PAC: Computing Join Queries with Semi-Covers

Heba Aamer
Vrije Universiteit Brussel, Belgium

Bas Ketsman
Vrije Universiteit Brussel, Belgium

Abstract

An increased and growing interest in large-scale data processing has triggered a demand for specialized algorithms that thrive in massively parallel shared-nothing systems. To answer the question of how to efficiently compute join queries in this setting, a rich line of research has emerged specifically for the Massively Parallel Communication (MPC) model. In the MPC model, algorithms are executed in rounds, with each round consisting of a synchronized communication phase and a separate local computation phase. The main cost measure is the load of the algorithm, defined as the maximum number of messages received by any server in any round.

We study worst-case optimal algorithms for the join query evaluation problem in the constant-round MPC model. In the single-round variant of MPC, the worst-case optimal load for this problem is well understood and algorithms exist that guarantee this load for any join query. In the constant-round variant of MPC, queries can often be computed with a lower load compared to the single-round variant, but the worst-case optimal load is only known for specific classes of join queries, including graph-like and acyclic join queries, and the associated algorithms use very different techniques. In this paper, we propose a new constant-round MPC algorithm for computing join queries. Our algorithm is correct for every join query and its load matches (up to a polylog factor) the worst-case optimal load for at least all join queries that are acyclic or graph-like.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Logic and databases; Theory of computation → Abstract machines

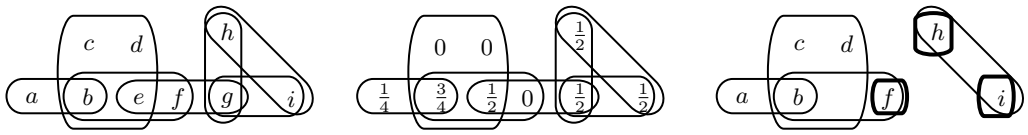
Keywords and phrases Worst-case optimal load, MPC model, join queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.6

Funding This work is partially funded by FWO-grant G062721N.

1 Introduction

In this paper, we study the query evaluation problem in distributed shared-nothing systems with very large numbers of servers. More precisely, we present a new algorithm for the computation of join queries (also called full self-join free conjunctive queries) in the MPC model. The MPC model is a well-established model within the database theory community for studying distributed algorithms. The model takes the number p as a parameter, representing the number of available servers. Servers have no direct access to each other's memory but they are connected through a network and can communicate with each other via private (one-to-one) message-passing. Algorithms for the MPC model are executed in rounds, with each round consisting of a *local computation* phase followed by a *global communication* phase. During the local computation phase, every server performs a computation over the data it has locally. During this phase, no messages are sent or received. During the communication phase, every server can send messages to other servers in the network. During this phase, no computation is performed by the servers. The communication phase ends with a global synchronization step after which all messages sent during the communication phase arrive in batch at their destination. We refer to the number of messages (*i.e.,* tuples) received by a server during the communication phase as the server's *load* in that round.



(a) The hypergraph of $q_\otimes$.

(b) Optimal fractional vertex cover for $q_\otimes$.

(c) Query $q_{\otimes\{e,g\}}$ (thin and thick hyperedges) and $(q_{\otimes\{e,g\}})\downarrow$ (only thin hyperedges).

Figure 1 Illustration of concepts on join query $q_\otimes := R_1[a,b] \bowtie R_2[b,c,d,e] \bowtie R_3[b,e,f] \bowtie R_4[e,f,g] \bowtie R_5[g,h] \bowtie R_6[g,i] \bowtie R_7[h,i]$.

The cost of an algorithm in the MPC model is measured in terms of maximum load and number of rounds (the number of synchronization barriers, to be precise). By maximum load, we mean the maximum load of any server during any of the algorithm's rounds. In this paper, we focus on *worst-case optimality* results for join queries. That is, we are interested in algorithms that optimize the maximum load w.r.t. given cardinality constraints rather than for the specific database instance over which the query is computed. We assume that initially the database instance D is evenly (and randomly) partitioned over the p servers, hence requiring a *linear* load of $|D|/p$ with $|D|$ being the number of tuples in D, which is essentially the best load one can hope for. Clearly, for the join query evaluation problem, a linear load of $|D|/p$ in the worst-case will be unrealistically low for most queries. In this context, we remark that some care is necessary with respect to the number of allowed rounds, as with $\geq p$ rounds one server can learn the entire database with a load as low as $|D|/p$ by having every other server send all its data in a round, after which computing any query over D becomes trivial. For this reason, it is common to add a restriction on the number of rounds to the model, with single-round and constant-round algorithms being of particular interest in the literature.

An algorithm in the MPC model evaluates a join query correctly, when every tuple in the output can be produced by any of the servers at any round. Several lower bounds for the join query evaluation problem in the MPC model are known. Since many of these bounds [10, 7, 12, 6, 13] rely on the assumption that relation instances have equal cardinalities m, we will also assume this condition throughout the paper. In other words, $|D| = r \cdot m$, with r the number of relations of the query and m the cardinality of the individual relation instances. The bounds relevant to this paper are all of the form $|D|/p^{1/k}$, with k a number depending on the structure of the target query where k varies depending on the considered variant of the problem. It is noteworthy that the bounds are in terms of $|D|$ (and not the number of bits) because we only consider tuple-based algorithms, similar to other works.

Early work mostly focuses on single-round MPC algorithms [2, 4, 5, 3, 10] and in this variant of the model the worst-case load is now well-understood. An algorithm of particular importance in this context is HyperCube [2, 4], also called the *shares* algorithm. The HyperCube algorithm is parameterized with an assignment of values for the query attributes that represent their shares. These shares are often computed based on a *fractional vertex cover* which is a function assigning rational weights between 0 and 1 to the attributes of the query such that, for every relation, the sum of weights assigned to its attributes is at least 1 (*i.e.,* the relation is *covered*). An example of a fractional vertex cover of the query $q_\otimes$, whose hypergraph is in Figure 1a, is given in Figure 1b. The sum of the assigned weights is called the *weight* of the fractional vertex cover. The choice of cover influences the load that the HyperCube will guarantee. More precisely, the lower the weight of the chosen vertex cover is, the lower the load gets. The optimal (*i.e.,* least) weight is called the *fractional vertex*

covering number, denoted τ. The fractional vertex cover in Figure 1b is an example of an optimal cover for $q_{\otimes}$ and hence $\tau(q_{\otimes}) = 3$. When the HyperCube is parameterized with an optimal fractional vertex cover, the algorithm is optimal in the sense that its load matches a $|D|/p^{1/\tau}$ lower bound [5] up to a polylog factor and with high probability when the input database is *skew-free*. Skew-freeness means that the degrees of attribute values (and by extension, partial tuples over attributes) do not exceed certain threshold values. While the precise definition of the threshold values is unimportant for the discussion, we remark that they depend on the considered fractional vertex cover. Intuitively: a higher weight on an attribute requires a lower degree for the attribute's values. In particular, a weight of 0 means that no constraints apply on the attribute's values.

Without constraints on the database, the worst-case optimal load of single-round MPC algorithms is $|D|/p^{1/\psi}$ and an algorithm exists that can achieve it w.h.p. and up to a polylog factor [10]. The number ψ is the *edge quasi-packing number* and equals the maximum fractional vertex covering number of residuals of the target query. A residual of a query q is the query obtained after removing some of the attributes of q (from all relations that have these attributes). Since every query is a residual of itself, it is immediate that $\tau \leq \psi$. An example of a residual query of $q_{\otimes}$ is given in Figure 1c. A simple algorithm to compute join queries with load $|D|/p^{1/\psi}$ [10] goes as follows: first partition the database in fragments such that in each fragment, for every attribute of the query, all values for that attribute either all have a low degree or all have a high degree (with respect to some threshold value). Given a specific such database fragment, say with H the set of heavy attributes, HyperCube is applied parameterized with a fractional vertex cover that assigns weights 0 to all the attributes in H. Since the threshold values will guarantee that the fragment is skew-free w.r.t. *any* fractional vertex cover assigning weights 0 to attributes in H, the load of this computation is decided by the weight of the considered fractional vertex cover, and is thus at best dependent on the fractional vertex covering number of the residual of q based on H. Since the total number of considered fragments is constant (under data complexity), the overall algorithm computes the target query over all fragments in parallel using the same set of p servers.

The main advantage of using multiple rounds lies in the key insight that the evaluation of some (residual) queries can be simplified by computing semi-joins in one round with linear load. For example, the evaluation of query $q_{\otimes\{e,g\}}$ in Figure 1c over some fragment can be simplified into the evaluation of query $(q_{\otimes\{e,g\}})\downarrow$ in the same figure (over a modification of the fragment with *similar* size) by applying semi-joins and hence lowering the load from $|D|/p^{1/4}$ (as in the single-round algorithm, because $\tau = 4$) to $|D|/p^{1/3}$. Leveraging this technique, multi-round join algorithms rely on the following *reduction* procedure: A set H of heavy attributes is chosen based on some threshold values. Now, suppose there are s possible heavy tuples over the H attributes in the considered fragment. The set of available servers gets *equally* divided into $|s|$ (disjoint) groups such that each heavy tuple $\boldsymbol{h}$ gets a dedicated group of servers that is responsible for computing the query over the considered fragment *only* when the attributes of H are fixed as per the values in $\boldsymbol{h}$. The H attributes are then dropped from the query and, over every group of servers, semi-joins are applied. How to proceed and evaluate the simplified query afterwards varies depending on the algorithm under consideration.

For the multi-round variant, a few results are known and several questions remain open. It is known that the worst-case optimal load is at least $|D|/p^{1/\rho}$, where the number ρ is the *fractional edge covering number* of the query. This lower bound result was obtained in [10], and for some join queries this bound is tight. This is the case for all graph-like queries

(*i.e.,* join queries involving relations with at most two attributes) [10, 7, 12, 11, 8] as well as for all join queries that are acyclic [6, 13]. For a while, it has been thought that $|D|/p^{1/\rho}$ could be the optimal load in general, but this idea was recently debunked. In particular, it is shown in [6] that $|D|/p^{1/\tau}$ is a tighter lower bound for a specific class of cyclic queries over relations with an arbitrary number of attributes. We remark that both graph-like and acyclic join queries have an important property in common, namely $\tau \leq \rho$,[1] while for the class of queries considered in [6] we have $\rho \leq \tau$. It is worth noting that the worst-case optimal algorithms proposed in the literature for either the graph-like or the acyclic queries cannot straightforwardly be generalized to arbitrary join queries. Thus, it remained open whether there exists a unified multi-round MPC algorithm that can work on arbitrary join queries. Later, in [11] an algorithm is introduced that can compute all join queries with load based on a newly-specified number. This number strictly exceeds ψ for some queries and hence on such queries, using more rounds incurs more load. For this reason and for brevity, we no longer mention this algorithm in the rest of the paper and we will compare it against ours in future work.

In this paper, we propose a new distributed join algorithm whose load is bounded by $|D|/p^{1/\gamma}$ up to a polylog factor and with high probability, and with γ a new query related number that we call *PAC*. Our algorithm runs in three rounds and is relatively simple, yet it is powerful enough to show the following:

1. We show that $\rho \leq \gamma \leq \psi$ for all join queries. Moreover, we show that $\gamma \leq \rho$ for acyclic and graph-like join queries, *i.e.,* our algorithm is worst-case optimal for all classes of queries for which the worst-case optimal load is currently known (with one exception discussed in Section 7).
2. We also show that $\rho = \gamma < \psi$ (and hence worst-case optimal) for some join queries that are not considered by any other previous algorithms.

PAC refers to three main concepts (**P**atches, **A**nchors, and semi-**C**overs) that our number, γ, is based on and our algorithm is parameterized with. A high-level overview: both patches and anchors are sets of sets of attributes from the query, and semi-covers are a relaxation of normal fractional vertex covers in which only a subset of the relations of the query should be covered. Similar to before, our algorithm evaluates a query over a database by considering the different fragments separately. Over a particular fragment, the algorithm is parameterized with a choice of patches, anchors, and a semi-cover. The algorithm then generalizes the aforementioned reduction procedure in several aspects. The attributes in the chosen patches are simply the attributes of the set H. This results in evenly dividing the set of available servers among the possible heavy tuples over the attributes in H. Before applying the semi-joins, the algorithm performs the following extra step: every group of servers, for some heavy tuple, gets *unevenly* divided based on the attributes in the chosen anchors such that the higher the degree of the values in the anchors, the larger the subgroup of servers it gets assigned. The semi-join step gets applied in every subgroup separately, and afterwards, the simplified query gets evaluated using HyperCube parameterized with the chosen semi-cover and extra shares to cover the rest of the query.

The definition of γ is more complicated than the earlier considered numbers, but links quite directly to the underlying algorithm, similar to how the worst-case optimal single-round MPC algorithm relates to the definition of ψ. While this number is not quite as elegant as τ, ρ, or ψ, we believe that it brings us another step closer to a worst-case optimal algorithm for all join queries.

[1] Property $\tau \leq \rho$ for reduced acyclic queries was only conjectured in [6]. Interestingly, our construction in Section 6.2 gives a proof for this conjecture as a side result.

The paper is organized as follows: We give the preliminaries in Section 2. In Section 3 we introduce PAC, which is the central number in this paper. In Section 4 and Section 5 we introduce algorithmic techniques followed by the algorithm underlying γ. In Section 6 we relate γ to τ, ρ, and ψ. We conclude in Section 7.[2]

2 Preliminaries

For a set S, $\mathcal{P}^+(S) := \{A \mid \emptyset \subsetneq A \subseteq S\}$ denotes the set of all non-empty subsets of S. We write $[0,1]$ as abbreviation for the set $\{i \in \mathbb{Q}^+ \mid 0 \leq i \leq 1\}$ of rationals between (and including) 0 and 1.

Relations and join queries

Let **rels** and **attrs** be respectively an infinite domain of relation names and attribute names. For convenience of notation, specifically to avoid non-determinacy in some of our notations, we assume a total order $\leq_{\mathbf{rels}}$ over **rels**.

A *join query* q is a pair $(\mathsf{rels}_q, \mathsf{attrs}_q)$ consisting of a finite set $\mathsf{rels}_q \subseteq \mathbf{rels}$ of relation names and a mapping attrs_q associating each relation name $R \in \mathsf{rels}_q$ to a finite set $\mathsf{attrs}_q(R)$ of attribute names from **attrs**.[3]

For a join query q, we write $\mathsf{attrs}(q)$ and $\mathsf{ins}(q)$[4] to denote:

$$\mathsf{attrs}(q) := \{\alpha \mid \alpha \in \mathsf{attrs}_q(R), R \in \mathsf{rels}_q\} \text{ and}$$
$$\mathsf{ins}(q) := \{A \in \mathcal{P}^+(\mathsf{attrs}(q)) \mid A \subseteq \mathsf{attrs}_q(R), R \in \mathsf{rels}_q\},$$

where the former is the set of all attributes appearing in the relations of q, and the latter is the set of sets of attributes of q appearing together in at least one relation of q. For consistency, we also write $\mathsf{rels}(q)$ instead of rels_q.

Two attributes α, β from $\mathsf{attrs}(q)$ are called *adjacent* if they are different and there is a relation $R \in \mathsf{rels}(q)$ with $\{\alpha, \beta\} \subseteq \mathsf{attrs}_q(R)$. Moreover, for two different relations $R_1, R_2 \in \mathsf{rels}(q)$, we say that R_1 is *reducable* into R_2 *subject to a set of attributes* $A \subseteq \mathsf{attrs}(q)$, if $\mathsf{attrs}_q(R_1) \setminus A \subseteq \mathsf{attrs}_q(R_2)$. When $A = \emptyset$ we simply say that R_1 is *reducable* into R_2. Finally, we say that q is *reduced* if it has no *reducable* relations. When q is not reduced, we will write $(q)\downarrow$ to denote *the* reduced query of q, which is *the* query obtained from q by removing all reducable relations.[5]

Tuples and instances

A *tuple* $\boldsymbol{t}$ over a finite set of attributes $A \subseteq \mathbf{attrs}$ is a function from A to **dom**. We will use the term *A-tuple* to refer to a tuple $\boldsymbol{t}$ with specific domain A (in other words, $\boldsymbol{t}$ is precisely defined over the attributes of A). A finite set of tuples over the same set of attributes A is called a *relation instance over* A. For a tuple $\boldsymbol{t}$ over A and a subset $B \subseteq A$ of its attributes, we write $\boldsymbol{t}[B]$ to denote the tuple over B with, for every $\alpha \in B$, $\boldsymbol{t}[B](\alpha) = \boldsymbol{t}(\alpha)$ in which case we say that $\boldsymbol{t}$ is an *extension* of $\boldsymbol{t}[B]$. By convention, we consider every tuple an extension of itself and of the empty tuple. We say that an A-tuple $\boldsymbol{t}$ and a B-tuple $\boldsymbol{u}$ are *consistent* if $\boldsymbol{t}[A \cap B] = \boldsymbol{u}[A \cap B]$.

[2] Due to space limitations, most of the proofs are omitted and will appear in the full version of the paper.
[3] Our definition of join a query disallows self-joins and repeated occurrences of attributes in relations, which are restrictions also imposed by all previous papers on the topic.
[4] ins stands for **in**cidence (attribute) **s**ubsets.
[5] We assume that reductions are done using some deterministic procedure, making use of $\leq_{\mathbf{rels}}$ to break ties, such that $(q)\downarrow$ defines a unique query.

A *(database) instance* $D := (\llbracket \cdot \rrbracket_D)$ *for join query* q consists of a mapping $\llbracket \cdot \rrbracket_D$ associating every relation name $R \in \mathsf{rels}(q)$ with a relation instance $\llbracket R \rrbracket_D$ over $\mathsf{attrs}_q(R)$. We write $|D| := \sum_{R \in \mathsf{rels}(q)} |\llbracket R \rrbracket_D|$ to denote the total number of tuples in D.

For a tuple $\boldsymbol{t}$ over $A \in \mathsf{ins}(q)$, we use the following notations:

$$\mathsf{deg}_R^D(\boldsymbol{t}) := \begin{cases} |\{\boldsymbol{u} \in \llbracket R \rrbracket_D \mid \boldsymbol{u}[A] = \boldsymbol{t}\}| & A \subseteq \mathsf{attrs}_q(R) \\ 0 & \text{otherwise;} \end{cases}$$

$$\mathsf{deg}^D(\boldsymbol{t}) := \sum_{R \in \mathsf{rels}(q)} \mathsf{deg}_R^D(\boldsymbol{t}); \text{ and}$$

$$\mathsf{tup}^D(A) := \{\boldsymbol{u}[A] \mid \boldsymbol{u} \in \llbracket R \rrbracket_D, A \subseteq \mathsf{attrs}_q(R), R \in \mathsf{rels}(q)\},$$

denoting, respectively, number of tuples in $\llbracket R \rrbracket_D$ that are extensions of $\boldsymbol{t}$; number of tuples in D that are extensions of $\boldsymbol{t}$; and the set of all A-tuples for which an extension in D exists.

Join query semantics

Given an instance D for some join query q and a tuple $\boldsymbol{t}$ over $A \subseteq \mathsf{attrs}(q)$, we say that $\boldsymbol{t}$ is *consistent* with D if, for every $R \in \mathsf{rels}(q)$, $\llbracket R \rrbracket_D$ contains a tuple that is consistent with $\boldsymbol{t}$. We denote the set of all A-tuples consistent with D as $\mathsf{joins}^D(A) := \{\boldsymbol{t} \mid \boldsymbol{t} \text{ is an } A\text{-tuple consistent with } D\}$.

The *output* of a join query q over an instance D can then be defined as $\llbracket q \rrbracket_D := \mathsf{joins}^D(\mathsf{attrs}(q))$.

Weight mappings

In this paper we will often consider weight mappings for join queries q, which are functions $f : A \to [0, 1]$ over some set of attributes $A \subseteq \mathsf{attrs}(q)$ associating a non-negative rational weight $f(\alpha) \in [0, 1]$ to every attribute $\alpha \in A$. Remark that we do not require a weight mapping to assign weights to all attributes of the query. Henceforth, given a particular weight mapping f for q, we will use A_f to denote the set of attributes f is defined over. For simplicity of notation, we often write $f(B)$ with a set of attributes $B \subseteq A_f$, to denote the sum $f(B) = \sum_{\alpha \in B} f(\alpha)$ of the respective attribute weights. We remark that $f(B)$ can be larger than 1. Moreover, we associate a cost $\mathsf{c}(f)$ to a weight mapping f with $\mathsf{c}(f) := \max\{1, f(A_f)\}$.

We say that a weight mapping f for q *covers a relation* $R \in \mathsf{rels}(q)$ if $f(\mathsf{attrs}_q(R) \cap A_f) \geq 1$. Given a weight mapping f and a join query q (it is possible that f is not defined for q), we define $\mathsf{cover}_{q,f}$ to be the join-query with:

$$\mathsf{rels}(\mathsf{cover}_{q,f}) := \{R \in \mathsf{rels}(q) \mid R \text{ is covered by } f\} \text{ and}$$

$$\mathsf{attrs}_{\mathsf{cover}_{q,f}}(R) := \mathsf{attrs}_q(R), \text{ for every } R \in \mathsf{rels}(\mathsf{cover}_{q,f}).$$

Consistent with the literature, we call f a *fractional vertex cover for* q if $A_f = \mathsf{attrs}(q)$ and f covers every relation of q (in which case $\mathsf{cover}_{q,f}$ is query q itself and we say that f is a cover for q).[6]

[6] We note that every fractional vertex cover for a query q is a cover for q and that every cover f for q can be extended to a fractional vertex cover f' of q with same cost (*i.e.,* $f(A_f) = f'(\mathsf{attrs}(q))$) by assigning a weight $f'(\alpha) = 0$ to attributes $\alpha \in \mathsf{attrs}(q) \setminus A_f$.

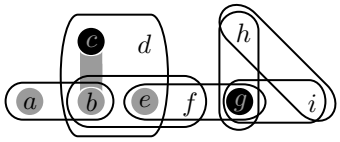


Figure 2 A visual depiction of configuration $\mathcal{C}_\otimes$ of $q_\otimes$ in which grey-scale shades represent spectrum values, with white, grey, and black representing, respectively, the values 1, $\frac{1}{2}$, and 0. Remark that the absence of a shade between sets of variables thus means a weight of 1. For instance, $\mathcal{C}_\otimes(\{g\}) = 0$, $\mathcal{C}_\otimes(\{b\}) = \frac{1}{2}$, $\mathcal{C}_\otimes(\{d\}) = 1$, $\mathcal{C}_\otimes(\{b,c\}) = \frac{1}{2}$, while $\mathcal{C}_\otimes(\{c,e\}) = 1$.

Residual queries

For a join query q and subset $A \subseteq \mathsf{attrs}(q)$, the *residual query* q_A is the join query with properties

$$\mathsf{rels}(q_A) := \{R \in \mathsf{rels}(q) \mid \mathsf{attrs}_q(R) \not\subseteq A\} \text{ and}$$
$$\mathsf{attrs}_{q_A}(R) := \mathsf{attrs}_q(R) \setminus A, \text{ for every } R \in \mathsf{rels}(q_A).$$

In Figure 1c, we give the residual query $q_{\otimes\{e,g\}}$. Notice that $q_{\otimes\{e,g\}}$ also exemplifies that the residual query of a reduced query is not necessarily reduced.

3 Towards PAC

We define the PAC number for join queries. The PAC number is defined relative to a sequence $\boldsymbol{r}$ of rationals that we call a spectrum. More precisely, a *spectrum* $\boldsymbol{r} = r_1, \ldots, r_k$ is a finite sequence of strictly increasing weights from $[0,1]$ with $r_1 = 0$ and $r_k = 1$.

3.1 Configurations

A *configuration* $\mathcal{C}$ of a join query q is a monotone mapping $\mathcal{C} : \mathsf{ins}(q) \to [0,1]$ that assigns a weight to every set of attributes in q appearing together in at least one relation of q. By *monotone* we mean that for sets $A, B \in \mathsf{ins}(q)$ and $B \subseteq A$: $\mathcal{C}(B) \leq \mathcal{C}(A)$. In addition, we require $\mathcal{C}(\mathsf{attrs}_q(R)) = 1$ for relations $R \in \mathsf{rels}(q)$ for which no relation R' with $\mathsf{attrs}_q(R) \subsetneq \mathsf{attrs}_q(R')$ exists. We say that $\mathcal{C}$ *adheres* to $\boldsymbol{r}$ if its image contains only weights occurring in $\boldsymbol{r}$, *i.e.,* $\mathcal{C}(A) \in \boldsymbol{r}$ for every $A \in \mathsf{ins}(q)$.

We will use configurations as a type of histogram for database fragments, More precisely, $\mathcal{C}(B)$ will mean that the number of tuples extending any specific B-tuple is below a threshold value based on $\mathcal{C}(B)$ and above a threshold value based on the element following $\mathcal{C}(B)$ in $\boldsymbol{r}$ (if $\mathcal{C}(B) < 1$). The precise threshold values are unimportant for the moment and will be defined in Section 4.

We call a set of attributes $A \in \mathsf{ins}(q)$ *heavy* w.r.t. $\mathcal{C}$ if $\mathcal{C}(A) < 1$ and that an individual attribute $\alpha \in \mathsf{attrs}(q)$ is *heavy* if $\{\alpha\}$ is heavy. We say that a relation $R \in \mathsf{rels}(q)$ is *heavy* w.r.t. $\mathcal{C}$ if all its attributes are heavy w.r.t. $\mathcal{C}$. We denote the set of all heavy attributes and heavy relations of q w.r.t. $\mathcal{C}$ by $\mathsf{Heavyattrs}(q, \mathcal{C})$ and $\mathsf{Heavyrels}(q, \mathcal{C})$, respectively.

▶ **Example 1.** An example of a configuration for $q_\otimes$ is depicted in Figure 2. For this configuration, $\mathcal{C}_\otimes$, $\mathsf{Heavyattrs}(q_\otimes, \mathcal{C}_\otimes) = \{a, b, c, e, g\}$ and $\mathsf{Heavyrels}(q_\otimes, \mathcal{C}_\otimes) = \{R_1\}$. The sets $\{b,c\}$, $\{a\}, \{b\}, \{c\}, \{e\}, \{g\}$ of $\mathsf{ins}(q_\otimes)$ are the (only) heavy sets of attributes for $q_\otimes$ w.r.t. $\mathcal{C}_\otimes$.

Under the presence of a configuration, we will not be interested in arbitrary weight mappings for q, but only in those that are compatible with $\mathcal{C}$ in the following sense: A weight mapping f for q is *compatible* with $\mathcal{C}$ if $\min\{1, f(B)\} \leq \mathcal{C}(B)$ for every $B \in \mathsf{ins}(q)$ with $B \subseteq A_f$.

▶ **Example 2.** Weight mapping $f_\otimes$ for $q_\otimes$ given in Figure 1b is not compatible with configuration $\mathcal{C}_\otimes$ for $q_\otimes$ in Figure 2 because $f_\otimes(a) = \frac{1}{4} > 0 = \mathcal{C}_\otimes(\{a\})$. An example of a weight mapping for $q_\otimes$ that is compatible with $\mathcal{C}_\otimes$ is given in Figure 3. We remark though that $f'_\otimes$ is not a fractional vertex cover for $q_\otimes$.

In our algorithm, compatible weight mappings will be used like fractional vertex covers in HyperCube over skew-free databases (a formal argument is given in Section 4.4). Since compatible weight mappings covering all relations of the query do not always exist we next introduce two new concepts: *Anchors* and *Patches* that, intuitively, allow to *cover* the remaining relations of the query. In fact, even when compatible weight mappings exist that cover the entire query, they are not necessarily the best choice cost-wise.

3.2 Anchors

We call a pair (X, Z) of non-empty sets of attributes of q an *anchor for* q and refer to the relations $R \in \mathsf{rels}(q)$ with $X \cup Z \subseteq \mathsf{attrs}_q(R)$ as relations *anchored* by (X, Z). From now on we will only consider anchors for which at least one anchored relation exists. We remark that X and Z do not need to be disjoint.

For a set $\mathcal{A}$ of anchors for q we write $\mathsf{rels}(\mathcal{A})$ to denote the set of all anchored relations, that is, $\mathsf{rels}(\mathcal{A}) := \{R \in \mathsf{rels}(q) \mid X \cup Z \subseteq \mathsf{attrs}_q(R), (X, Z) \in \mathcal{A}\}$. We write $H(\mathcal{A})$ to denote the union of the Z's in $\mathcal{A}$, that is, $H(\mathcal{A}) := \{\alpha \mid \alpha \in Z, (X, Z) \in \mathcal{A}\}$.

We say that an anchor (X, Z) of q is *compatible* with $\mathcal{C}$ if $\mathcal{C}(X) = 1$ and $\mathcal{C}(Z) < 1$, and we call a (possibly empty) set of anchors for q that are compatible with $\mathcal{C}$ an *anchoring of* q *w.r.t.* $\mathcal{C}$. Moreover, we associate a cost $\mathsf{c}(\mathcal{A})$ to such an anchoring $\mathcal{A}$ with $\mathsf{c}(\mathcal{A}) := |\mathcal{A}|$.

▶ **Example 3.** Set $\mathcal{A}_\otimes := \{(\{b, e\}, \{e\})\}$ is an example anchoring of $q_\otimes$ w.r.t. $\mathcal{C}_\otimes$ with $\mathsf{rels}(\mathcal{A}_\otimes) = \{R_2, R_3\}$ and $\mathsf{c}(\mathcal{A}_\otimes) = 1$. Sets $(\{b\}, \{e\})$ and $(\{h\}, \{i\})$ are examples of anchors for $q_\otimes$ not compatible with $\mathcal{C}_\otimes$.

3.3 Patches

Formally, a *patch* $\mathcal{P}$ for q (w.r.t. $\mathcal{C}$) is a subset of $\mathsf{ins}(q)$ whose elements are all heavy and disjoint. We write $\mathsf{attrs}(\mathcal{P})$ for the set of all attributes occurring in elements of $\mathcal{P}$, *i.e.,* $\mathsf{attrs}(\mathcal{P}) := \bigcup_{B \in \mathcal{P}} B$. We also associate a cost to every set $B \in \mathcal{P}$ with $\mathsf{c}(B) := r_{i+1}$ where $r_i = \mathcal{C}(B)$. We recall that, by definition of a patch, $\mathcal{C}(B) < 1$, hence r_{i+1} indeed always exists. The cost of a patch $\mathcal{P}$, denoted $\mathsf{c}(\mathcal{P})$, is then defined as $\mathsf{c}(\mathcal{P}) := \sum_{B \in \mathcal{P}} \mathsf{c}(B)$.

▶ **Example 4.** For an example of a patch for $q_\otimes$, consider the set $\mathcal{P} = \{\{g\}, \{e\}, \{b, c\}\}$. Then, for $q_\otimes$ w.r.t. $\mathcal{C}_\otimes$ adhering to spectrum $0, \frac{1}{2}, 1$, we have $\mathsf{c}(\mathcal{P}) = \frac{5}{2}$. However, if the considered spectrum had weights $0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$, $\mathsf{c}(\mathcal{P})$ would be $\frac{7}{4}$. Furthermore, we remark that sets $\{\{b, e\}, \{b\}\}$ and $\{\{b, e\}\}$ cannot be chosen as patches for $q_\otimes$ w.r.t. $\mathcal{C}_\otimes$ because their elements are not disjoint (for the former) or not all heavy (for the latter).

3.4 Semi-Covers and Solutions

Finally, we combine all concepts in a *solution* for a query q w.r.t. a configuration $\mathcal{C}$. Before giving the definition, we make formal what we expect from a weight mapping not covering all relations:

▶ **Definition 5.** *We call a weight mapping f a* semi-cover *for q w.r.t. a configuration $\mathcal{C}$ of q, a set $H \subseteq \mathsf{attrs}(q)$, and a set $S \subseteq \mathsf{rels}(q)$, if $A_f \cap H = \emptyset$ and for every relation $R \in \mathsf{rels}(q) \setminus \mathsf{Heavyrels}(q, \mathcal{C})$, there is a relation $R' \in \mathsf{rels}(q)$ such that R is reducable into R' subject to H, and R' is either covered by f or in the set S.*

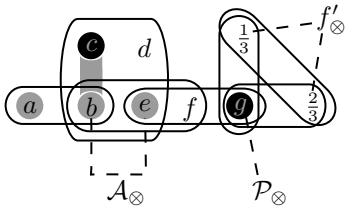


Figure 3 A visual depiction of the configuration $\mathcal{C}_\otimes$ along with a solution for $q_\otimes := R_1[a,b] \bowtie R_2[b,c,d,e] \bowtie R_3[b,e,f] \bowtie R_4[e,f,g] \bowtie R_5[g,h] \bowtie R_6[g,i] \bowtie R_7[h,i]$ w.r.t. $\mathcal{C}_\otimes$.

▶ **Example 6.** Remark that $f_\otimes$ is a semi-cover for $q_\otimes$ w.r.t. any configuration of $q_\otimes$, any set H, and any set S because $f_\otimes$ covers $q_\otimes$ (*i.e.,* $f_\otimes$ is a fractional vertex cover for $q_\otimes$).

For a more interesting example, consider the weight mapping $f'_\otimes$ in Figure 3. We can see that R_7 is the only relation that is covered by $f'_\otimes$. Now, take H to be $\{e, g\}$ and S to be $\{R_2, R_3\}$. Then $f'_\otimes$ is a semi-cover for $q_\otimes$ w.r.t. $\mathcal{C}_\otimes$, H, and S. Indeed, $R_1 \in \mathsf{Heavyrels}(q_\otimes, \mathcal{C}_\otimes)$, R_4 is reducable into $R_3 \in S$ subject to H, both R_5 and R_6 are reducable into R_7 subject to H, and R_7 is covered by $f'_\otimes$.

A *solution for q w.r.t. $\mathcal{C}$* is a triple $(\mathcal{A}, f, \mathcal{P})$ with $\mathcal{A}$ an anchoring of q w.r.t. $\mathcal{C}$, $\mathcal{P}$ a patch for q w.r.t. $\mathcal{C}$, and f a compatible semi-cover for q w.r.t. $\mathcal{C}$, set $H(\mathcal{A}) \cup \mathsf{attrs}(\mathcal{P})$, and set $\mathsf{rels}(\mathcal{A})$. The cost we associate with a solution $\mathcal{S}$ is defined $\mathsf{c}(\mathcal{S}) := \mathsf{c}(\mathcal{A}) + \mathsf{c}(f) + \mathsf{c}(\mathcal{P})$.

3.5 The PAC Number

As mentioned earlier, configurations correspond to database fragments. We will show later that the cost of a solution w.r.t. a given configuration is the cost of computing the query over the corresponding fragment using the different components of the solution. We can now define the PAC number $\gamma(q)$ for a join query q w.r.t. a spectrum $\boldsymbol{r}$,[7] as

$$\gamma(q) := \max_{\mathcal{C}} \left\{ \min_{\mathcal{A},f,\mathcal{P}} \left\{ \mathsf{c}(\mathcal{A}) + \mathsf{c}(f) + \mathsf{c}(\mathcal{P}) \right\} \right\}$$

with $\mathcal{C}$ ranging over all configurations of q adhering to $\boldsymbol{r}$ and $(\mathcal{A}, f, \mathcal{P})$ ranging over all possible solutions for q w.r.t. $\mathcal{C}$. Intuitively, the formula of the PAC number states that the cost of a query q is determined (and bounded) by its most computationally difficult configuration. Since there could be multiple solutions w.r.t. the same configuration, we are only interested in any solution with minimal cost.

▶ **Example 7.** Following from Examples 3 and 6, we see that $\mathcal{S}_\otimes := (\mathcal{A}_\otimes, f'_\otimes, \mathcal{P}_\otimes)$ with $\mathcal{P}_\otimes := \{\{g\}\}$ is a solution for $q_\otimes$ w.r.t. $\mathcal{C}_\otimes$ with $\mathsf{c}(\mathcal{S}_\otimes) = \frac{5}{2}$. A visual depiction of $\mathcal{S}_\otimes$ is given in Figure 3. Notice that every other solution for $q_\otimes$ does not have lower cost w.r.t. $\mathcal{C}_\otimes$.

The reader may wonder if the complexity of the PAC number is justified and whether all of its concepts are necessary. The answer to this question is open, but we will show in the full version of the paper that all obvious simplifications of our number, like restricting/forbidding anchors, considering only spectrum $\boldsymbol{r} = 0, 1$, etc, all give a number that is *strictly less tight* than γ for some join queries.

[7] To be precise, $\gamma(q)$ should be parameterized with the spectrum $\boldsymbol{r}$ considered. We omit this parameterization as it will significantly overload the notation throughout the paper. Except where specified differently all results hold for any choice of $\boldsymbol{r}$.

The remainder of the paper is devoted to showing the below theorem and how γ relates to the existing numbers τ, ρ, and ψ.

▶ **Theorem 8.** *Let q be a join query, D an instance, and $\boldsymbol{r}$ a fixed spectrum. Then, q can be computed over D using p servers in three rounds with load $\tilde{\mathcal{O}}(|D|/p^{1/\gamma(q)})$ with high probability.*

4 General Techniques

In our algorithm, we will make use of techniques and insights that follow directly or with minor modifications from classical results in the literature. In this section, we state these results.

4.1 Database Fragments

Like several of the earlier proposed algorithms in the literature [10, 7, 12, 8], our algorithm will first divide the input database instance in sub-databases. This is where configurations adhering to a spectrum become useful: The consecutive weights in a spectrum $\boldsymbol{r} = r_1, \ldots, r_k$ define intervals of threshold values that can be used to partition sets of tuples based on their degrees. With this intuition in place, a configuration adhering to $\boldsymbol{r}$ thus pinpoints an interval $\mathcal{C}(A) = r_i$ for all sets $A \in \mathsf{ins}(q)$. The intervals implied by a spectrum are based on the weights in $\boldsymbol{r}$, but also take D, p and q into account. Formally, a tuple $\boldsymbol{t} \in \mathsf{tup}^D(A)$ is in the interval r_i of $\boldsymbol{r}$ if its degree $\mathsf{deg}^D(\boldsymbol{t})$ is below (or equal to) $|D|/p^{r_i/\gamma(q)}$ and (if $i < k$) strictly above $|D|/p^{r_{i+1}/\gamma(q)}$. Henceforth, we will refer to the subset of A-tuples in D in the interval $\mathcal{C}(A)$ as $F^D(A, \mathcal{C})$.

We can now define a database instance $\mathcal{F} = (\llbracket\cdot\rrbracket_{\mathcal{F}})$ over q with $\llbracket R\rrbracket_{\mathcal{F}} := \{\boldsymbol{t} \in \llbracket R\rrbracket_D \mid \boldsymbol{t}[A] \in F^D(A, \mathcal{C}),\ \text{for every } A \subseteq \mathsf{attrs}_q(R)\}$. Intuitively, $\mathcal{F}$ contains all tuples from D that respect *all* degree constraints imposed by $\mathcal{C}$. Henceforth, we will refer to $\mathcal{F}$ as $\mathsf{Fragment}(D, \boldsymbol{r}, \mathcal{C})$. We remark that some care must be taken when interpreting the imposed degree constraints in the context of $\mathcal{F}$, as they are defined w.r.t. D. For example, for a tuple $\boldsymbol{t} \in \llbracket R\rrbracket_{\mathcal{F}}$ it does not necessarily follow from $\mathsf{deg}^D(\boldsymbol{t}) > m$ that $\mathsf{deg}^{\mathcal{F}}(\boldsymbol{t}) > m$.

The next proposition, which states that the output of q over D is the same as the combined output of q over the different fragments of D, justifies our definition:

▶ **Proposition 9.** *For a join query q, an instance D for q, and a spectrum $\boldsymbol{r}$, we have*

$$\llbracket q\rrbracket_D = \bigcup_{\mathcal{C} \in \mathsf{confs}(\boldsymbol{r})} \llbracket q\rrbracket_{\mathsf{Fragment}(D, \boldsymbol{r}, \mathcal{C})}$$

with $\mathcal{C}$ ranging over all configurations for q adhering to $\boldsymbol{r}$. Moreover, when $\boldsymbol{r}$ is fixed, $|\mathsf{confs}(\boldsymbol{r})|$ is constant, with $\mathsf{confs}(\boldsymbol{r})$ denoting the set of configurations adhering to $\boldsymbol{r}$.

4.2 Tuple-Restricted Database Fragments

Sometimes we will consider a value assignment for some of the attributes of q (defined as a tuple $\boldsymbol{h}$ over a subset A of $\mathsf{attrs}(q)$), and then be interested in the subinstance of an instance D for q containing all and only those tuples of D consistent with $\boldsymbol{h}$. Formally, this instance for q is defined as $\mathcal{F} := (\llbracket\cdot\rrbracket_{\mathcal{F}})$ with $\llbracket R\rrbracket_{\mathcal{F}} := \{\boldsymbol{t} \in \llbracket R\rrbracket_D \mid \boldsymbol{t} \text{ consistent with } \boldsymbol{h}\}$. Henceforth, we refer to $\mathcal{F}$ by $\mathsf{Subfragment}(D, \boldsymbol{h})$. Accordingly, we can easily see that computing $\llbracket q\rrbracket_D$ amounts to the following:

▶ **Proposition 10.** *Let q be a join query, D an instance for q, and $A \subseteq \mathsf{attrs}(q)$, we have $\llbracket q\rrbracket_D = \bigcup_{\boldsymbol{t} \in \mathsf{joins}^D(A)} \llbracket q\rrbracket_{\mathsf{Subfragment}(D, \boldsymbol{t})}$.*

4.3 Semi-Joins and Reductions

It is well known that semi-joins and intersections can be computed with a single round of communication using p servers with a load not exceeding $|D|/p$ w.h.p., and that guarded join queries (i.e., join queries having a single relation where all the other relations of the query are reducible into) can be computed with precisely two rounds and with same load guarantees independent of the number of reducible relations [10]. The latter follows from the observation that a reducible relation with its guard is a special type of semi-join and we can compute all these semi-joins at the same time in one round. Since after this step there will exist many copies of the guard relation (one for each reducible relation) a second step is needed to compute the intersection of its copies. Formally,

▶ **Proposition 11.** *A database D for a join query q can be translated to a database $(D)\downarrow$ for $(q)\downarrow$ using p servers in two rounds with load not exceeding $|D|/p$ w.h.p.*

4.4 HyperCube-Style Query Evaluation

Compatible weight mappings f for q w.r.t. a configuration $\mathcal{C}$ are essentially generalizations of the weight mappings used by the HyperCube algorithm [9] and the degree constraints associated with so-called skew-free databases w.r.t. these weight mappings, as made formal by the below proposition.

▶ **Proposition 12.** *Let q be a join query, D an instance for q, $\boldsymbol{r}$ a spectrum, $\mathcal{C}$ a configuration of q adhering to $\boldsymbol{r}$, $\mathcal{F}$ the fragment $\mathsf{Fragment}(D, \boldsymbol{r}, \mathcal{C})$, and f a compatible weight mapping for q w.r.t. $\mathcal{C}$ with $f(A_f) \leq \gamma(q)$. Then, $\mathsf{cover}_{q,f}$ can be computed over $\mathcal{F}$ using $p^{f(A_f)/\gamma(q)}$ servers in one round with load $\tilde{\mathcal{O}}(|D|/p^{1/\gamma(q)})$ w.h.p.*

Proposition 12 is based on the analysis of the HyperCube algorithm [9]. A full proof is in Appendix A.1.

4.5 Join Query Decompositions

Some join queries can be decomposed into a set of subqueries, and then computed in one round by computing each subquery in that round, over its own share of servers. This is formulated in Proposition 13. For this, we say that a set of queries $Q = \{q_1, \ldots, q_n\}$ is a *decomposition* of a join query q if $\mathsf{rels}(q) = \bigcup_{q_i} \mathsf{rels}(q_i)$ and for every $R \in \mathsf{rels}(q_i)$ for $i = 1, \ldots, n$, we have $\mathsf{attrs}_{q_i}(R) = \mathsf{attrs}_q(R)$.

▶ **Proposition 13.** *Let q be a join query, D an instance for q, and $Q = \{q_1, \ldots, q_n\}$ a decomposition of q. Suppose for $i = 1, \ldots, n$, we know that q_i is computable over D using p_i servers in one round with load $\tilde{\mathcal{O}}(L_i)$. Then, q can be computed over D using $\prod_i p_i$ servers in one round with load $\tilde{\mathcal{O}}(\max_i\{L_i\})$ w.h.p.*

The observation follows from viewing the evaluation of a join query as a Cartesian product of its decomposition. When distinct queries share common attributes, the Cartesian product returns copies of those attributes (one for each occurrence in a query).

The formal argument of the above load analysis is based on a result in [8, Lemma 3.2], in which this result is proven for decompositions with join queries not sharing any attributes. It however follows immediately that this technique also works if the involved queries share attributes, only requiring an additional local computation step during which tuples are combined taking the values of common attributes into account (see [6] for further discussion).

5 Algorithm

In this section, we show how to compute a join query q over a specific fragment $\mathcal{F} := \mathsf{Fragment}(D, \boldsymbol{r}, \mathcal{C})$ of the database instance D using p servers, based on some spectrum $\boldsymbol{r}$ and configuration $\mathcal{C}$ for q adhering to $\boldsymbol{r}$ using a specific solution $\mathcal{S}$ for q w.r.t. $\mathcal{C}$. More precisely, this section is entirely devoted to showing the following:

▶ **Theorem 14.** *Let q be a join query, D an instance, and $\mathcal{C}$ a configuration for q adhering to a spectrum $\boldsymbol{r}$. For any solution $\mathcal{S}$ for q w.r.t. $\mathcal{C}$, we can compute q over $\mathsf{Fragment}(D, \boldsymbol{r}, \mathcal{C})$ using p servers in three rounds with load $\tilde{\mathcal{O}}(|D|/p^{1/\mathsf{c}(\mathcal{S})})$ with high probability.*

When $\boldsymbol{r}$ is fixed, Theorem 8 is then a corollary of Theorem 14 and Proposition 9.

It is worth mentioning that the number "three" refers to the number of communication rounds of the described algorithm. For the sake of precision, we remark that extra rounds are needed if the required degree information is not readily present. It is not uncommon to assume that this information is available for each server at the start of the algorithm [7].

5.1 Server Organization

Towards the algorithm proving Theorem 14, we first show how the p servers are organized. For this, let $H := H(\mathcal{A}) \cup \mathsf{attrs}(\mathcal{P})$. We partition the available p servers in disjoint groups, with equally many groups as there are H-tuples in the fragment $\mathcal{F}$. We refer to this set of H-tuples by $\boldsymbol{H}$; formally defined as follows:

$$\boldsymbol{H} := \{\boldsymbol{t} \mid \boldsymbol{t}[Z] \in \mathsf{tup}^{\mathcal{F}}(Z) \text{ for every } (X, Z) \in \mathcal{A} \text{ and } \boldsymbol{t}[B] \in \mathsf{tup}^{\mathcal{F}}(B) \text{ for every } B \in \mathcal{P}\}.$$

For each such H-tuple $\boldsymbol{h} \in \boldsymbol{H}$, the associated group consists of precisely $p_{\boldsymbol{h}}$ servers:

$$p_{\boldsymbol{h}} := \underbrace{p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})}}_{p_f} \times \prod_{a=(X,Z)\in\mathcal{A}} \underbrace{p^{1/\mathsf{c}(\mathcal{S})} \cdot \frac{\mathsf{deg}^{D}(\boldsymbol{h}[Z])}{\mathsf{degs}^{\mathcal{F}}(Z)}}_{p_a},$$

with $\mathsf{degs}^{\mathcal{F}}(Z) := \sum_{\boldsymbol{t}\in\mathsf{tup}^{\mathcal{F}}(Z)} \mathsf{deg}^{D}(\boldsymbol{t})$ (intuitively denoting the total number of Z-tuples having values in fragment $\mathcal{F}$). The next lemma shows some desirable properties of the server organization (a proof is given in Appendix B.1):

▶ **Lemma 15.**
1. $\sum_{\boldsymbol{h}\in\boldsymbol{H}} p_{\boldsymbol{h}} \leq p$; *and*
2. $p_{\boldsymbol{h}} \geq p^{1/\mathsf{c}(\mathcal{S})}$.

5.2 Communication and Computation

Our algorithm proceeds with a computation of q for each instance $\mathsf{Subfragment}(\mathcal{F}, \boldsymbol{h})$ apart, using its designated group of $p_{\boldsymbol{h}}$ servers. Correctness of this approach follows directly from Lemma 15 (1), Proposition 10 and the following lemma.

▶ **Lemma 16.** $\bigcup_{\boldsymbol{t}\in\boldsymbol{H}} [\![q]\!]_{\mathsf{Subfragment}(\mathcal{F},\boldsymbol{t})} = \bigcup_{\boldsymbol{t}\in\mathsf{joins}^{\mathcal{F}}(H)} [\![q]\!]_{\mathsf{Subfragment}(\mathcal{F},\boldsymbol{t})}$.

Now the computation of q over $F := \mathsf{Subfragment}(\mathcal{F}, \boldsymbol{h})$ for a specific $\boldsymbol{h} \in \boldsymbol{H}$ goes as follows. Here, we assume w.l.o.g. that all servers are aware of the tuples in $\boldsymbol{H}$ as well as their associated server groups.

Step 1: Instance F is transformed in instance $F_{\boldsymbol{h}} := (F_H)\downarrow$ for $q_{\boldsymbol{h}} := (q_H)\downarrow$ by dropping the attributes of $\boldsymbol{h}$ from all relations in F and computing semi-joins as explained in Section 4.3 using the $p_{\boldsymbol{h}}$ servers.

Step 2: The join query $q_{\boldsymbol{h}}$ is now viewed as a set of join queries $\{q_f, q_{a_1}, \ldots, q_{a_\ell}, q_b\}$ (clearly a decomposition of q), with a query q_{a_i} for every anchor $a_i \in \mathcal{A}$ having precisely the relations of $q_{\boldsymbol{h}}$ anchored by a_i; with q_b the query consisting of all relations from $\mathsf{Heavyrels}(q, \mathcal{C})$ remaining in $q_{\boldsymbol{h}}$; and with $q_f := \mathsf{cover}_{q_{\boldsymbol{h}}, f}$. In this step, we broadcast all tuples from relations in q_b to all servers of the group. Moreover, we hash-partition the relations of q_f over the share p_f using HyperCube parameterized with f, and, for each anchor $a = (X, Z) \in \mathcal{A}$, we hash-partition the relations of the query q_a over share p_a using the values of attributes X (pretending they are a single value).[8]

5.3 Load Analysis

We argue that the outlined algorithm computes q over F using $p_{\boldsymbol{h}}$ servers with the desired load. A key aspect of the analysis is that our algorithm requires only communication to servers in the group $p_{\boldsymbol{h}}$. In other words, although this algorithm will be applied in parallel for all tuples of $\boldsymbol{H}$ over their respective database fragments, the load of servers will be decided entirely by the specific run of the algorithm for the tuple $\boldsymbol{h}$ they are responsible for.

In the remainder of this subsection, we will thus focus on the load of the individual steps 1 and 2. For Step 1, the load follows directly from Proposition 11 and Lemma 15. We also remark that the result of a semi-join (and by extension the size of relations in $F_{\boldsymbol{h}}$) do not exceed the size of relations they are based on (the size of relations in F, respectively). For Step 2, the desired load follows from Proposition 13 and the fact that the individual queries q_{a_i} as well as q_f and q_b can be computed with desired load over their respective shares.

▶ **Lemma 17.** *The load of computing each of the following in one round is* $\tilde{\mathcal{O}}(|D|/p^{1/c(\mathcal{S})})$ *w.h.p.*

1. $[\![q_b]\!]_{F_{\boldsymbol{h}}}$ *using* 1 *server;*
2. $[\![q_f]\!]_{F_{\boldsymbol{h}}}$ *using* p_f *servers; and*
3. $[\![q_a]\!]_{F_{\boldsymbol{h}}}$ *using* p_a *servers, for each anchor* a*.*

A proof for Lemma 17 is given in Appendix B.2.

6 PAC in Relation to Other Numbers

In this section, we show how γ relates to the different existing numbers ψ, ρ, and τ, that have been considered in the literature. The main result of this section is the following:

▶ **Theorem 18.** *For any choice of spectrum:*

(a) $\rho(q) \leq \gamma(q) \leq \psi(q)$ *for all join queries* q*; and*

(b) $\tau(q) \leq \gamma(q)$ *for reduced join queries* q*.*

For any spectrum including weight $1/2$*:*

(i) $\rho(q) = \gamma(q)$ *for join queries* q *that are acyclic or graph-like;*

(ii) $\rho(q) = \gamma(q) < \psi(q)$ *for some join queries* q *that are neither acyclic nor graph-like.*

[8] We remark that it is not an issue if this requires applying hash functions over attributes α from H (which are not present in $q_{\boldsymbol{h}}$ nor $F_{\boldsymbol{h}}$) as in that case there is only one value to consider, namely $\boldsymbol{h}[\alpha]$.

Property $\rho(q) \leq \gamma(q)$ is a direct consequence of the correctness of our algorithm (see Theorem 8) and the known lower bound on the load of such algorithms based on $\rho(q)$ [10]. Properties $\tau(q) \leq \gamma(q)$ (for reduced queries q) and $\gamma(q) \leq \psi(q)$ follow from the below Lemmas 19 and 20. Intuitively, Lemma 19 shows that a solution w.r.t. the configuration that has no heavy attribute sets always makes use of a solution $\mathcal{S}$ whose semi-cover f is also a cover for q, thus with $\mathsf{c}(f) = \mathsf{c}(\mathcal{S})$.

▶ **Lemma 19.** *Let $\mathcal{C}$ be a configuration for some join query q with $\mathcal{C}(A) = 1$, for every $A \in \mathsf{ins}(q)$. Then, for every solution $\mathcal{S} := (\mathcal{A}, f, \mathcal{P})$ for q w.r.t. $\mathcal{C}$, we have the following properties:*
1. *$\mathcal{A} = \mathcal{P} = \emptyset$; and*
2. *f is a cover for q.*

To see $\gamma(q) \leq \psi(q)$, it suffices to consider a minimal spectrum $\boldsymbol{r} = 0, 1$ and observe that, for every configuration $\mathcal{C}$ adhering to $\boldsymbol{r}$, a fractional vertex cover for q_H is a semi-cover for q compatible with $\mathcal{C}$.

▶ **Lemma 20.** *Let f be a fractional vertex cover for q_H. Then, $(\emptyset, f, \emptyset)$ is a solution for q w.r.t. $\mathcal{C}$. In other words:*
1. *f is a semi-cover for q w.r.t. $\mathcal{C}$, $\emptyset$, and $\emptyset$; and*
2. *f is compatible with $\mathcal{C}$.*

For graph-like and acyclic join queries (which are the main classes of join queries for which the worst-case optimal load is known), we have an even stronger result, namely that the load of our algorithm matches the optimal load.

▶ **Theorem 21.** *For join queries q that are graph-like or acyclic we have that $\rho(q) = \gamma(q)$.*

In the next two subsections, we show why $\gamma(q) \leq \rho(q)$ for graph-like, respectively, acyclic join queries.

As for item (ii) in Theorem 18, it is enough to give an example of a join query showing that our algorithm is also optimal for join queries that are neither acyclic nor graph-like. A simple example is

$$q := R_1[a, b] \bowtie R_2[a, c] \bowtie R_3[b, c, d] \bowtie R_4[d, e].$$

Notice that $\tau(q) = 2$ and $\rho(q) = 2\frac{1}{2}$. With spectrum $\boldsymbol{r} = 0, \frac{1}{2}, 1$, we obtain that $\gamma(q) = 2\frac{1}{2}$, which is clearly the worst-case optimal cost for this join query. Verifying that $\gamma(q)$ yields this cost is left as an exercise for the reader. We remark that $\psi(q) = 3$ and thus that our algorithm has a strictly better load than any previously known algorithm that could compute this query.

6.1 Optimality for Graph-like Joins

To show $\gamma(q) \leq \rho(q)$ for reduced graph-like join queries, it is sufficient to consider spectrum $\boldsymbol{r} = 0, \frac{1}{2}, 1$. The remainder of this section will be devoted to showing for an arbitrary configuration $\mathcal{C}$ adhering to $\boldsymbol{r}$ that there exists a solution $\mathcal{S}$ with $\mathsf{c}(\mathcal{S}) \leq \rho(q)$.

For the construction, we need extra terminology and call an attribute $\alpha \in \mathsf{attrs}(q)$ based on $\mathcal{C}$

- *relevant* if it occurs in a relation $R \in \mathsf{rels}(q) \setminus \mathsf{Heavyrels}(q, \mathcal{C})$;
- *heavy* if it is relevant and $\mathcal{C}(\{\alpha\}) = 0$;
- *isolated* if $\mathcal{C}(\{\alpha\}) = 1$ and all adjacent attributes are heavy;
- *light* if it is relevant non-isolated and $\mathcal{C}(\{\alpha\}) > 0$.

We denote the set of isolated attributes by I, the light attributes by L, and the heavy attributes by H.

For the construction of $\mathcal{S} = (\mathcal{A}, f, \mathcal{P})$, we first choose an anchoring $\mathcal{A}$ for q. For this, let us call a relation $R \in \mathsf{rels}(q)$ an *anchor candidate* if it has an attribute from I (and therefore also an attribute from H). Now let $\mathcal{R}$ be any maximal set of anchor candidates with property that the relations in $\mathcal{R}$ share no attributes. From $\mathcal{R}$ we then construct the desired anchoring $\mathcal{A} := \{(\{\alpha\}, \{\beta\}) \mid R \in \mathcal{R}, \{\alpha\} = I \cap \mathsf{attrs}_q(R), \beta \in \mathsf{attrs}_q(R) \cap H\}$ of q. For convenience, we will refer by $I_{\mathcal{R}} \subseteq I$ to the isolated attributes occurring in $\mathcal{R}$ and by $H_{\mathcal{R}} \subseteq H$ to the remaining attributes in $\mathcal{R}$. As patch, we choose $\mathcal{P} = \{\{\alpha\} \mid \alpha \in H \setminus H_{\mathcal{R}}\}$. Finally, as weight mapping we take $f : L \cup (I \setminus I_{\mathcal{R}}) \to [0, 1]$ assigning weight 1 to all attributes in $I \setminus I_{\mathcal{R}}$ and weight $1/2$ to attributes in L. Now, the desired result follows from the next proposition.

▶ **Proposition 22.**

1. *f is compatible with $\mathcal{C}$;*
2. *f is a semi-cover for q w.r.t. $\mathcal{C}$, $H(\mathcal{A}) \cup \mathsf{attrs}(\mathcal{P})$, and $\mathsf{rels}(\mathcal{A})$;*
3. *$\mathsf{c}(\mathcal{S}) \leq \rho(q)$.*

6.2 Optimality for Acyclic Joins

To show $\gamma(q) \leq \rho(q)$ for reduced acyclic join queries q, it is sufficient to consider spectrum $\boldsymbol{r} = 0, 1$. In the remainder of the section, we show how, for an arbitrary configuration $\mathcal{C}$ adhering to $\boldsymbol{r}$, a solution $\mathcal{S}$ for q and $\mathcal{C}$ can be constructed with $\mathsf{c}(\mathcal{S}) \leq \rho(q)$.

Before going into the details of the construction, we remark that every reduced acyclic join query has an *integral* fractional edge cover [6]. That is, there is a set $\mathsf{EC} \subseteq \mathsf{rels}(q)$ with property $|\mathsf{EC}| = \rho(q)$ and every attribute $\alpha \in \mathsf{attrs}(q)$ must appear in at least one relation in EC. Since q is acyclic it also has a join-tree T [1]: a tree having the relations of q as nodes and such that any two relations $R_1, R_3 \in \mathsf{rels}(q)$ sharing an attribute $\alpha \in \mathsf{attrs}_q(R_1) \cap \mathsf{attrs}_q(R_3)$ are connected by a path in the tree with property that every relation R_2 on this path has $\alpha \in \mathsf{attrs}_q(R_2)$.

In the special case $\mathsf{EC} \subseteq \mathsf{Heavyrels}(q, \mathcal{C})$ it follows inevitably (by definition of $\mathsf{Heavyrels}(q, \mathcal{C})$) that $\mathsf{rels}(q) \subseteq \mathsf{Heavyrels}(q, \mathcal{C})$ and hence that $\mathcal{S} = (\emptyset, f : \emptyset \to [0, 1], \emptyset)$ is the desired solution with trivial cost $\mathsf{c}(\mathcal{S}) = 1 \leq \rho(q)$.

Therefore, we continue with the assumption that there is a relation $R \in \mathsf{EC} \setminus \mathsf{Heavyrels}(q, \mathcal{C})$ and we make use of this assumption to also assume w.l.o.g. an orientation of T making R its root. The solution that we will construct for this case is of the form $\mathcal{S} = (\mathcal{A}, f, \emptyset)$ with $f : A_f \to [0, 1] : \alpha \mapsto 1$, *i.e.,* with empty patch and a weight mapping using only weights 1.

Next, we describe an iterative procedure applied on T. During this procedure we pinpoint the anchors of $\mathcal{A}$ as well as the attributes of A_f. The procedure keeps track of a set V of already visited relations from $\mathsf{rels}(q)$, which is initially the empty set. The procedure continues as long as $V \neq \mathsf{rels}(q)$ but will eventually terminate as every step increases V by one or more relations. The procedure goes as follows: Take a relation $R_c \in \mathsf{rels}(q) \setminus V$ whose ancestors (according to T) are already in V. We distinguish between three cases:

1. R_c is the root of T;
2. a relation $R_{\mathsf{EC}} \in \mathsf{EC}$ exists that is a descendant of R_c (according to T) and there is a non-empty set $X \subseteq \mathsf{attrs}_q(R_c) \cap \mathsf{attrs}_q(R_{\mathsf{EC}})$ of attributes with $\mathcal{C}(X) = 1$; or
3. there is no such set of attributes for R_c.

In case (1), we know $R_c \in \mathsf{EC} \setminus \mathsf{Heavyrels}(q, \mathcal{C})$. We add R_c to V and one of its attributes $\alpha \in \mathsf{attrs}_q(R_c)$ with $\mathcal{C}(\{\alpha\}) = 1$ to A_f.

In case (2), note that it is possible that R_{EC} and R_c are the same relation. We can assume w.l.o.g. that none of the relations between R_{EC} and R_c in T are in EC (beside R_{EC} itself). Indeed, if there is such a relation $R' \in \mathsf{EC}$, we know that $X \subseteq \mathsf{attrs}_q(R')$ by definition of join-tree and thus we can substitute R_{EC} by R'. If there are multiple choices for the set X, we look at the relation R_X with $X \subseteq \mathsf{attrs}_q(R_X)$ occurring closest to the root in T (remark that, by definition of join-tree, R_X is unique). We then choose the set $X \subseteq \mathsf{attrs}_q(R_c) \cap \mathsf{attrs}_q(R_{\mathsf{EC}})$ with $\mathcal{C}(X) = 1$ whose R_X occurs closest to the root of T in T.

Now, if there is an attribute $\alpha \in X$ with $\mathcal{C}(\{\alpha\}) = 1$ and either R_X has no parent, *or*, R_X has a parent R_p with $\mathsf{attrs}_q(R_p) \cap \mathsf{attrs}_q(R_{\mathsf{EC}}) = \emptyset$, then in both cases we add α to A_f. Otherwise, we add (X, Z) as anchor to $\mathcal{A}$, with Z a singleton containing an attribute $\alpha \in X$ with $\mathcal{C}(\{\alpha\}) < 1$ (if R_X has no parent) or $Z = \mathsf{attrs}_q(R_p) \cap \mathsf{attrs}_q(R_{\mathsf{EC}})$ (if R_X has a parent R_p). We add all relations on the path from R_{EC} to R_c in T to V.

In case (3), we simply add R_c to V.

The aforementioned construction is indeed a solution w.r.t. $\mathcal{C}$ as stated in the following lemma.

▶ **Lemma 23.** *$(\mathcal{A}, f, \emptyset)$ is a solution for q w.r.t. $\mathcal{C}$.*

For the cost of $\mathsf{c}(\mathcal{S})$ we remark that every step of the iterative procedure contributing to $\mathcal{S}$ (namely cases (1 and 2) creates either one anchor *or* adds one attribute to A_f. Since both steps add exactly one relation from $\mathsf{EC} \setminus V$ to V, it is immediate that $|\mathcal{A}| + |A_f| \leq |\mathsf{EC}|$. Furthermore, since $|A_f| \geq 1$ (by choice of the root of T which falls in case (1)) we have that $\mathsf{c}(\mathcal{S}) = |\mathcal{A}| + |A_f| \leq |\mathsf{EC}|$, as desired.

7 Conclusion and Future Work

Our algorithm can compute every join query q in the constant-round MPC model with a load bounded by the PAC number $\gamma(q)$. Since $\gamma(q) \leq \psi(q)$, this load is at least as low as the optimal load for q in the single-round MPC model. Moreover, $\gamma(q) = \rho(q)$ for any join query for which the worst-case optimal load is known, including all join queries that are graph-like or acyclic. We remark that the queries in [6] shown to have an optimal load w.r.t. $\tau(q)$ have property $\tau(q) = \psi(q)$ and hence are also computed optimally by our algorithm. There is one exception: the Loomis-Whitney (LW) join [10]. For some configurations of an LW join query q, we only have to apply step 1 of our algorithm since the query $(q_H)\downarrow$ consists of exactly one relation. In that case, our algorithm requires more load than what is expected "optimally". The reason for this is that our analysis for step 1 is not fine-grained enough to show that, in some peculiar cases, step 1 can be computed with a better load guarantee. Capturing this case requires a more fine-grained analysis.

Although our algorithm is relatively simple (compared to other worst-case optimal algorithms described in the literature) the number γ is complicated and relative to a spectrum that needs to be decided in advance. Although the spectrum $0, 1/2, 1$ is sufficient for all the proven relationships with ψ, τ, and ρ, there exist queries for which this choice of spectrum is suboptimal. In general, it is not known if a bound on spectra can be assumed or if it is computable from the structure of the query. We remark that it is currently also still unknown what the worst-case optimal load for arbitrary join queries in the constant-round MPC model is.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

2 Foto N. Afrati and Jeffrey D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011. doi:10.1109/TKDE.2011.47.

3 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and transferability for conjunctive queries. In Tova Milo and Diego Calvanese, editors, *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS, 2015*, pages 47–58. ACM, 2015. doi:10.1145/2745754.2745759.

4 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, 2013*, pages 273–284. ACM, 2013. doi:10.1145/2463664.2465224.

5 Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 212–223. ACM, 2014. doi:10.1145/2594538.2594558.

6 Xiao Hu. Cover or pack: New upper and lower bounds for massively parallel joins. In Leonid Libkin, Reinhard Pichler, and Paolo Guagliardo, editors, *PODS'21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, 2021*, pages 181–198. ACM, 2021. doi:10.1145/3452021.3458319.

7 Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 417–428. ACM, 2017. doi:10.1145/3034786.3034788.

8 Bas Ketsman, Dan Suciu, and Yufei Tao. A near-optimal parallel algorithm for joining binary relations. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:6)2022.

9 Paraschos Koutris. *Query Processing for Massively Parallel Systems*. PhD thesis, University of Washington, USA, 2015. URL: https://hdl.handle.net/1773/33697.

10 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-case optimal algorithms for parallel query processing. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory, ICDT 2016*, volume 48 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICDT.2016.8.

11 Miao Qiao and Yufei Tao. Two-attribute skew free, isolated cp theorem, and massively parallel joins. In *PODS*, pages 166–180, 2021. doi:10.1145/3452021.3458321.

12 Yufei Tao. A simple parallel algorithm for natural joins on binary relations. In Carsten Lutz and Jean Christoph Jung, editors, *23rd International Conference on Database Theory, ICDT 2020*, volume 155 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICDT.2020.25.

13 Yufei Tao. Parallel acyclic joins with canonical edge covers. In Dan Olteanu and Nils Vortmeier, editors, *25th International Conference on Database Theory, ICDT 2022*, volume 220 of *LIPIcs*, pages 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICDT.2022.9.

A Proofs for Section 4

A.1 Proof of Proposition 12

One of the most well-known algorithms in the literature is the HyperCube algorithm [2, 9], which is known to compute join queries with a load based on shares assigned to the attributes of q given that some degree constraints on relations in the input database are true. The

shares assigned based on fractional vertex covers for q have a special interest, because they are sufficient to find an optimal assignment of shares. The following result (translated to our terminology) is known:

▶ **Theorem 24** ([9]). *Let q be a join query and D an instance for q. Let $p_{\alpha_1}, \ldots, p_{\alpha_k}$ be the shares of the HyperCube algorithm with each p_{α_i} is for a specific attribute $\alpha_i \in \mathsf{attrs}(q)$. Suppose that for every relation $R \in \mathsf{rels}(q)$ and every B-tuple $\boldsymbol{t}$ with $B \subseteq \mathsf{attrs}_q(R)$, we have $\mathsf{deg}_R^D(\boldsymbol{t}) \leq \frac{|[\![R]\!]_D|}{\beta^{|B|} \cdot \prod_{\alpha \in B} p_\alpha}$ for some constant $\beta > 0$. Then w.h.p. the maximum load per server is $\tilde{\mathcal{O}}(\max_{R \in \mathsf{rels}(q)}\{\frac{|[\![R]\!]_D|}{\prod_{\alpha \in \mathsf{attrs}_q(R)} p_\alpha}\})$.*

Its proof follows from the next lemma:

▶ **Lemma 25** ([9]). *Let q be a join query and D an instance for q. Let $R \in \mathsf{rels}(q)$ with $\alpha_1, \ldots, \alpha_r$ be the attributes of R listed in some order. Let $p_{\alpha_1}, \ldots, p_{\alpha_r}$ be integers and let $p = \prod_i p_{\alpha_i}$. Suppose that we hash each tuple $\boldsymbol{t} \in [\![R]\!]_D$ to the bin $(h_1(\boldsymbol{t}[\{\alpha_1\}]), \ldots, h_r(\boldsymbol{t}[\{\alpha_r\}]))$, where $h_1, \ldots, h_r$ are independent and perfectly random hash functions from the domain $\boldsymbol{dom}$ to $p_{\alpha_1}, \ldots, p_{\alpha_r}$ respectively. Then suppose that for every B-tuple $\boldsymbol{t}$ with $B \subseteq \mathsf{attrs}_q(R)$ we have*

$$\mathsf{deg}_R^D(\boldsymbol{t}) \leq \frac{|[\![R]\!]_D|}{\beta^{|B|} \cdot \prod_{\alpha \in B} p_\alpha}$$

for some constant $\beta > 0$. Then the probability that the maximum load exceeds $\tilde{\mathcal{O}}(\frac{|[\![R]\!]_D|}{p})$ is exponentially small in p.

Next, we show that compatible weight mappings resemble the degree constraints required by the HyperCube algorithm. More precisely, let q be a join query, D an instance for q, $\boldsymbol{r}$ a spectrum, $\mathcal{C}$ a configuration of q adhering to $\boldsymbol{r}$, $\mathcal{F}$ the fragment $\mathsf{Fragment}(D, \boldsymbol{r}, \mathcal{C})$, and f a compatible weight mapping for q w.r.t. $\mathcal{C}$ with $f(A_f) \leq \gamma(q)$ and with $A_f = \mathsf{attrs}(q)$. (Notice that in case $A_f \subsetneq \mathsf{attrs}(q)$, then we assume $f(\alpha) = 0$ for every $\alpha \notin \mathsf{attrs}(q)$.) Moreover, let p be some integer and $p_\alpha := p^{f(\alpha)/\gamma(q)}$ be the share used by HyperCube for attribute α. Now, suppose that a relation $R \in \mathsf{rels}(q)$ is covered by f (*i.e.,* $f(\mathsf{attrs}_q(R)) \geq 1$ and $R \in \mathsf{rels}(\mathsf{cover}_{q,f})$), then we argue next that w.h.p. the load of R over $\mathcal{F}$ using HyperCube does not exceed $\tilde{\mathcal{O}}(|D|/p^{1/\gamma(q)})$.

To compute the load of R over $\mathcal{F}$, we will rely on the HyperCube, as already mentioned, by assigning a share of $p_\alpha := p^{f(\alpha)/\gamma(q)}$ to every attribute α. When $f(\mathsf{attrs}_q(R)) \leq 1$, then we can verify that it satisfies the degree constraints of Lemma 25, and hence, the load of R is in $\tilde{\mathcal{O}}(|D|/p^{1/\gamma(q)})$. Indeed, since f is compatible w.r.t. $\mathcal{C}$ and $f(\mathsf{attrs}_q(R)) \leq 1$, we know that $f(B) \leq \mathcal{C}(B) \leq 1$ for every $B \subseteq \mathsf{attrs}_q(R)$. Accordingly, the following holds for any m: $m/p^{\mathcal{C}(B)/\gamma(q)} \leq m/p^{f(B)/\gamma(q)}$.

Since every B-tuple $\boldsymbol{t}$ in $\mathcal{F}$ is light w.r.t. $\mathcal{C}(B)$, we obtain that

$$\mathsf{deg}_R^{\mathcal{F}}(\boldsymbol{t}) \leq \mathsf{deg}^{\mathcal{F}}(\boldsymbol{t}) \leq \frac{|D|}{p^{\mathcal{C}(B)/\gamma(q)}} \leq \frac{|D|}{p^{f(B)/\gamma(q)}}.$$

Moreover, we can see that

$$\frac{|D|}{p^{f(B)/\gamma(q)}} = \frac{k.|[\![R]\!]_D|}{p^{f(B)/\gamma(q)}} = \frac{|[\![R]\!]_D|}{1/k \cdot p^{f(B)/\gamma(q)}} \leq \frac{|[\![R]\!]_D|}{(1/k)^{|B|} \cdot p^{f(B)/\gamma(q)}} = \frac{|[\![R]\!]_D|}{\beta^{|B|} \cdot p^{f(B)/\gamma(q)}}$$

with $k = |\mathsf{rels}(q)|$ and $\beta = 1/k$, which yields that $\mathsf{deg}_R^{\mathcal{F}}(\boldsymbol{t}) \leq \frac{|[\![R]\!]_D|}{\beta^{|B|} \cdot p^{f(B)/\gamma(q)}}$. Thus, we obtain that the load of R does not exceed $\tilde{\mathcal{O}}(\frac{|[\![R]\!]_D|}{p^{f(\mathsf{attrs}_q(R))/\gamma(q)}}) = \tilde{\mathcal{O}}(\frac{|D|}{p^{f(\mathsf{attrs}_q(R))/\gamma(q)}})$ w.h.p. by Lemma 25 and the fact that $|D| = k \cdot |[\![R]\!]_D|$. Moreover, from the fact that R is covered by f (*i.e.,* $f(\mathsf{attrs}_q(R)) = 1$), we guarantee that the load of R does not exceed $\tilde{\mathcal{O}}(|D|/p^{1/\gamma(q)})$.

Now, consider the case when $f(\mathsf{attrs}_q(R)) > 1$. In order to show that the load will not exceed the required load, we will show that if we hash partition R using slightly smaller buckets by assigning a lower share compared to the shares assigned by f (*i.e.,* less bins) instead, we can still guarantee a load of $\tilde{\mathcal{O}}(|D|/p^{1/\gamma(q)})$. Hence, the load can only improve when we assign the larger shares.

Thereto, for a relation R that is covered by f, we use the shares $p'_\alpha := p^{f'(\alpha)/\gamma(q)}$ with $f'(\alpha) := f(\alpha)/f(\mathsf{attrs}_q(R))$ for every $\alpha \in \mathsf{attrs}_q(R)$. Notice that for each α, we can verify that $p'_\alpha \leq p_\alpha$ since $f(\mathsf{attrs}_q(R)) > 1$. From the compatibility of f, we know that, for every $B \subseteq \mathsf{attrs}_q(R)$, $\min\{1, f(B)\} \leq \mathcal{C}(B)$ which implies that $f'(B) \leq \mathcal{C}(B)$. Remark that $f'(\mathsf{attrs}_q(R)) = 1$ according to this construction. Therefore, using the argument mentioned above for the case when $f(\mathsf{attrs}_q(R)) \leq 1$, we see that with $p^{f'(\mathsf{attrs}_q(R))/\gamma(q)}$ servers, the load of R is indeed $\tilde{\mathcal{O}}(|D|/p^{1/\gamma(q)})$.

B Proofs for Section 5

B.1 Proof of Lemma 15

In what follows, let $a_1 = (X_1, Z_1), \ldots, a_k = (X_k, Z_k)$ be the set of anchors in $\mathcal{A}$ in some order. Recall that each group of $p_{\boldsymbol{h}}$ servers is defined as $p_{\boldsymbol{h}} := p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \times \prod_{a_i \in \mathcal{A}} p_{a_i}$, where $p_{a_i} := p^{1/\mathsf{c}(\mathcal{S})} \cdot \frac{\mathsf{deg}^D(\boldsymbol{h}[Z_i])}{\mathsf{degs}^{\mathcal{F}}(Z_i)}$. Moreover, the set $\boldsymbol{H}$ that $\boldsymbol{h}$ ranges over is defined as

$$\{\boldsymbol{t} \mid \boldsymbol{t}[Z] \in \mathsf{tup}^{\mathcal{F}}(Z) \text{ for every } (X, Z) \in \mathcal{A} \text{ and } \boldsymbol{t}[B] \in \mathsf{tup}^{\mathcal{F}}(B) \text{ for every } B \in \mathcal{P}\}.$$

First, we establish item (1) in Lemma 15 in the following lemma.

▶ **Lemma 26.** *For a join query q, a fragment $\mathcal{F}$ of D w.r.t. $\boldsymbol{r}$ and $\mathcal{C}$, and a solution $\mathcal{S} = (\mathcal{A}, f, \mathcal{P})$ for q w.r.t. $\mathcal{C}$, we have*

1. *the number of tuples over $\mathsf{attrs}(\mathcal{P})$ in $\mathcal{F}$ is at most $p^{\mathsf{c}(\mathcal{P})/\mathsf{c}(\mathcal{S})}$;*
2. *$\sum_{\boldsymbol{h}[H(\mathcal{A})]} \prod_{a_i \in \mathcal{A}} p_{a_i} \leq p^{\mathsf{c}(\mathcal{A})/\mathsf{c}(\mathcal{S})}$; and*
3. *$\sum_{\boldsymbol{h}} p_{\boldsymbol{h}} \leq p$.*

Proof. (1) By definition, we know that $\mathsf{c}(B) > \mathcal{C}(B)$ for any set of attributes B in $\mathcal{P}$, and that all of B-tuples in $\mathcal{F}$ are heavy for $\mathsf{c}(B)$. Consequently, we have at most $p^{\mathsf{c}(B)/\mathsf{c}(\mathcal{S})}$ heavy tuples in $\mathcal{F}$. Now recall that the sets of attributes in $\mathcal{P}$ are pair-wise disjoint. Hence, the total number of heavy tuples over $\mathsf{attrs}(\mathcal{P})$ is at most $\prod_{B \in \mathcal{P}} p^{\mathsf{c}(B)/\mathsf{c}(\mathcal{S})} = p^{\mathsf{c}(\mathcal{P})/\mathsf{c}(\mathcal{S})}$.

(2) First remark that the values for $\boldsymbol{h}[Z_i]$ ranges over the tuples in $S_i = \mathsf{tup}^{\mathcal{F}}(Z_i)$, while $\boldsymbol{h}[H(\mathcal{A})]$ ranges over the tuples from $\boldsymbol{H}[H(\mathcal{A})]$. We can see that $\boldsymbol{H}[H(\mathcal{A})] \subseteq S_1 \times \ldots \times S_k$. Indeed, $\boldsymbol{H}[H(\mathcal{A})]$ is a restriction of the values in $S_1 \times \ldots \times S_k$.

$$\begin{aligned}
\sum_{\boldsymbol{h}[H(\mathcal{A})]} \prod_{a_i \in \mathcal{A}} p_{a_i} &= \sum_{\boldsymbol{h}[H(\mathcal{A})]} \prod_{a_i \in \mathcal{A}} p^{1/\mathsf{c}(\mathcal{S})} \cdot \frac{\mathsf{deg}^D(\boldsymbol{h}[Z])}{\mathsf{degs}^{\mathcal{F}}(Z)} = p^{\mathsf{c}(\mathcal{A})/\mathsf{c}(\mathcal{S})} \cdot \sum_{\boldsymbol{h}[H(\mathcal{A})]} \prod_{a_i \in \mathcal{A}} \frac{\mathsf{deg}^D(\boldsymbol{h}[Z_i])}{\mathsf{degs}^{\mathcal{F}}(Z_i)} \\
&\leq p^{\mathsf{c}(\mathcal{A})/\mathsf{c}(\mathcal{S})} \cdot \sum_{(\boldsymbol{h}[Z_1], \ldots, \boldsymbol{h}[Z_k]) \in S_1 \times \ldots \times S_k} \prod_{a_i \in \mathcal{A}} \frac{\mathsf{deg}^D(\boldsymbol{h}[Z_i])}{\mathsf{degs}^{\mathcal{F}}(Z_i)} \\
&= p^{\mathsf{c}(\mathcal{A})/\mathsf{c}(\mathcal{S})} \cdot \prod_{a_i \in \mathcal{A}} \sum_{\boldsymbol{h}[Z_i] \in S_i} \frac{\mathsf{deg}^D(\boldsymbol{h}[Z_i])}{\mathsf{degs}^{\mathcal{F}}(Z_i)} = p^{\mathsf{c}(\mathcal{A})/\mathsf{c}(\mathcal{S})} \cdot \prod_{a_i \in \mathcal{A}} \frac{\mathsf{degs}^{\mathcal{F}}(Z_i)}{\mathsf{degs}^{\mathcal{F}}(Z_i)} \\
&= p^{\mathsf{c}(\mathcal{A})/\mathsf{c}(\mathcal{S})}.
\end{aligned}$$

The inequality holds since the possible values for $\boldsymbol{h}[H(\mathcal{A})]$ are a subset of the values possible for $(\boldsymbol{h}[Z_1], \ldots, \boldsymbol{h}[Z_k])$. Moreover, the second to last equality follow from the definition of $\mathsf{degs}^{\mathcal{F}}(Z_i)$ (recall that $\mathsf{degs}^{\mathcal{F}}(Z) := \sum_{\boldsymbol{t} \in \mathsf{tup}^{\mathcal{F}}(Z)} \mathsf{deg}^{D}(\boldsymbol{t})$).

(3) Now, we verify that $\sum_{\boldsymbol{h}} p_{\boldsymbol{h}} \leq p$ as follows:

$$\begin{aligned}
\textstyle\sum_{\boldsymbol{h}} p_{\boldsymbol{h}} &= \textstyle\sum_{\boldsymbol{h}} p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \times \prod_{a_i \in \mathcal{A}} p_{a_i} && = p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \cdot \textstyle\sum_{\boldsymbol{h}} \prod_{a_i \in \mathcal{A}} p_{a_i} \\
&\leq p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \cdot \textstyle\sum_{\boldsymbol{h}[\mathsf{attrs}(\mathcal{P})]} \sum_{\boldsymbol{h}[H(\mathcal{A})]} \prod_{a_i \in \mathcal{A}} p_{a_i} && \\
&\leq p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \cdot p^{\mathsf{c}(\mathcal{P})/\mathsf{c}(\mathcal{S})} \cdot \textstyle\sum_{\boldsymbol{h}[H(\mathcal{A})]} \prod_{a_i \in \mathcal{A}} p_{a_i} && \\
&\leq p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \cdot p^{\mathsf{c}(\mathcal{P})/\mathsf{c}(\mathcal{S})} \cdot p^{|\mathcal{A}|/\mathsf{c}(\mathcal{S})} && = p
\end{aligned}$$

where the first inequality follows from the fact that possible values of $\boldsymbol{h}$ are a subset of the ones possible over $\mathsf{attrs}(\mathcal{P})$ and $H(\mathcal{A})$ disjointly. The second inequality follows from item (1), and the last inequality follows from item (2). ◀

Next, we establish item (2) in Lemma 15 through the following lemma.

▶ **Lemma 27.** *For a join query q, a fragment $\mathcal{F}$ of D w.r.t. $\boldsymbol{r}$ and $\mathcal{C}$, and a solution $\mathcal{S} = (\mathcal{A}, f, \mathcal{P})$ for q w.r.t. $\mathcal{C}$, we have*

1. *$p_{a_i} \geq 1$ for every anchor $a_i \in \mathcal{A}$ and every tuple $\boldsymbol{h}$ over $H(\mathcal{A}) \cup \mathsf{attrs}(\mathcal{P})$ in $\mathcal{F}$; and*
2. *$p_{\boldsymbol{h}} \geq p^{1/\mathsf{c}(\mathcal{S})}$.*

Proof. (1) Equivalently, we want to show that $\mathsf{deg}^{D}(\boldsymbol{h}[Z_i]) \geq \mathsf{degs}^{\mathcal{F}}(Z_i)/p^{1/\mathsf{c}(\mathcal{S})}$. Since $\mathcal{C}(Z_i) < 1$, we obtain that indeed $\mathsf{deg}^{D}(\boldsymbol{h}[Z_i]) \geq |D|/p^{1/\gamma(q)}$. Moreover, it is clear that $|D| \geq \mathsf{degs}^{\mathcal{F}}(Z_i)$ and it is safe to assume that $\mathsf{c}(\mathcal{S}) \leq \gamma(q)$.

(2) Next, we show that $p_{\boldsymbol{h}} \geq p^{1/\mathsf{c}(\mathcal{S})}$. Indeed, by definition, it is clear that $p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \geq p^{1/\mathsf{c}(\mathcal{S})}$. Moreover, from item (1), we guarantee that $\prod_{a_i \in \mathcal{A}} p_{a_i} \geq 1$. Hence,

$$\left(p^{\mathsf{c}(f)/\mathsf{c}(\mathcal{S})} \times \prod_{a_i \in \mathcal{A}} p_{a_i}\right) \geq p^{1/\mathsf{c}(\mathcal{S})}.$$ ◀

B.2 Proof of Lemma 17

Proof. Since $\mathsf{rels}(q_b) \subseteq \mathsf{Heavyrels}(q, \mathcal{C})$ and given that the size of a relation in $F_{\boldsymbol{h}}$ cannot exceed its size in $\mathcal{F}$, it is sufficient to show that for every relation $R \in \mathsf{Heavyrels}(q, \mathcal{C})$, we have that $|[\![R]\!]_{\mathcal{F}}| < |D|/p^{1/\gamma(q)}$ in order to establish our proof of item (1). Let R be a relation from the set $\mathsf{Heavyrels}(q, \mathcal{C})$. By definition, $\mathcal{C}(\{\alpha\}) < 1$ for every $\alpha \in \mathsf{attrs}_q(R)$. Moreover, by definition, we know that $\mathsf{deg}^{D}(\boldsymbol{t}) > |D|/p^{1/\gamma(q)}$ for any $\{\alpha\}$-tuple $\boldsymbol{t}$ in D. Therefore, the total number of possible $\{\alpha\}$-tuples in D respecting $C(\{\alpha\}) < 1$ is clearly less than $p^{1/\gamma(q)}$. Consequently, the total number of $(\mathsf{attrs}_q(R))$-tuples in D is less than $p^{|\mathsf{attrs}_q(R)|/\gamma(q)}$. Since $\mathcal{F}$ is the instance that corresponds to such a configuration, we see that $|[\![R]\!]_{\mathcal{F}}| < p^{|\mathsf{attrs}_q(R)|/\gamma(q)}$ which is considered a low load compared to $|D|/p^{1/\gamma(q)}$ by our the assumption that $|D| \gg p$ (and precisely, the load is larger than $\max_{R \in \mathsf{rels}(q)}\{p^{|\mathsf{attrs}_q(R)|/\gamma(q)}\}$).

The proof of item (2) directly follows from Proposition 12. As for item (3), it is sufficient to show that the set of relations anchoring $a = (X, Z) \in \mathcal{A}$ can be hash-partitioned based on the attributes of X with the required load. Accordingly, Lemma 25 requires that $\mathsf{deg}^{\mathcal{F}}(\boldsymbol{t}) \leq |D|/p_a$ for every X-tuple $\boldsymbol{t}$. Recall that by definition of anchor, we know that $\mathsf{deg}^{\mathcal{F}}(\boldsymbol{t}) \leq \mathsf{deg}^{D}(\boldsymbol{t}) \leq |D|/p^{1/\gamma(q)} \leq |D|/p^{1/\mathsf{c}(\mathcal{S})}$ and we also know that $|D|/p^{1/\mathsf{c}(\mathcal{S})} \leq |D|/p_a$ because $p_a \leq p^{1/\mathsf{c}(\mathcal{S})}$ (recall that $p_a = p^{1/\mathsf{c}(\mathcal{S})} \cdot \frac{\mathsf{deg}^{D}(\boldsymbol{h}[Z])}{\mathsf{degs}^{\mathcal{F}}(Z)}$ for some tuple $\boldsymbol{h}$.) ◀