Parallel Acyclic Joins with Canonical Edge Covers

Yufei Tao ✉
Chinese University of Hong Kong, Hong Kong

Abstract

In PODS'21, Hu presented an algorithm in the massively parallel computation (MPC) model that processes any acyclic join with an asymptotically optimal load. In this paper, we present an alternative analysis of her algorithm. The novelty of our analysis is in the revelation of a new mathematical structure – which we name *canonical edge cover* – for acyclic hypergraphs. We prove non-trivial properties for canonical edge covers that offer us a graph-theoretic perspective about why Hu's algorithm works.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms

Keywords and phrases Joins, Conjunctive Queries, MPC Algorithms, Parallel Computing

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.9

Related Version *Full Version*: `https://arxiv.org/abs/2201.03832` [22]

Funding This work was partially supported by GRF projects 142078/20 and 142034/21 from HKRGC.

1 Introduction

Massively parallel join processing has attracted considerable attention in recent years. This line of research makes two types of contributions. The first consists of algorithms that promise excellent performance. The second, more subtle, type of contributions comprises knowledge revealing *mathematical structures* in the underlying problems. The latter is a necessary side-product of the former. In general, as human beings switch to a more generic setting, their knowledge from restrictive settings often proves insufficient, which then necessitates deeper investigation into the problem characteristics. Traditional studies have focused on joins in the RAM computation model [4, 14, 16, 18, 19], a degenerated "parallel" setup having only one machine. Designing algorithms to work with any number of machines poses serious challenges and demands novel findings [3, 7, 8, 10, 12, 13, 15, 20, 21] beyond the RAM literature.

This paper will focus on *acyclic joins*, a class of joins with profound importance in database systems [1, 7–9, 11, 23]. Recently, Hu [8] developed a worst-case optimal massively parallel algorithm for acyclic joins. In the current work, we will provide an alternative, hopefully more accessible, analysis of her elegant algorithm. The real excitement from our analysis is the identification of a new mathematical structure – we call "canonical edge cover" – for acyclic hypergraphs. The structure reveals a unique characteristic of acyclic joins and is a core reason why Hu's algorithm works.

1.1 Problem Definition

Acyclic Joins. Let **att** be a set where each element is called an *attribute*. Let **dom** be another set where each element is called a *value*. We assume a total order on **dom**; if not, manually impose one by ordering the values arbitrarily.

25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier; Article No. 9; pp. 9:1–9:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

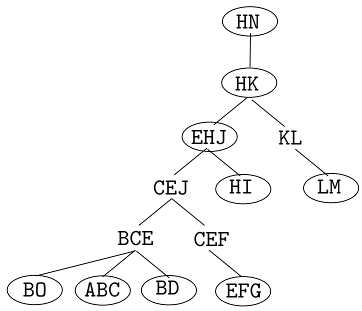


Figure 1 A hyperedge tree example.

A *tuple* over a set $U \subseteq \textbf{att}$ is a function $\boldsymbol{u} : U \to \textbf{dom}$. 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 $\boldsymbol{u}[U']$ – the *projection* of $\boldsymbol{u}$ on 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 $scheme(R) = U$. If U is the empty set $\emptyset$, then R is also $\emptyset$.

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 – denoted as $Join(Q)$ – is the following relation over $attset(Q)$

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

If the relations in Q are $R_1, R_2, ..., R_{|Q|}$, we may represent $Join(Q)$ also as $R_1 \bowtie R_2 \bowtie ... \bowtie R_{|Q|}$.

Q can be characterized by a hypergraph $G = (V, E)$ where each vertex in V is a distinct attribute in $attset(Q)$, and each hyperedge in E is the scheme of a distinct relation in Q. E may contain identical hyperedges because two (or more) relations in Q can have the same scheme. The term "hyper" suggests that a hyperedge can have more than two attributes.

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

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

We will call T a *hyperedge tree* of G (also known as the *join tree* of Q in the literature).

▶ **Example 1.** Consider the hypergraph $G = (V, E)$ where $V = \{\texttt{A}, \texttt{B}, ..., \texttt{O}\}$ and $E = \{\texttt{ABC}, \texttt{BD}, \texttt{BO}, \texttt{EFG}, \texttt{BCE}, \texttt{CEF}, \texttt{CEJ}, \texttt{HI}, \texttt{LM}, \texttt{EHJ}, \texttt{KL}, \texttt{HK}, \texttt{HN}\}$. Figure 1 shows a hyperedge tree T of G. To understand the connectedness requirement, observe the connected subtree formed by the five hyperedges involving E. ⌟

As G and T both contain "vertices" and "edges", for better clarity we will obey several conventions throughout the paper. 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 hyperedges, we will always refer to an edge in T as a *link*.

We use m to denote the *input size* of Q, defined as $\sum_{R \in Q} |R|$, namely, the total number of tuples in the relations participating in the join.

Computation Model. We assume the *massively parallel computation* (MPC) model which is popular in designing massively parallel algorithms [3, 7, 8, 10, 12, 13, 15, 20, 21]. In this model, we have p machines, each storing $\Theta(m/p)$ tuples from the relations of a query Q initially.

An algorithm executes in *rounds*, each having two phases: in the first phase, each machine performs local computation; in the second, the machines exchange messages (every message must have been generated at the end of the first phase). An algorithm must finish in a constant number of rounds, and when it does, every tuple in $Join(Q)$ must reside 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. The objective is to design an algorithm with the smallest load.

Math Conventions. The number p of machines is considered to be at most $m^{1-\epsilon}$, for some arbitrarily small constant $\epsilon > 0$. Every value in **dom** can be represented with $O(1)$ words. Our discussion focuses on *data complexities*, namely, we are interested in the influence of m on algorithm performance. For that reason, we assume that the hypergraph G of Q has $O(1)$ vertices. Given an integer $x \geq 1$, the notation $[x]$ represents the set $\{1, 2, ..., x\}$.

1.2 Previous Results

Fractional Edge Coverings and the AGM bound. Consider a query Q (which may or may not be acyclic) with hypergraph $G = (V, E)$. Associate every hyperedge $e \in E$ with a real-valued *weight* w_e, which falls between 0 and 1. Impose a constraint on every attribute $X \in V$: $\sum_{e \in E: X \in e} w_e \geq 1$, i.e., the total weight of all the hyperedges covering X must be at least 1. A set of weights $\{w_e \mid e \in E\}$ fulfilling all the constraints is a *fractional edge covering* of G. If we define $\sum_{e \in E} w_e$ as the *total weight* of the fractional edge covering, the *fractional edge covering number* of G – denoted as ρ – is the minimum total weight of all possible fractional edge coverings. A fractional edge covering is *optimal* if its total weight equals ρ.

The *AGM bound*, proved by Atserias, Grohe, and Marx [5], states that the size of $Join(Q)$ is always bounded by $O(m^\rho)$; recall that m is the input size of Q. Furthermore, the bound is tight: in the worst case, $|Join(Q)|$ can indeed reach $\Omega(m^\rho)$ [5].

Simplification for Acyclic Queries: Edge Covers. When Q is acyclic, $G = (V, E)$ always admits an optimal fractional edge covering with *integral* weights [8]. Recall that all the weights w_e ($e \in E$) must fall between 0 and 1. Hence, every weight in an optimal fractional edge covering must be either 0 or 1. This pleasant property allows the reader to connect ρ to edge "covers". A subset $S \subseteq E$ is an *edge cover*[1] of G if every attribute of V appears in at least one hyperedge of S. Thus, the value of ρ is simply the minimum size of all edge covers, namely, the smallest number of hyperedges that we must pick to cover all the attributes.

Join Algorithms in RAM. An algorithm able to answer Q using $O(m^\rho)$ time in the RAM model is worst-case optimal. Indeed, as $|Join(Q)|$ can be $\Omega(m^\rho)$, we need $\Theta(m^\rho)$ time just to output $Join(Q)$ in the worst case. Ngo et al. [17] designed the first algorithm that guarantees a running time of $O(m^\rho)$ for all queries. Since then, the community has discovered more algorithms [4,14,16,18,19] that are all worst-cast optimal (sometimes up to a polylogarithmic factor) but differ in their own features. For an acyclic Q, an algorithm due to Yannakakis [23] achieves a stronger sense of optimality: his algorithm runs in $O(m + |Join(Q)|)$ time, which is clearly the best regardless of $|Join(Q)|$.

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

Join Algorithms in MPC. Koutris, Beame, and Suciu [15] showed that, in the MPC model, the AGM bound implies a worst-case lower bound of $\Omega(m/p^{1/\rho})$ on the load of any algorithm that answers a query Q, where m is the input size of Q and ρ is the fractional edge covering number of the hypergraph $G = (V, E)$ defined by Q.

The above negative result has motivated considerable research looking for MPC algorithms whose loads are bounded by $O(m/p^{1/\rho})$, ignoring polylogarithmic factors; such algorithms are worst-case optimal. The goal has been realized only on four query classes. The first consists of all the cartesian-product queries (i.e., the relations in Q have disjoint schemes); see [3, 6, 13] for several optimal algorithms on such queries. The second is the so-called Loomis-Whitney join, where E consists of all the $|V|$ possible hyperedges of $|V| - 1$ attributes; see [15] for an optimal algorithm for such joins. The third class includes every query where all the hyperedges in G contain *at most* two attributes; see [12, 13, 21] for the optimal algorithms. The fourth class comprises all the acyclic queries, which were recently solved by Hu [8] optimally. It is worth pointing out that Hu's algorithm subsumes an earlier algorithm of [9] which is worst-case optimal on a subclass of acyclic queries.

Although it still remains elusive what other query classes can be settled with load $O(m/p^{1/\rho})$, now we know that this is *unachievable* for certain queries. In [8], Hu constructed a class of queries for which every algorithm must incur a load of $\omega(m/p^{1/\rho})$ in the worst case. The result of [8] suggests that additional parameters – other than m, p, and ρ – are needed to describe the worst-case optimality of an ideal MPC algorithm. We will not delve into the issue further because it does not apply to acyclic queries (the focus of this paper), but the reader may consult the recent works [8, 20] for the latest development on that issue. Finally, we remark that several algorithms [2, 9, 10] are able to achieve a load sensitive to the join size $|Join(Q)|$.

1.3 Our Contributions

The first, easy-to-discern, contribution of our paper is a new analysis of Hu's algorithm [8] for acyclic queries. Our second contribution is the introduction of *canonical edge cover* as a mathematical structure inherent in acyclic queries. We prove a suite of graph-theoretic properties for canonical edge covers and use them to give a more fundamental interpretation of the design choices in Hu's algorithm. The rest of the section will provide an overview of our results and techniques.

Clustering, k-Groups, k-Products, and Induced Loads. We first create a conceptual framework to state Hu's and our results on a common ground. Define a *clustering* of E (the hyperedge set of G) as a set $\{E_1, E_2, ..., E_s\}$ for some $s \geq 1$ where (i) each E_i is a subset of E, $i \in [s]$, and (ii) $\bigcup_i E_i = E$. We call each E_i a *cluster*; note that the clusters need *not* be disjoint.

Fix an arbitrary clustering $C = \{E_1, E_2, ..., E_s\}$. Given an integer $k \geq 1$, define a *k-group* of C as a collection of k hyperedges, each taken from a distinct cluster.

▶ **Example 2.** Let $G = (V, E)$ be the hypergraph in Example 1 (Figure 1). $C = \{\{$`BO`, `BCE`, `CEJ`$\}$, $\{$`ABC`, `BCE`, `CEJ`$\}$, $\{$`BD`, `BCE`, `CEJ`$\}$, $\{$`EFG`, `CEF`, `CEJ`$\}$, $\{$`HI`$\}$, $\{$`EHJ`$\}$, $\{$`LM`, `KL`$\}$, $\{$`HK`$\}$, $\{$`HN`$\}\}$ is a clustering of E. A 3-group example is $\{$`ABC`, `BD`, `EFG`$\}$. Note that the hyperedges in a k-group need not be distinct. For example, $\{$`CEJ`, `CEJ`, `CEJ`$\}$ is also a 3-group: the first `CEJ` is taken from the cluster $\{$`ABC`, `BCE`, `CEJ`$\}$, the second from $\{$`BD`, `BCE`, `CEJ`$\}$, and the third from $\{$`EFG`, `CEF`, `CEJ`$\}$. For a non-example, $\{$`ABC`, `LM`, `KL`$\}$ is not a 3-group. ⌟

For each hyperedge $e \in E$, let $R(e)$ represent the relation in Q whose scheme is e. Given a k-group K of the clustering C, we define the Q*-product* of K as $\prod_{e \in K} |R(e)|$ (i.e., the cartesian-product size of all the relevant relations). Given an integer k, we define the *max* (k, Q)*-product* of C – denoted as $P_k(Q, C)$ – as the maximum Q-product of all the k-groups of C.

▶ **Example 3.** Continuing on the previous example, the Q-product of the 3-group $\{\mathtt{ABC}, \mathtt{BCE}, \mathtt{CEJ}\}$ is $|R(\mathtt{ABC})| \cdot |R(\mathtt{BCE})| \cdot |R(\mathtt{CEJ})|$, while that of the 3-group $\{\mathtt{CEJ}, \mathtt{CEJ}, \mathtt{CEJ}\}$ is $|R(\mathtt{CEJ})|^3$. ⌟

Define the Q*-induced load* of C as

$$\max_{k=1}^{s} \left(P_k(Q, C)/p\right)^{1/k} \tag{1}$$

As $P_k(Q, C) \le m^k$ for any $k \in [s]$, it must hold that $(1) \le m/p^{1/s}$.

We can now give a more detailed account of Hu's result [8]. She proved that the load of her algorithm is bounded by $O(L)$, where L is the Q-induced load of a *certain* clustering with size $s = \rho$, and ρ is the fractional edge covering number of G. It thus follows immediately that $L \le m/p^{1/s}$. In [8], Hu presented a recursive procedure to identify the clustering C whose Q-induced load equals the target L. The procedure, however, is somewhat sophisticated, making it difficult to describe the target C in a succinct manner. Such difficulty is unjustified, especially given the algorithm's elegance, and indicates the existence of a hidden mathematical structure.

Our Results and Techniques. A hypergraph G can have many optimal edge covers (all of which must have size ρ). While Hu's analysis [8] assumes an arbitrary optimal edge cover, we will be choosy about what we work with. In Figure 1, the 9 circled nodes constitute a *canonical edge cover* F of G. Let us give an informal but intuitive explanation of how to construct this F. After rooting the tree in Figure 1 at `HN`, we add to F all the leaf nodes: `BO`, `ABC`, `BD`, `EFG`, `HI`, `LM`. Then, we process the non-leaf nodes bottom up. In processing `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 F already cover `B`? The answer is yes, due to the existence of `BO`; we therefore do *not* include `BCE` in F. We process `BCE`, `CEF`, and `CEJ` similarly, none of which enters F. At `EHJ`, we find disappearing attributes `E` and `J`. In general, as long as one disappearing attribute has not been covered by F, we pick the node; this is why `EHJ` is in F. The other nodes `HK` and `HN` in F are chosen based on the same reasoning.

We show that a canonical edge cover determined this way has appealing properties which fit the recursive strategy behind Hu's algorithm very well. At a high level, Hu's algorithm works by simplifying G into a number of "residual" hypergraphs to be processed recursively. Interestingly, with trivial modifications (such as removing the attributes that have become irrelevant), a canonical edge cover of G *remains canonical on every residual hypergraph*. This is the most crucial property we utilize to relate the load of the original query to those of the "residual queries" in forming up a working recurrence.

Our techniques also provide a simple and natural way to pinpoint a clustering C that can be used to bound the algorithm's load. Consider the canonical edge cover F shown in Figure 1 (the circled nodes). For each node in F, take a "signature path" by walking up and stopping right before reaching its lowest proper ancestor in F. For example, the signature path of `ABC` is {`ABC`, `BCE`, `CEJ`} (note: the path does not contain `EHJ`). Likewise, the signature path of `LM` is {`LM`, `KL`}. The signature paths of all the nodes in F together produce the clustering C given in Example 2. Our main result (Theorem 22) states that the Q-induced load of C is an upper bound on the load of Hu's algorithm. Because C has a size at most ρ, the algorithm's load is thus bounded by $O(m/p^{1/\rho})$.

2 Canonical Edge Covers for Acyclic Hypergraphs

This section is purely graph theoretic: we will establish several new properties for acyclic hypergraphs. Let $G = (V, E)$ be an acyclic hypergraph. A hyperedge $e_1 \in E$ is *subsumed* if it is a subset of another hyperedge $e_2 \in E$, i.e., $e_1 \subseteq e_2$. If an attribute X appears in only a single hyperedge, we call X an *exclusive attribute*; otherwise, X is *non-exclusive*. Unless otherwise stated, we allow G to be an arbitrary acyclic hypergraph. In particular, this means that E can contain two or more hyperedges with the same attributes (nonetheless, they are still distinct hyperedges) and may even have empty hyperedges (i.e., with no attributes at all). G is *clean* if E has no subsumed edges. Some of our results will apply only to clean hypergraphs.

Denote by T a hyperedge tree of G (the existence of T is guaranteed; see Section 1.1). By rooting T at an arbitrary leaf, we can regard T as a *rooted* tree. Make all the links[2] of T point *downwards*, i.e., from parent to child. This way, T becomes a directed acyclic graph.

Now that there are two views of T (i.e., undirected and directed), we ought to be careful with 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 (i.e., a raw leaf can be a leaf or the root under the directed view)

2.1 Fundamental Definitions and Properties

Summits and Disappearing Attributes. 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 each attribute $X \in V$, we define the *summit* of X as the highest node (a.k.a. a hyperedge) that contains X. If node e is the summit of X, we call X a *disappearing* attribute in e. By acyclicity's connectedness 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 4.** Let $G = (V, E)$ be the hypergraph in Example 1 whose (rooted) hypergraph 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`. Hence, both `E` and `J` are disappearing attributes of `EHJ`. ◀

Canonical Edge Cover. We say that a subset $S \subseteq E$ *covers* an attribute $X \in V$ if S has a hyperedge containing X. Recall 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:

---

[2] Remember that we refrain from saying "edges" of T; see Section 1.1.

edge-cover (T) /* T is rooted */

1. $F_{\mathrm{tmp}} = \emptyset$
2. obtain a reverse topological order $e_1, e_2, ..., e_{|E|}$ of the nodes (i.e., hyperedges) in T
3. **for** $i = 1$ 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}

▶ **Lemma 5.** *The output of* edge-cover *– denoted as F – is an optimal edge cover of G, and does <u>not</u> depend on the reverse topological order at Line 2. Furthermore, if G is clean, F includes all the raw leaves of T.*

All the missing proofs can be found in the full version [22]. We refer to F as the *canonical edge cover* (CEC) of G induced by T. The size of F is precisely the fractional edge covering number ρ of Q.

▶ **Example 6.** Continuing on the previous example, consider the reverse topological order of T: `ABC`, `BD`, `BO`, `BCE`, `EFG`, `CEF`, `CEJ`, `HI`, `EHJ`, `LM`, `KL`, `HK`, `HN`. When processing `ABC`, edge-cover adds it to F_{tmp} because `ABC` has a disappearing attribute `A` and yet $F_{\mathrm{tmp}} = \emptyset$. When processing `BCE`, $F_{\mathrm{tmp}} = \{$`ABC`, `BD`, `BO`$\}$. `BCE` has a disappearing attribute `B`, which, however, has been covered by F_{tmp}. Thus, `B` is not added to F_{tmp}. The final output of the algorithm is $F = \{$`ABC`, `BD`, `BO`, `EFG`, `HI`, `LM`, `EHJ`, `HK`, `HN`$\}$, which is the CEC of G induced by T. ◀

Signature Paths. Whenever F includes the root of T, we can define a *signature path* – denoted as $\mathrm{sigpath}(f, T)$ – for each node (i.e., hyperedge) $f \in F$. Specifically, $\mathrm{sigpath}(f, T)$ is a set of nodes defined as follows:

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

▶ **Example 7.** Consider the set F obtained in the previous example. If $f =$ `HN`, then the signature path of f is $\{$`HN`$\}$. If $f =$ `ABC`, then $\hat{f} =$ `EHJ`; and the signature path of f is $\{$`ABC`, `BCE`, `CEJ`$\}$. ◀

(Clean G) Clustering, Anchor Leaf, and Anchor Attribute. Consider $G = (V, E)$ now as a clean hypergraph. Let F be the CEC of G induced by a hyperedge tree T of G. As F contains the root and leaves of T (Lemma 5), $\{\mathrm{sigpath}(f, T) \mid f \in F\}$ is a clustering of E. If f is not the root of T, we call $\mathrm{sigpath}(f, T)$ a *non-root* cluster.[3]

Let f° be a leaf node in F, and $\hat{f}$ be the lowest proper ancestor of f° in F. We call f° an *anchor leaf* of T if two conditions are satisfied:

- $\hat{f}$ has no non-leaf proper descendants in F.
- f° has an attribute A° such that $A^\circ \notin \hat{f}$ but $A^\circ \in e$ for every node $e \in \mathrm{sigpath}(f^\circ, T)$.

A° will be referred to as an *anchor attribute* of f°.

▶ **Lemma 8.** *If G is clean, F always contains an anchor leaf.*

▶ **Example 9.** From the F constructed earlier, we obtain the clustering $C = \{\{$`BO`, `BCE`, `CEJ`$\}$, $\{$`ABC`, `BCE`, `CEJ`$\}$, $\{$`BD`, `BCE`, `CEJ`$\}$, $\{$`EFG`, `CEF`, `CEJ`$\}$, $\{$`HI`$\}$, $\{$`EHJ`$\}$, $\{$`LM`, `KL`$\}$, $\{$`HK`$\}$, $\{$`HN`$\}\}$. Other than $\{$`HN`$\}$, all the clusters in C are non-root clusters. `ABC` is an anchor leaf of T

[3] If f is the root of T, $\mathrm{sigpath}(f, T)$ contains just f itself.

with an anchor 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$(\texttt{BD}, T) = \{\texttt{BD}, \texttt{BCE}, \texttt{CEJ}\}$. Furthermore, LM is not an anchor leaf because HK, the lowest proper ancestor of LM in F, has a non-leaf proper descendant in F (i.e., EHJ). ⌟

2.2 (Clean G) Properties on Residual Hypergraphs

This subsection assumes $G = (V, E)$ to be clean. Let T be a hyperedge tree of G and F be the CEC induced by T. Fix an arbitrary anchor leaf f° of T and an anchor attribute A° of f°. We will analyze how the CEC changes as G is simplified based on f° and A°.

2.2.1 Simplification 1

The first simplification is based on removing attribute A° from G.

Residual Hypergraph. Let $G' = (V', E')$ be the *residual hypergraph* obtained by eliminating A° from G: $V' = V \setminus \{A^\circ\}$, and E' collects a hyperedge $e' = e \setminus \{A^\circ\}$ for every $e \in E$.[4] We characterize the one-one correspondence between E' and E by introducing a function $map(e) = e'$ and its inverse function $map^{-1}(e') = e$. Let T' be the hyperedge tree of G' obtained by discarding A° from every node in T (note: G' is not necessarily clean).

Canonical Edge Cover. Define

$$F' = \begin{cases} F \setminus \{f^\circ\} & \text{if } map(f^\circ) \text{ is subsumed in } G' \\ \{map(f) \mid f \in F\} & \text{otherwise} \end{cases} \tag{2}$$

▶ **Example 10.** Continuing on the previous example, if we choose $f^\circ = \texttt{ABC}$ with $A^\circ = \texttt{C}$ and eliminate C from the tree T in Figure 1, we obtain the hyperedge tree T' in Figure 2a, where the circled nodes constitute the set F'. Similarly, if we choose $f^\circ = \texttt{HI}$ with $A^\circ = \texttt{I}$, then T' and F' are as demonstrated in Figure 2b. ⌟

▶ **Lemma 11.** *If G is clean, F' is the CEC of G' induced by T'. Furthermore, if $map(f^\circ)$ is subsumed in G', then A° must be an exclusive attribute in f°.*

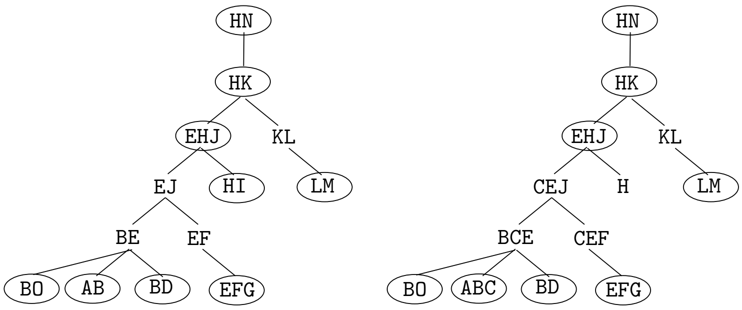
As a corollary, if $map(f^\circ)$ is subsumed in G', then every hyperedge of G, except f°, is directly retained in G'; furthermore, $map(f^\circ)$ is the only subsumed edge in G'. The next lemma gives another property of F' that holds no matter if G is clean.

▶ **Lemma 12.** *If a hyperedge e' of G' is subsumed, then $e' \notin F'$*

Cleansing. Even though G is clean, the residual hypergraph G' may contain subsumed hyperedges. Next, we describe a *cleansing* procedure which converts G' into a clean hypergraph $G^* = (V', E^*)$ (note that G^* has the same vertices as G') and converts T' into a rooted hyperedge tree T^* of G^*.

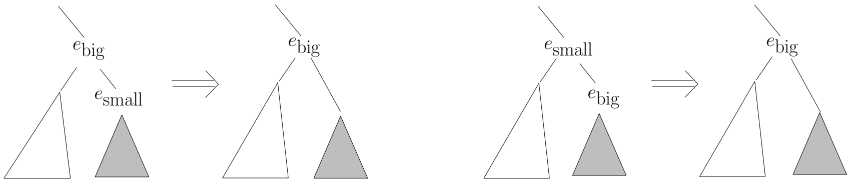
Cleansing is simple if $map(f^\circ)$ is subsumed in G'. In this case, G^* is the hypergraph obtained by removing $map(f^\circ)$ from G', and T^* is the tree obtained by removing the leaf $map(f^\circ)$ from T'. If $map(f^\circ)$ is not subsumed, the cleansing algorithm is:

[4] If $e = \{A^\circ\}$, E' collects $e' = \emptyset$.



(a) T' and F' after removing `C`. (b) T' and F' after removing `I`.

Figure 2 Residual hypergraphs.



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

Figure 3 Two cases of cleansing.

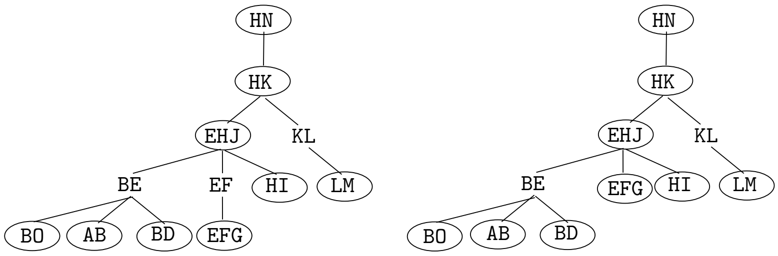
cleanse (G', T') /* condition: $map(f^\circ)$ not subsumed */
1. $G^* = G', T^* = T'$
2. **while** G^* has hyperedges e_{small} and e_{big} such that $e_{\text{small}} \subseteq e_{\text{big}}$ and they are connected by a link in T^* **do**
3. remove e_{small} from G^* and T^*
   /* $e_{\text{small}} \notin F'$ by Lemma 12 */
4. **if** e_{big} was the parent of e_{small} in T^* **then**
5. make e_{big} the new parent for all the child nodes of e_{small}; see Figure 3a
   else
6. 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 3b
7. **return** G^* and T^*

At the end of cleansing, we always set $F^* = F'$, regardless of whether $map(f^\circ)$ is subsumed.

▶ **Lemma 13.** *After cleansing, F^* is the CEC of G^* induced by T^*.*

▶ **Example 14.** In Example 10, the residual hypergraph G' in Figure 2a has two subsumed hyperedges `EJ` and `EF`, each removed by an iteration of cleanse. Suppose that the first iteration sets $e_{\text{small}} = $ `EJ` and $e_{\text{big}} = $ `EHJ` (this is a case of Figure 3a). Figure 4a illustrates the T^* after removing `EJ`. The next iteration sets $e_{\text{small}} = $ `EF` and $e_{\text{big}} = $ `EFG` (a case of Figure 4b). Figure 4b illustrates the T^* after removing `EF`. In both Figure 4a and 4b, the circled nodes constitute the CEC of G^* induced by T^*. ⌟

Distinct Clusters Lemma. The next property concerns the hypergraph $G^* = (V', E^*)$ after cleansing and the original hypergraph $G = (V, E)$. Recall that T^* and T are hyperedge trees of G^* and G, respectively. Before proceeding, the reader should recall that every hyperedge $e^* \in E^*$ corresponds to a distinct hyperedge $e \in E$, which is the hyperedge given by $map^{-1}(e^*)$.



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

Figure 4 Simplification 1.

Consider once again the CEC F of G, i.e., the original hypergraph, induced by T. As mentioned in Section 2.1, $C = \{\text{sigpath}(f, T) \mid f \in F\}$ is a clustering of E. By the same reasoning, because F^* is the CEC of G^* induced by T^* (Lemma 13), $C^* = \{\text{sigpath}(f^*, T^*) \mid f^* \in F^*\}$ must be a clustering of E^*. The following lemma draws a connection between C and C^*:

▶ **Lemma 15** (Distinct Clusters Lemma). *For any $1 \le k \le |F^*|$, if $\{e_1^*, ..., e_k^*\}$ is a k-group of C^*, then $\{map^{-1}(e_1^*), ..., map^{-1}(e_k^*)\}$ is a k-group of C.*

By definition of k-group, $e_1^*, ..., e_k^*$ originate from k distinct clusters in C^*. The lemma promises k different clusters in C each containing a distinct hyperedge in $\{map^{-1}(e_1^*), ..., map^{-1}(e_k^*)\}$.

▶ **Example 16.** Consider the T^* (and hence G^*) and F^* illustrated in Figure 4b. The clustering C^* is {{AB, BE}, {BO, BE}, {BD, BE}, {EFG}, {EHJ}, {HI}, {LM, KL}, {HK}, {HN}}. Because {BE, EFG, KL} is a 3-group of C^*, Lemma 15 asserts that $\{map^{-1}(\text{BE}), map^{-1}(\text{EFG}), map^{-1}(\text{KL})\}$ = {BCE, EFG, KL} must be a 3-group of the clustering C in Example 9. ⌟

2.2.2 Simplification 2

The second simplification decomposes G into multiple hypergraphs based on $\text{sigpath}(f^\circ, T)$.

Decomposition. Define Z to be the set of nodes z in T satisfying: z is not in $\text{sigpath}(f^\circ, T)$ but the parent of z is. 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 $\bar{T}^*$ as the rooted tree obtained by removing from T the subtree rooted at the highest node in $\text{sigpath}(f^\circ, T)$.

From each T_z^*, generate a hypergraph $G_z^* = (V_z^*, E_z^*)$. Specifically, E_z^* includes all and only the nodes (each being a hyperedge) in T_z^*, and V_z^* is the set of attributes appearing in at least one hyperedge in E_z^*. Likewise, from $\bar{T}^*$, generate a hypergraph $\bar{G}^* = (\bar{V}^*, \bar{E}^*)$ where $\bar{E}^*$ includes all and only the nodes in $\bar{T}^*$, and $\bar{V}^*$ is the set of attributes appearing in at least one hyperedge in $\bar{E}^*$.

Because G is clean, so must be all the generated hypergraphs. Furthermore, each of them has fewer edges than G.[5] For each $z \in Z$, T_z^* is a hyperedge tree of G_z^*; similarly, $\bar{T}^*$ is a hyperedge tree of $\bar{G}^*$.

[5] Because f° does not appear in any of the generated hypergraphs.

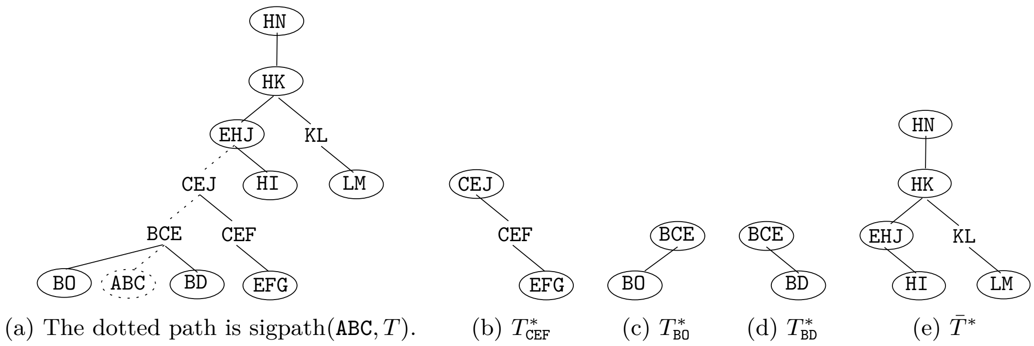


Figure 5 Decomposition.

▶ **Example 17.** In our running example, $f^\circ = \texttt{ABC}$, whose signature path is $\text{sigpath}(f^\circ, T) = \{\texttt{ABC}, \texttt{BCE}, \texttt{CEJ}\}$; see Figure 5a. $Z = \{\texttt{BO}, \texttt{BD}, \texttt{CEF}\}$. Figure 5b, 5c, and 5d illustrate T^*_z for $z = \texttt{CEF}$, BO, and BD, respectively. Figure 5e gives $\bar{T}^*_z$. ⌟

Canonical Edge Covers. Recall that F is the CEC of G induced by T. Next, we derive the CECs of the hypergraphs generated from the decomposition. For each $z \in Z$, define

$$F^*_z = \{\text{parent of } z\} \cup (F \cap E^*_z). \tag{3}$$

Also, define

$$\bar{F}^* = F \cap \bar{E}^*. \tag{4}$$

▶ **Lemma 18.** *For each node $z \in Z$, F^*_z is the CEC of G^*_z induced by T^*_z. Furthermore, $\bar{F}^*$ is the CEC of $\bar{G}^*$ induced by $\bar{T}^*$.*

▶ **Example 19.** We have circled the nodes in F^*_z in Figure 5b, 5c, and 5d for $z = \texttt{CEF}$, BO, and BD, respectively. Similarly, the circled nodes in Figure 5e constitute $\bar{F}^*_z$. ⌟

Distinct Clusters Lemma 2. We close the section with a property resembling Lemma 15.

Consider any $z \in Z$. Because $G^*_z = (V^*_z, E^*_z)$ is clean and F^*_z is the CEC of G^*_z induced by T^*_z, $C^*_z = \{\text{sigpath}(f^*, T^*_z) \mid f^* \in F^*_z\}$ is a clustering of E^*_z. Similarly, regarding $\bar{G}^* = (\bar{V}^*, \bar{E}^*)$, $\bar{C}^* = \{\text{sigpath}(f^*, \bar{T}^*) \mid f^* \in \bar{F}^*\}$ is a clustering of $\bar{E}^*$.

Define a *super-k-group* to be a set of hyperedges $K = \{e_1, e_2, ..., e_k\}$ satisfying:

- Each e_i, $i \in [k]$, is taken from a cluster of $\bar{C}^*$ or a non-root cluster[6] of C^*_z for some $z \in Z$.
- No two hyperedges in K are taken from the same cluster.

Before delving into the next lemma, the reader should recall that $\{\text{sigpath}(f, T) \mid f \in F\}$ is a clustering of E.

▶ **Lemma 20** (Distinct Clusters Lemma 2)**.** *If $\{e_1, e_2, ..., e_k\}$ is a super-k-group, then $\{e_1, e_2, ..., e_k\}$ must be a k-group of the clustering $\{\text{sigpath}(f, T) \mid f \in F\}$.*

▶ **Example 21.** In Figure 5, $C^*_{\texttt{CEF}} = \{\{\texttt{EFG}, \texttt{CEF}\}, \{\texttt{CEJ}\}\}$, $C^*_{\texttt{BO}} = \{\{\texttt{BO}\}, \{\texttt{BCE}\}\}$, $C^*_{\texttt{BD}} = \{\{\texttt{BD}\}, \{\texttt{BCE}\}\}$, $\bar{C}^* = \{\{\texttt{HI}\}, \{\texttt{EHJ}\}, \{\texttt{HK}\}, \{\texttt{HN}\}, \{\texttt{LM}, \texttt{KL}\}\}$. A super-4-group is {CEF, BO, BD, KL}. Lemma 20 assures us that {CEF, BO, BD, KL} must be a 4-group in the clustering C given in Example 9. ⌟

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

3 An MPC Algorithm

The rest of the paper will apply the theory of CECs to solve acyclic queries in the MPC model. We will describe a variant of Hu's algorithm [8] in this section[7] and present our analysis in the next section. Denote by Q the acyclic query to be answered. Let $G = (V, E)$ be the hypergraph of Q. We assume G to be clean; otherwise, Q can be converted to a clean query having the same result with load $O(m/p)$ [8]. We will also assume that Q has at least two relations; otherwise, the query is trivial and requires no communication.

3.1 Configurations

Let T be a hyperedge tree of G and F be the CEC of G induced by T. The size of F is precisely ρ, the fractional edge covering number of Q (Section 1.2). As explained in Section 2.1, when G is clean,

$$C = \{\text{sigpath}(f, T) \mid f \in F\} \tag{5}$$

is a clustering of E. Let f° be an anchor leaf of T and A° an anchor attribute of f° (Section 2.1); remember that A° appears in all the hyperedges of sigpath(f°, T). Define

$$L = \text{the } Q\text{-induced load of } C. \tag{6}$$

The reader can review Equation (1) for the definition of "Q-induced load".

For each hyperedge $e \in E$, as before $R(e)$ denotes the relation in Q corresponding to e. Fix a value $x \in \mathbf{dom}$. Given an $e \in \text{sigpath}(f^\circ, T)$, we define the *$A^\circ$-frequency* of x in $R(e)$ as the number of tuples $\boldsymbol{u} \in R(e)$ such that $\boldsymbol{u}(A^\circ) = x$. Further define the *signature-path A°-frequency* of x as the sum of its A°-frequencies in the $R(e)$ of all $e \in \text{sigpath}(f^\circ, T)$. A value $x \in \mathbf{dom}$ is

- *heavy*, if its signature-path A°-frequency is at least L;
- *light*, otherwise.

Divide **dom** into disjoint intervals such that the light values in each interval have a total signature-path A°-frequency of $\Theta(L)$. We will refer to those intervals as the *light intervals* of A°. The total number of heavy values and light intervals is at most

$$\sum_{e \in \text{sigpath}(f^\circ, T)} \frac{|R(e)|}{L} = O\left(\max_{e \in \text{sigpath}(f^\circ, T)} \frac{|R(e)|}{L}\right) = O\left(\frac{\text{max } (1, Q)\text{-product of } C}{L}\right) = O(p) \tag{7}$$

where the first equality used the fact that sigpath(f°, T) has $O(1)$ edges and the second equality applied the definition of max (k, Q)-product (see Section 1.3).

A *configuration* η is either a heavy value or a light interval. Equation (7) implies that the number of configurations is $O(p)$. For each hypergraph $e \in E$, 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 $\boldsymbol{u}(A^\circ) = \eta$;
- if η is a light interval, $R(e, \eta)$ includes all and only the tuples $\boldsymbol{u} \in R(e)$ where $\boldsymbol{u}(A^\circ)$ is a light value in η.

[7] Our algorithm follows Hu's ideas [8] but differs in certain details. For example, Hu's algorithm takes an arbitrary optimal edge cover of G as the input, while we insist on working with a CEC.

Note that $R(e, \eta) = R(e)$ if $A^\circ \notin e$. Let Q_η be the query defined by $\{R(e, \eta) \mid e \in E\}$. Our objective is to compute $Join(Q_\eta)$ for all η in parallel. The final result $Join(Q)$ is simply $\bigcup_\eta Join(Q_\eta)$.

The rest of the section will explain how to solve $Join(Q_\eta)$ for an arbitrary η. We allocate

$$p_\eta = \Theta\left(1 + \max_{k=1}^{|F|} \frac{P_k(Q_\eta, C)}{L^k}\right) \quad (8)$$

machines for this purpose, where $P_k(Q_\eta, C)$ is the max (k, Q_η)-product of C.

3.2 Solving Q_η When η is a Heavy Value

Define the residual hypergraph $G' = (V', E')$ after removing A°, and also functions $map(.)$ and $map^{-1}(.)$ as in Section 2.2.1. We compute $Join(Q_\eta)$ 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 $e' = map(e) = e \setminus \{A^\circ\}$. Specifically, $R^*(e', \eta)$ is a copy of $R(e, \eta)$ but with A° discarded, or formally, $R^*(e', \eta) = \{\boldsymbol{u}[e'] \mid \text{tuple } \boldsymbol{u} \in R(e, \eta)\}$. No communication occurs as each machine simply discards A° from every tuple $\boldsymbol{u} \in R(e, \eta)$ in the local storage.

Step 3. Cleanse G' into $G^* = (V', E^*)$. As explained in Section 2.2.1, this may or may not require calling algorithm cleanse. If called, cleanse identifies in each iteration two hyperedges e_{small} and e_{big} in the current G^* and removes e_{small}. Accordingly, we perform a *semi-join* between $R^*(e_{\text{small}}, \eta)$ and $R^*(e_{\text{big}}, \eta)$, which 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. Let Q^*_η be the query defined by the relation set $\{R^*(e^*, \eta) \mid e^* \in E^*\}$. Compute $Join(Q^*_\eta)$ using p_η machines recursively. Note that the number of participating attributes has decreased by 1 for the recursion.

Step 5. We output $Join(Q_\eta)$ by augmenting each tuple $\boldsymbol{u} \in Join(Q^*_\eta)$ with $\boldsymbol{u}(A^\circ) = \eta$. No communication is needed.

3.3 Solving Q_η When η is a Light Interval

Define Z, $G^*_z = (V^*_z, E^*_z)$ (for each $z \in Z$), C^*_z, $\bar{G}^* = (\bar{V}^*, \bar{E}^*)$, and $\bar{C}^*$ all in the way described in Section 2.2.2. We compute $Join(Q_\eta)$ in four steps.

Step 1. Same as Step 1 of the algorithm in Section 3.2.

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

Step 3. For each $z \in Z$, define a query $Q^*_{\eta,z} = \{R(e, \eta) \mid e \in E^*_z\}$. Similarly, for $\bar{G}^*$, define a query $\bar{Q}^*_\eta = \{R(e, \eta) \mid e \in \bar{E}^*\}$. Next, we compute the cartesian product of $Join(\bar{Q}^*_\eta)$ and the $Join(Q^*_{\eta,z})$ of all the $z \in Z$ – namely $\left(\times_{z \in Z} Join(Q^*_{\eta,z})\right) \times Join(\bar{Q}^*_\eta)$ – using p_η machines. Towards that purpose, define for each $z \in Z$

$$p_{\eta,z} = \Theta\left(1 + \max_{k=1}^{|F^*_z|} \frac{P_k(Q^*_{\eta,z}, C^*_z)}{L^k}\right) \quad (9)$$

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

$$\bar{p}_\eta = \Theta\left(1 + \max_{k=1}^{|\bar{F}^*|} \frac{P_k(\bar{Q}^*_\eta, \bar{C}^*)}{L^k}\right) \quad (10)$$

where $P_k(\bar{Q}^*_\eta, \bar{C}^*)$ is the max $(k, \bar{Q}^*_\eta)$-product of the clustering $\bar{C}^*$. We will prove later that each $Q^*_{\eta,z}$ can be answered with load $O(L)$ using $p_{\eta,z}$ machines, and $\bar{Q}^*_\eta$ can be answered with load $O(L)$ using $\bar{p}_\eta$ machines. Therefore, applying the cartesian product algorithm given in Lemma 6 of [12] (see also Lemma 4 of [13]), we can compute $\left(\times_{z\in Z} \mathit{Join}(Q^*_{\eta,z})\right) \times \mathit{Join}(\bar{Q}^*_\eta)$ with load $O(L)$ using $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z}$ machines. As proved later, we can adjust the constants in (9) and (10) to make sure $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} \le p_\eta$, where p_η is given in (8).

Step 4. We combine the cartesian product $\left(\times_{z\in Z} \mathit{Join}(Q^*_{\eta,z})\right) \times \mathit{Join}(\bar{Q}^*_\eta)$ with the tuples broadcast in Step 2 to derive $\mathit{Join}(Q_\eta)$ with no more communication. Specifically, for each tuple $\boldsymbol{u}$ in the cartesian product, the machine where $\boldsymbol{u}$ resides outputs $\{\boldsymbol{u}\} \bowtie \left(\bowtie_{e\in \mathrm{sigpath}(f^\circ,T)} R(e,\eta)\right)$. It is rudimentary to verify that all the tuples of $\mathit{Join}(Q_\eta)$ will be produced this way.

4 Analysis of the Algorithm

This section will establish:

▶ **Theorem 22.** *Consider any join query Q defined in Section 1.1 whose hypergraph is G. The algorithm of Section 3 answers Q with load $O(L)$, where L (given in (6)) is the Q-induced load of the clustering obtained from a canonical edge cover of G.*

We will prove the theorem by induction on the number of participating attributes (i.e., $|V|$) and the number of participating relations (i.e., $|Q|$). If $|Q| = 1$, the theorem trivially holds. If $|V| = 1$, Q has only one relation (because Q is clean) and the theorem also holds. Next, assuming that the theorem holds on any query with *either* strictly less participating attributes *or* strictly less participating relations than Q, we will prove the theorem's correctness on Q.

Our analysis will answer three questions. First, why do we have enough machines to handle all configurations in parallel? In particular, we must show that $\sum_\eta p_\eta \le p$, where p_η is given in (8). Second, why does each step in Section 3.2 and 3.3 entail a load of $O(L)$? Third, why do we have $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} \le p_\eta$ in Step 3 of Section 3.3? Settling these questions will complete the proof of Theorem 22.

All the notations in this section follow those in Section 3.

4.1 Total Number of Machines for All Configurations

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

$$\frac{1}{L^k} \sum_\eta P_k(Q_\eta, C) \quad = \quad O(p) \tag{11}$$

which will yield

$$\begin{aligned}
\sum_\eta p_\eta &= \sum_\eta O\left(1 + \max_{k=1}^{|F|} \frac{P_k(Q_\eta, C)}{L^k}\right) \\
&= \sum_\eta O\left(1 + \sum_{k=1}^{|F|} \frac{P_k(Q_\eta, C)}{L^k}\right) = O(p) + \sum_{k=1}^{|F|} O\left(\sum_\eta \frac{P_k(Q_\eta, C)}{L^k}\right) \\
&= O(p)
\end{aligned}$$

where the second equality used $|F| = O(1)$ and the third equality used $\sum_\eta 1 = O(p)$.[8]

[8] $\sum_\eta 1$ is the number of configurations which is $O(p)$ as shown in (7).

Henceforth, fix the value of k. For any η, the hypergraph of Q_η is always G (i.e., the hypergraph of Q). Consider an arbitrary k-group K of the clustering C (given in Equation 5). The Q_η-product of K is $\prod_{e\in K}|R(e,\eta)|$.[9] For any K, we will prove

$$\frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e,\eta)| \quad = \quad O(p). \tag{12}$$

As C has $O(1)$ k-groups K, the above yields

$$\begin{aligned}\sum_\eta\frac{P_k(Q_\eta,C)}{L^k} &= \sum_\eta\frac{1}{L^k}\max_K\prod_{e\in K}|R(e,\eta)| \\ &= O\left(\sum_\eta\frac{1}{L^k}\sum_K\prod_{e\in K}|R(e,\eta)|\right) = O\left(\sum_K\frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e,\eta)|\right) \\ &= \sum_K O(p) = O(p)\end{aligned}$$

as claimed in (11).

Let us first consider the case where $K\cap\text{sigpath}(f^\circ,T)\neq\emptyset$, namely, K has a hyperedge e_0 picked from the cluster $\text{sigpath}(f^\circ,T)$. We have:

$$\sum_\eta\prod_{e\in K}|R(e,\eta)| \quad = \quad \sum_\eta\Big(|R(e_0,\eta)|\cdot\prod_{e\in K\setminus\{e_0\}}|R(e,\eta)|\Big) \tag{13}$$

For each $e\in K\setminus\{e_0\}$, obviously $|R(e,\eta)|\le|R(e)|$. Regarding e_0, because A° must be an attribute of e_0, the relations $R(e_0,\eta)$ of all the configurations η form a *partition* of $R(e_0)$.[10] Hence:

$$\begin{aligned}(13) \quad &\le \quad \Big(\prod_{e\in K\setminus\{e_0\}}|R(e)|\Big)\Big(\sum_\eta|R(e_0,\eta)|\Big) = \Big(\prod_{e\in K\setminus\{e_0\}}|R(e)|\Big)\cdot|R(e_0)| = \prod_{e\in K}|R(e)| \\ &\le \quad \max\ (k,Q)\text{-product of } C.\end{aligned}$$

Therefore, the left hand side of (12) is bounded by $\frac{\max\ (k,Q)\text{-product of } C}{L^k}$, which is at most p (by definition of L).

Next, we consider $K\cap\text{sigpath}(f^\circ,T)=\emptyset$. In this case, we must have $k=|K|\le|F|-1$, because the hyperedges in K need to come from distinct clusters of C, and C has $|F|$ clusters (one of them is $\text{sigpath}(f^\circ,T)$, which now must be excluded). Applying the trivial fact $|R(e,\eta)|\le|R(e)|$ (for any e) and the fact that $\sum_\eta 1$ is bounded by (7), we have

$$\begin{aligned}\frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e,\eta)| \quad &\le \quad \frac{1}{L^k}\sum_\eta\prod_{e\in K}|R(e)| = O\left(\frac{1}{L^k}\prod_{e\in K}|R(e)|\cdot\max_{e\in\text{sigpath}(f^\circ,T)}\frac{|R(e)|}{L}\right) \\ &= \quad O\left(\frac{\max\ (k+1,Q)\text{-product of } C}{L^{k+1}}\right)\end{aligned}$$

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

---

9 For the definition of "a k-group's Q-product", review Section 1.3.
10 The $R(e_0,\eta)$ of all the η are mutually disjoint and their union equals $R(e_0)$.

4.2 Heavy Q_η

This subsection will prove that the algorithm in Section 3.2 has load $O(L)$. Step 2 and 5 demand no communication. The loads of Step 1 and 3 can all be bounded[11] by $O(\frac{1}{p_\eta}\sum_{e\in E}|R(e,\eta)|) = O(\frac{1}{p_\eta}\max_{e\in E}|R(e,\eta)|) = O(P_1(Q_\eta, C)/p_\eta) = O(L)$.

To analyze Step 4, let T^* be the hyperedge tree of G^* (produced by cleansing) and F^* be the CEC of G^*. By definition, the Q^*_η-induced load of the clustering $C^* = \{\text{sigpath}(f^*, T^*) \mid f^* \in F^*\}$ is

$$L^*_\eta \quad = \quad \max_{k=1}^{|F^*|}\left(\frac{P_k(Q^*_\eta, C^*)}{p_\eta}\right)^{1/k} \tag{14}$$

where $P_k(Q^*_\eta, C^*)$ is the max (k, Q^*_η)-product of C^*. By our inductive assumption (that Theorem 22 holds on Q^*_η), Step 4 incurs load $O(L^*_\eta)$. We will prove $P_k(Q^*_\eta, C^*) \le P_k(Q_\eta, C)$ for every k which, together with (8) and (14), will tell us $L^*_\eta = O(L)$.

Before proceeding, the reader should recall that, for any hyperedge e^* of G^*, $map^{-1}(e^*)$ gives a hyperedge in G. We must have $|R^*(e^*,\eta)| \le |R(map^{-1}(e^*),\eta)|$. 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 $P_k(Q^*_\eta, C^*) \le P_k(Q_\eta, C)$, consider any k-group K^* of C^*. By Lemma 15, $K = \{map^{-1}(e^*) \mid e^* \in K^*\}$ must be a k-group of C. Since $|R^*(e^*,\eta)| \le |R(map^{-1}(e^*),\eta)|$ for any $e^* \in K^*$, we have $\prod_{e^*\in K^*}|R^*(e^*,\eta)| \le \prod_{e\in K}|R(e,\eta)| \le P_k(Q_\eta, C)$. Therefore:

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

4.3 Light Q_η

This subsection will concentrate on the algorithm of Section 3.3.

Load. Step 1 incurs load $O(L)$ (same analysis as in Section 3.2). 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.

To analyze Step 3, let us first consider $\bar{Q}^*_\eta$. The $\bar{Q}^*_\eta$-induced load of the clustering $\bar{C}^*$ is

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

where $P_k(\bar{Q}^*_\eta, \bar{C}^*)$ as the max $(k, \bar{Q}^*_\eta)$-product of $\bar{C}^*$. By our inductive assumption (that Theorem 22 holds on $\bar{Q}^*_\eta$), answering $\bar{Q}^*_\eta$ with $\bar{p}_\eta$ machines requires load $O(\bar{L}^*_\eta)$, which is $O(L)$ given the $\bar{p}_\eta$ in (10). A similar argument shows that answering each $Q^*_{\eta,z}$ with $p_{\eta,z}$ machines – with $p_{\eta,z}$ given in (9) – incurs a load of $O(L)$. Thus, the cartesian product at Step 3 can be computed with load $O(L)$.

Number of machines in Step 3. Next, we will prove that $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} \le p_\eta$ always holds in Step 3. It suffices to show $\bar{p}_\eta \cdot \prod_{z\in Z} p_{\eta,z} = O(p_\eta)$ which, as we will see, relies on Lemma 20 and the fact that $|R(e,\eta)| \le L$ for every node $e \in \text{sigpath}(f^\circ, T)$.

[11] Step 3 performs $O(1)$ semi joins, each of which can be performed by sorting. For sorting in the MPC model, see Section 2.2.1 of [10]. The stated bound for Step 1 and 3 requires the assumption $p \le m^{1-\epsilon}$.

Consider an arbitrary $z \in Z$. The root of T_z^* – denoted as e_{root} – must belong to sigpath(f°, T). Recall that a k-group K of C_z^* takes a hyperedge from a distinct cluster in C_z^*. Call K a *non-root k-group* if $e_{root} \notin K$, or a *root k-group*, otherwise. Define

$$\begin{aligned} P_k(Q_{\eta,z}^*, C_z^*) &= \max\ (k, Q_{\eta,z}^*)\text{-product of } C_z^* \\ P_k^{non}(Q_{\eta,z}^*, C_z^*) &= \max\ (k, Q_{\eta,z}^*)\text{-product of all the } \textit{non-root } k\text{-groups of } C_z^*. \end{aligned}$$

As a special case, define $P_0^{non}(Q_{\eta,z}^*, C_z^*) = 1$. For any k, we observe

$$P_k(Q_{\eta,z}^*, C_z^*) \leq \max\{P_k^{non}(Q_{\eta,z}^*, C_z^*), L \cdot P_{k-1}^{non}(Q_{\eta,z}^*, C_z^*)\}. \tag{15}$$

To prove the inequality, fix K to the k-group with the largest $Q_{\eta,z}^*$-product ($= P_k(Q_{\eta,z}^*, C_z^*)$). If K is a non-root k-group, (15) obviously holds. Consider, instead, that K is a root k-group. Since $e_{root} \in$ sigpath(f°, 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 (15), we can now derive from (9):

$$\begin{aligned} p_{\eta,z} &= O\left(1 + \max_{k=1}^{|F_z^*|} \frac{\max\{P_k^{non}(Q_{\eta,z}^*, C_z^*), L \cdot P_{k-1}^{non}(Q_{\eta,z}^*, C_z^*)\}}{L^k}\right) \\ &= O\left(1 + \max_{k=1}^{|F_z^*|-1} \frac{P_k^{non}(Q_{\eta,z}^*, C_z^*)}{L^k}\right) \end{aligned} \tag{16}$$

where the second equality used the fact that, when $k = |F_z^*|$, a k-group must be a root k-group.

We are now ready to prove $\bar{p}_\eta \cdot \prod_{z \in Z} p_{\eta,z} = O(p_\eta)$. For each $z \in Z$, define integer k_z and a set K_z of hyperedges as follows:

- If (16) $= \Theta(P_k^{non}(Q_{\eta,z}^*, C_z^*)/L^k)$ for some $k \in [1, |F_z^*| - 1]$, set $k_z = k$ and K_z to the non-root k-group whose $Q_{\eta,z}^*$-product equals $P_k^{non}(Q_{\eta,z}^*, C_z^*)$.
- Otherwise (we must have $p_{\eta,z} = \Theta(1)$), set $k_z = 0$ and $K_z = \emptyset$; furthermore, define the $Q_{\eta,z}^*$-product of K_z to be 1.

Similarly, regarding $\bar{p}_\eta$ in (10), define integer $\bar{k}$ and a set $\bar{K}$ of hyperedges as follows:

- If (10) $= \Theta(P_k(\bar{Q}_\eta^*, \bar{C}^*)/L^k)$ for some $k \in [1, |\bar{F}^*|]$, set $\bar{k} = k$ and $\bar{K}$ to the k-group of the clustering $\bar{C}^*$ whose $\bar{Q}_\eta^*$-product equals $P_k(\bar{Q}_\eta^*, \bar{C}^*)$.
- Otherwise, set $\bar{k} = 0$ and $\bar{K} = \emptyset$; furthermore, define the $\bar{Q}_\eta^*$-product of $\bar{K}$ to be 1.

Define $K_{super} = \bar{K} \cup \left(\bigcup_{z \in Z} K_z\right)$. If $K_{super} = \emptyset$, then $p_{\eta,z} = \Theta(1)$ for all $z \in Z$ and $\bar{p}_\eta = \Theta(1)$, which leads to

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

If $K_{super} \neq \emptyset$, K_{super} is a super-$|K_{super}|$-group[12]. By Lemma 20, K_{super} is a $|K_{super}|$-group of T. We thus have:

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

This completes the whole proof of Theorem 22.

[12] For the definition of super-k-group, review Section 2.2.2.

References

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

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

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

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

5 Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *SIAM Journal of Computing*, 42(4):1737–1767, 2013.

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

7 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 303–318, 2017.

8 Xiao Hu. Cover or pack: New upper and lower bounds for massively parallel joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 181–198, 2021.

9 Xiao Hu and Ke Yi. Instance and output optimal parallel algorithms for acyclic joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 450–463, 2019.

10 Xiao Hu, Ke Yi, and Yufei Tao. Output-optimal massively parallel algorithms for similarity joins. *ACM Transactions on Database Systems (TODS)*, 44(2):6:1–6:36, 2019.

11 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 1259–1274. ACM, 2017.

12 Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 417–428, 2017.

13 Bas Ketsman, Dan Suciu, and Yufei Tao. A near-optimal parallel algorithm for joining binary relations. *CoRR*, abs/2011.14482, 2020.

14 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 213–228, 2015.

15 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-case optimal algorithms for parallel query processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 8:1–8:18, 2016.

16 Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 234–245, 2014.

17 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2012.

18 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018.

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

20 Miao Qiao and Yufei Tao. Two-attribute skew free, isolated CP theorem, and massively parallel joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 166–180, 2021.

21 Yufei Tao. A simple parallel algorithm for natural joins on binary relations. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 25:1–25:18, 2020.

22 Yufei Tao. Parallel acyclic joins with canonical edge covers. *CoRR*, abs/2201.03832, 2022.

23 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of Very Large Data Bases (VLDB)*, pages 82–94, 1981.