Logical Methods in Computer Science
Volume 18, Issue 2, 2022, pp. 6:1–6:22
https://lmcs.episciences.org/

Submitted Dec. 01, 2020
Published May 05, 2022

A NEAR-OPTIMAL PARALLEL ALGORITHM FOR JOINING BINARY RELATIONS

BAS KETSMAN, DAN SUCIU, AND YUFEI TAO

[a] Vrije Universiteit Brussel
e-mail address: bas.ketsman@vub.be

[b] University of Washington
e-mail address: suciu@cs.washington.edu

[c] Chinese University of Hong Kong, Hong Kong
e-mail address: taoyf@cse.cuhk.edu.hk

Abstract. We present a constant-round algorithm in the massively parallel computation (MPC) model for evaluating a natural join where every input relation has two attributes. Our algorithm achieves a load of $\tilde{O}(m/p^{1/\rho})$ where m is the total size of the input relations, p is the number of machines, ρ is the join's fractional edge covering number, and $\tilde{O}(.)$ hides a polylogarithmic factor. The load matches a known lower bound up to a polylogarithmic factor. At the core of the proposed algorithm is a new theorem (which we name *the isolated cartesian product theorem*) that provides fresh insight into the problem's mathematical structure. Our result implies that the *subgraph enumeration problem*, where the goal is to report all the occurrences of a constant-sized subgraph pattern, can be settled optimally (up to a polylogarithmic factor) in the MPC model.

1. Introduction

Understanding the hardness of joins has been a central topic in database theory. Traditional efforts have focused on discovering fast algorithms for processing joins in the *random access machine* (RAM) model (see [1, 5, 16–18, 21, 22] and the references therein). Nowadays, massively parallel systems such as Hadoop [8] and Spark [2] have become the mainstream architecture for analytical tasks on gigantic volumes of data. Direct adaptations of RAM algorithms, which are designed to reduce CPU time, rarely give satisfactory performance on that architecture. In systems like Hadoop and Spark, it is crucial to minimize communication across the participating machines because usually the overhead of message exchanging overwhelms the CPU calculation cost. This has motivated a line of research — which

Key words and phrases: Joins, Conjunctive Queries, Parallel Computing, Database Theory.

The research of Bas Ketsman was partially supported by FWO-grant G062721N. The research of Dan Suciu was partially supported by projects NSF IIS 1907997 and NSF-BSF 2109922. The research of Yufei Tao was partially supported by GRF projects 142034/21 and 142078/20 from HKRGC, and an AIR project from the Alibaba group.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-18(2:6)2022

includes this work — that aims to understand the communication complexities of join problems.

1.1. **Problem Definition.** We will first give a formal definition of the join operation studied in this paper and then elaborate on the computation model assumed.

Joins. Let **att** be a finite set where each element is called an *attribute*, and **dom** be a countably infinite set where each element is called a *value*. A *tuple* over a set $U \subseteq \mathbf{att}$ is a function $\boldsymbol{u} : U \to \mathbf{dom}$. Given a subset V of U, define $\boldsymbol{u}[V]$ as the tuple $\boldsymbol{v}$ over V such that $\boldsymbol{v}(X) = \boldsymbol{u}(X)$ for every $X \in V$. We say that $\boldsymbol{u}[V]$ is the *projection* of $\boldsymbol{u}$ on V.

A *relation* is a set R of tuples over the same set U of attributes. We say that the *scheme* of R is U, and write this fact as $scheme(R) = U$. R is *unary* or *binary* if $|scheme(R)| = 1$ or 2, respectively. A value $x \in \mathbf{dom}$ *appears* in R if there exist a tuple $\boldsymbol{u} \in R$ and an attribute $X \in U$ such that $\boldsymbol{u}(X) = x$; we will also use the expression that x is "a value on the attribute X in R".

A *join query* (sometimes abbreviated as a "join" or a "query") is a set $\mathcal{Q}$ of relations. Define $attset(\mathcal{Q}) = \bigcup_{R \in \mathcal{Q}} scheme(R)$. The *result* of the query, denoted as $Join(\mathcal{Q})$, is the following relation over $attset(\mathcal{Q})$

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

$\mathcal{Q}$ is

- *simple* if no distinct $R, S \in \mathcal{Q}$ satisfy $scheme(R) = scheme(S)$;
- *binary* if every $R \in \mathcal{Q}$ is binary.

Our objective is to design algorithms for answering simple binary queries.

The integer

$$m = \sum_{R \in \mathcal{Q}} |R| \tag{1.1}$$

is the *input size* of $\mathcal{Q}$. Concentrating on *data complexity*, we will assume that both $|\mathcal{Q}|$ and $|attset(\mathcal{Q})|$ are constants.

Computation Model. We will assume the *massively parallel computation* (MPC) model, which is a widely-accepted abstraction of today's massively parallel systems. Denote by p the number of machines. In the beginning, the input elements are evenly distributed across these machines. For a join query, this means that each machine stores $\Theta(m/p)$ tuples from the input relations (we consider that every value in **dom** can be encoded in a single word).

An algorithm is executed in *rounds*, each having two phases:

- In the first phase, every machine performs computation on the data of its local storage.
- In the second phase, the machines communicate by sending messages to each other.

All the messages sent out in the second phase must be prepared in the first phase. This prevents a machine from, for example, sending information based on the data received *during* the second phase. Another round is launched only if the current round has not solved the problem. In our context, *solving* a join query means that, for every tuple $\boldsymbol{u}$ in the join result,

at least one of the machines has $\boldsymbol{u}$ in the local storage; furthermore, no tuples outside the join result should be produced.

The *load* of a round is the largest number of words received by a machine in this round, that is, if machine $i \in [1, p]$ receives x_i words, the load is $\max_{i=1}^{p} x_i$. The performance of an algorithm is measured by two metrics: (i) the number of rounds, and (ii) the *load* of the algorithm, defined as the total load of all rounds. CPU computation is for free. We will be interested only in algorithms finishing in a constant number of rounds. The load of such an algorithm is asymptotically the same as the maximum load of the individual rounds.

The number p of machines is assumed to be significantly less than m, which in this paper means $p^3 \leq m$. For a randomized algorithm, when we say that its load is at most L, we mean that its load is bounded by L with probability at least $1 - 1/p^c$ where c can be set to an arbitrarily large constant. The notation $\tilde{O}(.)$ hides a factor that is polylogarithmic to m and p.

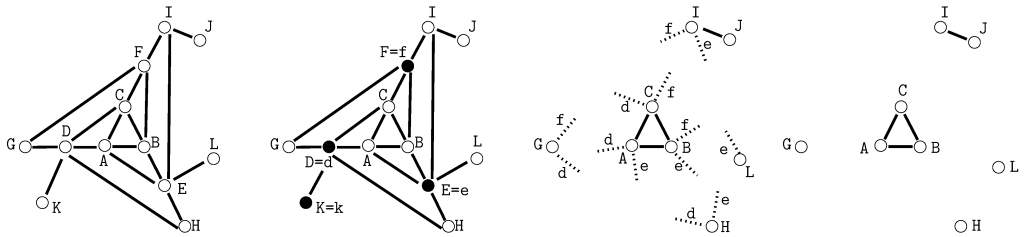
1.2. **Previous Results.** Early work on join processing in the MPC model aimed to design algorithms performing only one round. Afrati and Ullman [3] explained how to answer a query $\mathcal{Q}$ with load $O(m/p^{1/|\mathcal{Q}|})$. Later, by refining their prior work in [6], Koutris, Beame, and Suciu [13] described an algorithm that can guarantee a load of $\tilde{O}(m/p^{1/\psi})$, where ψ is the query's *fractional edge quasi-packing number*. To follow our discussion in Section 1, the reader does not need the formal definition of ψ (which will be given in Section 2); it suffices to understand that ψ is a positive constant which can vary significantly depending on $\mathcal{Q}$. In [13], the authors also proved that any one-round algorithm must incur a load of $\Omega(m/p^{1/\psi})$, under certain assumptions on the statistics available to the algorithm.

Departing from the one-round restriction, subsequent research has focused on algorithms performing multiple, albeit still a constant number of, rounds. The community already knows [13] that any constant-round algorithm must incur a load of $\Omega(m/p^{1/\rho})$ answering a query, where ρ is the query's *fractional edge covering number*. As far as Section 1 is concerned, the reader does not need to worry about the definition of ρ (which will appear in Section 2); it suffices to remember two facts:

- Like ψ, ρ is a positive constant which can vary significantly depending on the query $\mathcal{Q}$.
- On the same $\mathcal{Q}$, ρ never exceeds ψ, but can be much smaller than ψ (more details in Section 2).

The second bullet indicates that $m/p^{1/\rho}$ can be far less than $m/p^{1/\psi}$, suggesting that we may hope to significantly reduce the load by going beyond only one round. Matching the lower bound $\Omega(m/p^{1/\rho})$ with a concrete algorithm has been shown possible for several special query classes, including star joins [3], cycle joins [13], clique joins [13], line joins [3, 13], Loomis-Whitney joins [13], etc. The simple binary join defined in Section 1.1 captures cycle, clique, and line joins as special cases. Guaranteeing a load of $O(m/p^{1/\rho})$ for arbitrary simple binary queries is still open.

1.3. **Our Contributions.** The paper's main algorithmic contribution is to settle any simple binary join $\mathcal{Q}$ under the MPC model with load $\tilde{O}(m/p^{1/\rho})$ in a constant number rounds (Theorem 6.2). The load is optimal up to a polylogarithmic factor. Our algorithm owes to



(a) A join query (b) A residual query (c) After deleting black verticess (d) After semi-join reduction

Figure 1: Processing a join by constraining heavy values

a new theorem — we name the *isolated cartesian product theorem* (Theorem 5.1; see also Theorem 5.4) — that reveals a non-trivial fact on the problem's mathematical structure.

Overview of Our Techniques. Consider the join query $\mathcal{Q}$ illustrated by the graph in Figure 1a. An edge connecting vertices X and Y represents a relation $R_{\{X,Y\}}$ with scheme $\{X, Y\}$. $\mathcal{Q}$ contains all the 18 relations represented by the edges in Figure 1a; $attset(\mathcal{Q}) = \{\mathtt{A}, \mathtt{B}, ..., \mathtt{L}\}$ has a size of 12.

Set $\lambda = \Theta(p^{1/(2\rho)})$ where ρ is the fractional edge covering number of $\mathcal{Q}$ (Section 2). A value $x \in \mathbf{dom}$ is *heavy* if at least m/λ tuples in an input relation $R \in \mathcal{Q}$ carry x on the same attribute. The number of heavy values is $O(\lambda)$. A value $x \in \mathbf{dom}$ is *light* if x appears in at least one relation $R \in \mathcal{Q}$ but is not heavy. A tuple in the join result may take a heavy or light value on each of the 12 attributes $\mathtt{A}, ..., \mathtt{L}$. As there are $O(\lambda)$ choices on each attribute (i.e., either a light value or one of the $O(\lambda)$ heavy values), there are $t = O(\lambda^{12})$ "choice combinations" from all attributes; we will refer to each combination as a *configuration*. Our plan is to partition the set of p servers into t subsets of sizes $p_1, p_2, ..., p_t$ with $\sum_{i=1}^{t} p_i = p$, and then dedicate p_i servers ($1 \leq i \leq t$) to computing the result tuples of the i-th configuration. This can be done in parallel for all $O(\lambda^{12})$ configurations. The challenge is to compute the query on each configuration with a load $O(m/p^{1/\rho})$, given that only p_i (which can be far less than p) servers are available for that subtask.

Figure 1b illustrates one possible configuration where we constrain attributes `D`, `E`, `F`, and `K` respectively to heavy values `d`, `e`, `f`, and `k` and the other attributes to light values. Accordingly, vertices `D`, `E`, `F`, and `K` are colored black in the figure. The configuration gives rise to a *residual query* $\mathcal{Q}'$:

- For each edge $\{X, Y\}$ with two white vertices, $\mathcal{Q}'$ has a relation $R'_{\{X,Y\}}$ that contains only the tuples in $R_{\{X,Y\}} \in \mathcal{Q}$ using light values on both X and Y;
- For each edge $\{X, Y\}$ with a white vertex X and a black vertex Y, $\mathcal{Q}'$ has a relation $R'_{\{X,Y\}}$ that contains only the tuples in $R_{\{X,Y\}} \in \mathcal{Q}$ each using a light value on X and the constrained heavy value on Y;
- For each edge $\{X, Y\}$ with two black vertices, $\mathcal{Q}'$ has a relation $R'_{\{X,Y\}}$ with only one tuple that takes the constrained heavy values on X and Y, respectively.

For example, a tuple in $R'_{\{\mathtt{A},\mathtt{B}\}}$ must use light values on both `A` and `B`; a tuple in $R'_{\{\mathtt{D},\mathtt{G}\}}$ must use value `d` on `D` and a light value on `G`; $R'_{\{\mathtt{D},\mathtt{K}\}}$ has only a single tuple with values `d` and `k` on `D` and `K`, respectively. Finding all result tuples for $\mathcal{Q}$ under the designated configuration amounts to evaluating $\mathcal{Q}'$.

Since the black attributes have had their values fixed in the configuration, they can be deleted from the residual query, after which some relations in $\mathcal{Q}'$ become unary or even disappear. Relation $R'_{\{\mathtt{A},\mathtt{D}\}} \in \mathcal{Q}'$, for example, can be regarded as a unary relation over $\{\mathtt{A}\}$ where every tuple is "piggybacked" the value $\mathtt{d}$ on $\mathtt{D}$. Let us denote this unary relation as $R'_{\{\mathtt{A}\}|\mathtt{d}}$, which is illustrated in Figure 1c with a dotted edge extending from $\mathtt{A}$ and carrying the label $\mathtt{d}$. The deletion of $\mathtt{D}, \mathtt{E}$, $\mathtt{F}$, and $\mathtt{K}$ results in 13 unary relations (e.g., two of them are over $\{\mathtt{A}\}$: $R'_{\{\mathtt{A}\}|\mathtt{d}}$ and $R'_{\{\mathtt{A}\}|\mathtt{e}}$). Attributes $\mathtt{G}$, $\mathtt{H}$, and $\mathtt{L}$ become *isolated* because they are not connected to any other vertices by solid edges. Relations $R'_{\{\mathtt{A},\mathtt{B}\}}$, $R'_{\{\mathtt{A},\mathtt{C}\}}$, $R'_{\{\mathtt{B},\mathtt{C}\}}$, and $R'_{\{\mathtt{I},\mathtt{J}\}}$ remain binary, whereas $R'_{\{\mathtt{D},\mathtt{K}\}}$ has disappeared (more precisely, if $R_{\{\mathtt{D},\mathtt{K}\}}$ does not contain a tuple taking values $\mathtt{d}$ and $\mathtt{k}$ on $\mathtt{D}$ and $\mathtt{K}$ respectively, then $\mathcal{Q}'$ has an empty answer; otherwise, we proceed in the way explained next).

Our algorithm solves the residual query $\mathcal{Q}'$ of Figure 1c as follows:

(1) *Perform a semi-join reduction.* There are two steps. First, for every vertex X in Figure 1c, intersect all the unary relations over $\{X\}$ (if any) into a single list $R''_{\{X\}}$. For example, the two unary relations $R'_{\{\mathtt{A}\}|\mathtt{d}}$ and $R'_{\{\mathtt{A}\}|\mathtt{e}}$ of $\mathtt{A}$ are intersected to produce $R''_{\{\mathtt{A}\}}$; only the values in the intersection can appear in the join result. Second, for every non-isolated attribute X in Figure 1c, use $R''_{\{X\}}$ to shrink each binary relation $R'_{\{X,Y\}}$ (for all relevant Y) to eliminate tuples whose X-values are absent in $R''_{\{X\}}$. This reduces $R'_{\{X,Y\}}$ to a subset $R''_{\{X,Y\}}$. For example, every tuple in $R''_{\{\mathtt{A},\mathtt{B}\}}$ uses an $\mathtt{A}$-value from $R''_{\{\mathtt{A}\}}$ and a $\mathtt{B}$-value from $R''_{\{\mathtt{B}\}}$.

(2) *Compute a cartesian product.* The residual query $\mathcal{Q}'$ can now be further simplified into a join query $\mathcal{Q}''$ which includes (i) the relation $R''_{\{X\}}$ for every isolated attribute X, and (ii) the relation $R''_{\{X,Y\}}$ for every solid edge in Figure 1c. Figure 1d gives a neater view of $\mathcal{Q}''$; clearly, $Join(\mathcal{Q}'')$ is the cartesian product of $R''_{\{\mathtt{G}\}}$, $R''_{\{\mathtt{H}\}}$, $R''_{\{\mathtt{L}\}}$, $R''_{\{\mathtt{I},\mathtt{J}\}}$, and the result of the "triangle join" $\{R''_{\{\mathtt{A},\mathtt{B}\}}, R''_{\{\mathtt{A},\mathtt{C}\}}, R''_{\{\mathtt{B},\mathtt{C}\}}\}$.

As mentioned earlier, we plan to use only a small subset of the p servers to compute $\mathcal{Q}'$. It turns out that the load of our strategy depends heavily on the cartesian product of the *unary* relations $R''_{\{X\}}$ (one for every isolated attribute X, i.e., $R''_{\{\mathtt{G}\}}$, $R''_{\{\mathtt{H}\}}$, and $R''_{\{\mathtt{L}\}}$ in our example) in a configuration. Ideally, if the cartesian product of *every* configuration is small, we can prove a load of $\tilde{O}(m/p^{1/\rho})$ easily. Unfortunately, this is not true: in the worst case, the cartesian products of various configurations can differ dramatically.

Our isolated cartesian product theorem (Theorem 5.1) shows that the cartesian product size is small when *averaged over all the possible configurations*. This property allows us to allocate a different number of machines to process each configuration in parallel while ensuring that the total number of machines required will not exceed p. The theorem is of independent interest and may be useful for developing join algorithms under other computation models (e.g., the external memory model [4]; see Section 7).

1.4. **An Application: Subgraph Enumeration.** The joins studied in this paper bear close relevance to the *subgraph enumeration problem*, where the goal is to find all occurrences of a pattern subgraph $G' = (V', E')$ in a graph $G = (V, E)$. This problem is NP-hard [7] if the size of G' is unconstrained, but is polynomial-time solvable when G' has only a constant

symbol	meaning	definition
p	number of machines	Sec 1.1
$\mathcal{Q}$	join query	Sec 1.1
m	input size of $\mathcal{Q}$	(1.1)
$Join(\mathcal{Q})$	result of $\mathcal{Q}$	Sec 1.1
$attset(\mathcal{Q})$	set of attributes in the relations of $\mathcal{Q}$	Sec 1.1
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	hypergraph of $\mathcal{Q}$	Sec 2
W	fractional edge covering/packing of $\mathcal{G}$	Sec 2
$W(e)$	weight of an edge $e \in \mathcal{E}$	Sec 2
ρ (or τ)	fractional edge covering (or packing) number of $\mathcal{G}$	Sec 2
R_e ($e \in \mathcal{E}$)	relation $R \in \mathcal{Q}$ with $scheme(R) = e$	Sec 2
λ	heavy parameter	Sec 4
$\mathcal{H}$	set of heavy attributes in $attset(\mathcal{Q})$	Sec 4
$config(\mathcal{Q}, \mathcal{H})$	set of configurations of $\mathcal{H}$	Sec 4
$\boldsymbol{\eta}$	configuration	Sec 4
$R'_e(\boldsymbol{\eta})$	residual relation of $e \in \mathcal{E}$ under $\boldsymbol{\eta}$	Sec 4
$\mathcal{Q}'(\boldsymbol{\eta})$	residual query under $\boldsymbol{\eta}$	(4.1)
k	size of $attset(\mathcal{Q})$	Lemma 4.1
$m_{\boldsymbol{\eta}}$	input size of $\mathcal{Q}'(\boldsymbol{\eta})$	Lemma 4.1
$\mathcal{L}$	set of light attributes in $attset(\mathcal{Q})$	(5.2)
$\mathcal{I}$	set of isolated attributes in $attset(\mathcal{Q})$	(5.3)
$R''_X(\boldsymbol{\eta})$	relation on attribute X after semi-join reduction	(5.4)
$R''_e(\boldsymbol{\eta})$	relation on $e \in \mathcal{E}$ after semi-join reduction	Sec 5.2
$\mathcal{Q}''_{isolated}(\boldsymbol{\eta})$	query on the isolated attributes after semi-join reduction	(5.5)
$\mathcal{Q}''_{light}(\boldsymbol{\eta})$	query on the light edges after semi-join reduction	(5.6)
$\mathcal{Q}''(\boldsymbol{\eta})$	reduced query under $\boldsymbol{\eta}$	(5.7)
$W_{\mathcal{I}}$	total weight of all vertices in $\mathcal{I}$ under fractional edge packing W	(5.10)
$\mathcal{J}$	non-empty subset of $\mathcal{I}$	Sec 5.4
$\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta})$	query on the isolated attributes in $\mathcal{J}$ after semi-join reduction	(5.14)
$W_{\mathcal{J}}$	total weight of all vertices in $\mathcal{J}$ under fractional edge packing W	(5.15)

Table 1: Frequently used notations

number of vertices. In the MPC model, the edges of G are evenly distributed onto the p machines at the beginning, whereas an algorithm must produce every occurrence on at least one machine in the end. The following facts are folklore regarding a constant-size G':

- Every constant-round subgraph enumeration algorithm must incur a load of $\Omega(|E|/p^{1/\rho})$,[1] where ρ is the fractional edge covering number (Section 2) of G'.
- The subgraph enumeration problem can be converted to a simple binary join with input size $O(|E|)$ and the same fractional edge covering number ρ.

Given a constant-size G', our join algorithm (Theorem 6.2) solves subgraph enumeration with load $\tilde{O}(|E|/p^{1/\rho})$, which is optimal up to a polylogarithmic factor.

1.5. **Remarks.** This paper is an extension of [12] and [20]. Ketsman and Suciu [12] were the first to discover a constant-round algorithm to solve simple binary joins with an asymptotically optimal load. Tao [20] introduced a preliminary version of the isolated

[1]Here, we consider $|E| \geq |V|$ because vertices with no edges can be discarded directly.

cartesian product theorem and applied it to simplify the algorithm of [12]. The current work features a more powerful version of the isolated cartesian product theorem (see the remark in Section 5.5). Table 1 lists the symbols that will be frequently used.

2. Hypergraphs and the AGM Bound

We define a *hypergraph* $\mathcal{G}$ as a pair $(\mathcal{V}, \mathcal{E})$ where:

- $\mathcal{V}$ is a finite set, where each element is called a *vertex*;
- $\mathcal{E}$ is a set of subsets of $\mathcal{V}$, where each subset is called a *(hyper-)edge*.

An edge e is *unary* or *binary* if $|e| = 1$ or 2, respectively. $\mathcal{G}$ is *binary* if all its edges are binary.

Given a vertex $X \in \mathcal{V}$ and an edge $e \in \mathcal{E}$, we say that X and e are *incident* to each other if $X \in e$. Two distinct vertices $X, Y \in \mathcal{V}$ are *adjacent* if there is an $e \in \mathcal{E}$ containing X and Y. All hypergraphs discussed in this paper have the property that every vertex is incident to at least one edge.

Given a subset $\mathcal{U}$ of $\mathcal{V}$, we define the subgraph *induced* by $\mathcal{U}$ as $(\mathcal{U}, \mathcal{E}_{\mathcal{U}})$ where $\mathcal{E}_{\mathcal{U}} = \{\mathcal{U} \cap e \mid e \in \mathcal{E}\}$.

Fractional Edge Coverings and Packings. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a hypergraph and W be a function mapping $\mathcal{E}$ to real values in $[0, 1]$. We call $W(e)$ the *weight* of edge e and $\sum_{e \in \mathcal{E}} W(e)$ the *total weight* of W. Given a vertex $X \in \mathcal{V}$, we refer to $\sum_{e \in \mathcal{E}: X \in e} W(e)$ (i.e., the sum of the weights of all the edges incident to X) as the *weight* of X.

W is a *fractional edge covering* of $\mathcal{G}$ if the weight of every vertex $X \in \mathcal{V}$ is at least 1. The *fractional edge covering number* of $\mathcal{G}$ — denoted as $\rho(\mathcal{G})$ — equals the smallest total weight of all the fractional edge coverings. W is a *fractional edge packing* if the weight of every vertex $X \in \mathcal{V}$ is at most 1. The *fractional edge packing number* of $\mathcal{G}$ — denoted as $\tau(\mathcal{G})$ — equals the largest total weight of all the fractional edge packings. A fractional edge packing W is *tight* if it is simultaneously also a fractional edge covering; likewise, a fractional edge covering W is *tight* if it is simultaneously also a fractional edge packing. Note that in a tight fractional edge covering/packing, the weight of every vertex must be exactly 1.

Binary hypergraphs have several interesting properties:

Lemma 2.1. *If $\mathcal{G}$ is binary, then:*

- *$\rho(\mathcal{G}) + \tau(\mathcal{G}) = |\mathcal{V}|$; furthermore, $\rho(\mathcal{G}) \geq \tau(\mathcal{G})$, where the equality holds if and only if $\mathcal{G}$ admits a tight fractional edge packing (a.k.a. tight fractional edge covering).*
- *$\mathcal{G}$ admits a fractional edge packing W of total weight $\tau(\mathcal{G})$ such that*
  (1) *the weight of every vertex $X \in \mathcal{V}$ is either 0 or 1;*
  (2) *if $\mathcal{Z}$ is the set of vertices in $\mathcal{V}$ with weight 0, then $\rho(\mathcal{G}) - \tau(\mathcal{G}) = |\mathcal{Z}|$.*

Proof. The first bullet is proved in Theorem 2.2.7 of [19]. The fractional edge packing W in Theorem 2.1.5 of [19] satisfies Property (1) of the second bullet. Regarding such a W, we have

$$\tau(\mathcal{G}) = \text{total weight of } W = \frac{1}{2} \sum_{X \in \mathcal{V}} (\text{weight of } X) = (|\mathcal{V}| - |\mathcal{Z}|)/2.$$

Plugging this into $\rho(\mathcal{G}) + \tau(\mathcal{G}) = |\mathcal{V}|$ yields $\rho(\mathcal{G}) = (|\mathcal{V}| + |\mathcal{Z}|)/2$. Hence, Property (2) follows. □

Example. Suppose that $\mathcal{G}$ is the binary hypergraph in Figure 1a. It has a fractional edge covering number $\rho(\mathcal{G}) = 6.5$, as is achieved by the function W_1 that maps {G, F}, {D, K}, {I, J}, {E, H}, and {E, L} to 1, {A, B}, {A, C}, and {B, C} to 1/2, and the other edges to 0. Its fractional edge packing number is $\tau(\mathcal{G}) = 5.5$, achieved by the function W_2 which is the same as W_1 except that W_2 maps {E, L} to 0. Note that W_2 satisfies both properties of the second bullet (here $\mathcal{Z} = \{\mathtt{L}\}$). □

Hypergraph of a Join Query and the AGM Bound. Every join $\mathcal{Q}$ defines a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = attset(\mathcal{Q})$ and $\mathcal{E} = \{scheme(R) \mid R \in \mathcal{Q}\}$. When $\mathcal{Q}$ is simple, for each edge $e \in \mathcal{E}$ we denote by R_e the input relation $R \in \mathcal{Q}$ with $e = scheme(R)$. The following result is known as the *AGM bound*:

Lemma 2.2 [5]**.** *Let $\mathcal{Q}$ be a simple binary join and W be any fractional edge covering of the hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ defined by $\mathcal{Q}$. Then, $|Join(\mathcal{Q})| \leq \prod_{e \in \mathcal{E}} |R_e|^{W(e)}$.*

The fractional edge covering number of $\mathcal{Q}$ equals $\rho(\mathcal{G})$ and, similarly, the fractional edge packing number of $\mathcal{Q}$ equals $\tau(\mathcal{G})$.

Remark on the Fractional Edge Quasi-Packing Number. Although the technical development in the subsequent sections is irrelevant to "fractional edge quasi-packing number", we provide a full definition of the concept here because it enables the reader to better distinguish our solution and the one-round algorithm of [13] (reviewed in Section 1.2). Consider a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For each subset $\mathcal{U} \subseteq \mathcal{V}$, let $\mathcal{G}_{\backslash \mathcal{U}}$ be the graph obtained by removing $\mathcal{U}$ from all the edges of $\mathcal{E}$, or formally: $\mathcal{G}_{\backslash \mathcal{U}} = (\mathcal{V} \setminus \mathcal{U}, \mathcal{E}_{\backslash \mathcal{U}})$ where $\mathcal{E}_{\backslash \mathcal{U}} = \{e \setminus \mathcal{U} \mid e \in \mathcal{E} \text{ and } e \setminus \mathcal{U} \neq \emptyset\}$. The *fractional edge quasi-packing number* of $\mathcal{G}$ — denoted as $\psi(\mathcal{G})$ — is

$$\psi(\mathcal{G}) \quad = \quad \max_{\text{all } \mathcal{U} \subseteq \mathcal{V}} \tau(\mathcal{G}_{\backslash \mathcal{U}})$$

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

In [13], Koutris, Beame, and Suciu proved that $\psi(\mathcal{G}) \geq \rho(\mathcal{G})$ holds on any $\mathcal{G}$ (which need not be binary). In general, $\psi(\mathcal{G})$ can be considerably higher than $\rho(\mathcal{G})$. In fact, this is true even on "regular" binary graphs, about which we mention two examples (both can be found in [13]):

- when $\mathcal{G}$ is a clique, $\psi(\mathcal{G}) = |\mathcal{V}| - 1$ but $\rho(\mathcal{G})$ is only $|\mathcal{V}|/2$;
- when $\mathcal{G}$ is a cycle, $\psi(\mathcal{G}) = \lceil 2(|\mathcal{V}| - 1)/3 \rceil$ and $\rho(\mathcal{G})$ is again $|\mathcal{V}|/2$.

If $\mathcal{G}$ is the hypergraph defined by a query $\mathcal{Q}$, $\psi(\mathcal{G})$ is said to be the query's fractional edge covering number. It is evident from the above discussion that, when $\mathcal{G}$ is a clique or a cycle, the load $\tilde{O}(m/p^{1/\rho(\mathcal{G})})$ of our algorithm improves the load $\tilde{O}(m/p^{1/\psi(\mathcal{G})})$ of [13] by a polynomial factor.

3. Fundamental MPC Algorithms

This subsection will discuss several building-block routines in the MPC model useful later.

Cartesian Products. Suppose that R and S are relations with disjoint schemes. Their *cartesian product*, denoted as $R \times S$, is a relation over $scheme(R) \cup scheme(S)$ that consists of all the tuples $\boldsymbol{u}$ over $scheme(R) \cup scheme(S)$ such that $\boldsymbol{u}[scheme(R)] \in R$ and $\boldsymbol{u}[scheme(S)] \in S$.

The lemma below gives a deterministic algorithm for computing the cartesian product:

Lemma 3.1. *Let $\mathcal{Q}$ be a set of $t = O(1)$ relations $R_1, R_2, ..., R_t$ with disjoint schemes. The tuples in R_i ($1 \leq i \leq t$) have been labeled with ids 1, 2, ..., $|R_i|$, respectively. We can deterministically compute $Join(\mathcal{Q}) = R_1 \times R_2 \times ... \times R_t$ in one round with load*

$$O\left(\max_{\text{non-empty } \mathcal{Q}' \subseteq \mathcal{Q}} \frac{|Join(\mathcal{Q}')|^{\frac{1}{|\mathcal{Q}'|}}}{p^{\frac{1}{|\mathcal{Q}'|}}}\right) \tag{3.1}$$

using p machines. Alternatively, if we assume $|R_1| \geq |R_2| \geq ... \geq |R_t|$, then the load can be written as

$$O\left(\max_{i=1}^{t} \frac{|Join(\{R_1, R_2, ..., R_i\})|^{\frac{1}{i}}}{p^{\frac{1}{i}}}\right). \tag{3.2}$$

In (3.1) and (3.2), the constant factors in the big-O depend on t.

Proof. For each $i \in [1, t]$, define $\mathcal{Q}_i = \{R_1, ..., R_i\}$ and $L_i = |Join(\mathcal{Q}_i)|^{\frac{1}{i}}/p^{\frac{1}{i}}$. Let t' be the largest integer satisfying $|R_i| \geq L_i$ for all $i \in [1, t']$; t' definitely exists because $|R_1| \geq L_1 = |R_1|/p$. Note that this means $|R_t| \leq |R_{t-1}| \leq ... \leq |R_{t'+1}| < L_{t'+1}$ if $t' < t$.

Next, we will explain how to obtain $Join(\mathcal{Q}_{t'})$ with load $O(L_{t'})$. If $t' < t$, this implies that $Join(\mathcal{Q})$ can be obtained with load $O(L_{t'} + L_{t'+1})$ because $R_{t'+1}, ..., R_t$ can be broadcast to all the machines with an extra load $O(L_{t'+1} \cdot (t - t')) = O(L_{t'+1})$.

Align the machines into a t'-dimensional $p_1 \times p_2 \times ... \times p_{t'}$ grid where

$$p_i = \lfloor |R_i|/L_{t'} \rfloor$$

for each $i \in [1, t']$. This is possible because $|R_i| \geq |R_{t'}| \geq L_{t'}$ and $\prod_{i=1}^{t'} \frac{|R_i|}{L_{t'}} = \frac{|Join(\mathcal{Q}_{t'})|}{(L_{t'})^{t'}} = p$. Each machine can be uniquely identified as a t'-dimensional point $(x_1, ..., x_{t'})$ in the grid where $x_i \in [1, p_i]$ for each $i \in [1, t']$. For each R_i, we send its tuple with id $j \in [1, |R_i|]$ to all the machines whose coordinates on dimension i are $(j \mod p_i) + 1$. Hence, a machine receives $O(|R_i|/p_i) = O(L_{t'})$ tuples from R_i; and the overall load is $O(L_{t'} \cdot t') = O(L_{t'})$. For each combination of $\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_{t'}$ where $\boldsymbol{u}_i \in R_i$, some machine has received all of $\boldsymbol{u}_1, ..., \boldsymbol{u}_{t'}$. Therefore, the algorithm is able to produce the entire $Join(\mathcal{Q}_{t'})$. □

The load in (3.2) matches a lower bound stated in Section 4.1.5 of [14]. The algorithm in the above proof generalizes an algorithm in [10] for computing the cartesian product of $t = 2$ relations. The randomized hypercube algorithm of [6] incurs a load higher than (3.2) by a logarithmic factor and can fail with a small probability.

Composition by Cartesian Product. If we already know how to solve queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ separately, we can compute the cartesian product of their results efficiently:

Lemma 3.2. *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two join queries satisfying the condition $attset(\mathcal{Q}_1) \cap attset(\mathcal{Q}_2) = \emptyset$. Let m be the total number of tuples in the input relations of $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Suppose that*

- *with probability at least* $1-\delta_1$*, we can compute in one round* $Join(\mathcal{Q}_1)$ *with load* $\tilde{O}(m/p_1^{1/t_1})$ *using* p_1 *machines;*
- *with probability at least* $1-\delta_2$*, we can compute in one round* $Join(\mathcal{Q}_2)$ *with load* $\tilde{O}(m/p_2^{1/t_2})$ *using* p_2 *machines.*

Then, with probability at least $1-\delta_1-\delta_2$*, we can compute* $Join(\mathcal{Q}_1) \times Join(\mathcal{Q}_2)$ *in one round with load* $\tilde{O}(\max\{m/p_1^{1/t_1}, m/p_2^{1/t_2}\})$ *using* p_1p_2 *machines.*

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the algorithm for $\mathcal{Q}_1$ and $\mathcal{Q}_2$, respectively. If a tuple $\boldsymbol{u} \in Join(\mathcal{Q}_1)$ is produced by $\mathcal{A}_1$ on the i-th ($i \in [1, p_1]$) machine, we call $\boldsymbol{u}$ an *i-tuple*. Similarly, if a tuple $\boldsymbol{v} \in Join(\mathcal{Q}_2)$ is produced by $\mathcal{A}_2$ on the j-th ($j \in [1, p_2]$) machine, we call $\boldsymbol{v}$ a *j-tuple*.

Arrange the p_1p_2 machines into a matrix where each row has p_1 machines and each column has p_2 machines (note that the number of rows is p_2 while the number of columns is p_1). For each row, we run A_1 using the p_1 machines on that row to compute $Join(\mathcal{Q}_1)$; this creates p_2 instances of A_1 (one per row). If A_1 is randomized, we instruct all those instances to take the same random choices.[2] This ensures:

- with probability at least $1-\delta_1$, all the instances succeed simultaneously;
- for each $i \in [1, p_1]$, all the machines at the i-th column produce exactly the same set of i-tuples.

The load incurred is $\tilde{O}(m/p_1^{1/t_1})$. Likewise, for each column, we run A_2 using the p_2 machines on that column to compute $Join(\mathcal{Q}_2)$. With probability at least $1-\delta_2$, for each $j \in [1, p_2]$, all the machines at the j-th row produce exactly the same set of j-tuples. The load is $\tilde{O}(m/p_2^{1/t_2})$.

Therefore, it holds with probability at least $1-\delta_1-\delta_2$ that, for each pair (i, j), some machine has produced all the i- and j-tuples. Hence, every tuple of $Join(\mathcal{Q}_1) \times Join(\mathcal{Q}_2)$ appears on a machine. The overall load is the larger between $\tilde{O}(m/p_1^{1/t_1})$ and $\tilde{O}(m/p_2^{1/t_2})$. □

Skew-Free Queries. It is possible to solve a join query $\mathcal{Q}$ on binary relations in a single round with a small load if no value appears too often. To explain, denote by m the input size of $\mathcal{Q}$; set $k = |attset(\mathcal{Q})|$, and list out the attributes in $attset(\mathcal{Q})$ as $X_1, ..., X_k$. For $i \in [1, k]$, let p_i be a positive integer referred to as the *share* of X_i. A relation $R \in \mathcal{Q}$ with scheme $\{X_i, X_j\}$ is *skew-free* if every value $x \in \mathbf{dom}$ fulfills both conditions below:

- R has $O(m/p_i)$ tuples $\boldsymbol{u}$ with $\boldsymbol{u}(X_i) = x$;
- R has $O(m/p_j)$ tuples $\boldsymbol{u}$ with $\boldsymbol{u}(X_j) = x$.

Define $share(R) = p_i \cdot p_j$. If every $R \in \mathcal{Q}$ is skew-free, $\mathcal{Q}$ is *skew-free*. We know:

[2]The random choices of an algorithm can be modeled as a sequence of random bits. Once the sequence is fixed, a randomized algorithm becomes deterministic. An easy way to "instruct" all instances of A_1 to make the same random choices is to ask all the participating machines to pre-agree on the random-bit sequence. For example, one machine can generate all the random bits and send them to the other machines. Such communication happens *before* receiving $\mathcal{Q}$ and hence does not contribute to the query's load. The above approach works for a single $\mathcal{Q}$ (which suffices for proving Lemma 3.2). There is a standard technique [15] to extend the approach to work for any number of queries. The main idea is to have the machines pre-agree on a sufficiently large number of random-bit sequences. Given a query, a machine randomly picks a specific random-bit sequence and broadcasts the sequence's id (note: only the id, not the sequence itself) to all machines. As shown in [15], such an id can be encoded in $\tilde{O}(1)$ words. Broadcasting can be done in constant rounds with load $O(p^\epsilon)$ for an arbitrarily small constant $\epsilon > 0$.

Lemma 3.3 [6]. *With probability at least* $1 - 1/p^c$ *where* $p = \prod_{i=1}^{k} p_i$ *and* $c \geq 1$ *can be set to an arbitrarily large constant, a skew-free query* $\mathcal{Q}$ *with input size* m *can be answered in one round with load* $\tilde{O}(m/\min_{R\in\mathcal{Q}} share(R))$ *using* p *machines.*

4. A Taxonomy of the Join Result

Given a simple binary join $\mathcal{Q}$, we will present a method to partition $Join(\mathcal{Q})$ based on the value frequencies in the relations of $\mathcal{Q}$. Denote by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the hypergraph defined by $\mathcal{Q}$ and by m the input size of $\mathcal{Q}$.

Heavy and Light Values. Fix an arbitrary integer $\lambda \in [1, m]$. A value $x \in \mathbf{dom}$ is

- *heavy* if $|\{\boldsymbol{u} \in R \mid \boldsymbol{u}(X) = x\}| \geq m/\lambda$ for some relation $R \in \mathcal{Q}$ and some attribute $X \in scheme(R)$;
- *light* if x is not heavy, but appears in at least one relation $R \in \mathcal{Q}$.

It is easy to see that each attribute has at most λ heavy values. Hence, the total number of heavy values is at most $\lambda \cdot |attset(\mathcal{Q})| = O(\lambda)$. We will refer to λ as the *heavy parameter*.

Configurations. Let $\mathcal{H}$ be an arbitrary (possibly empty) subset of $attset(\mathcal{Q})$. A *configuration* of $\mathcal{H}$ is a tuple $\boldsymbol{\eta}$ over $\mathcal{H}$ such that $\boldsymbol{\eta}(X)$ is heavy for every $X \in \mathcal{H}$. Let $config(\mathcal{Q}, \mathcal{H})$ be the set of all configurations of $\mathcal{H}$. It is clear that $|config(\mathcal{Q}, \mathcal{H})| = O(\lambda^{|\mathcal{H}|})$.

Residual Relations/Queries. Consider an edge $e \in \mathcal{E}$; define $e' = e \setminus \mathcal{H}$. We say that e is *active* on $\mathcal{H}$ if $e' \neq \emptyset$, i.e., e has at least one attribute outside $\mathcal{H}$. An active e defines a *residual relation* under $\boldsymbol{\eta}$ — denoted as $R'_e(\boldsymbol{\eta})$ — which

- is over e' and
- consists of every tuple $\boldsymbol{v}$ that is the projection (on e') of some tuple $\boldsymbol{w} \in R_e$ "consistent" with $\boldsymbol{\eta}$, namely:
  - $\boldsymbol{w}(X) = \boldsymbol{\eta}(X)$ for every $X \in e \cap \mathcal{H}$;
  - $\boldsymbol{w}(Y)$ is light for every $Y \in e'$;
  - $\boldsymbol{v} = \boldsymbol{w}[e']$.

The *residual query* under $\boldsymbol{\eta}$ is

$$\mathcal{Q}'(\boldsymbol{\eta}) = \left\{ R'_e(\boldsymbol{\eta}) \;\middle|\; e \in \mathcal{E},\ e \text{ active on } \mathcal{H} \right\}. \tag{4.1}$$

Note that if $\mathcal{H} = attset(\mathcal{Q})$, $\mathcal{Q}'(\boldsymbol{\eta})$ is empty.

Example. Consider the query $\mathcal{Q}$ in Section 1.3 (hypergraph $\mathcal{G}$ in Figure 1a) and the configuration $\boldsymbol{\eta}$ of $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$ where $\boldsymbol{\eta}[\mathtt{D}] = \mathtt{d}$, $\boldsymbol{\eta}[\mathtt{E}] = \mathtt{e}$, $\boldsymbol{\eta}[\mathtt{F}] = \mathtt{f}$, and $\boldsymbol{\eta}[\mathtt{K}] = \mathtt{k}$. If e is the edge $\{\mathtt{A}, \mathtt{D}\}$, then $e' = \{\mathtt{A}\}$ and $R'_e(\boldsymbol{\eta})$ is the relation $R'_{\{\mathtt{A}\}|\mathtt{d}}$ mentioned in Section 1.3. If e is the edge $\{\mathtt{A}, \mathtt{B}\}$, then $e' = \{\mathtt{A}, \mathtt{B}\}$ and $R'_e(\boldsymbol{\eta})$ is the relation $R'_{\{\mathtt{A},\mathtt{B}\}}$ in Section 1.3. The residual query $\mathcal{Q}'(\boldsymbol{\eta})$ is precisely the query $\mathcal{Q}'$ in Section 1.3. □

It is rudimentary to verify

$$Join(\mathcal{Q}) = \bigcup_{\mathcal{H}} \Big(\bigcup_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} Join(\mathcal{Q}'(\boldsymbol{\eta})) \times \{\boldsymbol{\eta}\} \Big). \tag{4.2}$$

Lemma 4.1. *Let $\mathcal{Q}$ be a simple binary join with input size m and $\mathcal{H}$ be a subset of $attset(\mathcal{Q})$. For each configuration $\boldsymbol{\eta} \in config(\mathcal{Q}, \mathcal{H})$, denote by $m_{\boldsymbol{\eta}}$ the total size of all the relations in $\mathcal{Q}'(\boldsymbol{\eta})$. We have:*

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} m_{\boldsymbol{\eta}} \quad \leq \quad m \cdot \lambda^{k-2}$$

where $k = |attset(\mathcal{Q})|$.

Proof. Let e be an edge in $\mathcal{E}$ and fix an arbitrary tuple $\boldsymbol{u} \in R_e$. Tuple $\boldsymbol{u}$ contributes 1 to the term $m_{\boldsymbol{\eta}}$ only if $\boldsymbol{\eta}(X) = \boldsymbol{u}(X)$ for every attribute $X \in e \cap \mathcal{H}$. How many such configurations $\boldsymbol{\eta}$ can there be? As these configurations must have the same value on every attribute in $e \cap \mathcal{H}$, they can differ only in the attributes of $\mathcal{H} \setminus e$. Since each attribute has at most λ heavy values, we conclude that the number of those configurations η is at most $\lambda^{|\mathcal{H} \setminus e|}$. $|\mathcal{H} \setminus e|$ is at most $k - 2$ because $|\mathcal{H}| \leq k$ and e has two attributes. The lemma thus follows. □

5. A Join Computation Framework

Answering a simple binary join $\mathcal{Q}$ amounts to producing the right-hand side of (4.2). Due to symmetry, it suffices to explain how to do so for an arbitrary subset $\mathcal{H} \subseteq attset(\mathcal{Q})$, i.e., the computation of

$$\bigcup_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} Join(\mathcal{Q}'(\boldsymbol{\eta})). \tag{5.1}$$

At a high level, our strategy (illustrated in Section 1.3) works as follows. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the hypergraph defined by $\mathcal{Q}$. We will remove the vertices in $\mathcal{H}$ from $\mathcal{G}$, which disconnects $\mathcal{G}$ into connected components (CCs). We divide the CCs into two groups: (i) the set of CCs each involving at least 2 vertices, and (ii) the set of all other CCs, namely those containing only 1 vertex. We will process the CCs in Group 1 *together* using Lemma 3.3, process the CCs in Group 2 *together* using Lemma 3.1, and then compute the cartesian product between Groups 1 and 2 using Lemma 3.2.

Sections 5.1 and 5.2 will formalize the strategy into a processing framework. Sections 5.3 and 5.4 will then establish two important properties of this framework, which are the key to its efficient implementation in Section 6.

5.1. Removing the Attributes in $\mathcal{H}$.

We will refer to each attribute in $\mathcal{H}$ as a *heavy attribute*. Define

$$\mathcal{L} \quad = \quad attset(\mathcal{Q}) \setminus \mathcal{H}; \tag{5.2}$$

and call each attribute in $\mathcal{L}$ a *light attribute*. An edge $e \in \mathcal{E}$ is

- a *light edge* if e contains two light attributes, or
- a *cross edge* if e contains a heavy attribute and a light attribute.

A light attribute $X \in \mathcal{L}$ is a *border attribute* if it appears in at least one cross edge e of $\mathcal{G}$. Denote by $\mathcal{G}' = (\mathcal{L}, \mathcal{E}')$ the subgraph of $\mathcal{G}$ induced by $\mathcal{L}$. A vertex $X \in \mathcal{L}$ is *isolated* if $\{X\}$ is the only edge in $\mathcal{E}'$ incident to X. Define

$$\mathcal{I} \quad = \quad \{X \in \mathcal{L} \mid X \text{ is isolated}\}. \tag{5.3}$$

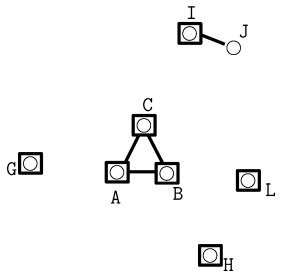


Figure 2: Subgraph induced by $\mathcal{L}$

Example (cont.). Consider again the join query $\mathcal{Q}$ whose hypergraph $\mathcal{G}$ is shown in Figure 1a. Set $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$. Set $\mathcal{L}$ includes all the white vertices in Figure 1b. Edge $\{\mathtt{A}, \mathtt{B}\}$ is a light edge, $\{\mathtt{A}, \mathtt{D}\}$ is a cross edge, while $\{\mathtt{D}, \mathtt{K}\}$ is neither a light edge nor a cross edge. All the vertices in $\mathcal{L}$ except J are border vertices. Figure 2 shows the subgraph of $\mathcal{G}$ induced by $\mathcal{L}$, where a unary edge is represented by a box and a binary edge by a segment. The isolated vertices are G, H, and L. □

5.2. **Semi-Join Reduction.** Recall from Section 4 that every configuration $\boldsymbol{\eta}$ of $\mathcal{H}$ defines a residual query $\mathcal{Q}'(\boldsymbol{\eta})$. Next, we will simplify $\mathcal{Q}'(\boldsymbol{\eta})$ into a join $\mathcal{Q}''(\boldsymbol{\eta})$ with the same result.

Observe that the hypergraph defined by $\mathcal{Q}'(\boldsymbol{\eta})$ is always $\mathcal{G}' = (\mathcal{L}, \mathcal{E}')$, regardless of $\boldsymbol{\eta}$. Consider a border attribute $X \in \mathcal{L}$ and a cross edge e of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ incident to X. As explained in Section 4, the input relation $R_e \in \mathcal{Q}$ defines a unary residual relation $R'_e(\boldsymbol{\eta}) \in \mathcal{Q}'(\boldsymbol{\eta})$. Note that $R'_e(\boldsymbol{\eta})$ has scheme $\{X\}$. We define:

$$R''_X(\boldsymbol{\eta}) \quad = \bigcap_{\text{cross edge } e \in \mathcal{E} \text{ s.t. } X \in e} R'_e(\boldsymbol{\eta}). \tag{5.4}$$

Example (cont.). Let $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$, and consider its configuration $\boldsymbol{\eta}$ with $\boldsymbol{\eta}(\mathtt{D}) = \mathtt{d}$, $\boldsymbol{\eta}(\mathtt{E}) = \mathtt{e}$, $\boldsymbol{\eta}(\mathtt{F}) = \mathtt{f}$, and $\boldsymbol{\eta}(\mathtt{K}) = \mathtt{k}$. Set X to the border attribute A. When e is $\{\mathtt{A}, \mathtt{D}\}$ or $\{\mathtt{A}, \mathtt{E}\}$, $R'_e(\boldsymbol{\eta})$ is the relation $R'_{\{\mathtt{A}\}|\mathtt{d}}$ or $R'_{\{\mathtt{A}\}|\mathtt{e}}$ mentioned in Section 1.3, respectively. $\{\mathtt{A}, \mathtt{D}\}$ and $\{\mathtt{A}, \mathtt{E}\}$ are the only cross edges containing A. Hence, $R''_A(\boldsymbol{\eta}) = R'_{\{\mathtt{A}\}|\mathtt{d}} \cap R'_{\{\mathtt{A}\}|\mathtt{e}}$, which is the relation $R''_{\{A\}}$ in Section 1.3. □

Recall that every light edge $e = \{X, Y\}$ in $\mathcal{G}$ defines a residual relation $R'_e(\boldsymbol{\eta})$ with scheme e. We define $R''_e(\boldsymbol{\eta})$ as a relation over e that contains every tuple $\boldsymbol{u} \in R'_e(\boldsymbol{\eta})$ satisfying:

- (applicable only if X is a border attribute) $\boldsymbol{u}(X) \in R''_X(\boldsymbol{\eta})$;
- (applicable only if Y is a border attribute) $\boldsymbol{u}(Y) \in R''_Y(\boldsymbol{\eta})$.

Note that if neither X nor Y is a border attribute, then $R''_e(\boldsymbol{\eta}) = R'_e(\boldsymbol{\eta})$.

Example (cont.). For the light edge $e = \{\mathtt{A}, \mathtt{B}\}$, $R'_e(\boldsymbol{\eta})$ is the relation $R'_{\{\mathtt{A},\mathtt{B}\}}$ mentioned in Section 1.3. Because A and B are border attributes, $R''_e(\boldsymbol{\eta})$ includes all the tuples in $R'_{\{\mathtt{A},\mathtt{B}\}}$ that take an A-value from $R''_{\mathtt{A}}(\boldsymbol{\eta})$ and a B-value from $R''_{\mathtt{B}}(\boldsymbol{\eta})$. This $R''_e(\boldsymbol{\eta})$ is precisely the relation $R''_{\{\mathtt{A},\mathtt{B}\}}$ in Section 1.3. □

Every vertex $X \in \mathcal{I}$ must be a border attribute and, thus, must now be associated with $R''_X(\boldsymbol{\eta})$. We can legally define:

$$\mathcal{Q}''_{isolated}(\boldsymbol{\eta}) = \{R''_X(\boldsymbol{\eta}) \mid X \in \mathcal{I}\} \quad (5.5)$$

$$\mathcal{Q}''_{light}(\boldsymbol{\eta}) = \{R''_e(\boldsymbol{\eta}) \mid \text{light edge } e \in \mathcal{E}\} \quad (5.6)$$

$$\mathcal{Q}''(\boldsymbol{\eta}) = \mathcal{Q}''_{light}(\boldsymbol{\eta}) \cup \mathcal{Q}''_{isolated}(\boldsymbol{\eta}). \quad (5.7)$$

Example (cont.). $\mathcal{Q}''_{isolated}(\boldsymbol{\eta}) = \{R''_{\{\mathtt{G}\}}, R''_{\{\mathtt{H}\}}, R''_{\{\mathtt{L}\}}\}$ and $\mathcal{Q}''_{light}(\boldsymbol{\eta}) = \{R''_{\{\mathtt{A},\mathtt{B}\}}, R''_{\{\mathtt{A},\mathtt{C}\}}, R''_{\{\mathtt{B},\mathtt{C}\}}, R''_{\{\mathtt{I},\mathtt{J}\}}\}$, where all the relation names follow those in Section 1.3. □

We will refer to the conversion from $\mathcal{Q}'(\boldsymbol{\eta})$ to $\mathcal{Q}''(\boldsymbol{\eta})$ as *semi-join reduction* and call $\mathcal{Q}''(\boldsymbol{\eta})$ the *reduced query* under $\boldsymbol{\eta}$. It is rudimentary to verify:

$$Join(\mathcal{Q}'(\boldsymbol{\eta})) = Join(\mathcal{Q}''(\boldsymbol{\eta})) = Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta})) \times Join(\mathcal{Q}''_{light}(\boldsymbol{\eta})). \quad (5.8)$$

5.3. **The Isolated Cartesian Product Theorem.** As shown in (5.5), $\mathcal{Q}''_{isolated}(\boldsymbol{\eta})$ contains $|\mathcal{I}|$ unary relations, one for each isolated attribute in $\mathcal{I}$. Hence, $Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))$ is the cartesian product of all those relations. The size of $Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))$ has a crucial impact on the efficiency of our join strategy because, as shown in Lemma 3.1, the load for computing a cartesian product depends on the cartesian product's size. To prove that our strategy is efficient, we want to argue that

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} \left| Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta})) \right| \quad (5.9)$$

is low, namely, the cartesian products of all the configurations $\boldsymbol{\eta} \in config(\mathcal{Q}, \mathcal{H})$ have a small size overall.

It is easy to place an upper bound of $\lambda^{|\mathcal{H}|} \cdot m^{|\mathcal{I}|}$ on (5.9). As each relation (trivially) has size at most m, we have $|Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))| \leq m^{|\mathcal{I}|}$. Given that $\mathcal{H}$ has at most $\lambda^{|\mathcal{H}|}$ different configurations, (5.9) is at most $\lambda^{|\mathcal{H}|} \cdot m^{|\mathcal{I}|}$. Unfortunately, the bound is not enough to establish the claimed performance of our MPC algorithm (to be presented in Section 6). For that purpose, we will need to prove a tighter upper bound on (5.9) — this is where the isolated cartesian product theorem (described next) comes in.

Given an arbitrary fractional edge packing W of the hypergraph $\mathcal{G}$, we define

$$W_{\mathcal{I}} = \sum_{Y \in \mathcal{I}} \text{weight of } Y \text{ under } W. \quad (5.10)$$

Recall that the weight of a vertex Y under W is the sum of $W(e)$ for all the edges $e \in \mathcal{E}$ containing Y.

Theorem 5.1 (*The isolated cartesian product theorem*)**.** *Let $\mathcal{Q}$ be a simple binary query whose relations have a total size of m. Denote by $\mathcal{G}$ the hypergraph defined by $\mathcal{Q}$. Consider an arbitrary subset $\mathcal{H} \subseteq attset(\mathcal{Q})$, where $attset(\mathcal{Q})$ is the set of attributes in the relations of $\mathcal{Q}$. Let $\mathcal{I}$ be the set of isolated vertices defined in* (5.3)*. Take an arbitrary fractional edge packing W of $\mathcal{G}$. It holds that*

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} \left| Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta})) \right| \leq \lambda^{|\mathcal{H}| - W_{\mathcal{I}}} \cdot m^{|\mathcal{I}|} \quad (5.11)$$

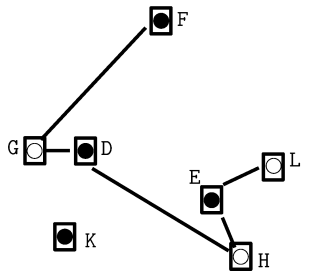


Figure 3: Illustration of $\mathcal{Q}^*$

where λ is the heavy parameter (Section 4), $config(\mathcal{Q}, \mathcal{H})$ is the set of configurations of $\mathcal{H}$ (Section 4), $\mathcal{Q}''_{isolated}(\boldsymbol{\eta})$ is defined in (5.5)*, and $W_{\mathcal{I}}$ is defined in* (5.10)*.*

Theorem 5.1 is in the strongest form when $W_{\mathcal{I}}$ is maximized. Later in Section 5.5, we will choose a specific W that yields a bound sufficient for us to prove the efficiency claim on our join algorithm.

Proof of Theorem 5.1. We will construct a set $\mathcal{Q}^*$ of relations such that $Join(\mathcal{Q}^*)$ has a result size at least the left-hand side of (5.11). Then, we will prove that the hypergraph of $\mathcal{Q}^*$ has a fractional edge covering that (by the AGM bound; Lemma 2.2) implies an upper bound on $|Join(\mathcal{Q}^*)|$ matching the right-hand side of (5.11).

Initially, set $\mathcal{Q}^*$ to $\emptyset$. For every cross edge $e \in \mathcal{E}$ incident to a vertex in $\mathcal{I}$, add to $\mathcal{Q}^*$ a relation $R^*_e = R_e$. For every $X \in \mathcal{H}$, add a unary relation $R^*_{\{X\}}$ to $\mathcal{Q}^*$ which consists of all the heavy values on X; note that $R^*_{\{X\}}$ has at most λ tuples. Finally, for every $Y \in \mathcal{I}$, add a unary relation $R^*_{\{Y\}}$ to $\mathcal{Q}^*$ which contains all the heavy and light values on Y.

Define $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ as the hypergraph defined by $\mathcal{Q}^*$. Note that $\mathcal{V}^* = \mathcal{I} \cup \mathcal{H}$, while $\mathcal{E}^*$ consists of all the cross edges in $\mathcal{G}$ incident to a vertex in $\mathcal{I}$, $|\mathcal{H}|$ unary edges $\{X\}$ for every $X \in \mathcal{H}$, and $|\mathcal{I}|$ unary edges $\{Y\}$ for every $Y \in \mathcal{I}$.

Example (cont.). Figure 3 shows the hypergraph of the $\mathcal{Q}^*$ constructed. As before, a box and a segment represent a unary and a binary edge, respectively. Recall that $\mathcal{H} = \{\mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{K}\}$ and $\mathcal{I} = \{\mathtt{G}, \mathtt{H}, \mathtt{L}\}$. □

Lemma 5.2. $\sum_{\boldsymbol{\eta}' \in config(\mathcal{Q},\mathcal{H})} \left| Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}')) \right| \leq |Join(\mathcal{Q}^*)|$.

Proof. We will prove

$$\bigcup_{\boldsymbol{\eta}' \in config(\mathcal{Q},\mathcal{H})} Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}')) \times \{\boldsymbol{\eta}'\} \quad \subseteq \quad Join(\mathcal{Q}^*). \tag{5.12}$$

from which the lemma follows.

Take a tuple $\boldsymbol{u}$ from the left-hand side of (5.12), and set $\boldsymbol{\eta}' = \boldsymbol{u}[\mathcal{H}]$. Based on the definition of $\mathcal{Q}''_{isolated}(\boldsymbol{\eta}')$, it is easy to verify that $\boldsymbol{u}[e] \in R_e$ for every cross edge $e \in \mathcal{E}$ incident a vertex in $\mathcal{I}$; hence, $\boldsymbol{u}[e] \in R^*_e$. Furthermore, $\boldsymbol{u}(X) \in R^*_{\{X\}}$ for every $X \in \mathcal{H}$

because $\boldsymbol{u}(X) = \boldsymbol{\eta'}(X)$ is a heavy value. Finally, obviously $\boldsymbol{u}(Y) \in R^*_{\{Y\}}$ for every $Y \in \mathcal{I}$. All these facts together ensure that $\boldsymbol{u} \in \mathit{Join}(\mathcal{Q}^*)$. □

Lemma 5.3. *$\mathcal{G}^*$ admits a tight fractional edge covering W^* satisfying $\sum_{X\in\mathcal{H}} W^*(\{X\}) = |\mathcal{H}| - W_{\mathcal{I}}$.*

Proof. We will construct a desired function W^* from the fractional edge packing W in Theorem 5.1.

For every cross edge $e \in \mathcal{E}$ incident to a vertex in $\mathcal{I}$, set $W^*(e) = W(e)$. Every edge in $\mathcal{E}$ incident to $Y \in \mathcal{I}$ must be a cross edge. Hence, $\sum_{\text{binary } e\in\mathcal{E}^*:Y\in e} W^*(e)$ is precisely the weight of Y under W.

Next, we will ensure that each attribute $Y \in \mathcal{I}$ has a weight 1 under W^*. Since W is a fractional edge packing of $\mathcal{G}$, it must hold that $\sum_{\text{binary } e\in\mathcal{E}^*:Y\in e} W(e) \leq 1$. This permits us to assign the following weight to the unary edge $\{Y\}$:

$$W^*(\{Y\}) \;=\; 1 - \sum_{\text{binary } e\in\mathcal{E}^*:Y\in e} W(e).$$

Finally, in a similar way, we make sure that each attribute $X \in \mathcal{H}$ has a weight 1 under W^* by assigning:

$$W^*(\{X\}) \;=\; 1 - \sum_{\text{binary } e\in\mathcal{E}^*:X\in e} W(e).$$

This finishes the design of W^*, which is now a tight fractional edge covering of $\mathcal{G}^*$.

Clearly:

$$\sum_{X\in\mathcal{H}} W^*(\{X\}) \;=\; |\mathcal{H}| - \sum_{X\in\mathcal{H}} \left(\sum_{\text{binary } e\in\mathcal{E}^*:X\in e} W(e) \right). \tag{5.13}$$

Every binary edge $e \in \mathcal{E}^*$ contains a vertex in $\mathcal{H}$ and a vertex in $\mathcal{I}$. Therefore:

$$\sum_{X\in\mathcal{H}} \left(\sum_{\text{binary } e\in\mathcal{E}^*:X\in e} W(e) \right) \;=\; \sum_{Y\in\mathcal{I}} \left(\sum_{\text{binary } e\in\mathcal{E}^*:Y\in e} W(e) \right) = W_{\mathcal{I}}.$$

Putting together the above equation with (5.13) completes the proof. □

The AGM bound in Lemma 2.2 tells us that

$$\begin{aligned}
\mathit{Join}(\mathcal{Q}^*) \;&\leq\; \prod_{e\in\mathcal{E}^*} |R^*_e|^{W^*(e)} \\
&=\; \Big(\prod_{X\in\mathcal{H}} |R^*_{\{X\}}|^{W^*(\{X\})} \Big) \Big(\prod_{Y\in\mathcal{I}} \prod_{e\in\mathcal{E}^*:Y\in e} |R^*_e|^{W^*(e)} \Big) \\
&\leq\; \Big(\prod_{X\in\mathcal{H}} \lambda^{W^*(\{X\})} \Big) \Big(\prod_{Y\in\mathcal{I}} \prod_{e\in\mathcal{E}^*:Y\in e} m^{W^*(e)} \Big) \\
&\quad \text{(applying } |R^*_{\{X\}}| \leq \lambda \text{ and } |R^*_e| \leq m\text{)} \\
&\leq\; \lambda^{|\mathcal{H}|-W_{\mathcal{I}}} \cdot m^{|\mathcal{I}|} \\
&\quad \text{(by Lemma 5.3 and } \sum_{e\in\mathcal{E}^*:Y\in e} W^*(e) = 1 \text{ for each } Y \text{ due to tightness of } W^*\text{)}
\end{aligned}$$

which completes the proof of Theorem 5.1. □

5.4. **A Subset Extension of Theorem 5.1.** Remember that $\mathcal{Q}''_{isolated}(\boldsymbol{\eta})$ contains a relation $R''_X(\boldsymbol{\eta})$ (defined in (5.4)) for every attribute $X \in \mathcal{I}$. Given a non-empty subset $\mathcal{J} \subseteq \mathcal{I}$, define

$$\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta}) \quad = \quad \{R''_X(\boldsymbol{\eta}) \;\big|\; X \in \mathcal{J}\}. \tag{5.14}$$

Note that $Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta}))$ is the cartesian product of the relations in $\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta})$.

Take an arbitrary fractional edge packing W of the hypergraph $\mathcal{G}$. Define

$$W_{\mathcal{J}} \quad = \quad \sum_{Y \in \mathcal{J}} \text{weight of } Y \text{ under } W. \tag{5.15}$$

We now present a general version of the isolated cartesian product theorem:

Theorem 5.4. *Let $\mathcal{Q}$ be a simple binary query whose relations have a total size of m. Denote by $\mathcal{G}$ the hypergraph defined by $\mathcal{Q}$. Consider an arbitrary subset $\mathcal{H} \subseteq attset(\mathcal{Q})$, where $attset(\mathcal{Q})$ is the set of attributes in the relations of $\mathcal{Q}$. Let $\mathcal{I}$ be the set of isolated vertices defined in* (5.3) *and $\mathcal{J}$ be any non-empty subset of $\mathcal{I}$. Take an arbitrary fractional edge packing W of $\mathcal{G}$. It holds that*

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} \left| Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta})) \right| \quad \leq \quad \lambda^{|\mathcal{H}|-W_{\mathcal{J}}} \cdot m^{|\mathcal{J}|}. \tag{5.16}$$

where λ is the heavy parameter (see Section 4), $config(\mathcal{Q},\mathcal{H})$ is the set of configurations of $\mathcal{H}$ (Section 4), $\mathcal{Q}''_{\mathcal{J}}$ is defined in (5.14)*, and $W_{\mathcal{J}}$ is defined in* (5.15)*.*

Proof. We will prove the theorem by reducing it to Theorem 5.1. Define $\overline{\mathcal{J}} = \mathcal{I} \setminus \mathcal{J}$ and

$$\tilde{\mathcal{Q}} \quad = \quad \{R \in Q \mid scheme(R) \cap \overline{\mathcal{J}} = \emptyset\}.$$

One can construct $\tilde{\mathcal{Q}}$ alternatively as follows. First, discard from $\mathcal{Q}$ every relation whose scheme contains an attribute in $\overline{\mathcal{J}}$. Then, $\tilde{\mathcal{Q}}$ consists of the relations remaining in $\mathcal{Q}$.

Denote by $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ the hypergraph defined by $\tilde{\mathcal{Q}}$. Set $\tilde{\mathcal{H}} = \mathcal{H} \cap attset(\tilde{\mathcal{Q}})$ and $\tilde{\mathcal{L}} = attset(\tilde{\mathcal{Q}}) \setminus \tilde{\mathcal{H}}$. $\mathcal{J}$ is precisely the set of isolated attributes decided by $\tilde{\mathcal{Q}}$ and $\tilde{\mathcal{H}}$.[3]

Define a function $\tilde{W} : \tilde{\mathcal{E}} \to [0,1]$ by setting $\tilde{W}(e) = W(e)$ for every $e \in \tilde{\mathcal{E}}$. $\tilde{W}$ is a fractional edge packing of $\tilde{\mathcal{G}}$. Because every edge $e \in \mathcal{E}$ containing an attribute in $\mathcal{J}$ is preserved in $\tilde{\mathcal{E}}$,[4] we have $W_{\mathcal{J}} = \tilde{W}_{\mathcal{J}}$. Applying Theorem 5.1 to $\tilde{\mathcal{Q}}$ gives:

$$\sum_{\tilde{\boldsymbol{\eta}} \in config(\tilde{\mathcal{Q}},\tilde{\mathcal{H}})} \left| Join(\tilde{\mathcal{Q}}''_{isolated}(\tilde{\boldsymbol{\eta}})) \right| \;\leq\; \lambda^{|\tilde{\mathcal{H}}|-\tilde{W}_{\mathcal{J}}} \cdot m^{|\mathcal{J}|} \;=\; \lambda^{|\tilde{\mathcal{H}}|-W_{\mathcal{J}}} \cdot m^{|\mathcal{J}|}. \tag{5.17}$$

[3] Let $\tilde{\mathcal{I}}$ be the set of isolated attributes after removing $\tilde{\mathcal{H}}$ from $\tilde{\mathcal{G}}$. We want to prove $\mathcal{J} = \tilde{\mathcal{I}}$. It is easy to show $\mathcal{J} \subseteq \tilde{\mathcal{I}}$. To prove $\tilde{\mathcal{I}} \subseteq \mathcal{J}$, suppose that there is an attribute X such that $X \in \tilde{\mathcal{I}}$ but $X \notin \mathcal{J}$. As X appears in $\tilde{\mathcal{G}}$, we know $X \notin \mathcal{I}$. Hence, $\mathcal{G}$ must contain an edge $\{X, Y\}$ with $Y \notin \mathcal{H}$. This means $Y \notin \mathcal{I}$, because of which the edge $\{X, Y\}$ is disjoint with $\overline{\mathcal{J}}$ and thus must belong to $\tilde{\mathcal{G}}$. But this contradicts the fact $X \in \tilde{\mathcal{I}}$.

[4] Suppose that there is an edge $e = \{X, Y\}$ such that $X \in \mathcal{J}$ and yet $e \notin \tilde{\mathcal{E}}$. It means that $Y \in \bar{\mathcal{J}} \subseteq \mathcal{I}$. But then e is incident on two attributes in $\mathcal{I}$, which is impossible.

It remains to show

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} \left| Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta})) \right| \;\leq\; \lambda^{|\mathcal{H}|-|\tilde{\mathcal{H}}|} \sum_{\tilde{\boldsymbol{\eta}} \in config(\tilde{\mathcal{Q}},\tilde{\mathcal{H}})} \left| Join(\tilde{\mathcal{Q}}''_{isolated}(\tilde{\boldsymbol{\eta}})) \right| \tag{5.18}$$

after which Theorem 5.4 will follow from (5.17) and (5.18).

For each configuration $\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})$, we can find $\tilde{\boldsymbol{\eta}} = \boldsymbol{\eta}[\tilde{\mathcal{H}}] \in config(\tilde{Q},\tilde{H})$ such that $Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta})) = Join(\tilde{\mathcal{Q}}''_{isolated}(\tilde{\boldsymbol{\eta}}))$. The correctness of (5.18) follows from the fact that at most $\lambda^{|\mathcal{H}|-|\tilde{\mathcal{H}}|}$ configurations $\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})$ correspond to the same $\tilde{\boldsymbol{\eta}}$. □

5.5. **A Weaker Result.** One issue in applying Theorem 5.4 is that the quantity $|\mathcal{H}| - W_{\mathcal{J}}$ is not directly related to the fractional edge covering number ρ of $\mathcal{Q}$. The next lemma gives a weaker result that addresses the issue to an extent sufficient for our purposes in Section 6:

Lemma 5.5. *Let $\mathcal{Q}$ be a simple binary query who relations have a total size of m. Denote by $\mathcal{G}$ the hypergraph defined by $\mathcal{Q}$. Consider an arbitrary subset $\mathcal{H} \subseteq attset(\mathcal{Q})$, where $attset(\mathcal{Q})$ is the set of attributes in the relations of $\mathcal{Q}$. Define $\mathcal{L} = attset(\mathcal{Q}) \setminus \mathcal{H}$ and $\mathcal{I}$ as the set of isolated vertices in $\mathcal{L}$ (see* (5.3)*). For any non-empty subset $\mathcal{J} \subseteq \mathcal{I}$, it holds that*

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{Q},\mathcal{H})} \left| Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta})) \right| \;\leq\; \lambda^{2\rho-|\mathcal{J}|-|\mathcal{L}|} \cdot m^{|\mathcal{J}|} \tag{5.19}$$

where ρ is the fractional edge covering number of $\mathcal{G}$, λ is the heavy parameter (Section 4), $config(\mathcal{Q},\mathcal{H})$ is the set of configurations of $\mathcal{H}$ (Section 4), and $\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta})$ is defined in (5.14)*.*

Proof. Let W be an arbitrary fractional edge packing of $\mathcal{G}$ satisfying the second bullet of Lemma 2.1. Specifically, the weight of W is the fractional edge packing number τ of $\mathcal{G}$; and the weight of every vertex in $\mathcal{G}$ is either 0 or 1. Denote by Z the set of vertices in $\mathcal{G}$ whose weights under W are 0. Lemma 2.1 tells us $\tau + \rho = |attset(\mathcal{Q})|$ and $\rho - \tau = |\mathcal{Z}|$. Set $\mathcal{J}_0 = \mathcal{J} \cap Z$ and $\mathcal{J}_1 = \mathcal{J} \setminus \mathcal{J}_0$. Because $\mathcal{J}_0 \subseteq \mathcal{Z}$, we can derive:

$$\begin{aligned}
\tau + |\mathcal{J}_0| &\leq \rho \;\Rightarrow \\
|attset(\mathcal{Q})| - \rho + |\mathcal{J}_0| &\leq \rho \;\Rightarrow \\
(|\mathcal{H}| + |\mathcal{L}|) + (|\mathcal{J}| - |\mathcal{J}_1|) &\leq 2\rho \;\Rightarrow \\
|\mathcal{H}| - |\mathcal{J}_1| &\leq 2\rho - |\mathcal{J}| - |\mathcal{L}|.
\end{aligned}$$

Lemma 5.5 now follows from Theorem 5.4 due to $|\mathcal{J}_1| = W_{\mathcal{J}}$, which holds because every vertex in $\mathcal{J}_1$ has weight 1 under W. □

Remark. The above lemma was the "isolated cartesian product theorem" presented in the preliminary version [20] of this work. The new version (i.e., Theorem 5.4) is more powerful and better captures the mathematical structure underneath.

6. An MPC Join Algorithm

This section will describe how to answer a simple binary join $\mathcal{Q}$ in the MPC model with load $\tilde{O}(m/p^{1/\rho})$.

We define a *statistical record* as a tuple (R, X, x, cnt), where R is a relation in $\mathcal{Q}$, X an attribute in $scheme(R)$, x a value in **dom**, and cnt the number of tuples $\boldsymbol{u} \in R$ with $\boldsymbol{u}(X) = x$. Specially, $(R, \emptyset, nil, cnt)$ is also a statistical record where cnt gives the number of tuples in R that use only light values. A *histogram* is defined as the set of statistical records for all possible R, X, and x satisfying (i) $cnt = \Omega(m/p^{1/\rho})$ or (ii) $X = \emptyset$ (and, hence $x = nil$); note that there are only $O(p^{1/\rho})$ such records. We assume that every machine has a local copy of the histogram. By resorting to standard MPC sorting algorithms [9, 10], the assumption can be satisfied with a preprocessing that takes constant rounds and load $\tilde{O}(p^{1/\rho} + m/p)$.

Henceforth, we will fix the heavy parameter

$$\lambda \quad = \quad \Theta(p^{1/(2\rho)})$$

and focus on explaining how to compute (5.1) for an arbitrary subset $\mathcal{H}$ of $attset(\mathcal{Q})$. As $attset(\mathcal{Q})$ has $2^k = O(1)$ subsets (where $k = |attset(\mathcal{Q})|$), processing them all in parallel increases the load by only a constant factor and, as guaranteed by (4.2), discovers the entire $Join(\mathcal{Q})$.

Our algorithm produces (5.1) in three steps:

(1) Generate the input relations of the residual query $\mathcal{Q}'(\boldsymbol{\eta})$ of every configuration $\boldsymbol{\eta}$ of $\mathcal{H}$ (Section 5.1).
(2) Generate the input relations of the reduced query $\mathcal{Q}''(\boldsymbol{\eta})$ of every $\boldsymbol{\eta}$ (Section 5.2).
(3) Evaluate $\mathcal{Q}''(\boldsymbol{\eta})$ for every $\boldsymbol{\eta}$.

The number of configurations of $\mathcal{H}$ is $O(\lambda^{|\mathcal{H}|}) = O(\lambda^k) = O(p^{k/(2\rho)})$, which is $O(p)$ because $\rho \geq k/2$ by the first bullet of Lemma 2.1. Next, we elaborate on the details of each step.

Step 1. Lemma 4.1 tells us that the input relations of all the residual queries have at most $m \cdot \lambda^{k-2}$ tuples in total. We allocate $p'_{\boldsymbol{\eta}} = \lceil p \cdot \frac{m_{\boldsymbol{\eta}}}{\Theta(m \cdot \lambda^{k-2})} \rceil$ machines to store the relations of $\mathcal{Q}'(\boldsymbol{\eta})$, making sure that $\sum_{\boldsymbol{\eta}} p'_{\boldsymbol{\eta}} \leq p$. Each machine keeps on average

$$O(m_{\boldsymbol{\eta}}/p'_{\boldsymbol{\eta}}) = O(m \cdot \lambda^{k-2}/p) = O(m/p^{1/\rho})$$

tuples, where the last equality used $\rho \geq k/2$. Each machine $i \in [1, p]$ can use the histogram to calculate the input size $m_{\boldsymbol{\eta}}$ of $\mathcal{Q}'(\boldsymbol{\eta})$ precisely for each $\boldsymbol{\eta}$; it can compute locally the id range of the $m_{\boldsymbol{\eta}}$ machines responsible for $Q'(\boldsymbol{\eta})$. If a tuple $\boldsymbol{u}$ in the local storage of machine i belongs to $Q'(\boldsymbol{\eta})$, the machine sends $\boldsymbol{u}$ to a random machine within that id range. Standard analysis shows that each of the $m_{\boldsymbol{\eta}}$ machines receives asymptotically the same number of tuples of $\mathcal{Q}'(\boldsymbol{\eta})$ (up to an $\tilde{O}(1)$ factor) with probability at least $1 - 1/p^c$ for an arbitrarily large constant c. Hence, Step 1 can be done in a single round with load $\tilde{O}(m/p^{1/\rho})$ with probability at least $1 - 1/p^c$.

Step 2. Now that all the input relations of each $\mathcal{Q}'(\boldsymbol{\eta})$ have been stored on $p'_{\boldsymbol{\eta}}$ machines, the semi-join reduction in Section 5.2 that converts $\mathcal{Q}'(\boldsymbol{\eta})$ to $\mathcal{Q}''(\boldsymbol{\eta})$ is a standard process that can be accomplished [10] with sorting in $O(1)$ rounds entailing a load of $\tilde{O}(m_{\boldsymbol{\eta}}/p'_{\boldsymbol{\eta}}) = \tilde{O}(m/p^{1/\rho})$; see also [13] for a randomized algorithm that performs fewer rounds.

Step 3. This step starts by letting each machine know about the value of $|Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))|$ for every $\boldsymbol{\eta}$. For this purpose, each machine broadcasts to all other machines how many tuples it has in $R''_X(\boldsymbol{\eta})$ for every $X \in \mathcal{I}$ and every $\boldsymbol{\eta}$. Since there are $O(p)$ different $\boldsymbol{\eta}$, $O(p)$ numbers are sent by each machine, such that the load of this round is $O(p^2)$. From the numbers received, each machine can independently figure out the values of all $|Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))|$.

We allocate

$$p''_{\boldsymbol{\eta}} = \Theta\left(\lambda^{|\mathcal{L}|} + p \cdot \sum_{\text{non-empty } \mathcal{J} \subseteq \mathcal{I}} \frac{|Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta}))|}{\lambda^{2\rho-|\mathcal{J}|-|\mathcal{L}|} \cdot m^{|\mathcal{J}|}}\right) \tag{6.1}$$

machines for computing $\mathcal{Q}''(\boldsymbol{\eta})$. Notice that

$$\sum_{\boldsymbol{\eta}} p''_{\boldsymbol{\eta}} = O\left(\sum_{\boldsymbol{\eta}} \lambda^{|\mathcal{L}|}\right) + O\left(p \cdot \sum_{\text{non-empty } \mathcal{J} \subseteq \mathcal{I}} \sum_{\boldsymbol{\eta}} \frac{|Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta}))|}{\lambda^{2\rho-|\mathcal{J}|-|\mathcal{L}|} \cdot m^{|\mathcal{J}|}}\right) = O(p)$$

where the equality used Lemma 5.5, the fact that $\mathcal{I}$ has constant non-empty subsets, and that $\sum_{\boldsymbol{\eta}} \lambda^{|\mathcal{L}|} \leq \lambda^{|\mathcal{H}|} \cdot \lambda^{|\mathcal{L}|} = \lambda^k \leq p$. We can therefore adjust the constants in (6.1) to make sure that the total number of machines needed by all the configurations is at most p.

Lemma 6.1. *$\mathcal{Q}''(\boldsymbol{\eta})$ can be answered in one round with load $O(m/p^{1/\rho})$ using $p''_{\boldsymbol{\eta}}$ machines, subject to a failure probability of at most $1/p^c$ where c can be set to an arbitrarily large constant.*

Proof. As shown in (5.8), $Join(\mathcal{Q}''(\boldsymbol{\eta}))$ is the cartesian product of $Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))$ and $Join(\mathcal{Q}''_{light}(\boldsymbol{\eta}))$. We deploy $\Theta(p''_{\boldsymbol{\eta}}/\lambda^{|\mathcal{L}|-|\mathcal{I}|})$ machines to compute $Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))$ in one round. By Lemma 3.1, the load is

$$\tilde{O}\left(\frac{|Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta}))|^{1/|\mathcal{J}|}}{\left(\frac{p''_{\boldsymbol{\eta}}}{\lambda^{|\mathcal{L}|-|\mathcal{I}|}}\right)^{1/|\mathcal{J}|}}\right) \tag{6.2}$$

for some non-empty $\mathcal{J} \subseteq \mathcal{I}$. (6.1) guarantees that

$$p''_{\boldsymbol{\eta}} = \Omega\left(p \cdot \frac{|Join(\mathcal{Q}''_{\mathcal{J}}(\boldsymbol{\eta}))|}{\lambda^{2\rho-|\mathcal{J}|-|\mathcal{L}|} \cdot m^{|\mathcal{J}|}}\right)$$

with which we can derive

$$(6.2) = \tilde{O}\left(\frac{m \cdot \lambda^{\frac{2\rho-|\mathcal{J}|-|\mathcal{I}|}{|\mathcal{J}|}}}{p^{1/|\mathcal{J}|}}\right) = \tilde{O}\left(\frac{m \cdot \lambda^{\frac{2\rho-2|\mathcal{J}|}{|\mathcal{J}|}}}{p^{1/|\mathcal{J}|}}\right) = \tilde{O}\left(\frac{m \cdot p^{\frac{2\rho-2|\mathcal{J}|}{2\rho|\mathcal{J}|}}}{p^{1/|\mathcal{J}|}}\right) = \tilde{O}\left(\frac{m}{p^{1/\rho}}\right).$$

Regarding $\mathcal{Q}''_{light}(\boldsymbol{\eta})$, first note that $attset(\mathcal{Q}''_{light}(\boldsymbol{\eta})) = \mathcal{L} \setminus \mathcal{I}$. If $\mathcal{L} \setminus \mathcal{I}$ is empty, no $\mathcal{Q}''_{light}(\boldsymbol{\eta})$ exists and $Join(\mathcal{Q}''(\boldsymbol{\eta})) = Join(\mathcal{Q}''_{isolated}(\boldsymbol{\eta}))$. The subsequent discussion considers that $\mathcal{L} \setminus \mathcal{I}$ is not empty. As the input relations of $\mathcal{Q}''_{light}(\boldsymbol{\eta})$ contain only light values, $\mathcal{Q}''_{light}(\boldsymbol{\eta})$ is skew-free if a share of λ is assigned to each attribute in $\mathcal{L} \setminus \mathcal{I}$. By Lemma 3.3, $Join(\mathcal{Q}''_{light}(\boldsymbol{\eta}))$ can be computed in one round with load $\tilde{O}(m/\lambda^2) = \tilde{O}(m/p^{1/\rho})$ using $\Theta(\lambda^{|\mathcal{L}\setminus\mathcal{I}|})$ machines, subject to a certain failure probability δ. As $\lambda^{|\mathcal{L}\setminus\mathcal{I}|} \geq \lambda$ which is a polynomial of p, Lemma 3.3 allows us to make sure $\delta \leq 1/p^c$ for any constant c.

By combining the above discussion with Lemma 3.2, we conclude that $Join(\mathcal{Q}''(\boldsymbol{\eta}))$ can be computed in one round with load $\tilde{O}(m/p^{1/\rho})$ using $p''_{\boldsymbol{\eta}}$ machines, subject to a failure probability at most $\delta \leq 1/p^c$. □

Overall, the load of our algorithm is $\tilde{O}(p^{1/\rho} + p^2 + m/p^{1/\rho})$. This brings us to our second main result:

Theorem 6.2. *Given a simple binary join query with input size $m \geq p^3$ and a fractional edge covering number ρ, we can answer it in the MPC model using p machines in constant rounds with load $\tilde{O}(m/p^{1/\rho})$, subject to a failure probability of at most $1/p^c$ where c can be set to an arbitrarily large constant.*

7. Concluding Remarks

This paper has introduced an algorithm for computing a natural join over binary relations under the MPC model. Our algorithm performs a constant number of rounds and incurs a load of $\tilde{O}(m/p^{1/\rho})$ where m is the total size of the input relations, p is the number of machines, and ρ is the fractional edge covering number of the query. The load matches a known lower bound up to a polylogarithmic factor. Our techniques heavily rely on a new finding, which we refer to as the *isolated cartesian product theorem*, on the join problem's mathematical structure.

We conclude the paper with two remarks:

- The assumption $p^3 \leq m$ can be relaxed to $p \leq m^{1-\epsilon}$ for an arbitrarily small constant $\epsilon > 0$. Recall that our algorithm incurs a load of $\tilde{O}(p^{1/\rho} + p^2 + m/p^{1/\rho})$ where the terms $\tilde{O}(p^{1/\rho})$ and $\tilde{O}(p^2)$ are both due to the computation of statistics (in preprocessing and Step 2, respectively). In turn, these statistics are needed to allocate machines for subproblems. By using the machine-allocation techniques in [10], we can avoid most of the statistics communication and reduce the load to $\tilde{O}(p^\epsilon + m/p^{1/\rho})$.
- In the external memory (EM) model [4], we have a machine equipped with M words of internal memory and an unbounded disk that has been formatted into *blocks* of size B words. An I/O either reads a block of B words from the disk to the memory, or overwrites a block with B words in the memory. A join query $\mathcal{Q}$ is considered solved if every tuple $\boldsymbol{u} \in \mathcal{Q}$ has been generated in memory at least once. The challenge is to design an algorithm to achieve the purpose with as few I/Os as possible. There exists a reduction [13] that can be used to convert an MPC algorithm to an EM counterpart. Applying the reduction on our algorithm gives an EM algorithm that solves $\mathcal{Q}$ with $\tilde{O}(\frac{m^\rho}{B \cdot M^{\rho-1}})$ I/Os, provided that $M \geq m^c$ for some positive constant $c < 1$ that depends on $\mathcal{Q}$. The I/O complexity can be shown to be optimal up to a polylogarithmic factor using the lower-bound arguments in [11, 18]. We suspect that the constraint $M \geq m^c$ can be removed by adapting the isolated cartesian product theorem to the EM model.

References

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

[2] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):922–933, 2009.

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

[5] Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *SIAM Journal on 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] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 137–150, 2004.

[9] Michael T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal of Computing*, 29(2):416–432, 1999.

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

[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] 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.

[14] Paraschos Koutris, Semih Salihoglu, and Dan Suciu. Algorithmic aspects of parallel data processing. *Foundations and Trends in Databases*, 8(4):239–370, 2018.

[15] Ilan Newman. Private vs. common random bits in communication complexity. *Information Processing Letters (IPL)*, 39(2):67–71, 1991.

[16] 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.

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

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

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

[20] 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.

[21] Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

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