is the set of attributes appearing in at least one edge in $\overline{E^*}$.

Because G is reduced, so must be all the residual hypergraphs. For each $z \in Z$, T_z^* is an edge tree of G_z^*; similarly, $\overline{T^*}$ is an edge tree of $\overline{G^*}$.



(a) The dotted path is sigpath(ABC, T) (b) T^*_{CEF} (c) T^*_{BO} (d) T^*_{BD} (e) $\overline{T^*}$

Fig. 6. Residual hypergraphs after removing a signature path.

Example 3.9. In Example 3.5 (see Figure 1 for the edge tree T, where the circled edges constitute $\mathcal{F}$), suppose that we choose $f_{\text{anc}} = \mathsf{ABC}$, whose signature path is sigpath(f_{anc}, T) = {ABC, BCE, CEJ}; see Figure 6(a). Then, Z = {BO, BD, CEF}. Figures 6(b), 6(c), and 6(d) illustrate T^*_z for z = CEF, BO, and BD, respectively, while Figure 6(e) gives $\overline{T^*}$.

Recall that $\mathcal{F}$ is the CEC of G induced by T. The next lemma shows that the CECs of the residual hypergraphs can be derived from $\mathcal{F}$ effortlessly.

Lemma 13. *For each node $z \in Z$, the CEC of G^*_z induced by T^*_z is*

$$\mathcal{F}^*_z = \{\textit{parent of } z\} \cup (\mathcal{F} \cap E^*_z). \tag{22}$$

The CEC of $\overline{G^*}$ induced by $\overline{T^*}$ is

$$\overline{\mathcal{F}^*} = \mathcal{F} \cap \overline{E^*}. \tag{23}$$

The proof can be found in Appendix E.

Example 3.10. Continuing on Example 3.6, we have circled the nodes of $\mathcal{F}^*_z$ in Figures 6(b), 6(c), and 6(d) for z = CEF, BO, and BD, respectively. Similarly, the circled nodes in Figure 6(e) constitute $\overline{\mathcal{F}^*}$.

3.4.2 Preserving k-Groups. We close the section with a property resembling Lemma 12. Define a *super-k-group* to be a set of edges $K = \{e_1, e_2, \ldots, e_k\}$ satisfying:

— each e_i, $i \in [k]$, is taken from either a cluster of the $\overline{T^*}$-clustering of $\overline{G^*}$ or a non-root cluster[6] of the T^*_z-clustering of G^*_z for some $z \in Z$;
— no two edges in K are taken from the same cluster.

Then, we have:

Lemma 14. *Any super-k-group $\{e_1, e_2, \ldots, e_k\}$ must also be a k-group of the T-clustering of G.*

The proof can be found in Appendix F.

Example 3.11. In Figure 6, the T^*_z-clusterings of G^*_z for z = CEF, BO, and BD are C^*_{CEF} = {{EFG, CEF}, {CEJ}}, C^*_{BO} = {{BO}, {BCE}}, and C^*_{BD} = {{BD}, {BCE}}, respectively, while the $\overline{T^*}$-clustering of $\overline{G^*}$ is $\overline{C^*}$ = {{HI}, {EHJ}, {HK}, {HN}, {LM, KL}}. A super-3-group example is {CEF, BO, KL}. Lemma 14 assures us that {CEF, BO, KL} must be a 3-group in the T-clustering of G given in

[6]Namely, e_i cannot be the root of T^*_z.

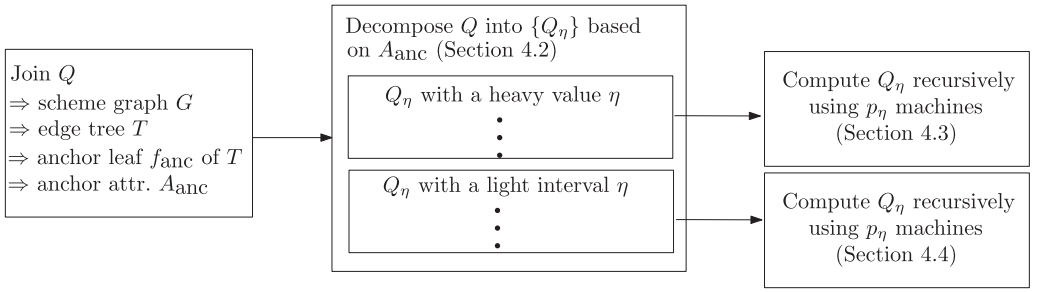


Fig. 7. Overview of our MPC Algorithm in Section 4.

Example 3.4 (see Figure 1 for the edge tree T, where the circled edges constitute $\mathcal{F}$). As non-examples, {CEJ, BO, KL} is not a super-3-group because CEJ does not come from a non-root cluster of C^*_{CEF}, and neither is {EFG, CEF, KL} because EFG and CEF must come from the same cluster of C^*_{CEF}.

4 AN MPC ALGORITHM FOR ACYCLIC QUERIES

The following sections will apply the theory of CECs to solve acyclic joins in the MPC model. Specifically, we will describe a new algorithm in this section and present its analysis in Section 5. Figure 7 provides an overview of our algorithm.

4.1 Fundamental Definitions

In this subsection, we will introduce several basic definitions applicable to general acyclic queries. Consider an acyclic query Q whose schema graph is $G = (V, E)$. Fix an arbitrary edge tree T of G and use the edge-cover algorithm (in Section 3.1) to compute the CEC $\mathcal{F}$ of G induced by T. Let C be the T-clustering of G in (19).

Recall that a *k-group* of C is a collection of k edges, each taken from a distinct cluster in C. Given a k-group K of C, we define

$$Q\text{-product of } K = \prod_{e \in K} |R_e| \tag{24}$$

where R_e is the (only) relation in Q with scheme e; the Q-product of K is simply the Cartesian-product size of all the input relations corresponding to the edges in K. Given an integer $k \in [|\mathcal{F}|]$, we define the *max-(k, Q)-product* of C as the largest Q-product of all k-groups K, or formally:

$$P_k(Q, C) = \max_{k\text{-group } K \text{ of } C} Q\text{-product of } K. \tag{25}$$

As the Q-product of any k-group is at most N^k where N is the input size of Q, we always have $P_k(Q, C) \leq N^k$. Finally, define

$$Q\text{-induced load of } C = \max_{k=1}^{|C|} \left(\frac{P_k(Q, C)}{p} \right)^{1/k} \tag{26}$$

where $|C|$ is the number of clusters in C.

As $P_k(Q, C) \leq N^k$ for any $k \in [|C|]$, the Q-induced load of C is at most $N/p^{1/|C|}$. Another useful fact is $P_1(Q, C) = \Theta(N)$ (which holds because Q has a constant number of relations). This means that the Q-induced load of C is $\Omega(N/p)$.

Example 4.1. For an illustration, let us revisit Example 3.4 (see Figure 1 for the edge tree T, where the circled edges constitute $\mathcal{F}$). In that example, we obtained the T-clustering $C = \{\{\text{BO}, \text{BCE}, \text{CEJ}\}, \{\text{ABC}, \text{BCE}, \text{CEJ}\}, \{\text{BD}, \text{BCE}, \text{CEJ}\}, \{\text{EFG}, \text{CEF}, \text{CEJ}\}, \{\text{HI}\}, \{\text{EHJ}\}, \{\text{LM}, \text{KL}\}, \{\text{HK}\}, \{\text{HN}\}\}$. For the 3-group

$\{\mathtt{ABC}, \mathtt{BD}, \mathtt{EFG}\}$, its Q-product is $|R_{\mathtt{ABC}}| \cdot |R_{\mathtt{BD}}| \cdot |R_{\mathtt{EFG}}|$. As $P_k(Q, C) \leq N^k$ for any $k \leq |C| = 9$, the Q-induced load of C is at most $N/p^{1/9}$.

4.2 Configurations

Henceforth, we fix Q to be the acyclic query to be answered. Denote by $G = (V, E)$ the schema graph of Q. We assume G to be reduced; otherwise, Q can be converted in load $O(N/p)$ to a query that has the same result but with a reduced schema graph [14, 17]. We will also assume that Q has at least two relations; otherwise, the query is trivial and requires no communication.

Choose an arbitrary edge tree T of G and compute the CEC $\mathcal{F}$ of G induced by T. Define C as the T-clustering of G given in (19). Choose an arbitrary anchor leaf f_{anc} of T and an arbitrary anchor attribute A_{anc} of f_{anc}; remember that A_{anc} appears in all the edges of $\text{sigpath}(f_{\text{anc}}, T)$.

For each edge $e \in E$, let R_e represent the relation in Q whose scheme is e. Fix a value $x \in \mathbf{dom}$. Given an edge $e \in \text{sigpath}(f_{\text{anc}}, T)$, we define the *$A_{anc}$-frequency* of x in R_e as the number of tuples $\boldsymbol{u} \in R_e$ such that $\boldsymbol{u}(A_{\text{anc}}) = x$. Moreover, define the *signature-path A_{anc}-frequency* of x as the sum of its A_{anc}-frequencies in the R_e of all $e \in \text{sigpath}(f_{\text{anc}}, T)$.

Example 4.2. Continuing on Example 4.1 (see Figure 1 for the edge tree T, where the circled edges constitute $\mathcal{F}$), let us choose the anchor leaf f_{anc} to be $\mathtt{ABC}$, which has an anchor attribute $A_{\text{anc}} = \mathtt{C}$ and the signature path $\text{sigpath}(\mathtt{ABC}, T) = \{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}$. Fix a value $c \in \mathbf{dom}$. The $\mathtt{C}$-frequency of c in relation $R_{\mathtt{ABC}}$ is the number of tuples $\boldsymbol{u} \in R_{\mathtt{ABC}}$ satisfying $\boldsymbol{u}(\mathtt{C}) = c$. The signature-path $\mathtt{C}$-frequency of c is the total number of tuples $\boldsymbol{u}$ in $R_{\mathtt{ABC}}, R_{\mathtt{BCE}}, R_{\mathtt{CEJ}}$ satisfying $\boldsymbol{u}(\mathtt{C}) = c$.

We will use L to represent the Q-induced load of C defined in (26). Given a value $x \in \mathbf{dom}$, we say that x is

- *heavy*, if its signature-path A_{anc}-frequency is at least L;
- *light*, otherwise.

Divide $\mathbf{dom}$ into disjoint intervals such that either the entire $\mathbf{dom}$ is one interval or the light values in each interval have a total signature-path A_{anc}-frequency of $\Theta(L)$. We will refer to those intervals as the *light intervals* of A_{anc}.

A *configuration* η is either a heavy value or a light interval of A_{anc}. The number of configurations, which is the total number of heavy values and light intervals, is at most

$$\begin{aligned}\sum_{\eta} 1 &= \sum_{e \in \text{sigpath}(f_{\text{anc}}, T)} O\left(1 + \frac{|R_e|}{L}\right) \\ &= O\left(1 + \max_{e \in \text{sigpath}(f_{\text{anc}}, T)} \frac{|R_e|}{L}\right) = O\left(1 + \frac{\text{max-}(1, Q)\text{-product of } C}{L}\right) = O(p) \qquad (27)\end{aligned}$$

where the second equality used the fact that $\text{sigpath}(f_{\text{anc}}, T)$ has $O(1)$ edges and the third equality applied the definition of the max-(k, Q)-product of C in (25) and the definition of L, i.e., the Q-induced load of C in (26).

For each edge $e \in E$, define a relation $R(e, \eta)$ as follows:

- if η is a heavy value, $R(e, \eta)$ includes all and only the tuples $\boldsymbol{u} \in R_e$ satisfying $\boldsymbol{u}(A_{\text{anc}}) = \eta$;
- if η is a light interval, $R(e, \eta)$ includes all and only the tuples $\boldsymbol{u} \in R_e$ where $\boldsymbol{u}(A_{\text{anc}})$ is a light value in η.

Note that $R(e, \eta) = R_e$ if $A_{\text{anc}} \notin e$. We associate the configuration with a query

$$Q_\eta = \{R(e, \eta) \mid e \in E\}.$$

Our objective is to compute $\mathcal{J}oin(Q_\eta)$ for all η in parallel. The final result $\mathcal{J}oin(Q)$ is simply $\bigcup_\eta \mathcal{J}oin(Q_\eta)$.

Note that Q_η has the same schema graph G as Q. Recall that C is the T-clustering of G. The rest of the section will explain how to solve $\mathcal{J}oin(Q_\eta)$ for an arbitrary η using

$$p_\eta = \Theta\left(1 + \max_{k=1}^{|C|} \frac{P_k(Q_\eta, C)}{L^k}\right) \tag{28}$$

machines, where $P_k(Q_\eta, C)$ is the max-(k, Q_η)-product of C. As will be proved in Section 5, we can adjust the constants in (28) to make sure $\sum_\eta p_\eta \leq p$.

Example 4.3. To illustrate the above steps, we continue the discussion in Example 4.2. Recall that in this context the schema graph of the query Q has the edge set $E = \{\mathtt{ABC}, \mathtt{BD}, \mathtt{BO}, \mathtt{EFG}, \mathtt{BCE}, \mathtt{CEF}, \mathtt{CEJ}, \mathtt{HI}, \mathtt{LM}, \mathtt{EHJ}, \mathtt{KL}, \mathtt{HK}, \mathtt{HN}\}$. The edge tree T is shown in Figure 1, where the circled edges constitute $\mathcal{F}$. We have chosen the anchor leaf $f_{\text{anc}} = \mathtt{ABC}$ and the anchor attribute $A_{\text{anc}} = \mathtt{C}$.

Suppose that **dom** is the integer domain. Let $h_1, h_2, \ldots, h_x$ be the heavy values of attribute C, and let $l_1, l_2, \ldots, l_y$ — in ascending order — be the light values of C that appear in at least one input relation of Q. The set $\{h_1, h_2, \ldots, h_x, l_1, l_2, \ldots, l_y\}$ is sometimes called the "active domain" of C.

There are multiple approaches to create the desired light intervals. In the following, we describe one such approach. The (disjoint) intervals we are creating will all have the form $(c_{left}, c_{right}]$ and will be generated in ascending order of c_{left} (hence, also in ascending order of c_{right}). To create the first interval, we set $c_{left} = -\infty$ and scan $l_1, l_2, \ldots, l_y$ until reaching the largest i satisfying the condition that the total signature-path C-frequency of $l_1, \ldots, l_i$ does not exceed $2L$. If $i = y$, we set $c_{right} = \infty$, in which case the interval covers the entire **dom**. Otherwise, set $c_{right} = l_i$, i.e., the first interval is $(-\infty, l_i]$, in which case the total signature-path C-frequency of $l_1, \ldots, l_i$ must be at least L.

Iteratively, suppose that we have just obtained an interval ending at l_i for some $i \in [1, y-1]$. To create the next interval, we set its $c_{left} = l_i$ and scan $l_{i+1}, l_{i+2}, \ldots, l_y$ until reaching the largest j satisfying the condition that the total signature-path C-frequency of $l_{i+1}, \ldots, l_j$ does not exceed $2L$. If $j = y$, we set $c_{right} = \infty$, in which case no more intervals will be created. Otherwise, set $c_{right} = l_j$, i.e., the new interval is $(l_i, l_j]$, in which case the total signature-path C-frequency of $l_{i+1}, \ldots, l_j$ must be at least L.

There is a small issue with the above strategy: we may create more than one interval, but the last interval could have a total signature-path C-frequency less than L. The issue can be easily fixed by simply merging the last two intervals into one; the total signature-path C-frequency of the merged interval is at most $4L$. It is rudimentary to implement the above strategy under the MPC model in $O(1)$ rounds with load $O(N/p)$, which the reader can verify to be $O(L)$.

A configuration η is either a heavy value of attribute C or a light interval obtained earlier. To illustrate Q_η, consider first that η takes a heavy value h_i for some $i \in [x]$. For the edge $e = \mathtt{ABC}$, the relation $R(e, \eta)$ — namely, $R(\mathtt{ABC}, h_i)$ — is the set of tuples $\boldsymbol{u} \in R_{\mathtt{ABC}}$ with $\boldsymbol{u}(\mathtt{C}) = h_i$. The relations $R(\mathtt{BCE}, h_i)$, $R(\mathtt{CEJ}, h_i)$, and $R(\mathtt{CEF}, h_i)$ are defined similarly. For any edge $e \in \{\mathtt{BD}, \mathtt{BO}, \mathtt{EFG}, \mathtt{HI}, \mathtt{LM}, \mathtt{EHJ}, \mathtt{KL}, \mathtt{HK}, \mathtt{HN}\}$, the relation $R(e, h_i)$ is identical to R_e because $\mathtt{C} \notin e$. The query Q_η involves all the relations $R(e, \eta)$ thus defined for all $e \in E$. Now, consider that η takes a light interval I. For the edge $e = \mathtt{ABC}$, the relation $R(e, \eta)$ — namely, $R(\mathtt{ABC}, I)$ — is the set of tuples $\boldsymbol{u} \in R_{\mathtt{ABC}}$ satisfying $\boldsymbol{u}(\mathtt{C}) \in I$. The relations $R(\mathtt{BCE}, I)$, $R(\mathtt{CEJ}, I)$, and $R(\mathtt{CEF}, I)$ are defined similarly. Again, for any edge $e \in \{\mathtt{BD}, \mathtt{BO}, \mathtt{EFG}, \mathtt{HI}, \mathtt{LM}, \mathtt{EHJ}, \mathtt{KL}, \mathtt{HK}, \mathtt{HN}\}$, the relation $R(e, I)$ is identical to R_e. The query Q_η involves all the relations $R(e, \eta)$ thus defined for all $e \in E$.

4.3 Solving Q_η When η is a Heavy Value

Remove A_{anc} from G, and define the residual hypergraph $G' = (V', E')$, as well as the functions $map(.)$ and $map^{-1}(.)$, in the way explained in Section 3.3. We compute $\mathcal{J}oin(Q_\eta)$ in five steps.

Step 1. Send the tuples of $R(e, \eta)$, for all $e \in E$, to the p_η allocated machines such that each machine receives $\Theta(\frac{1}{p_\eta} \sum_{e \in E} |R(e, \eta)|)$ tuples.

Step 2. For each $e \in E$, convert $R(e, \eta)$ to $R^*(e', \eta)$ where $e' = map(e) = e \setminus \{A_{\text{anc}}\}$. Specifically, $R^*(e', \eta)$ is a copy of $R(e, \eta)$ but with A_{anc} discarded, or formally:

$$R^*(e', \eta) = \{\boldsymbol{u}[e'] \mid \text{tuple } \boldsymbol{u} \in R(e, \eta)\}. \tag{29}$$

Note that if $A_{\text{anc}} \notin e$, then $R^*(e', \eta) = R(e, \eta)$. No communication occurs as each machine simply discards A_{anc} from every tuple $\boldsymbol{u} \in R(e, \eta)$ in the local storage.

Step 3. Cleanse G' into $G^* = (V', E^*)$ by calling the cleanse algorithm in Section 3.3.1. Every time cleanse performs an iteration in Lines 5-9 with edges e_{small} and e_{big}, we perform a *semi-join* between $R^*(e_{\text{small}}, \eta)$ and $R^*(e_{\text{big}}, \eta)$. The semi-join removes every tuple $\boldsymbol{u}$ from $R^*(e_{\text{big}}, \eta)$ with the property that $\boldsymbol{u}[e_{\text{small}}]$ is absent from $R^*(e_{\text{small}}, \eta)$. $R^*(e_{\text{small}}, \eta)$ is discarded after the semi-join.

Step 4. Define a sub-query:

$$Q^*_\eta = \{R^*(e^*, \eta) \mid e^* \in E^*\}$$

Note that Q^*_η involves one less attribute than Q (because A_{anc} no longer exists). Compute $\mathcal{J}oin(Q^*_\eta)$ using p_η machines recursively.

Step 5. Output $\mathcal{J}oin(Q_\eta)$ by augmenting each tuple $\boldsymbol{u} \in \mathcal{J}oin(Q^*_\eta)$ with $\boldsymbol{u}(A_{\text{anc}}) = \eta$. No communication is needed.

Example 4.4. We will illustrate the above steps by continuing Example 4.2. Suppose that the configuration η is a heavy value h of attribute C. In Example 4.2, we have explained how to obtain relation $R(e, \eta)$ — namely, $R(e, h)$ — for each $e \in E$ (the content of E is listed at the beginning of Example 4.2). In Step 1, the relations $\{R(e, h) \mid e \in E\}$ are sent to the p_η machines allocated exclusively to η such that each machine receives roughly the same number of tuples.

In Step 2, each of the p_η machines examines the tuples in its local storage. If a tuple $\boldsymbol{u} \in R(e, h)$ is found, where $e \in \{\text{ABC}, \text{BCE}, \text{CEJ}, \text{CEF}\}$, the machine removes the C-value of $\boldsymbol{u}$ to create a new tuple $\boldsymbol{v} = \boldsymbol{u}[e']$, where $e' = e \setminus \{\text{C}\}$. Generating all such $\boldsymbol{v}$ effectively creates the relation $R^*(e', h)$. For any $e \in \{\text{BD}, \text{BO}, \text{EFG}, \text{HI}, \text{LM}, \text{EHJ}, \text{KL}, \text{HK}, \text{HN}\}$, we have $e' = e \setminus \{\text{C}\} = e$, and $R^*(e', h)$ is simply $R(e, h)$.

Recall that the edge tree T of the query Q is illustrated in Figure 1. After discarding attribute C, the edge tree T' is transformed into the one depicted in Figure 3(a). In Step 3, the cleanse procedure first removes edge EJ from T', as shown in Figure 5(a), and then edge EF, as shown in Figure 5(b). As a result, Step 3 performs a semi-join between $R^*(\text{EJ}, h)$ and $R^*(\text{EHJ}, h)$, after which $R^*(\text{EHJ}, h)$ can only shrink and $R^*(\text{EJ}, h)$ is discarded. This is followed by another semi-join between $R^*(\text{EF}, h)$ and $R^*(\text{EFG}, h)$, after which $R^*(\text{EFG}, h)$ can only shrink and $R^*(\text{EF}, h)$ is discarded.

The sub-query Q^*_η — that is, Q^*_h — now contains only relations $R^*(\text{AB}, h)$, $R^*(\text{BE}, h)$, $R^*(\text{BO}, h)$, $R^*(\text{BD}, h)$, $R^*(\text{EHJ}, h)$, $R^*(\text{EFG}, h)$, $R^*(\text{HI}, h)$, $R^*(\text{LM}, h)$, $R^*(\text{KL}, h)$, $R^*(\text{HK}, h)$, and $R^*(\text{HN}, h)$. Step 4 computes Q^*_h using the p_η allocated machines recursively. Finally, in Step 5, each tuple in the result of Q^*_h is then augmented with the C-value h to produce a result tuple for Q_η.

Tuple-Based Implementation. For the benefit of reader comprehension, we have deliberately presented our algorithm in a way that aligns conceptually with the discussion in Section 3.3. However, this approach may inadvertently lead to the misconception that our algorithm does not handle each tuple in the original relations of Q as atoms, as mandated by the class of tuple-based algorithms outlined in Section 1.1. Specifically, the confusion lies in removing the attribute A_{anc} in Step 2 and "concatenating" it back in Step 5.

Nevertheless, once the reader has grasped the underlying rationale, it is rudimentary to resolve the issue by electing for a tuple-based implementation. First, it is important to remember that η is the sole value under attribute A_{anc} for the sub-query Q_η we are processing. As mentioned, for each edge $e \in E$ containing A_{anc}, Step 2 removes A_{anc} from relation $R(e, \eta)$ by generating the relation

$R^*(e', \eta)$ of (29), where $e' = e \setminus \{A_{\text{anc}}\}$. Specifically, for each tuple $\boldsymbol{u} \in R(e, \eta)$, Step 2 adds the tuple $\boldsymbol{v} = \boldsymbol{u}[e']$ to $R^*(e', \eta)$, effectively retaining all values of $\boldsymbol{u}$ except $\boldsymbol{u}(A_{\text{anc}}) = \eta$. This gives rise to a sub-query devoid of attribute A_{anc}. Steps 3-4 evaluate this sub-query by moving tuples like $\boldsymbol{v}$ among the p_η allocated machines. Each result tuple of the sub-query needs to be augmented with the value η on attribute A_{anc} before being returned (Step 5). To allow each machine to perform such augmentation locally, we can broadcast η to all the p_η allocated machines (this increases the load by only one). This way, whenever $\boldsymbol{v}$ is communicated between two machines, we are in fact transmitting a pair $(\boldsymbol{v}, \eta)$, which is equivalent to sending the tuple $\boldsymbol{u}$ itself as an atom. This, thus, yields a tuple-based implementation of our algorithm.

4.4 Solving Q_η When η is a Light Interval

Remove sigpath(f_{anc}, T) from G, and define Z, G_z^* and T_z^* for each $z \in Z$, $\overline{G^*}$, and $\overline{T^*}$ all in the way described in Section 3.4. We compute $\mathit{Join}(Q_\eta)$ in four steps.

Step 1. Same as Step 1 of the algorithm in Section 4.3.

Step 2. For each $e \in$ sigpath(f_{anc}, T), broadcast $R(e, \eta)$ to all p_η machines. By definition of light interval, the size of every such $R(e, \eta)$ is at most L.

Step 3. For each $z \in Z$, define for $G_z^* = (V_z^*, E_z^*)$:

$$\begin{aligned} C_z^* &= \text{the } T_z^*\text{-clustering of } G_z^* \\ Q_{\eta,z}^* &= \text{the join } \{R(e, \eta) \mid e \in E_z^*\}. \end{aligned}$$

Similarly, define for $\overline{G^*} = (\overline{V^*}, \overline{E^*})$:

$$\begin{aligned} \overline{C^*} &= \text{the } \overline{T^*}\text{-clustering of } \overline{G^*} \\ \overline{Q_\eta^*} &= \text{the join } \{R(e, \eta) \mid e \in \overline{E^*}\}. \end{aligned}$$

Note that $\overline{Q_\eta^*}$ and the $Q_{\eta,z}^*$ of each $z \in Z$ have at least one less relation than Q due to the disappearance of f_{anc}.

Next, we compute the Cartesian product

$$\left(\times_{z \in Z} \mathit{Join}(Q_{\eta,z}^*)\right) \times \mathit{Join}(\overline{Q_\eta^*}) \tag{30}$$

using p_η machines. For that purpose, define for each $z \in Z$

$$p_{\eta,z} = \Theta\left(1 + \max_{k=1}^{|C_z^*|} \frac{P_k(Q_{\eta,z}^*, C_z^*)}{L^k}\right) \tag{31}$$

where $P_k(Q_{\eta,z}^*, C_z^*)$ is the max-$(k, Q_{\eta,z}^*)$-product of the clustering C_z^*. Similarly, define

$$\overline{p_\eta} = \Theta\left(1 + \max_{k=1}^{|\overline{C^*}|} \frac{P_k(\overline{Q_\eta^*}, \overline{C^*})}{L^k}\right) \tag{32}$$

where $P_k(\overline{Q_\eta^*}, \overline{C^*})$ is the max-$(k, \overline{Q_\eta^*})$-product of the clustering $\overline{C^*}$. We will prove later that $\mathit{Join}(Q_{\eta,z}^*)$ of each $z \in Z$ can be evaluated with load $O(L)$ using $p_{\eta,z}$ machines, and $\mathit{Join}(\overline{Q_\eta^*})$ can be evaluated with load $O(L)$ using $\overline{p_\eta}$ machines. Therefore, applying the Cartesian product algorithm given in Lemma 4 of [17], we can compute (30) with load $O(L)$ using

$$\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} \tag{33}$$

machines. As proved in Section 5, we can adjust the constants in (31) and (32) to make sure that (33) is at most the value p_η given in (28).

Step 4. We combine the Cartesian product in (30) with the tuples broadcast in Step 2 to derive $\mathcal{J}oin(Q_\eta)$ with no more communication. Specifically, for each tuple $\boldsymbol{u}$ in the Cartesian product, the machine where $\boldsymbol{u}$ resides outputs $\{\boldsymbol{u}\} \bowtie \left(\bowtie_{e \in \text{sigpath}(f_{\text{anc}}, T)} R(e, \eta)\right)$. It is easy to verify that all the tuples of $\mathcal{J}oin(Q_\eta)$ will be produced this way.

Example 4.5. Next, we illustrate the above steps by continuing Example 4.2. Suppose that the configuration η is a light interval I of attribute C. In Example 4.2, we have explained how to obtain relation $R(e, \eta)$ — namely, $R(e, I)$ — for each $e \in E$. In Step 1, the relations $\{R(e, h) \mid e \in E\}$ are sent to the p_η machines allocated exclusively to η such that each machine receives roughly the same number of tuples. In Step 2, all the tuples in relations $R(\mathsf{ABC}, I)$, $R(\mathsf{BCE}, I)$ and $R(\mathsf{CEJ}, I)$ are broadcast to all the p_η machines.

Recall that the edge tree T of the query Q is illustrated in Figure 1. After discarding edges ABC, BCE, and CEJ, we have $Z = \{\mathsf{CEF}, \mathsf{BO}, \mathsf{BD}\}$. For each edge $z \in Z$, we create a sub-query $Q^*_{\eta,z}$ as follows:

- For $z = \mathsf{CEF}$, the residual hypergraph G^*_{CEF} has an edge tree T^*_{CEF} depicted in Figure 6(b). Accordingly, $Q^*_{\eta,z}$ — or more specifically, $Q^*_{I,\mathsf{CEF}}$ — involves relations $R(\mathsf{CEJ}, I)$, $R(\mathsf{CEF}, I)$, and $R(\mathsf{EFG}, I)$.
- For $z = \mathsf{BO}$, the residual hypergraph G^*_{BO} has an edge tree T^*_{BO} depicted in Figure 6(c). Accordingly, $Q^*_{I,\mathsf{BO}}$ involves relations $R(\mathsf{BO}, I)$ and $R(\mathsf{BCE}, I)$.
- For $z = \mathsf{BD}$, the residual hypergraph G^*_{BD} has an edge tree T^*_{BD} depicted in Figure 6(d). Accordingly, $Q^*_{I,\mathsf{BD}}$ involves relations $R(\mathsf{BD}, I)$ and $R(\mathsf{BCE}, I)$.

In addition, the elimination of edges ABC, BCE, and CEJ also yields a residual hypergraph $\overline{G^*}$ with an edge tree $\overline{T^*}$ depicted in Figure 6(e). Accordingly, we create another sub-query $\overline{Q^*_\eta}$ — or more specifically, $\overline{Q^*_I}$ — which has relations $R(\mathsf{HI}, I)$, $R(\mathsf{LM}, I)$ $R(\mathsf{EHJ}, I)$, $R(\mathsf{KL}, I)$, $R(\mathsf{HK}, I)$, and $R(\mathsf{HN}, I)$.

We now have four sub-queries: $Q^*_{I,\mathsf{CEF}}$, $Q^*_{I,\mathsf{BO}}$, $Q^*_{I,\mathsf{BD}}$, and $\overline{Q^*_I}$. Step 3 computes the Cartesian product of their results using an algorithm developed by Kestman et al. [17]. They showed that if the following conditions hold:

- $\mathcal{J}oin(Q^*_{I,\mathsf{CEF}})$ can be computed with load $O(L)$ using $p_{I,\mathsf{CEF}}$ machines;
- $\mathcal{J}oin(Q^*_{I,\mathsf{BO}})$ can be computed with load $O(L)$ using $p_{I,\mathsf{BO}}$ machines;
- $\mathcal{J}oin(Q^*_{I,\mathsf{BD}})$ can be computed with load $O(L)$ using $p_{I,\mathsf{BD}}$ machines;
- $\mathcal{J}oin(\overline{Q^*_I})$ can be computed with load $O(L)$ using $\overline{p_I}$ machines.

then the Cartesian product can be computed with load $O(L)$ using $p_{I,\mathsf{CEF}} \cdot p_{I,\mathsf{BO}} \cdot p_{I,\mathsf{BD}} \cdot \overline{p_I}$ machines. The values of $p_{I,\mathsf{CEF}}$, $p_{I,\mathsf{BO}}$, and $p_{I,\mathsf{BD}}$ are computed using (31), while $\overline{p_I}$ is computed using (32). We will prove in the next section that $p_{I,\mathsf{CEF}} \cdot p_{I,\mathsf{BO}} \cdot p_{I,\mathsf{BD}} \cdot \overline{p_I}$ is at most p_η, i.e., the total number of machines allocated to configuration η. Each of the sub-queries $Q^*_{I,\mathsf{CEF}}$, $Q^*_{I,\mathsf{BO}}$, $Q^*_{I,\mathsf{BD}}$, and $\overline{Q^*_I}$ can be processed recursively to satisfy the four conditions mentioned earlier.

In Step 4, every time a machine computes a tuple $\boldsymbol{u}$ in the Cartesian product from Step 3, it outputs $\{\boldsymbol{u}\} \bowtie R(\mathsf{ABC}, I) \bowtie R(\mathsf{BCE}, I) \bowtie R(\mathsf{CEJ}, I)$ locally. Note that $R(\mathsf{ABC}, I)$, $R(\mathsf{BCE}, I)$, and $R(\mathsf{CEJ}, I)$ have been broadcast to all the p_η machines. By combining the outputs of the p_η machines, we obtain the result of Q_η.

5 ANALYSIS OF THE ALGORITHM

This section will establish another main result of the article:

Theorem 15. *Any acyclic join query Q as defined in Section 1.1 can be solved with load $O(N/p^{1/\rho^*})$, where N is the input size of Q, ρ^* is the fractional edge covering number of Q, and p is the number of machines.*

We will actually prove a stronger claim:

Lemma 16. *Consider an acyclic join query Q whose schema graph G is reduced. Let T be any edge tree of G and C be the T-clustering of G defined in (19). Our algorithm in Section 4 answers Q with load $O(L)$, where L is the Q-induced load of C defined in (26).*

Before proving the lemma, let us first clarify how it leads to Theorem 15. First, if G is not reduced, we can convert Q into another query with the same result whose schema graph is reduced, which can be done with load $O(N/p)$ using algorithms from [14] and [17]. If, on the other hand, G is reduced, the Q-induced load L of C is at most $N/p^{1/|C|}$ (as discussed in Section 4.1). As can be seen from (19), $|C|$ equals the size of $\mathcal{F}$, which by Lemma 8 is ρ^*. Therefore, $L = O(N/p^{1/\rho^*})$ and Theorem 15 follows.

The rest of the section serves as a proof of Lemma 16. All the notations below follow those in Section 4. Our proof is via induction on the number of participating attributes (i.e., $|V|$) and the number of participating relations (i.e., $|Q|$). If $|Q| = 1$, the lemma trivially holds. If $|V| = 1$, Q has only one relation (because Q is reduced) and the lemma again holds trivially. Next, assuming that the lemma holds on any query with *either* strictly less participating attributes *or* strictly less participating relations than Q, we will prove the lemma's correctness on Q. Our analysis will answer three questions:

(1) Why do we have enough machines to handle all configurations in parallel? In particular, we must show that $\sum_\eta p_\eta \leq p$, where p_η is the number of machines allocated to η, as is given in (28).
(2) Why does each step in Section 4.3 and 4.4 entail a load of $O(L)$?
(3) Why do we have $\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} \leq p_\eta$ in Step 3 of Section 4.4?

Settling these questions will complete the proof of Lemma 16.

Remark on Memory Usage. As a corollary of Theorem 15, our algorithm utilizes $O(N/p^{1/\rho^*})$ words of memory on each machine. More specifically, each machine receives in total $O(N/p^{1/\rho^*})$ "atom tuples" from the relations of Q, a.k.a., a subset of each relation in Q. Then, the machine locally computes the join induced by those subsets. Such computation can be done with no extra memory asymptotically[7] — recall that the join result is output by emission, rather than physically stored.

5.1 Total Number of Machines for All Configurations

It suffices to prove $\sum_\eta p_\eta = O(p)$ because adjusting the hidden constants will then ensure $\sum_\eta p_\eta \leq p$. For every $k \in [|C|]$, we will show

$$\frac{1}{L^k} \sum_\eta P_k(Q_\eta, C) = O(p), \tag{34}$$

where $P_k(Q_\eta, C)$ is the max-(k, Q)-product of the T-clustering C, as defined in (25). It will then follow that

$$\begin{aligned}
\sum_\eta p_\eta &= \sum_\eta O\left(1 + \max_{k=1}^{|C|} \frac{P_k(Q_\eta, C)}{L^k}\right) && \text{(by definition of } p_\eta \text{ in (28))} \\
&= \sum_\eta O\left(1 + \sum_{k=1}^{|C|} \frac{P_k(Q_\eta, C)}{L^k}\right) && \text{(because } |C| = O(1)\text{)}
\end{aligned}$$

[7] As CPU time is for free in our model, one can compute the join on those subsets using a nested loop. For better practical efficiency, one can apply any of the join algorithms [5, 18, 21–25, 33] in RAM, which all require memory at the same order as the input size.

$$= O(p) + \sum_{k=1}^{|C|} O\left(\sum_{\eta} \frac{P_k(Q_\eta, C)}{L^k}\right) \quad \text{(because } \sum_{\eta} 1 = O(p), \text{ as shown in (27))}$$
$$= O(p) \quad \text{(by (34)).}$$

Now, fix k to an arbitrary integer in $[|C|]$. For any configuration η, the schema graph of Q_η is always G (i.e., same as the schema graph of Q). Consider an arbitrary k-group K of C (the concept of k-group was defined in Section 3.2). The Q_η-product of K, defined in (24), is $\prod_{e\in K} |R(e,\eta)|$. Given any K, we will prove

$$\frac{1}{L^k} \sum_{\eta} \prod_{e\in K} |R(e,\eta)| = O(p). \tag{35}$$

The above will then yield

$$\begin{aligned}
\sum_{\eta} \frac{P_k(Q_\eta, C)}{L^k} &= \sum_{\eta} \frac{1}{L^k} \max_K \prod_{e\in K} |R(e,\eta)| \\
&\quad \text{(by definition of } P_k(Q_\eta, C) \text{ in (25), applying also (24))} \\
&= O\left(\sum_{\eta} \frac{1}{L^k} \sum_K \prod_{e\in K} |R(e,\eta)|\right) \\
&\quad \text{(as } C \text{ has only a constant number of } k\text{-groups } K) \\
&= O\left(\sum_K \frac{1}{L^k} \sum_{\eta} \prod_{e\in K} |R(e,\eta)|\right) = \sum_K O(p) \quad \text{(using (35))} \\
&= O(p)
\end{aligned}$$

as claimed in (34).

It remains to prove (35). Let us first consider the case where $K \cap \text{sigpath}(f_{\text{anc}}, T) \neq \emptyset$, namely, K has an edge e_0 picked from the cluster $\text{sigpath}(f_{\text{anc}}, T)$. In this case, we have:

$$\sum_{\eta} \prod_{e\in K} |R(e,\eta)| = \sum_{\eta} \left(|R(e_0,\eta)| \cdot \prod_{e\in K\setminus\{e_0\}} |R(e,\eta)|\right) \tag{36}$$

For each $e \in K \setminus \{e_0\}$, obviously $|R(e,\eta)| \leq |R_e|$. Regarding e_0, because A_{anc} must be an attribute of e_0, the relations $R(e_0,\eta)$ of all the configurations η form a *partition* of $R(e_0)$.[8] Hence:

$$\begin{aligned}
(36) &\leq \left(\prod_{e\in K\setminus\{e_0\}} |R_e|\right)\left(\sum_{\eta} |R(e_0,\eta)|\right) = \left(\prod_{e\in K\setminus\{e_0\}} |R_e|\right) \cdot |R(e_0)| = \prod_{e\in K} |R_e| \\
&\leq \text{max-}(k,Q)\text{-product of } C.
\end{aligned}$$

Therefore, the left-hand side of (35) is bounded by $\left(1/L^k\right) \cdot \text{max-}(k,Q)$-product of C, which is at most p by definition of L (recall that L is the Q-induced load of C, defined in (26)).

Example 5.1. To illustrate the analysis, let us revisit Example 4.2 (refer to Figure 1 for the edge tree T, where the circled edges form the CEC). We have chosen the anchor leaf $f_{\text{anc}} = \texttt{ABC}$ and

[8] The $R(e_0, \eta)$ of all η are mutually disjoint and their union equals $R(e_0)$.

the anchor attribute $A_{\text{anc}} = \mathtt{C}$, resulting in the signature path $\text{sigpath}(\mathtt{ABC}, T) = \{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}$. Consider $K = \{\mathtt{ABC}, \mathtt{EHJ}, \mathtt{HI}\}$, which has an edge $\mathtt{ABC}$ in $\text{sigpath}(\mathtt{ABC}, T)$ (i.e., $e_0 = \mathtt{ABC}$). In this case,

$$\begin{aligned}\frac{1}{L^3} \cdot \sum_{\eta} |R(\mathtt{ABC}, \eta)| \cdot |R(\mathtt{EHJ}, \eta)| \cdot |R(\mathtt{HI}, \eta)| &= \frac{1}{L^3} \cdot |R_{\mathtt{EHJ}}| \cdot |R_{\mathtt{HI}}| \cdot \sum_{\eta} |R(\mathtt{ABC}, \eta)| \\ &\leq \frac{|R_{\mathtt{EHJ}}| \cdot |R_{\mathtt{HI}}| \cdot |R_{\mathtt{ABC}}|}{L^3} \\ &\leq \frac{\text{max-}(3, Q)\text{-product of } C}{L^3} = O(p).\end{aligned}$$

Next, we consider $K \cap \text{sigpath}(f_{\text{anc}}, T) = \emptyset$. In this case, we must have $k = |K| \leq |\mathcal{F}| - 1$, because the edges in K need to come from distinct clusters of C, and C has $|\mathcal{F}|$ clusters (one of them is $\text{sigpath}(f_{\text{anc}}, T)$, which now must be excluded). We can derive:

$$\begin{aligned}\frac{1}{L^k} \sum_{\eta} \prod_{e \in K} |R(e, \eta)| &\leq \frac{1}{L^k} \sum_{\eta} \prod_{e \in K} |R_e| \quad \text{(applying the trivial fact } |R(e, \eta)| \leq |R_e|) \\ &= O\left(\frac{1}{L^k} \prod_{e \in K} |R_e| \cdot \sum_{\eta} 1\right) = O\left(\frac{1}{L^k} \prod_{e \in K} |R_e| \cdot \max_{e' \in \text{sigpath}(f_{\text{anc}}, T)} \frac{|R_{e'}|}{L}\right) \\ &\quad \text{(because } \sum_{\eta} 1 = O\left(\max_{e' \in \text{sigpath}(f_{\text{anc}}, T)} \frac{|R_{e'}|}{L}\right), \text{ as shown in (27))} \\ &= O\left(\frac{\text{max-}(k+1, Q)\text{-product of } C}{L^{k+1}}\right) \\ &\quad \text{notice that } \prod_{e \in K} |R_e| \cdot \max_{e' \in \text{sigpath}(f_{\text{anc}}, T)} |R_{e'}| \\ &\quad \text{is the } Q\text{-product of a } (k+1)\text{-group}\end{aligned}$$

which is at most p. This completes the proof of $\sum_{\eta} p_{\eta} = O(p)$.

Example 5.2. We again use the context of Example 4.2 to illustrate the analysis (see Figure 1 for the edge tree T, where the circled edges constitute the CEC). Recall that we have chosen the anchor leaf $f_{\text{anc}} = \mathtt{ABC}$ and the anchor attribute $A_{\text{anc}} = \mathtt{C}$, resulting in the signature path $\text{sigpath}(\mathtt{ABC}, T) = \{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}$. Consider $K = \{\mathtt{BO}, \mathtt{EHJ}, \mathtt{HI}\}$. In this case,

$$\begin{aligned}&\frac{1}{L^3} \cdot \sum_{\eta} |R(\mathtt{BO}, \eta)| \cdot |R(\mathtt{EHJ}, \eta)| \cdot |R(\mathtt{HI}, \eta)| \\ &\qquad \leq \frac{1}{L^3} \cdot |R_{\mathtt{BO}}| \cdot |R_{\mathtt{EHJ}}| \cdot |R_{\mathtt{HI}}| \cdot \sum_{\eta} 1 \\ &\qquad = \frac{1}{L^3} \cdot |R_{\mathtt{BO}}| \cdot |R_{\mathtt{EHJ}}| \cdot |R_{\mathtt{HI}}| \cdot O\left(\frac{\max\{|R_{\mathtt{ABC}}|, |R_{\mathtt{BCE}}|, |R_{\mathtt{CEJ}}|\}}{L}\right) \\ &\qquad \leq \frac{\text{max-}(4, Q)\text{-product of } C}{L^4} = O(p).\end{aligned}$$

5.2 Heavy Q_η

This subsection will prove that the algorithm in Section 4.3 has load $O(L)$. Steps 2 and 5 demand no communication. The following discussion focuses on the other steps.

Let us start with a technical lemma that will be useful later. Recall that G is the schema graph of query Q (and Q_η), T is an edge tree of G, and C is the T-clustering of G. For any $k \in [|C|]$, $P_k(Q_\eta, C)$ is the max-(k, Q)-product of C, defined in (25). Our technical lemma is:

Lemma 17. *For any $k \in [|C|]$, it holds that $(P_k(Q_\eta, C)/p_\eta)^{1/k} = O(L)$.*

Proof. Define

$$k' = \arg\max_{k \in [|C|]} \left(\frac{P_k(Q_\eta, C)}{p_\eta}\right)^{1/k}$$

To prove the lemma, it suffices to show that $(P_{k'}(Q_\eta, C)/p_\eta)^{1/k'} = O(L)$. Define

$$k'' = \arg\max_{k \in [|C|]} \frac{P_k(Q_\eta, C)}{L^k}.$$

From (28), we know that p_η is either $\Theta(\frac{P_{k''}(Q_\eta, C)}{L^{k''}})$ or $\Theta(1)$. In the former case, because $\frac{P_{k''}(Q_\eta, C)}{L^{k''}} \geq \frac{P_{k'}(Q_\eta, C)}{L^{k'}}$, we have $p_\eta = \Theta(\frac{P_{k''}(Q_\eta, C)}{L^{k''}}) = \Omega(\frac{P_{k'}(Q_\eta, C)}{L^{k'}})$ and hence

$$\left(\frac{P_{k'}(Q_\eta, C)}{p_\eta}\right)^{1/k'} = O\left(\left(\frac{P_{k'}(Q_\eta, C)}{P_{k'}(Q_\eta, C)/L^{k'}}\right)^{1/k'}\right) = O(L).$$

In the latter case (i.e., $p_\eta = \Theta(1)$), (28) implies $P_{k'}(Q_\eta, C) = O(L^{k'})$ and hence

$$\left(\frac{P_{k'}(Q_\eta, C)}{p_\eta}\right)^{1/k'} = \left(\frac{O(L^{k'})}{\Theta(1)}\right)^{1/k'} = O(L).$$

We thus complete the proof. □

We now continue the analysis of the algorithm in Section 4.3. The loads of Steps 1 and 3 are both bounded[9] by

$$O\left(\frac{1}{p_\eta}\sum_{e \in E}|R(e, \eta)|\right) = O\left(\frac{1}{p_\eta}\max_{e \in E}|R(e, \eta)|\right) = O\left(\frac{P_1(Q_\eta, C)}{p_\eta}\right) = O(L).$$

where the second equality applied the definition of $P_1(Q_\eta, C)$ in (25) — note that the Q_η-product of a 1-group K is merely the maximum size of the relations $R(e, \eta)$ of all $e \in K$ — and the third inequality applied Lemma 17.

For analyzing Step 4, let us first recall that, after removing the anchor attribute A_{anc}, we convert G into residual hypergraph G' and T into an edge tree T' of G'. Then, Step 3 cleanses G' into a reduced hypergraph G^* and, accordingly, converts T' into an edge tree T^* of G^*. Let C^* be the T^*-clustering of G^*. The discussion in Section 3.3 tells us $|C^*| \leq |C|$, where as mentioned before C is the T-clustering of G. By the definition in (26), the Q^*_η-induced load of C^* is

$$L^*_\eta = \max_{k=1}^{|C^*|}\left(\frac{P_k(Q^*_\eta, C^*)}{p_\eta}\right)^{1/k} \tag{37}$$

where $P_k(Q^*_\eta, C^*)$ is the max-(k, Q^*_η)-product of C^* (defined in (25)). By our inductive assumption (that Lemma 16 holds on Q^*_η), Step 4 incurs load $O(L^*_\eta)$. Next, we will argue that $O(L^*_\eta) = O(L)$.

Lemma 18. *For each $k \in [|C^*|]$, it holds that $P_k(Q^*_\eta, C^*) \leq P_k(Q_\eta, C)$.*

[9] Step 3 requires $O(1)$ semi-joins, each of which can be performed by sorting. For sorting in the MPC model, see Section 2.2.1 of [14]. The stated bound for Steps 1 and 3 requires the assumption $p \leq N^{1-\epsilon}$ introduced in Section 1.1.

Proof. Recall that every edge e^* of G^* corresponds to an edge $map^{-1}(e^*)$ in G, where the function $map^{-1}(.)$ was defined in Section 3.3.1. We must have

$$|R^*(e^*, \eta)| \leq |R(map^{-1}(e^*), \eta)|.$$

To see why, note that this is true when $|R^*(e^*, \eta)|$ is created in Step 2, whereas $R^*(e^*, \eta)$ can only shrink in Steps 3-5.

To prove the lemma, consider any k-group K^* of C^*. By Lemma 12, $K = \{map^{-1}(e^*) \mid e^* \in K^*\}$ must be a k-group of C. Since $|R^*(e^*, \eta)| \leq |R(map^{-1}(e^*), \eta)|$ for any $e^* \in K^*$, we have $\prod_{e^* \in K^*} |R^*(e^*, \eta)| \leq \prod_{e \in K} |R(e, \eta)| \leq P_k(Q_\eta, C)$. Therefore:

$$P_k(Q^*_\eta, C^*) = \max_{K^*} \prod_{e^* \in K^*} |R^*(e^*, \eta)| \leq P_k(Q_\eta, C)$$

as needed. □

Applying the above lemma to (37), we now have $L^*_\eta \leq \max_{k=1}^{|C^*|} (\frac{P_k(Q_\eta, C)}{p_\eta})^{1/k}$, which is $O(L)$ by Lemma 17.

Example 5.3. To illustrate the core of the above analysis, we will re-examine the scenario in Example 4.4. The schema graph G of the query Q has an edge tree T shown in Figure 1, where the circled nodes constitute the CEC. The T-clustering of G is

$$C = \{\{\mathtt{BO}, \mathtt{BCE}, \mathtt{CEJ}\}, \{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}, \{\mathtt{BD}, \mathtt{BCE}, \mathtt{CEJ}\}, \{\mathtt{EFG}, \mathtt{CEF}, \mathtt{CEJ}\}, \{\mathtt{HI}\}, \{\mathtt{EHJ}\}, \{\mathtt{LM}, \mathtt{KL}\}, \{\mathtt{HK}\}, \{\mathtt{HN}\}\}.$$

We have chosen the anchor leaf $f_{\text{anc}} = \mathtt{ABC}$ and the anchor attribute $A_{\text{anc}} = \mathtt{C}$, resulting in the signature path $\text{sigpath}(\mathtt{ABC}, T) = \{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}$. In Example 4.4, the configuration η is a heavy value h of C. The sub-query we need to process — Q^*_η, or specifically, Q^*_h — has a schema graph G^*, for which Figure 5(b) shows its edge tree T^*, where the circled nodes constitute the CEC. The T^*-clustering of G^* is

$$C^* = \{\{\mathtt{BO}, \mathtt{BE}\}, \{\mathtt{AB}, \mathtt{BE}\}, \{\mathtt{BD}, \mathtt{BE}\}, \{\mathtt{EFG}\}, \{\mathtt{HI}\}, \{\mathtt{EHJ}\}, \{\mathtt{LM}, \mathtt{KL}\}, \{\mathtt{HK}\}, \{\mathtt{HN}\}\}.$$

From our inductive assumption, our algorithm processes Q^*_h with load $O(L^*_\eta)$, where L^*_η — or specifically, L^*_h — is given in (38). Suppose, w.l.o.g., that (i) expression (38) is maximized at $k = 3$ and (ii) $P_3(Q^*_h, C^*)$ is the Q^*_h-product of the 3-group $K^* = \{\mathtt{AB}, \mathtt{BO}, \mathtt{EHJ}\}$ of C^*, meaning $P_k(Q^*_h, C^*) = |R^*(\mathtt{AB}, h)| \cdot |R^*(\mathtt{BO}, h)| \cdot |R^*(\mathtt{EHJ}, h)|$. The reader should take a moment to recall the meanings of $|R^*(\mathtt{AB}, h)|$, $|R^*(\mathtt{BO}, h)|$, and $|R^*(\mathtt{EHJ}, h)$ from Example 4.4.

Recall that every edge e^* in G^* corresponds to an edge e in G, as is captured by $e = map^{-1}(e^*)$. Here, edges AB, BD, and EHJ of G^* correspond to edges ABC, BD, and EHJ of G, respectively. Crucially, Lemma 12 guarantees that the set $K = \{\mathtt{ABC}, \mathtt{BO}, \mathtt{EHJ}\}$ must be a 3-group of C. It follows that

$$\begin{aligned} P_3(Q^*_h, C^*) &= |R^*(\mathtt{AB}, h)| \cdot |R^*(\mathtt{BO}, h)| \cdot |R^*(\mathtt{EHJ}, h)| \\ &\leq |R(\mathtt{ABC}, h)| \cdot |R(\mathtt{BO}, h)| \cdot |R(\mathtt{EHJ}, h)| \\ &\quad \text{(see Example 4.3 for the meanings of } R(\mathtt{ABC}, h), R(\mathtt{BO}, h), \text{ and } R(\mathtt{EHJ}, h)) \\ &\leq P_3(Q_h, C). \end{aligned}$$

As $L^*_h = O(P_3(Q^*_h, C^*)/p_\eta)^{1/3})$, we can now apply Lemma 17 to conclude that $L^*_h = O(L)$.

5.3 Light Q_η

This subsection will concentrate on the algorithm of Section 4.4.

Load. Step 1 incurs load $O(L)$ (same analysis as in Section 4.3). Step 2 also requires a load of $O(L)$ because every broadcast relation has a size of at most L. Step 4 needs no communication.

For analyzing Step 3, let us recall that, at this moment, we have removed all the edges in the signature path $\text{sigpath}(f_{\text{anc}}, T)$ from the schema graph G of Q. This yields set Z, defined in (21). For each edge $z \in Z$, we have obtained a sub-query $Q^*_{\eta,z}$, whose schema graph G^*_z has an edge tree T^*_z, which defines a T^*_z-clustering C^*_z of G^*_z. In addition, we have also obtained another sub-query $\overline{Q^*_\eta}$, whose schema graph $\overline{G^*}$ has an edge tree $\overline{T^*}$, which defines a $\overline{T^*}$-clustering $\overline{C^*}$ of $\overline{G^*}$.

Let us first consider $\overline{Q^*_\eta}$. The $\overline{Q^*_\eta}$-induced load of $\overline{C^*}$ is

$$\overline{L^*_\eta} = \max_{k=1}^{|\overline{C^*}|} \left(\frac{P_k(\overline{Q^*_\eta}, \overline{C^*})}{\overline{p_\eta}} \right)^{1/k}$$

where $P_k(\overline{Q^*_\eta}, \overline{C^*})$ is the max-$(k, \overline{Q^*_\eta})$-product of $\overline{C^*}$ (see the definition in (25)). Regarding the join $Q^*_{\eta,z}$ for each $z \in Z$, the $Q^*_{\eta,z}$-induced load of C^*_z is

$$L^*_{\eta,z} = \max_{k=1}^{|C^*_z|} \left(\frac{P_k(Q^*_{\eta,z}, C^*_z)}{p_{\eta,z}} \right)^{1/k} \tag{38}$$

where $P_k(Q^*_{\eta,z}, C^*_z)$ is the max-$(k, Q^*_{\eta,z})$-product of C^*_z. By our inductive assumption—namely, Lemma 16 holds on $\overline{Q^*_\eta}$ and the $Q^*_{\eta,z}$ of each $z \in Z$ —we know:

- evaluating $\overline{Q^*_\eta}$ with $\overline{p_\eta}$ machines requires load $O(\overline{L^*_\eta})$, which is $O(L)$ given the $\overline{p_\eta}$ in (32), following an argument similar to that used to prove (37) is $O(L)$;
- evaluating $Q^*_{\eta,z}$ of any $z \in Z$ with $p_{\eta,z}$ machines requires load $O(L^*_{\eta,z})$, which is $O(L)$ given the $p_{\eta,z}$ in (31), again following an argument similar to that used to prove (37) is $O(L)$.

Thus, the Cartesian product at Step 3 can be computed with load $O(L)$, as explained in Section 4.4.

Number of Machines in Step 3. To establish Lemma 16, it remains to prove that $\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} \leq p_\eta$ always holds in Step 3. It suffices to show $\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} = O(p_\eta)$ because we can then adjust the constants to ensure $\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} \leq p_\eta$.

Fix an arbitrary $z \in Z$. The root of T^*_z — denoted as e_{root} — must belong to $\text{sigpath}(f_{\text{anc}}, T)$. Recall that a k-group K of C^*_z takes edges from distinct clusters in C^*_z. Call K a

- *non-root k-group* of C^*_z if $e_{root} \notin K$, or
- a *root k-group* of C^*_z, otherwise.

A non-root k-group K must have a size $|K| \leq |C^*_z| - 1$ because e_{root} makes a cluster in C^*_z.

For each $k \in [|C^*_z|]$, define

$$P^{non}_k(Q^*_{\eta,z}, C^*_z) = \begin{cases} 1 & \text{if } k = 0 \\ \text{max-}(k, Q^*_{\eta,z})\text{-product of all the } \textit{non-root } k\text{-groups of } C^*_z & \text{if } 1 \leq k \leq |C^*_z| - 1 \\ -\infty & \text{if } k = |C^*_z| \end{cases}$$

We observe:

Lemma 19. *For any $z \in Z$ and any $k \in [|C^*_z|]$, it holds that*

$$P_k(Q^*_{\eta,z}, C^*_z) \quad \leq \quad \max \left\{ P^{non}_k(Q^*_{\eta,z}, C^*_z), L \cdot P^{non}_{k-1}(Q^*_{\eta,z}, C^*_z) \right\} \tag{39}$$

where $P_k(Q^*_{\eta,z}, C^*_z)$ is the max-$(k, Q^*_{\eta,z})$-product of C^*_z (see definition in (25)).

Proof. Define K as the k-group of C_z^* whose $Q_{\eta,z}^*$-product (see definition in (24)) is the greatest among all the k-groups of C_z^*. In other words, the $Q_{\eta,z}^*$-product of K is $P_k(Q_{\eta,z}^*, C_z^*)$. If K is a non-root k-group of C_z^*, (39) obviously holds. Consider, instead, that K is a root k-group of C_z^*. Since $e_{root} \in \text{sigpath}(f_{\text{anc}}, T)$, we know $|R(e_{root}, \eta)| \leq L$ and hence $\prod_{e\in K} |R(e,\eta)| \leq L \cdot \prod_{e\in K\setminus\{e_{root}\}} |R(e,\eta)|$. As $K \setminus \{e_{root}\}$ is a non-root $(k-1)$-group, $P_k(Q_{\eta,z}^*, C_z^*) \leq L \cdot P_{k-1}^{non}(Q_{\eta,z}^*, C_z^*)$ holds. □

Equipped with (39), we can now derive from (31):

$$\begin{aligned} p_{\eta,z} &= O\left(1 + \max_{k=1}^{|C_z^*|} \frac{\max\{P_k^{non}(Q_{\eta,z}^*, C_z^*), L \cdot P_{k-1}^{non}(Q_{\eta,z}^*, C_z^*)\}}{L^k}\right) \\ &= O\left(1 + \max_{k=1}^{|C_z^*|-1} \frac{P_k^{non}(Q_{\eta,z}^*, C_z^*)}{L^k}\right) \end{aligned} \tag{40}$$

where the second equality used the fact that $P_k^{non}(Q_{\eta,z}^*, C_z^*) = -\infty$ for $k = |C_z^*|$.

We are now ready to prove $\overline{p_\eta} \cdot \prod_{z\in Z} p_{\eta,z} = O(p_\eta)$. For each $z \in Z$, the value $p_{\eta,z}$ in (40) is either $\Theta(\max_{k=1}^{|C_z^*|-1} \frac{P_k^{non}(Q_{\eta,z}^*, C_z^*)}{L^k})$ or $\Theta(1)$. Depending on which case it is, we define integer k_z and a set K_z of edges differently, as explained next:

— If $p_{\eta,z} = \Theta(\frac{P_k^{non}(Q_{\eta,z}^*, C_z^*)}{L^k})$ for some $k \in [|C_z^*| - 1]$, then

$$\begin{aligned} k_z &= k \\ K_z &= \text{the non-root } k\text{-group of } C_z^* \text{ whose } Q_{\eta,z}^*\text{-product equals } P_k^{non}(Q_{\eta,z}^*, C_z^*) \end{aligned}$$

— Otherwise (namely, $p_{\eta,z} = \Theta(1)$), then

$$\begin{aligned} k_z &= 0 \\ K_z &= \emptyset \end{aligned}$$

In this case, define the $Q_{\eta,z}^*$-product of K_z to be 1.

The above definitions of k_z and K_z guarantee $p_{\eta,z} = \Theta(\frac{Q_{\eta,z}^*\text{-product of } K_z}{L^{k_z}})$ in all cases.

In the same fashion, concerning the value $\overline{p_\eta}$ in (32), we define integer $\overline{k}$ and a set $\overline{K}$ of edges as follows:

— If $\overline{p_\eta} = \Theta(\frac{P_k(\overline{Q_\eta^*}, \overline{C^*})}{L^k})$ for some $k \in [|\overline{C^*}|]$, then

$$\begin{aligned} \overline{k} &= k \\ \overline{K} &= \text{the } k\text{-group of } \overline{C^*} \text{ whose } \overline{Q_\eta^*}\text{-product equals } P_k(\overline{Q_\eta^*}, \overline{C^*}). \end{aligned}$$

— Otherwise (namely, $\overline{p_\eta} = \Theta(1)$), then

$$\begin{aligned} \overline{k} &= 0 \\ \overline{K} &= \emptyset \end{aligned}$$

In this case, define the $\overline{Q_\eta^*}$-product of $\overline{K}$ to be 1.

The above definitions of $\overline{k}$ and $\overline{K}$ guarantee $\overline{p}_\eta = \Theta(\frac{\overline{Q_\eta^*}\text{-product of } \overline{K}}{L^{\overline{k}}})$ in all cases.

Now, we can define

$$K_{super} = \overline{K} \cup \left(\bigcup_{z\in Z} K_z\right).$$

If $K_{super} = \emptyset$, then $p_{\eta,z} = \Theta(1)$ for all $z \in Z$ and $\overline{p_\eta} = \Theta(1)$, which leads to

$$\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} = O(1) = O(p_\eta).$$

If $K_{super} \neq \emptyset$, then K_{super} is a super-$|K_{super}|$-group (see Section 3.4 for the definition of "super-k-group"). By Lemma 14, K_{super} is a $|K_{super}|$-group of T. We thus have:

$$\begin{aligned}
\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} &= \Theta\left(\frac{\overline{Q^*_\eta}\text{-product of } \overline{K}}{L^{\overline{k}}}\right) \cdot \prod_{z \in Z} \Theta\left(\frac{Q^*_{\eta,z}\text{-product of } K_z}{L^{k_z}}\right) \\
&= \Theta\left(\frac{\prod_{e \in K_{super}} |R_e|}{L^{|K_{super}|}}\right) \\
&= O\left(\frac{\max\text{-}(|K_{super}|, Q_\eta)\text{-product of } C}{L^{|K_{super}|}}\right) \\
&= O(p_\eta)
\end{aligned}$$

where the last equality used the definition of p_η in (28).

We have shown that $\overline{p_\eta} \cdot \prod_{z \in Z} p_{\eta,z} = O(p_\eta)$ holds in all cases. This completes the whole proof of Lemma 16.

Example 5.4. To illustrate the core of the above analysis, we will re-visit the scenario in Example 4.5. The schema graph G of Q has an edge tree T shown in Figure 1, where the circled nodes constitute the CEC. We repeat the T-clustering of G here for the reader's convenience:

$$C = \{\{\mathtt{BO}, \mathtt{BCE}, \mathtt{CEJ}\}, \{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}, \{\mathtt{BD}, \mathtt{BCE}, \mathtt{CEJ}\}, \{\mathtt{EFG}, \mathtt{CEF}, \mathtt{CEJ}\}, \{\mathtt{HI}\}, \{\mathtt{EHJ}\}, \{\mathtt{LM}, \mathtt{KL}\}, \{\mathtt{HK}\}, \{\mathtt{HN}\}\}.$$

The anchor leaf is $f_{\text{anc}} = \mathtt{ABC}$, resulting in the signature path $\text{sigpath}(\mathtt{ABC}, T) = \{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}$. The configuration η is a light interval I of $\mathtt{C}$. As explained in Example 4.4, to show that Q_η — or specifically, Q_I — can be processed with load $O(L)$, we need to prove:

- $\mathcal{J}oin(Q^*_{I,\mathtt{CEF}})$ can be computed with load $O(L)$ using $p_{I,\mathtt{CEF}}$ machines,
- $\mathcal{J}oin(Q^*_{I,\mathtt{BO}})$ can be computed with load $O(L)$ using $p_{I,\mathtt{BO}}$ machines,
- $\mathcal{J}oin(Q^*_{I,\mathtt{BD}})$ can be computed with load $O(L)$ using $p_{I,\mathtt{BD}}$ machines,
- $\mathcal{J}oin(\overline{Q^*_I})$ can be computed with load $O(L)$ using $\overline{p_I}$ machines, and
- $p_{I,\mathtt{CEF}} \cdot p_{I,\mathtt{BO}} \cdot p_{I,\mathtt{BD}} \cdot \overline{p_I} \leq p_\eta$

where the values of $p_{I,\mathtt{CEF}}$, $p_{I,\mathtt{BO}}$, and $p_{I,\mathtt{BD}}$ are computed using (31), $\overline{p_I}$ is computed using (32), and p_η — or specifically, p_I — is computed using (28). The first four statements can be established following an argument similar to that used to prove (37) is $O(L)$. Next, we will illustrate our proof for $p_{I,\mathtt{CEF}} \cdot p_{I,\mathtt{BO}} \cdot p_{I,\mathtt{BD}} \cdot \overline{p_I} \leq p_\eta$.

Recall that

- For $Q^*_{I,\mathtt{CEF}}$, its schema graph of $G^*_{\mathtt{CEF}}$ has an edge tree $T^*_{\mathtt{CEF}}$ shown in Figure 6(b), and the $T^*_{\mathtt{CEF}}$-clustering of $G^*_{\mathtt{CEF}}$ is $C^*_{\mathtt{CEF}} = \{\{\mathtt{EFG}, \mathtt{CEF}\}, \{\mathtt{CEJ}\}\}$.
- For $Q^*_{I,\mathtt{BO}}$, its schema graph of $G^*_{\mathtt{BO}}$ has an edge tree $T^*_{\mathtt{BO}}$ shown in Figure 6(c), and the $T^*_{\mathtt{BO}}$-clustering of $G^*_{\mathtt{BO}}$ is $C^*_{\mathtt{BO}} = \{\{\mathtt{BO}\}, \{\mathtt{BCE}\}\}$.
- For $Q^*_{I,\mathtt{BD}}$, its schema graph of $G^*_{\mathtt{BD}}$ has an edge tree $T^*_{\mathtt{BD}}$ shown in Figure 6(d), and the $T^*_{\mathtt{BD}}$-clustering of $G^*_{\mathtt{BD}}$ is $C^*_{\mathtt{BD}} = \{\{\mathtt{BD}\}, \{\mathtt{BCE}\}\}$.
- For $\overline{Q^*_I}$, its schema graph of $\overline{G^*}$ has an edge tree $\overline{T^*}$ shown in Figure 6(e), and the $\overline{T^*}$-clustering of $\overline{G^*}$ is $\overline{C^*} = \{\{\mathtt{HI}\}, \{\mathtt{EHJ}\}, \{\mathtt{HK}\}, \{\mathtt{HN}\}, \{\mathtt{LM}, \mathtt{KL}\}\}$.

As shown in (31), $p_{I,\mathsf{CEF}}$ is determined by the maximum of three terms, which are 1, $P_1(Q^*_{I,\mathsf{CEF}}, C^*_{\mathsf{CEF}})/L$, and $P_2(Q^*_{I,\mathsf{CEF}}, C^*_{\mathsf{CEF}})/L^2$. Similar choices exist for $p_{I,\mathsf{BO}}$, $p_{I,\mathsf{BD}}$, and $\overline{p_I}$. In our illustration, we will focus on the following choices:

- $p_{I,\mathsf{CEF}} = \Theta(P_2(Q^*_{I,\mathsf{CEF}}, C^*_{\mathsf{CEF}})/L^2)$. Furthermore, suppose that $P_2(Q^*_{I,\mathsf{CEF}}, C^*_{\mathsf{CEF}})$ equals the $Q^*_{I,\mathsf{CEF}}$-product of the 2-group $K_1 = \{\mathsf{CEF}, \mathsf{CEJ}\}$, i.e., $P_2(Q^*_{I,\mathsf{CEF}}, C^*_{\mathsf{CEF}}) = |R(\mathsf{CEF}, I)| \cdot |R(\mathsf{CEJ}, I)|$.
- $p_{I,\mathsf{BO}} = \Theta(P_2(Q^*_{I,\mathsf{BO}}, C^*_{\mathsf{CEF}})/L^2)$. In this case, $P_2(Q^*_{I,\mathsf{BO}})$ must be the $Q^*_{I,\mathsf{BO}}$-product of the 2-group $K_2 = \{\mathsf{BO}, \mathsf{BCE}\}$, i.e., $P_2(Q^*_{I,\mathsf{BO}}, C^*_{\mathsf{BO}}) = |R(\mathsf{BO}, I)| \cdot |R(\mathsf{BCE}, I)|$
- $p_{I,\mathsf{BD}} = \Theta(P_1(Q^*_{I,\mathsf{BD}}, C^*_{\mathsf{CEF}})/L)$. Furthermore, suppose that $P_1(Q^*_{I,\mathsf{BD}})$ equals $Q^*_{I,\mathsf{BD}}$-product of the 1-group $K_3 = \{\mathsf{BD}\}$, i.e., $P_1(Q^*_{I,\mathsf{BD}}) = |R(\mathsf{BD}, I)|$.
- $\overline{p_I} = \Theta(P_3(\overline{Q^*}, \overline{C^*})/L^3)$. Furthermore, suppose that $P_3(\overline{Q^*}, \overline{C^*})$ equals the $\overline{Q^*}$-product of the 3-group $K_4 = \{\mathsf{HI}, \mathsf{KL}, \mathsf{HK}\}$, i.e., $P_3(\overline{Q^*}, \overline{C^*}) = |R(\mathsf{HI}, I)| \cdot |R(\mathsf{KL}, I)| \cdot |R(\mathsf{HK}, I)|$.

The ideas demonstrated below extend to the other choice combinations in a straightforward manner.

The 2-groups K_1 and K_2 are root 2-groups, while K_3 is a non-root 1-group (the concept of root/non-root k-group does not apply to K_4). In general, every root k-group must contain an edge $e \in \text{sigpath}(\mathsf{ABC}, T)$ such that $R(e, I)$ has a size bounded by L. Indeed, for K_1, we have $|R(\mathsf{CEJ}, I)| \leq L$, while for K_2, we have $|R(\mathsf{BCE}, I)| \leq L$.

Putting together all the above facts, we have:

$$\begin{aligned} & p_{I,\mathsf{CEF}} \cdot p_{I,\mathsf{BO}} \cdot p_{I,\mathsf{BD}} \cdot \overline{p_I} \\ &= O\left(\frac{|R(\mathsf{CEF}, I)| \cdot |R(\mathsf{CEJ}, I)|}{L^2} \cdot \frac{|R(\mathsf{BO}, I)| \cdot |R(\mathsf{BCE}, I)|}{L^2} \cdot \frac{|R(\mathsf{BD}, I)|}{L} \cdot \frac{|R(\mathsf{HI}, I)| \cdot |R(\mathsf{KL}, I)| \cdot |R(\mathsf{HK}, I)|}{L^3}\right) \\ &= O\left(\frac{|R(\mathsf{CEF}, I)|}{L} \cdot \frac{|R(\mathsf{BO}, I)|}{L} \cdot \frac{|R(\mathsf{BD}, I)|}{L} \cdot \frac{|R(\mathsf{HI}, I)| \cdot |R(\mathsf{KL}, I)| \cdot |R(\mathsf{HK}, I)|}{L^3}\right) \end{aligned} \tag{41}$$

The set $K = \{\mathsf{CEF}, \mathsf{BO}, \mathsf{BD}, \mathsf{HI}, \mathsf{KL}, \mathsf{HK}\}$ is a super-6-group. By Lemma 14, K must be a 6-group of the T-clustering of G. Therefore:

$$(41) = O\left(\frac{Q_I\text{-product of } K}{L^6}\right) = O\left(\frac{P_k(Q_I, C)}{L^6}\right) = O(p_I)$$

where the last equality used the definition of p_I in (28). Adjusting the hidden constants in the big-O gives $p_{I,\mathsf{CEF}} \cdot p_{I,\mathsf{BO}} \cdot p_{I,\mathsf{BD}} \cdot \overline{p_I} \leq p_I$.

6 CONCLUDING REMARKS

In this article, we have disproved the existence of any tuple-based algorithm that can evaluate an arbitrary join query in the MPC model with load $\tilde{O}(N/p^{1/\rho^*})$, where N is the query's input size, ρ^* is the query's fractional edge covering number, and p is the number of machines. Specifically, we have established a new load lower bound of $\Omega(N/p^{1/\tau^*})$ for an instance of boat joins (see Figure 2 for the schema graph of such joins), where $\tau^* = 3$ is the fractional edge packing number for such joins, and their ρ^* value equals 2. We can actually make the gap between $\tilde{O}(N/p^{1/\rho^*})$ and $\Omega(N/p^{1/\tau^*})$ arbitrarily large, by adapting our argument to a class of "generalized boat joins" (defined as follows. Fix any constant integer $k \geq 3$. The schema graph G of a generalized boat join has $2k$ attributes $X_1, X_2, \ldots, X_k$ and $Y_1, Y_2, \ldots, Y_k$, and $2k + 2$ edges: $\{X_1, X_2, \ldots, X_k\}$, $\{Y_1, Y_2, \ldots, Y_k\}$, and $\{X_i, Y_i\}$ for every $i \in [k]$. G has a fractional edge covering number $\rho^* = 2$ and yet a fractional edge packing number $\tau^* = k$. It is not difficult to modify our argument to prove that, for every k, $\Omega(N/p^{1/\tau^*}) = \Omega(N/p^{1/k})$ is a load lower bound on at least one instance of generalized boat joins.

Boat joins, as well as their generalized counterparts, have cyclic schema graphs. We have shown that cyclicity is indeed what prevents us from guaranteeing a load of $\tilde{O}(N/p^{1/\rho^*})$. For that purpose, we have presented an algorithm that can evaluate any acyclic join with load $O(N/p^{1/\rho^*})$ — without any polylogarithmic factor—which matches the well-known lower bound of $\Omega(N/p^{1/\rho^*})$ and is therefore asymptotically optimal. Our algorithm is made possible by canonical edge cover, a new mathematical structure of acyclic hypergraphs that we discover in this article. Every acyclic hypergraph has a canonical edge cover, which constitutes an integral optimal fractional edge covering and has many interesting properties useful for algorithm design.

An intriguing open problem left behind by this article is whether (cyclic) join evaluation in MPC can be fully characterized by putting *together* the fractional edge covering number ρ^* and fractional edge packing number τ^* of a join. Our lower bound argument does not rule out an algorithm with load $\tilde{O}(N/p^{1/\chi^*})$, where $\chi^* = \max\{\rho^*, \tau^*\}$. In fact, all the generalized boat joins can be evaluated with load $\tilde{O}(N/p^{1/\tau^*})$ using algorithms from [19] and [27]. Unfortunately, the algorithms in [19] and [27] fail to achieve load $\tilde{O}(N/p^{1/\chi^*})$ for arbitrary joins.

APPENDICES

A CHERNOFF BOUNDS

Let $X_1, X_2, \ldots, X_t$ be $t \geq 1$ independent Bernoulli random variables such that $\mathbf{Pr}[X_i = 1]$ is the same for all $i \in [1, t]$ (hence, so is $\mathbf{Pr}[X_i = 0]$). Let $X = \sum_{i=1}^{t}$ and $\mu = \mathbf{E}[X]$. For any $0 < \gamma \leq 1$, it holds that

$$\mathbf{Pr}[|X - \mu| \geq \gamma \cdot \mu] \quad \leq \quad 2\exp\left(-\frac{\gamma^2 \cdot \mu}{3}\right). \tag{42}$$

For any $\gamma \geq 2$, it holds that

$$\mathbf{Pr}[X \geq \gamma \cdot \mu) \quad \leq \quad \exp\left(-\frac{\gamma \cdot \mu}{6}\right). \tag{43}$$

Inequalities (42) and (43) are commonly known as "Chernoff bounds" (see [28] for the proofs).

B PROOF OF LEMMA 10

We will first prove that $\mathcal{F}'$ is the CEC of G' induced by T' by discussing in Section B.1 the scenario where $map(f_{\text{anc}}) = f_{\text{anc}} \setminus \{A_{\text{anc}}\}$ is a subsumed edge in G' and in Section B.2 the scenario where $map(f_{\text{anc}})$ is not subsumed in G'. Then, Section B.3 will explain why $\mathcal{F}'$ cannot contain any subsumed edge of G'.

B.1 $\mathcal{F}'$ is the CEC of G': the Scenario Where $map(f_{\text{anc}})$ Is Subsumed

Let $\hat{e}$ be the parent of f_{anc} in T. As $map(f_{\text{anc}})$ is subsumed in G', $map(f_{\text{anc}})$ must be a subset of $map(\hat{e})$. This implies $A_{\text{anc}} \notin \hat{e}$ (otherwise, $f_{\text{anc}} \subseteq \hat{e}$ and G is not reduced). Because A_{anc} needs to appear in all the nodes of sigpath(f_{anc}, T), $A_{\text{anc}} \notin \hat{e}$ indicates that $\hat{e} \notin$ sigpath(f_{anc}, T) and thus sigpath(f_{anc}, T) has only a single node f_{anc}. It thus follows that $\hat{e} \in \mathcal{F}$ and A_{anc} is an exclusive attribute in f_{anc}.

To show that $\mathcal{F}' = \mathcal{F} \setminus \{f_{\text{anc}}\}$ is the CEC of G' induced by T', it suffices to prove that $\mathcal{F}'$ is the output of edge-cover(T') on an arbitrary reverse topological order of T' (Lemma 8 tells us that the output is not sensitive to the reverse topological order). For this purpose, consider σ_0 as any reverse topological order of T where $\hat{e}$ succeeds f_{anc} (i.e., $\hat{e}$ is the immediate successor of f_{anc} in σ_0). Let σ_1 be the sequence obtained by removing f_{anc} from σ_0; σ_1 must be a reverse topological order of T'. Let e_{before} be the node preceding f_{anc} in σ_0 (i.e., e_{before} is the immediate predecessor of f_{anc} in σ_0) and hence preceding $\hat{e}$ in σ_1; define $e_{\text{before}} = nil$ if f_{anc} is the first in σ_0.

Let us compare the execution of edge-cover(T) on σ_0 to that of edge-cover(T') on σ_1. The two executions are identical till the moment right after e_{before} has been processed (by Line 4 of edge-cover). By the fact that edge-cover(T) adds $\hat{e}$ to F_{tmp} (we have proved earlier $\hat{e} \in \mathcal{F}$), $\hat{e}$ has a disappearing attribute not covered by F_{tmp} when it is processed. Hence, when $\hat{e}$ is processed by edge-cover(T'), it must also have a disappearing attribute not covered by F_{tmp} and thus is added to F_{tmp}. The rest execution of edge-cover(T) is the same as that of edge-cover(T') because every non-exclusive attribute of f_{anc} is in $\hat{e}$. Therefore, the output of edge-cover(T') is the same as that of edge-cover(T), except that the former does not include f_{anc}.

B.2 $\mathcal{F}'$ is the CEC of G': The Scenario Where $map(f_{\text{anc}})$ is Not Subsumed in G'

Let $\sigma_0 = (e_1, e_2, \ldots, e_{|E|})$ be an arbitrary reverse topological order of T. Define $e'_i = map(e_i) = e_i \setminus \{A_{\text{anc}}\}$ for $i \in [|E|]$. The sequence $\sigma_1 = (e'_1, e'_2, \ldots, e'_{|E|})$ is a reverse topological order of T'. We will compare the execution of edge-cover(T) on σ_0 to that of edge-cover(T') on σ_1. Define $F_0(e_i)$ (respectively, $F_1(e'_i)$) as the content of F_{tmp} after edge-cover(T) (respectively, edge-cover(T')) has processed e_i (respectively, e'_i).

Claim 1. *For any leaf e of T,* edge-cover(T') *must add $e' = map(e)$ to F_{tmp}.*

To prove the claim, first note that, because e is a leaf of T and G is reduced, e must have an exclusive attribute X. If edge-cover(T') does not add e' to F_{tmp}, e' has no exclusive attributes in T'. This implies $X = A_{\text{anc}}$, which further implies $f_{\text{anc}} = e$ (otherwise, A_{anc} appears in two distinct nodes and thus cannot be exclusive). However, in that case, e' must contain an exclusive attribute in T' (because $e' = map(f_{\text{anc}})$ is not subsumed in G'), thus giving a contradiction.

Claim 2. *For each i, $e_i \in F_0(e_i)$ if and only if $e'_i \in F_1(e'_i)$.*

We prove the claim by induction on i. Because e_1 is a leaf of T, Lemma 8 and Claim 1 guarantee $e_1 \in F_0(e_1)$ and $e'_1 \in F_1(e'_1)$, respectively. Thus, Claim 2 holds for $i = 1$.

Next, we prove the correctness on $i > 1$, assuming that it holds on e_{i-1} and e'_{i-1}. The inductive assumption implies that $F_0(e_{i-1})$ covers an attribute $X \neq A_{\text{anc}}$ if and only if $F_1(e'_{i-1})$ covers X. If $e_i \notin F_0(e_i)$, every disappearing attribute of e_i must be covered by $F_0(e_{i-1})$. Hence, $F_1(e'_{i-1})$ must cover all the disappearing attributes of e'_i and thus $e'_i \notin F_1(e'_i)$.

The rest of the proof assumes $e_i \in F_0(e_i)$, i.e., e_i has a disappearing attribute X not covered by $F_0(e_{i-1})$. If $X \neq A_{\text{anc}}$, X is a disappearing attribute in e'_i not covered by $F_1(e'_{i-1})$ and thus $e'_i \in F_1(e'_i)$. It remains to discuss the scenario $X = A_{\text{anc}}$. As A_{anc} is disappearing at e_i, A_{anc} cannot exist in any proper ancestor of e_i. Thus, f_{anc} must be a descendant of e_i. We can assert that $f_{\text{anc}} = e_i$; otherwise, the leaf f_{anc} is processed before e_i and must exist in $F_0(e_{i-1})$ (Lemma 8), contradicting the fact that A_{anc} is not covered by $F_0(e_{i-1})$. Then, $e'_i \in F_1(e'_i)$ follows from Claim 1.

We can now conclude that $\mathcal{F}'$ is always the CEC of G' induced by T'.

B.3 $\mathcal{F}'$ Cannot Contain Subsumed Edges

Consider any subsumed edge e' in E'. Define $e = map^{-1}(e')$; we know that e must contain A_{anc} (otherwise, e is subsumed in E and G is not reduced). Hence, $e = e' \cup \{A_{\text{anc}}\}$. If $e = f_{\text{anc}}$, then $map(f_{\text{anc}}) = map(e) = e'$ is subsumed in G', in which case $e' \notin \mathcal{F}'$ holds due to the explicit exclusion of f_{anc} from $\mathcal{F}'$ as shown in (20).

Next, we consider $e \neq f_{\text{anc}}$. To prove $e \notin \mathcal{F}'$, by the way $\mathcal{F}'$ is computed in (20), it suffices to show $e \notin \mathcal{F}$, where $\mathcal{F}$ is the CEC of G induced by T. Assume, on the contrary, that $e \in \mathcal{F}$. Let $\hat{f}$ be the lowest proper ancestor of f_{anc} in $\mathcal{F}$ (here, "ancestor" is defined with respect to T).

The definition of A_{anc} assures us $A_{\text{anc}} \notin \hat{f}$. Because $A_{\text{anc}} \in f_{\text{anc}}$ and $A_{\text{anc}} \in e$, e must be a proper descendant of $\hat{f}$ in T (connectedness of acyclicity). By definition of f_{anc}, $\hat{f}$ cannot have any non-leaf proper descendant in $\mathcal{F}$. Hence, e must be a leaf of T.

Because A_{anc} appears in two distinct leaves of T (i.e., f_{anc} and e), connectedness of acyclicity demands that A_{anc} should also exist in the parent $\hat{e}$ of e. As G is reduced, e must have an attribute X that does not appear in $\hat{e}$ and thus must be exclusive. It follows that $X \neq A_{\text{anc}}$. However, in that case, $e' = e \setminus \{A_{\text{anc}}\}$ contains X and thus cannot be subsumed in G' (X remains exclusive in G'), giving a contradiction.

C PROOF OF LEMMA 11

We discuss only the scenario where $map(f_{\text{anc}})$ is not subsumed in G' (the opposite case is easy and omitted). Our proof will establish a stronger claim:

Claim. *$\mathcal{F}^* = \mathcal{F}'$ is the CEC of G^* induced by T^* every time Line 5 of cleanse is executed.*

$G^* = G'$ and $T^* = T'$ at Line 1. $\mathcal{F}^* = \mathcal{F}'$ is the CEC of G^* induced by T^* at this moment (Lemma 10). Hence, the claim holds on the first execution of Line 5.

Inductively, assuming that the claim holds currently, we will show that it still does after cleanse deletes the next e_{small} from G^*. Let G_0^* and T_0^* (respectively, G_1^* and T_1^*) be the G^* and T^* before (respectively, after) the deletion of e_{small}, . The fact e_{small} being subsumed in G^* suggests e_{small} being subsumed in G'. By Lemma 10, $e_{\text{small}} \notin \mathcal{F}' = \mathcal{F}^*$.

Case 1: e_{big} parents e_{small}. Let σ_0 be a reverse topological order of T_0^* where e_{big} succeeds e_{small}. As $\mathcal{F}^*$ is the CEC of G_0^* induced by T_0^*, edge-cover(T_0^*) produces $\mathcal{F}^*$ if executed on σ_0 (Lemma 8).

Let σ_1 be a copy of σ_0 but with e_{small} removed; σ_1 is a reverse topological order of T_1^*. Every node in T_1^* retains the same disappearing attributes as in T_0^* (see Figure 4(a)), whereas e_{small} has no disappearing attributes. It is easy to verify that running edge-cover(T_1^*) on σ_1 has the same output $\mathcal{F}^*$ as running edge-cover(T_0^*) on σ_0.

Case 2: e_{small} parents e_{big}. Let σ_0 be a reverse topological order of T_0^* where e_{small} succeeds e_{big}. Let σ_1 be a copy of σ_0 but with e_{small} removed; σ_1 is a reverse topological order of T_1^*. We will argue that running edge-cover(T_1^*) on σ_1 also returns $\mathcal{F}^*$.

The reader should note several facts about disappearing attributes. If an attribute has e_{small} as the summit in T_0^*, the attribute's summit in T_1^* becomes e_{big} (see Figure 4(b)). If an attribute has $e \neq e_{\text{small}}$ as the summit in T_0^*, its summit in T_1^* is still e. Hence, every node in T_1^* except e_{big} retains the same disappearing attributes as in T_0^*, whereas the disappearing attributes of e_{big} in T_1^* contain those of e_{big} and e_{small} in T_0^*.

For each node e in σ_0 (respectively, σ_1), denote by $F_0(e)$ (respectively, $F_1(e)$) the content of F_{tmp} after edge-cover(T_0^*) (respectively, edge-cover(T_1^*)) has processed e. Let e_{before} be the node before e_{big} in σ_0.[10] It is easy to see that edge-cover(T_0^*) and edge-cover(T_1^*) behave the same way until finishing with e_{before}, which gives $F_0(e_{\text{before}}) = F_1(e_{\text{before}})$. It must hold that $e_{\text{small}} \notin F_0(e_{\text{small}})$ (otherwise, e_{small} would be a subsumed edge in $\mathcal{F}^*$, contradicting Lemma 10). Two possibilities apply to e_{big}:

(1) $e_{\text{big}} \in F_0(e_{\text{big}})$. Hence, e_{big} has a disappearing attribute in T_0^* not covered by $F_0(e_{\text{before}})$. This means that e_{big} also has a disappearing attribute in T_1^* not covered by $F_1(e_{\text{before}}) = F_0(e_{\text{before}})$. It follows that $e_{\text{big}} \in F_1(e_{\text{big}})$, meaning $F_1(e_{\text{big}}) = F_0(e_{\text{big}}) = F_0(e_{\text{small}})$.

[10] In the special case where e_{big} is the first in σ_0, define $e_{\text{before}} = nil$ with $F_0(e_{\text{before}}) = F_1(e_{\text{before}}) = \emptyset$.

(2) $e_{\text{big}} \notin F_0(e_{\text{big}})$. All the disappearing attributes of e_{big} and e_{small} in T_0^* are covered by $F_0(e_{\text{before}})$. Hence, the disappearing attributes of e_{big} in T_1^* are covered by $F_1(e_{\text{before}}) = F_0(e_{\text{before}})$. Therefore, $e_{\text{big}} \notin F_1(e_{\text{big}})$, meaning $F_0(e_{\text{small}}) = F_0(e_{\text{before}}) = F_1(e_{\text{before}}) = F_1(e_{\text{big}})$.

We now conclude that $F_1(e_{\text{big}}) = F_0(e_{\text{small}})$ always holds. Every remaining node in σ_0 and σ_1 has the same disappearing attributes in T_0^* and T_1^*. The rest execution of edge-cover(T_0^*) is identical to that of edge-cover(T_1^*).

D PROOF OF LEMMA 12

We will discuss only the scenario where $map(f_{\text{anc}})$ is not subsumed (the opposite scenario is easy and omitted).

Departing from acyclic queries, let us consider a more general problem on a rooted tree $\mathcal{T}$ where (i) every node is colored black or white, and (ii) the root and all the leaves are black. Denote by B the set of black nodes. Each black node $b \in B$ is associated with a *signature path*:

— If b is the root of $\mathcal{T}$, its signature path contains just b itself.
— Otherwise, let $\hat{b}$ be the lowest ancestor of b among all the nodes in B; the signature path of b is the set of nodes on the path from $\hat{b}$ to b, except $\hat{b}$.

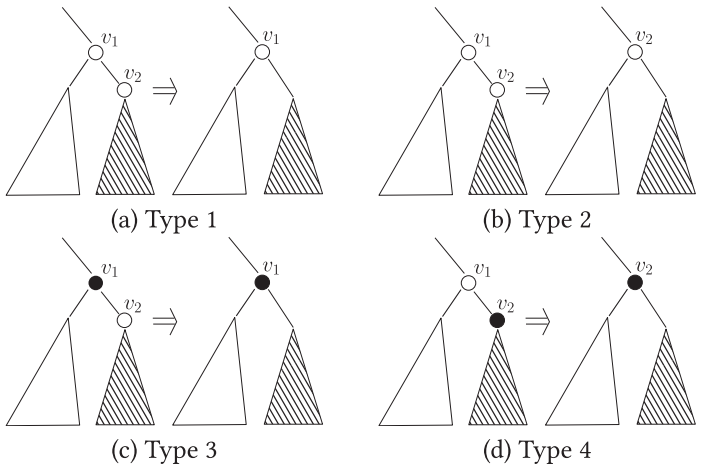


Fig. 8. Four types of contraction.

We define four types of contractions:

— *Type 1*: We are given two white nodes v_1 and v_2 such that v_1 parents v_2. The contraction removes v_2 from $\mathcal{T}$ and makes v_1 the new parent for all the child nodes of v_2. See Figure 8(a).
— *Type 2*: We are given two white nodes v_1 and v_2 such that v_1 parents v_2. The contraction removes v_1 from $\mathcal{T}$, makes v_2 the new parent for all the child nodes of v_1, and makes v_2 a child of the original parent of v_1. See Figure 8(b).
— *Type 3*: Same as Type 1, except that v_1 is black and v_2 is white. See Figure 8(c).
— *Type 4*: Same as Type 2, except that v_1 is white and v_2 is black. See Figure 8(d).

The facts below are evident:

— The number of black nodes remains the same after a contraction.
— After a contraction, each signature path either remains the same or shrinks.

We now draw correspondence between a contraction and an edge deletion in cleanse. $\mathcal{T}$ corresponds to the current edge tree T^* in cleanse. The set B of black nodes equals $\mathcal{F}^* = \mathcal{F}'$ for the entire execution of cleanse. The set $\{v_1, v_2\}$ corresponds to $\{e_{\text{small}}, e_{\text{big}}\}$. As shown in Lemma 10, e_{small} cannot exist in $\mathcal{F}^*$ and thus cannot correspond to a black node. If we denote by C (respectively, C^*) the set of signature paths at the beginning (respectively, end) of cleanse, each signature path in C^* is obtained by continuously shrinking a distinct signature path in C. This implies Lemma 12, noticing that $C = \{\text{sigpath}(f, T) \mid f \in \mathcal{F}\}$ and $C^* = \{\text{sigpath}(f^*, T^*) \mid f^* \in \mathcal{F}^*\}$.

E PROOF OF LEMMA 13

We will first prove that, for any $z \in Z$, $\mathcal{F}_z^*$ is the CEC of G_z^* induced by T_z^*. Let $\hat{z}$ be the parent of z. Recall that $\mathcal{F}$ is the CEC of G induced by T. Consider a reverse topological order σ_z of T satisfying the following condition: a prefix of σ_z is a permutation of the nodes in the subtree of T rooted at z. In other words, in σ_z, every node in the aforementioned subtree must rank before every node outside the subtree. Define σ_z^* to be the sequence obtained by deleting from σ_z all the nodes e such that $e \neq \hat{z}$ and e is outside the subtree of T rooted at z. It is clear that σ_z^* is a reverse topological order of T_z^*.

Let us compare the execution of edge-cover(T) on σ to that of edge-cover(T_z^*) on σ_z^*. They are exactly the same until z has been processed. Hence, every node in the F_{tmp} of edge-cover(T) at this moment must have been added to F_{tmp} by edge-cover(T_z^*). This means that all the nodes in $\mathcal{F}_z^*$, except $\hat{z}$, must appear in the final F_{tmp} output by edge-cover(T_z^*). Finally, the final F_{tmp} must also contain $\hat{z}$ as well due to Lemma 8 (notice that $\hat{z}$ is a raw leaf of T_z^*). This shows that $\mathcal{F}_z^*$ is the CEC of G_z^* induced by T_z^*.

Next, we prove that $\overline{\mathcal{F}^*}$ is the CEC of $\overline{G^*}$ induced by $\overline{T^*}$. Let $\overline{e}$ be the highest node in sigpath(f_{anc}, T). Consider a reverse topological order $\overline{\sigma}$ of T satisfying the following condition: a prefix of $\overline{\sigma}$ is a permutation of the nodes in the subtree of T rooted at $\overline{e}$. Define $\overline{\sigma^*}$ to be the sequence obtained by deleting that prefix from $\overline{\sigma}$. It is clear that $\overline{\sigma^*}$ is a reverse topological order of $\overline{T^*}$. Define $\hat{\overline{e}}$ to be the parent of $\overline{e}$ in T. Note that $\hat{\overline{e}}$ must belong to $\mathcal{F}$ due to the definitions of $\overline{e}$ and sigpath(f_{anc}, T)

We will compare the execution of edge-cover(T) on σ to that of edge-cover($\overline{T^*}$) on $\overline{\sigma^*}$. For each e in σ, define $F_0(e)$ as the content of F_{tmp} after edge-cover(T) has finished processing e. Similarly, for each e in $\overline{\sigma^*}$, define $F_1(e)$ as the content of F_{tmp} after edge-cover($\overline{T^*}$) has finished processing e. Divide σ into three segments: (i) σ_1, which includes the prefix of σ ending at (and including) $\overline{e}$, (ii) σ_2, which starts right after σ_1 and ends at (and includes) $\hat{\overline{e}}$, and (iii) σ_3, which is the rest of σ. Note that $\overline{\sigma^*}$ is the concatenation of σ_2 and σ_3.

Claim 1. *For any e in σ_2, $e \in F_0(e)$ if and only if $e \in F_1(e)$.*

We prove the claim by induction. As the base case, consider e as the first element in σ_2. In $\overline{T^*}$, e must be a leaf and, by Lemma 8, must be in $F_1(e)$. In T, e is either a leaf or $\hat{\overline{e}}$. In the former case, Lemma 8 assures us $e \in F_0(e)$. In the latter case, e is also in $F_0(e)$ because $\hat{\overline{e}} \in \mathcal{F}$.

Next, we prove the claim on every other node e in σ_2, assuming the claim's correctness on the node e_{before} preceding e in σ_2. This inductive assumption implies $F_1(e_{\text{before}}) \subseteq F_0(e_{\text{before}})$. If $e \in F_0(e)$, then e has a disappearing attribute X not covered by $F_0(e_{\text{before}})$. As $F_1(e_{\text{before}}) \subseteq F_0(e_{\text{before}})$, $F_1(e_{\text{before}})$ does not cover X, either. Hence, edge-cover($\overline{T^*}$) adds e to F_{tmp}, namely, $e \in F_1(e)$.

Let us now focus on the case where $e \in F_1(e)$. If $e = \hat{\overline{e}}$, the fact $\hat{\overline{e}} \in \mathcal{F}$ indicates $e \in F_0(e)$. Next, we consider $e \neq \hat{\overline{e}}$, meaning that e is a proper descendant of $\hat{\overline{e}}$. The fact $e \in F_1(e)$ suggests

that e has a disappearing attribute X not covered by $F_1(e_{\text{before}})$. If $e \notin F_0(e)$, $F_0(e_{\text{before}})$ must have a node e' containing X. Node e' must come from σ_1 (the inductive assumption prohibits e' from appearing in σ_2) and hence must be a descendant of $\overline{e}$. By acyclicity's connectedness requirement, X appearing in both e and e' means that X must belong to $\hat{\overline{e}}$. But this contradicts X disappearing at e. We thus conclude that $e \in F_0(e)$.

Claim 2. *For any e in σ_3, $e \in F_0(e)$ if and only if $e \in F_1(e)$.*

Claim 1 assures us that $F_1(\hat{\overline{e}}) \subseteq F_0(\hat{\overline{e}})$. Note also that $\hat{\overline{e}}$ belongs to $F_0(\hat{\overline{e}})$ (as explained before, $\hat{\overline{e}} \in \mathcal{F}$) and hence also to $F_1(\hat{\overline{e}})$ (Claim 1). Any node $e' \in F_0(\hat{\overline{e}}) \setminus F_1(\hat{\overline{e}})$ must appear in the subtree rooted at $\hat{\overline{e}}$ in T, whereas any node e in σ_3 must be outside that subtree. By acyclicity's connectedness requirement, if e' contains an attribute X in e, then $X \in \hat{\overline{e}}$ for sure. This means that $F_1(\hat{\overline{e}})$ covers a disappearing attribute of e if and only if $F_0(\hat{\overline{e}})$ does so. Therefore, edge-cover($\overline{T^*}$) processes each node of σ_3 in the same way as edge-cover(T). This proves the correctness of Claim 2.

By putting Claims 1 and 2 together, we conclude that edge-cover($\overline{T^*}$) returns all and only the attributes in $\sigma_2 \cup \sigma_3$ output by edge-cover(T). Therefore, the output of edge-cover($\overline{T^*}$) is $\mathcal{F} \cap \overline{E^*} = \overline{\mathcal{F}^*}$.

F PROOF OF LEMMA 14

For any $f^* \in \mathcal{F}_z^*$ and any $z \in Z$ that is not the root of T_z^*, it holds that sigpath$(f^*, T_z^*) \subseteq$ sigpath(f^*, T). Similarly, for any $f^* \in \overline{\mathcal{F}^*}$, it holds that sigpath$(f^*, \overline{T^*}) \subseteq$ sigpath(f^*, T). To prove the lemma, it suffices to show that, given a super-k-group $K = \{e_1, \ldots, e_k\}$, we can always assign each e_i, $i \in [k]$, to a distinct cluster in $\{\text{sigpath}(f, T) \mid f \in \mathcal{F}\}$. This is easy: if e_i is picked from sigpath(f^*, T_z^*) for some $z \in \mathcal{F}$ and $f^* \in \mathcal{F}_z^*$, assign e_i to sigpath(f^*, T); if e_i is picked from sigpath$(f^*, \overline{T^*})$ for some $f^* \in \overline{\mathcal{F}^*}$, assign e_i to sigpath(f^*, T).

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[2] Foto N. Afrati, Manas R. Joglekar, Christopher Ré, Semih Salihoglu, and Jeffrey D. Ullman. 2017. GYM: A multiround distributed join algorithm. In *Proceedings of the International Conference on Database Theory (ICDT'17)*. 4:1–4:18.

[3] Foto N. Afrati and Jeffrey D. Ullman. 2011. Optimizing multiway joins in a map-reduce environment. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 23, 9 (2011), 1282–1298.

[4] Alok Aggarwal and Jeffrey Scott Vitter. 1988. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)* 31, 9 (1988), 1116–1127.

[5] Kaleb Alway, Eric Blais, and Semih Salihoglu. 2021. Box covers and domain orderings for beyond worst-case join processing. In *Proceedings of the International Conference on Database Theory (ICDT'21)*. 3:1–3:23.

[6] Albert Atserias, Martin Grohe, and Daniel Marx. 2013. Size bounds and query plans for relational joins. *SIAM J. Comput.* 42, 4 (2013), 1737–1767.

[7] Paul Beame, Paraschos Koutris, and Dan Suciu. 2017. Communication steps for parallel query processing. *Journal of the ACM (JACM)* 64, 6 (2017), 40:1–40:58.

[8] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering conjunctive queries under updates. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'17)*. 303–318.

[9] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'04)*. 137–150.

[10] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2001. The complexity of acyclic conjunctive queries. *Journal of the ACM (JACM)* 48, 3 (2001), 431–498.

[11] Xiao Hu. 2021. Cover or pack: New upper and lower bounds for massively parallel joins. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'21)*. 181–198.

[12] Xiao Hu and Ke Yi. 2016. Towards a worst-case I/O-Optimal algorithm for acyclic joins. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'16)*. 135–150.

[13] Xiao Hu and Ke Yi. 2019. Instance and output optimal parallel algorithms for acyclic joins. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'19)*. 450–463.

[14] Xiao Hu, Ke Yi, and Yufei Tao. 2019. Output-optimal massively parallel algorithms for similarity joins. *ACM Transactions on Database Systems (TODS)* 44, 2 (2019), 6:1–6:36.
[15] Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. 2017. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *Proceedings of the ACM Management of Data (SIGMOD'17)*. ACM, 1259–1274.
[16] Bas Ketsman and Dan Suciu. 2017. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'17)*. 417–428.
[17] Bas Ketsman, Dan Suciu, and Yufei Tao. 2022. A near-optimal parallel algorithm for joining binary relations. *Log. Methods Comput. Sci.* 18, 2 (2022).
[18] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Re, and Atri Rudra. 2016. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems (TODS)* 41, 4 (2016), 22:1–22:45.
[19] Paraschos Koutris, Paul Beame, and Dan Suciu. 2016. Worst-case optimal algorithms for parallel query processing. In *Proceedings of the International Conference on Database Theory (ICDT'16)*. 8:1–8:18.
[20] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: Interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment (PVLDB)* 3, 1 (2010), 330–339.
[21] Gonzalo Navarro, Juan L. Reutter, and Javiel Rojas-Ledesma. 2020. Optimal joins using compact data structures. In *Proceedings of the International Conference on Database Theory (ICDT'20)*, Vol. 155. 21:1–21:21.
[22] Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. 2014. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'14)*. 234–245.
[23] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2012. Worst-Case optimal join algorithms: [Extended Abstract]. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'12)*. 37–48.
[24] Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. 2018. Worst-case optimal join algorithms. *Journal of the ACM (JACM)* 65, 3 (2018), 16:1–16:40.
[25] Hung Q. Ngo, Christopher Re, and Atri Rudra. 2013. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec.* 42, 4 (2013), 5–16.
[26] Anna Pagh and Rasmus Pagh. 2006. Scalable computation of acyclic joins. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'06)*. 225–232.
[27] Miao Qiao and Yufei Tao. 2021. Two-attribute skew free, isolated CP theorem, and massively parallel joins. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'21)*. 166–180.
[28] Cheng Sheng, Yufei Tao, and Jianzhong Li. 2012. Exact and approximate algorithms for the most connected vertex problem. *ACM Transactions on Database Systems (TODS)* 37, 2 (2012), 12:1–12:39.
[29] Yufei Tao. 2018. Massively parallel entity matching with linear classification in low dimensional space. In *Proceedings of the International Conference on Database Theory (ICDT'18)*, Vol. 98. 20:1–20:19.
[30] Yufei Tao. 2020. A simple parallel algorithm for natural joins on binary relations. In *Proceedings of the International Conference on Database Theory (ICDT'20)*. 25:1–25:18.
[31] Yufei Tao. 2022. Parallel acyclic joins with canonical edge covers. In *Proceedings of the International Conference on Database Theory (ICDT'22)*. 9:1–9:19.
[32] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy. 2010. HIVE - a petabyte scale data warehouse using Hadoop. In *Proceedings of the International Conference on Data Engineering (ICDE'10)*. 996–1005.
[33] Todd L. Veldhuizen. 2014. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of the International Conference on Database Theory (ICDT'14)*. 96–106.
[34] Mihalis Yannakakis. 1981. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases* (September 9–11, 1981, Cannes, France). 82–94.
[35] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*. 15–28.

Received 15 August 2022; revised 8 November 2023; accepted 10 November 2023