Communication Steps for Parallel Query Processing

PAUL BEAME, PARASCHOS KOUTRIS, and DAN SUCIU, University of Washington

We study the problem of computing conjunctive queries over large databases on parallel architectures without shared storage. Using the structure of such a query q and the skew in the data, we study tradeoffs between the number of processors, the number of rounds of communication, and the per-processor *load*—the number of bits each processor can send or can receive in a single round—that are required to compute q. Since each processor must store its received bits, the load is at most the number of bits of storage per processor.

When the data are free of skew, we obtain essentially tight upper and lower bounds for one round algorithms, and we show how the bounds degrade when there is skew in the data. In the case of skewed data, we show how to improve the algorithms when approximate degrees of the (necessarily small number of) heavy-hitter elements are available, obtaining essentially optimal algorithms for queries such as skewed simple joins and skewed triangle join queries.

For queries that we identify as *treelike*, we also prove nearly matching upper and lower bounds for multi-round algorithms for a natural class of skew-free databases. One consequence of these latter lower bounds is that for any $\varepsilon > 0$, using p processors to compute the connected components of a graph, or to output the path, if any, between a specified pair of vertices of a graph with m edges and per-processor load that is $O(m/p^{1-\varepsilon})$ requires $\Omega(\log p)$ rounds of communication.

Our upper bounds are given by simple structured algorithms using MapReduce. Our one-round lower bounds are proved in a very general model, which we call the *Massively Parallel Communication (MPC)* model, that allows processors to communicate arbitrary bits. Our multi-round lower bounds apply in a restricted version of the MPC model in which processors in subsequent rounds after the first communication round are only allowed to send tuples.

Categories and Subject Descriptors: H.2.4 [**Systems**]: Parallel Databases

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Parallel computation, lower bounds

ACM Reference format:
Paul Beame, Paraschos Koutris, and Dan Suciu. 2017. Communication Steps for Parallel Query Processing. *J. ACM* 64, 6, Article 40 (October 2017), 58 pages.
https://doi.org/10.1145/3125644

Preliminary versions of portions of this work appeared in "Communication Steps for Parallel Query Processing" and "Skew in Parallel Query Processing" which were published in the proceedings of the ACM Symposium on Principles of Database Systems (PODS) in 2013 and 2014, respectively (Beame et al. 2013, 2014).
Suciu was partially supported by NSF AITF 1535565 and NSF IIS 1247469. Beame was partially supported by NSF grants CCF-1524246 and CCF-1217099. Koutris was partially supported from the Wisconsin Alumni Research Foundation.
Authors' addresses: P. Beame, P. Koutris, and D. Suciu, P. Koutris University of Wisconsin-Madison; emails: beame@cs.washington.edu, paris@cs.wisc.edu, suciu@cs.washington.edu.

https://doi.org/10.1145/3125644

1 INTRODUCTION

Most of the time spent during big data analysis today is allocated in data processing tasks, such as identifying relevant data, cleaning, filtering, joining, grouping, transforming, extracting features, and evaluating results (Chaudhuri 2012; EMC Corporation). These tasks form the main bottleneck in big data analysis, and it is a major challenge to improve the performance and usability of data processing tools. The motivation for this article comes from the need to understand the complexity of query processing in big data management.

Query processing on big data is typically performed on a shared-nothing parallel architecture. In this setting, the data are stored on a large number of independent servers interconnected by a fast network. The servers perform local computations and can also communicate with each other to exchange data. Starting from MapReduce (Dean and Ghemawat 2004), the past decade has seen the development of several massively parallel frameworks that support big data processing, including PigLatin (Olston et al. 2008), Hive (Thusoo et al. 2009), Dremmel (Melnik et al. 2010), Spark (Spark), and Myria (Halperin et al. 2014).

Unlike traditional query processing, the time complexity is no longer dominated by the number of disk accesses. Typically, a query is evaluated by a sufficiently large number of servers such that the entire data can be kept in main memory. In these systems, the new complexity parameter is the *communication cost*, which depends on both the amount of data being exchanged and the number of global synchronization barriers (rounds).

Contributions. We define the *Massively Parallel Communication* (MPC) model, to analyze the tradeoff between the number of rounds and the amount of communication required in a massively parallel computing environment. We include the number of servers p as a parameter, and we allow each server to be infinitely powerful, subject only to the data to which it has access. The computation proceeds in *rounds*, where each round consists of local computation followed by global exchange of data between all servers.

An algorithm in the MPC model is characterized by the number of servers p, the number of rounds r, and the maximum number of bits L, or *maximum load*, that each server receives at any round. There are no other restrictions on communication between servers. Though the storage capacity of each server is not a separate parameter, since a server needs to store the data it has received to operate on it, the load L is always a lower bound on the storage capacity of each server. An ideal parallel algorithm with input size M would distribute the input data equally among the p servers, so each server would have a maximum load of M/p, and would perform the computation in a single round. In the degenerate case where $L = M$, the entire data can be sent to a single server, and thus there exists no parallelism.

The focus of the MPC model on the communication load captures a key property of the system architectures assumed by MapReduce and related programming abstractions.[1] Since there is no restriction on the form of communication or the kinds of operations allowed for processing of local data, the lower bounds we obtain in the MPC model apply much more generally than those bounds based on specific assumed primitives or communication structures such as those for MapReduce.

We establish both lower and upper bounds in the MPC model for computing a full conjunctive query q, in three different settings.

First, we restrict the computation to a single communication round and to input data without skew. In particular, given a query q over relations $S_1, S_2, \ldots$ such that an input relation S_j has size

[1]This focus is somewhat related to the generic approach to communication in Valiant's Bulk Synchronous Parallel (BSP) model (Valiant 1990), which the MPC model simplifies and strengthens. We discuss the relationship of the MPC model to a variety of MapReduce and parallel models in Section 6.

M_j (in bits), we examine the minimum load L for which it is possible to compute q in a single round. We show that any algorithm that correctly computes q requires a load

$$L \geq \max_{\mathbf{u}} \left(\frac{\prod_j M_j^{u_j}}{p} \right)^{1/\sum_j u_j},$$

where $\mathbf{u} = (u_1, \ldots, u_\ell)$ is a fractional *edge packing* for the hypergraph of q. Our lower bound applies to the strongest possible model in which servers can encode any information in their messages and have access to a common source of randomness. This is stronger than the lower bounds in Afrati et al. (2012) and Koutris and Suciu (2011), which assume that the units being exchanged are tuples. We further show that a simple algorithm, which we call the HyperCube algorithm, matches our lower bound for any conjunctive query when the input data have no skew. As an example, for the triangle query $C_3 = S_1(x, y), S_2(y, z), S_3(z, x)$ with sizes $M = |S_1| = |S_2| = |S_3|$, we show that the lower bound for the load is $\Omega(M/p^{2/3})$, and the HyperCube algorithm can match this bound.

Second, we study how skew influences the computation. A value in the database is *skewed* and is called a *heavy hitter* when it occurs with much higher frequency than some predefined threshold. Since data distribution is typically done using hash-partitioning, unless they are handled differently from other values, all tuples containing a heavy hitter will be sent to the same server, causing it to be overloaded. The standard technique that handles skew consists of first detecting the heavy hitters, then treating them differently from the other values, for example, by partitioning tuples with heavy hitters on the other attributes.

In analyzing the impact of skew, we first provide bounds on the behavior of algorithms that are not given special information about heavy hitters and hence are limited in their ability to deal with skew. We then consider a natural model for handling skew which assumes that at the beginning of the computation all servers know the identity of all heavy hitters, and the (approximate) frequency of each heavy hitter. (It will be easy to see that there can only be a small number of heavy hitters and this kind of information can be easily obtained in advance.[2]) Given these statistics, we present upper and lower bounds for the maximum load for full conjunctive queries. In particular, we present a general lower bound that holds for any conjunctive query. We next give matching upper bounds for the class of *star joins*, which are queries of the form $q(z, x_1, \ldots, x_k) = S_1(z, x_1), S_2(z, x_2), \ldots, S_k(z, x_k)$ (this includes the case of the simple join query for $k = 2$), as well as the triangle query.

Third, we establish lower bounds for multiple communication rounds, for a restricted version of the MPC model, called *tuple-based* MPC model. The messages sent in the first round are still unrestricted, but in subsequent rounds the servers can send only tuples, either base tuples in the input tables, or join tuples corresponding to a subquery; moreover, the destinations of each tuple may depend only on the tuple content, the message received in the first round, the server, and the round. We should note that many of the most frequently used MapReduce algorithms for query processing are tuple based (e.g., parallel hash-join, broadcast join). Here, we prove that the number of rounds required is, essentially, given by the depth of a query plan for the query, where each operator is a subquery that can be computed in one round with the required load. For example, to compute a length k chain query L_k, if we want to achieve $L = O(M/p)$, then the optimal computation is a bushy join tree, where each operator is L_2 (a two-way join) and the optimal number of rounds is $\log_2 k$. If we allow a larger load, $L = O(M/p^{1/2})$, then we can use L_4 as operator

[2]For example, we can compute all frequencies by performing a parallel aggregation, or we can find the heavy hitters by sampling; both of these methods can be executed in one round, with load $O(M/p)$.

Table 1. Roadmap for the Organization of the Results Presented in This Article

Number of Rounds	Data Distribution	Upper Bound	Lower Bound
1 round (MPC model)	no skew	Section 3.1	Section 3.2
	skew (oblivious)	Section 4.1	Section 4.1
	skew (with information)	Section 4.2.1	Section 4.2.3
multiple rounds (tuple-based MPC model)	no skew	Section 5.1	Section 5.2

(a four-way join), and the optimal number of rounds decreases to $\log_4 k$. More generally, we show nearly matching upper and lower bounds based on graph-theoretic properties of the query.

We further show that our results for conjunctive path queries imply that any tuple-based MPC algorithm with load $L < M$ requires $\Omega(\log p)$ rounds to compute the connected components of sparse undirected graphs of size M (in bits). This is an interesting contrast to the results of Karloff et al. (2010), which show that connected components (and indeed minimum spanning trees) of undirected graphs can be computed in only two rounds of MapReduce provided that the input graph is sufficiently dense.

By being explicit about the number of processors, in the MPC model we must directly handle issues of load balancing and skew in task (reducer) sizes that are often ignored in MapReduce algorithms but are actually critical for good performance (e.g., see Kwon et al. (2012)). When task sizes are similar, standard analysis shows that hash-based load balancing works well. However, standard bounds do not yield sharp results when there is significant deviation in sizes. To handle such situations, we prove a sharp Chernoff bound for weighted balls in bins that is particularly suited to the analysis of hash-based load balancing with skewed data. This bound, which is given in Section 3, should be of independent interest.

Organization. We start by presenting the MPC model and defining important notions in Section 2. In Section 3, we describe the upper and lower bounds for computation restricted to one round and data without skew. We study the effect of data skew in Section 4. In Section 5, we present upper and lower bounds for the case of multiple rounds. We conclude by discussing the related work in Section 6. In Table 1, the reader can view a more detailed roadmap for the results of this article.

2 MODEL

In this section, we present in detail the MPC model and introduce the necessary terminology and background.

2.1 Massively Parallel Communication

In the MPC model, computation is performed by p servers, or processors, connected by a complete network of private channels. The computation proceeds in *steps*, or *rounds*, where each round consists of two distinct phases:

Communication Phase. The servers exchange data, each by communicating with all other servers (sending and receiving data).

Computation Phase. Each server performs only local computation.

The input data of size M bits is initially uniformly partitioned among the p servers, that is, each server stores M/p bits of the data: This describes the way the data are typically partitioned in a distributed storage system. We do not assume that the data are partitioned in any particular

way, in other words, an algorithm in the MPC model must return the correct result no matter how the input data are partitioned. At the end of the execution, the output must be present in the union of the p processors. The lower bounds in this work hold even if we allow the output to have overlap among the processors; however, all algorithms guarantee that an output tuple will appear in exactly one processor.

The complexity of an algorithm in the MPC model is captured by two parameters:

The number of rounds r. This parameter denotes the number of synchronization barriers that an algorithm requires.

The maximum load L. This parameter denotes the *maximum load* among all servers at any round, where the load is the amount of data received by a server during a particular round.

Normally, all of the data are exchanged during the first communication round, so the load L is at least M/p. On the other hand, the load is strictly less than M: Otherwise, if we allowed a load $L = M$, then any problem can solved trivially in one round by simply sending the entire data to server 1 and then computing the answer locally. Typical loads will be of the form $M/p^{1-\varepsilon}$, for some $0 \leq \varepsilon < 1$ that depends on the query. The bounds on the load depend on both the amount of the input data and the size of the query. We assume that the query to be executed is known to all servers. In many cases of interest, the query is of small constant size, but in the worst case the dependence on the query size is at most linear.

We do not allow the number of rounds to reach $r = p$, because any problem can be solved trivially in p rounds by sending at each round M/p bits of data to server 1, until this server accumulates the entire data. In fact, all of the algorithms presented in this article require a constant number of rounds, $r = O(1)$.

Input Servers. As explained above, the data are initially distributed uniformly on the p servers; we call this form of input *partitioned input*. Our lower bounds for queries over a fixed relational vocabulary $S_1, \ldots, S_\ell$ consider an alternative, more powerful model, where each relation S_j is stored on a separate server, called an *input server*; during the first round the ℓ input servers distribute their data to the p workers and then no longer participate in the computation. The input-server model is potentially more powerful, since the jth input server has access to the entire relation S_j, whose size is much larger than M/p. We state and prove all our lower bounds for the input-server model. This is w.l.o.g., because any algorithm in the partitioned-input model with load L can be converted into an input-server algorithm with the same load, as follows. Denote $f_j = |S_j|/(\sum_i |S_i|)$ for all $j = 1, \ldots, \ell$: We assume these numbers are known by all input servers, because we assume the statistics $|S_j|$ known to the algorithm. Then, each input server j holding the relation S_j will partition S_j into $f_j p$ equal fragments and then will simulate $f_j p$ workers, each processing one of the fragments. Thus, our lower bounds for the input-sever model immediately apply to the partitioned-input model.

Randomization. The MPC model allows randomization. The random bits are available to all servers and are computed independently of the input data. The algorithm may fail to produce its output with a small probability $\eta > 0$, independent of the input. For example, we use randomization for load balancing and abort the computation if the amount of data received during a round would exceed the maximum load L, but this will only happen with exponentially small probability.

To prove lower bounds for randomized algorithms, we use Yao's Lemma (Yao 1977). We first prove bounds for *deterministic* algorithms, showing that any algorithm fails with probability at least η over inputs chosen randomly from a distribution μ. This implies, by Yao's Lemma, that every randomized algorithm with the same resource bounds will fail on some input (in the support of μ) with probability at least η over the algorithm's random choices.

2.2 Conjunctive Queries

In this article, we consider a particular class of problems for the MPC model, namely computing answers to conjunctive queries over a database. We fix an input vocabulary $S_1, \ldots, S_\ell$, where each relation S_j has a fixed arity a_j; we denote $a = \sum_{j=1}^{\ell} a_j$. The input data consist of one relation instance for each symbol.

We consider full conjunctive queries (CQs) without self-joins, denoted as follows:

$$q(x_1, \ldots, x_k) = S_1(\bar{x}_1), \ldots, S_\ell(\bar{x}_\ell). \tag{1}$$

$S_j(\bar{x}_j)$ is called an *atom*, where $\bar{x}_j$ denotes a vector of variables. The query is *full*, meaning that every variable in the body appears in the head (for example, $q(x) = S(x, y)$ is not full), and *without self-joins*, meaning that each relation name S_j appears only once (for example, $q(x, y, z) = S(x, y), S(y, z)$ has a self-join). The first restriction, to full conjunctive queries, is a limitation: Our lower bounds do not carry over to general conjunctive queries (but the upper bounds do carry over). The second restriction, to queries without self-joins, is w.l.o.g.[3]

The *hypergraph* of a query q is defined by introducing one node for each variable in the body and one hyperedge for each set of variables that occur in a single atom. We say that a conjunctive query is *connected* if the query hypergraph is connected. For example, $q(x, y) = R(x), S(y)$ is not connected, whereas $q(x, y) = R(x), S(y), T(x, y)$ is connected. We use $\text{vars}(S_j)$ to denote the set of variables in the atom S_j, and $\text{atoms}(x_i)$ to denote the set of atoms where x_i occurs; k and ℓ denote the number of variables and atoms in q, as in Equation (1). The *connected components* of q are the maximal connected subqueries of q.

Characteristic of a Query. The *characteristic* of a conjunctive query q as in Equation (1) is defined as $\chi(q) = a - k - \ell + c$, where $a = \sum_j a_j$ is the sum of arities of all atoms, k is the number of variables, ℓ is the number of atoms, and c is the number of connected components of q.[4] Intuitively, a larger value of the characteristic implies that variables will be more "shared" among the atoms, that is, they will occur more often. The characteristic of a query will play a crucial role at the design of multi-round algorithms in Section 5.

For a query q and a set of atoms $M \subseteq \text{atoms}(q)$, define q/M to be the query that results from contracting the edges in the hypergraph of q. As an example, if we define

$$L_k = S_1(x_0, x_1), S_2(x_1, x_2), \ldots, S_k(x_{k-1}, x_k),$$

then we have that $L_5/\{S_2, S_4\} = S_1(x_0, x_1), S_3(x_1, x_3), S_5(x_3, x_5)$.

Lemma 2.1. *The characteristic of a query q satisfies the following properties:*

(a) If $q_1, \ldots, q_c$ are the connected components of q, then $\chi(q) = \sum_{i=1}^{c} \chi(q_i)$.
(b) For any $M \subseteq \text{atoms}(q)$, $\chi(q/M) = \chi(q) - \chi(M)$.

[3] To see this, denote q' the query obtained from q by giving distinct names $S'_j, S''_j, \ldots$ to repeated occurrences of the same relation S_j. Any algorithm A' for q' can be used to compute q with the same load, by first having each server copy locally its fragment of S_j into new relations $S'_j, S''_j, \ldots$, then executing A' on the new inputs. Conversely, any algorithm A for q can also be converted into an algorithm A' for q', having the same load as A has on an input that is ℓ times larger. The algorithm A' is obtained as follows. First, each server will copy its fragment of S_j into new relations $S'_j, S''_j, \ldots$. Namely, for each atom $S'_j(x, y, z, \ldots)$ in q' derived from some atom $S_j(x, y, z, \ldots)$ in q, the server will insert for each tuple $(a, b, c, \ldots) \in S_j$ a new tuple $((a, \text{"}x\text{"}), (b, \text{"}y\text{"}), (c, \text{"}z\text{"}), \ldots) \in S'_j$. That is, each value a in the first column is replaced by the pair $(a, \text{"}x\text{"})$, where x is the variable occurring in that column and similarly for all other columns. This copy operation can be done locally by all servers, without communication, and each input relation is copied at most ℓ times. Finally, run the algorithm A' on the copied relations.

[4] In the preliminary version of this article (Beame et al. 2013), we defined $\chi(q)$ with the opposite sign (as $-a + k + \ell - c$); we find the current definition more natural since now $\chi(q) \geq 0$ for every q.

(c) $\chi(q) \geq 0$.
(d) For any $M \subseteq atoms(q)$, $\chi(q) \geq \chi(q/M)$.

Proof. Property (a) is immediate from the definition of χ, since the connected components of q are disjoint with respect to variables and atoms. Since q/M can be produced by contracting according to each connected component of M in turn, by property (a) and induction it suffices to show that property (b) holds in the case that M is connected. If a connected M has k_M variables, ℓ_M atoms, and total arity a_M, then the query after contraction, q/M, will have the same number of connected components, $k_M - 1$ fewer variables, and the terms for the number of atoms and total arity will be reduced by $a_M - \ell_M$ for a total reduction of $a_M - k_M - \ell_M + 1 = \chi(M)$. Thus, property (b) follows.

By property (a), it suffices to prove (c) when q is connected. If q is a single atom S_j, then $\chi(S_j) \geq 0$, since the number of variables is at most the arity a_j of the atom. If q has more than one atom, then let S_j be any such atom: Then $\chi(q) = \chi(q/S_j) + \chi(S_j) \geq \chi(q/S_j)$, because $\chi(S_j) \geq 0$. Property (d) follows from (b) using the fact that $\chi(M) \geq 0$. □

For a simple illustration of property (b), consider the example above $L_5/\{S_2, S_4\}$, which is equivalent to L_3. We have $\chi(L_5) = 10 - 6 - 5 + 1 = 0$, and $\chi(L_3) = 6 - 4 - 3 + 1 = 0$, and also $\chi(M) = 0$ (because M consists of two disconnected components, $S_2(x_1, x_2)$ and $S_4(x_3, x_4)$, each with characteristic 0). For a more interesting example, consider the query K_4 whose graph is the complete graph with four variables,

$$K_4 = S_1(x_1, x_2), S_2(x_1, x_3), S_3(x_2, x_3), S_4(x_1, x_4), S_5(x_2, x_4), S_6(x_3, x_4),$$

and denote $M = \{S_1, S_2, S_3\}$. Then $K_4/M = S_4(x_1, x_4), S_5(x_1, x_4), S_6(x_1, x_4)$, and the characteristics are as follows: $\chi(K_4) = 12 - 4 - 6 + 1 = 3$, $\chi(M) = 6 - 3 - 3 + 1 = 1$, $\chi(K_4/M) = 6 - 2 - 3 + 1 = 2$.

Finally, we define a class of queries that will be used later in the article.

Definition 2.2. A conjunctive query q is *treelike* if q is connected and $\chi(q) = 0$.

For example, the query L_k is treelike; in fact, a query over a binary vocabulary is treelike if and only if its hypergraph is a tree. Over non-binary vocabularies, if a query is treelike, then it is acyclic, but the converse does not hold even when the query is connected: $q = S_1(x_0, x_1, x_2), S_2(x_1, x_2, x_3)$ is acyclic but not treelike.[5] An important property of treelike queries is that every connected subquery will be also treelike.

Fractional Edge Packing. A *fractional edge packing* (also known as a *fractional matching*) of a query q is any feasible solution $\mathbf{u} = (u_1, \ldots, u_\ell)$ of the following linear constraints:

$$\begin{aligned} &\forall i \in [k] : \sum_{j : i \in S_j} u_j \leq 1 \\ &\forall j \in [\ell] : u_j \geq 0. \end{aligned} \tag{2}$$

The edge packing associates a non-negative weight u_j to each atom S_j such that for every variable x_i, the sum of the weights for the atoms that contain x_i do not exceed 1. If all inequalities in Equation (2) are equalities, then we say that the solution $\mathbf{u}$ is *tight*. The dual notion is a *fractional*

[5]There are a number of notions of hypergraph acyclicity in the literature, see Brault-Baron (2016) for a recent survey, but our notion is more restrictive than any of them because it prohibits any pair of variables being contained in more than one atom.

vertex cover of q, which is a feasible solution $\mathbf{v} = (v_1, \ldots, v_k)$ to the following linear constraints:

$$\forall j \in [\ell] : \sum_{i : i \in S_j} v_i \geq 1$$
$$\forall i \in [k] : v_i \geq 0.$$

At optimality, $\max_{\mathbf{u}} \sum_j u_j = \min_{\mathbf{v}} \sum_i v_i$; this quantity is denoted τ^* and is called the *fractional vertex covering number of* q.

Example 2.3. An edge packing of the query $L_3 = S_1(x_1, x_2), S_2(x_2, x_3), S_3(x_3, x_4)$ is any solution to $u_1 \leq 1$, $u_1 + u_2 \leq 1$, $u_2 + u_3 \leq 1$, and $u_3 \leq 1$. In particular, the solution $(1, 0, 1)$ is a tight edge packing; it is also an optimal packing, thus $\tau^* = 2$.

A *fractional edge cover* is a feasible solution $\mathbf{u} = (u_1, \ldots, u_\ell)$ to a system of linear constraints as above, where $\leq$ is replaced by $\geq$ in Equation (2). Every tight fractional edge packing is a tight fractional edge cover and vice versa. The optimal value of a fractional edge cover is denoted ρ^*. The fractional edge packing and cover have no connection, and there is no relationship between τ^* and ρ^*. For example, for $q = S_1(x, y), S_2(y, z)$, we have $\tau^* = 1 < \rho^* = 2$, while for $q = S_1(x), S_2(x, y), S_3(y)$ we have $\tau^* = 2 > \rho^* = 1$. The two notions coincide, however, when they are tight, meaning that a tight fractional edge cover is also a tight fractional edge packing and vice versa. The fractional edge cover has been used recently in several articles to prove bounds on query size and the running time of a sequential algorithm for the query (Atserias et al. 2008; Ngo et al. 2012); for the results in this article we need the fractional packing.

2.3 Entropy

Let us fix a finite probability space. For random variables X and Y, the *entropy* and the *conditional entropy* are defined, respectively, as follows:

$$H(X) = -\sum_x P(X = x) \log P(X = x), \tag{3}$$

$$H(X \mid Y) = \sum_y P(Y = y) H(X \mid Y = y). \tag{4}$$

The entropy satisfies the following basic inequalities:

$$H(X \mid Y) \leq H(X)$$
$$H(X, Y) = H(X \mid Y) + H(Y). \tag{5}$$

Assuming additionally that X has a support of size n:

$$H(X) \leq \log n. \tag{6}$$

2.4 Friedgut's Inequality

Friedgut (2004) introduces the following class of inequalities. Each inequality is described by a hypergraph, which in our article corresponds to a query, so we will describe the inequality using query terminology. Fix a query q as in Equation (1), and let $n > 0$ be an integer. For every atom $S_j(\bar{x}_j)$ of arity a_j, we introduce a set of n^{a_j} variables $w_j(\mathbf{a}_j) \geq 0$, where $\mathbf{a}_j \in [n]^{a_j}$. If $\mathbf{a} \in [n]^a$, then we denote by $\mathbf{a}_j$ the vector of size a_j that results from projecting on the variables of the relation S_j. Let $\mathbf{u} = (u_1, \ldots, u_\ell)$ be a fractional *edge cover* for q. Then:

$$\sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \leq \prod_{j=1}^{\ell} \left(\sum_{\mathbf{a}_j \in [n]^{a_j}} w_j(\mathbf{a}_j)^{1/u_j} \right)^{u_j}. \tag{7}$$

We illustrate Friedgut's inequality on the queries C_3 and L_3:

$$C_3(x,y,z) = S_1(x,y), S_2(y,z), S_3(z,x)$$
$$L_3(x,y,z,w) = S_1(x,y), S_2(y,z), S_3(z,w). \quad (8)$$

Consider the cover $(1/2, 1/2, 1/2)$ for C_3 and the cover $(1, 0, 1)$ for L_3. Then, we obtain the following inequalities, where α, β, γ stand for w_1, w_2, w_3, respectively:

$$\sum_{x,y,z\in[n]} \alpha_{xy}\cdot\beta_{yz}\cdot\gamma_{zx} \le \sqrt{\sum_{x,y\in[n]}\alpha_{xy}^2 \sum_{y,z\in[n]}\beta_{yz}^2 \sum_{z,x\in[n]}\gamma_{zx}^2}$$
$$\sum_{x,y,z,w\in[n]} \alpha_{xy}\cdot\beta_{yz}\cdot\gamma_{zw} \le \sum_{x,y\in[n]}\alpha_{xy}\cdot\max_{y,z\in[n]}\beta_{yz}\cdot\sum_{z,w\in[n]}\gamma_{zw},$$

where we used the fact that $\lim_{u\to 0}(\sum \beta_{yz}^{\frac{1}{u}})^u = \max \beta_{yz}$.

Friedgut's inequalities immediately imply a well-known result developed in a series of articles (Grohe and Marx 2006; Atserias et al. 2008; Ngo et al. 2012) that gives an upper bound on the size of a query answer as a function on the cardinality of the relations. For example, in the case of C_3, consider an instance S_1, S_2, S_3, and set $\alpha_{xy} = 1$ if $(x,y) \in S_1$; otherwise, $\alpha_{xy} = 0$ (and similarly for β_{yz}, γ_{zx}). We obtain then $|C_3| \le \sqrt{|S_1|\cdot|S_2|\cdot|S_3|}$. Note that all these results are expressed in terms of a fractional edge *cover*. When we apply Friedgut's inequality in Section 3 to a fractional edge *packing*, we ensure that the packing is tight.

3 ONE COMMUNICATION STEP WITHOUT SKEW

In this section, we consider the case where the data have no skew, and the computation is restricted to a single communication round.

We will say that a database is a *matching database* over a domain $[n]$ if, in each attribute of each relation, every domain value occurs at most once; equivalently, every attribute is a key. In particular, every value in each relation has degree bounded by 1, that is, the frequency of each value is exactly 1. Our lower bounds in this section will hold for such matching databases. The upper bound, and in particular the load analysis for the algorithm, hold for more general databases, with a small amount of skew, which we will formally define in Section 3.1.

We assume that all input servers know the cardinalities $m_1, \ldots, m_\ell$ of the relations $S_1, \ldots, S_\ell$, and denote $\mathbf{m} = (m_1, \ldots, m_\ell)$ the vector of cardinalities. We also denote $M_1, \ldots, M_\ell$ the number of bits needed to represent each relation, and call $\mathbf{M} = (M_1, \ldots, M_\ell)$ *derived statistics*. Since each tuple of S_j requires $a_j \log n$ bits to represent, where n is the size of the domain of the database, we have $M_j \le a_j m_j \log n$. Our upper bounds are stated in terms of the cardinalities $\mathbf{m}$; in that case, we define the derived statistics $M_j \stackrel{\text{def}}{=} a_j m_j \log n$. Our lower bounds are stated in terms of $\mathbf{M}$ only and apply to more general encodings; this is necessary, because for restricted classes of databases M_j can be much smaller than $a_j m_j \log n$. For example, the standard encoding of a relation of arity 2 and cardinality m_j uses $M_j = 2m_j \log n$ bits, but if the relation is restricted to be a matching of cardinality n (equivalently, the relation is a permutation over the domain $[n]$), then it requires only $M_j = \log n!$ bits. As an extreme case, if S_j is a unary relation, then it can be encoded using a bitmap, and $M_j = \log n$.

3.1 The HyperCube Algorithm

We describe here an algorithm that computes a conjunctive query in one step. Such an algorithm was introduced by Afrati and Ullman (2010) for MapReduce, is similar to an algorithm by Suri and Vassilvitskii (2011) to count triangles, and also uses ideas that can be traced back to Ganguly et al.

(1992) for parallel processing of Datalog programs. We call this the HyperCube (HC) algorithm, following Beame et al. (2013).

The HC algorithm is parametrized by a vector of *shares* : Each variable x_i, where $i = 1, \ldots, k$, is assigned a share p_i, such that $\prod_{i=1}^{k} p_i = p$. The name of the algorithm results from the fact that each server is mapped to a distinct point in the k-dimensional hypercube $\mathcal{P} = [p_1] \times \cdots \times [p_k]$. The description of the HC algorithm is as follows:

ALGORITHM 1 : HyperCube Algorithm

Input: shares $\mathbf{p} = (p_1, \ldots, p_k)$, query q, relations $S_1, \ldots, S_\ell$.

1: Assign each server in $[p]$ to a distinct point in $\mathcal{P} = [p_1] \times \cdots \times [p_k]$
2: Choose k independent hash functions $h_i : [n] \to [p_i]$
3: **Communication**: send each tuple $S_j(t)$ to the subcube

$$\mathcal{D}(t) = \{\mathbf{y} \in \mathcal{P} \mid \forall s = 1, \ldots, a_j : h_{i_s}(t[i_s]) = \mathbf{y}_{i_s}\}$$

4: **Computation**: each server computes q on its local instance

The correctness of the HC algorithm follows from the observation that, for every potential tuple $(a_1, \ldots, a_k)$, the server $(h_1(a_1), \ldots, h_k(a_k))$ contains all the necessary information to decide whether it belongs in the answer or not.

Example 3.1. We illustrate how to compute the triangle query $C_3(x_1, x_2, x_3) = S_1(x_1, x_2), S_2(x_2, x_3), S_3(x_3, x_1)$. Consider the shares $p_1 = p_2 = p_3 = p^{1/3}$. Each of the p servers is uniquely identified by a triple (y_1, y_2, y_3), where $y_1, y_2, y_3 \in [p^{1/3}]$. In the first communication round, the input server storing S_1 sends each tuple $S_1(\alpha_1, \alpha_2)$ to all servers with index $(h_1(\alpha_1), h_2(\alpha_2), y_3)$ for all $y_3 \in [p^{1/3}]$: Notice that each tuple is replicated $p^{1/3}$ times. The input servers holding S_2 and S_3 proceed similarly with their tuples. After round 1, any three tuples $S_1(\alpha_1, \alpha_2), S_2(\alpha_2, \alpha_3), S_3(\alpha_3, \alpha_1)$ that contribute to the output tuple $C_3(\alpha_1, \alpha_2, \alpha_3)$ will be seen by the server $\mathbf{y} = (h_1(\alpha_1), h_2(\alpha_2), h_3(\alpha_3))$: Any server that detects three matching tuples outputs them.

3.1.1 Analysis of the HC Algorithm. The main technical result of this section is the analysis of the HC algorithm. Theorem 3.2 considers the hypercube partition of a single relation R and gives conditions under which this partition is balanced; the analysis of the HC algorithm follows immediately. To state and prove Theorem 3.2, it is convenient to allow the input relation R to be a bag rather than a set, that is, each tuple may occur multiple times in R; the size $m = |R|$, or cardinality of R, is defines as the total number of tuples, counting multiplicities.

Let R be a relation of arity r, that is, a bag of tuples. For a tuple J over a subset $U \subseteq [r]$, we define the *degree* of the tuple J in relation R, denoted $d_J(R)$, as $d_J(R) = |\{t \in R \mid t[U] = J\}|$. For a full tuple J (i.e., $U = [r]$), $d_J(R)$ is the multiplicity of J in R, and for the empty tuple $J = ()$, $d_J(R) = |R|$. Note that in the special case of matching databases, for every non-empty tuple J we have $d_J(R) = 1$.

Theorem 3.2. *Let $R(A_1, \ldots, A_r)$ be a relation of arity r of size m. Let $p_1, \ldots, p_r$ be integers, $p_i \geq 2$ for all i, and let $p = \prod_i p_i$. We hash partition R into p bins as follows. We identify each bin with an r-tuple in $[p_1] \times \cdots \times [p_r]$, and sent each tuple $(a_1, \ldots, a_r)$ is to the bin $(h_1(a_1), \ldots, h_r(a_r))$, where $h_1, \ldots, h_r$ are independent and perfectly random hash functions, with co-domains $[p_1], \ldots, [p_r]$, respectively. Then the expected load in every bin is m/p and the following hold:*

(1) Fix $\Delta > 0$ a constant. If for every non-empty tuple J, $d_J(R) \leq \Delta$, then for any constant $\delta > 0$, the probability that the maximum load of any server exceeds $(1 + \delta)m/p$ is is exponentially small in $(m/(p\Delta))^{1/r}$.

(2) *For every* $c > 0$ *there is a* $c' > 0$ *such that for* $2 \geq \delta > \frac{\ln p}{c' r^2 p^{1/2}}$ *and* $\beta = \frac{\delta}{cr^3 \ln p}$ *if for every tuple* J *over* $U \subseteq [r]$, $d_J(R) \leq \beta^{|U|} \frac{m}{\prod_{d \in U} p_d}$, *then the probability that the maximum load of any server exceeds* $(1+\delta)m/p$ *is at most* p^{-cr}.

(3) *If for every tuple* J *over* $U \subseteq [r]$, $d_J(R) \leq \frac{m}{\prod_{d \in U} p_d}$, *then for every* $c > 0$ *there is a* $c' > 0$ *such that the probability that the maximum load of any server exceeds* $(\frac{c' \ln p}{\ln \ln p})^r m/p$ *is at most* p^{-c}.

Denote $L_{\mathbf{y}}$ the load of a server $\mathbf{y} = (y_1, \ldots, y_r) \in [p_1] \times \cdots \times [p_r]$; it is straightforward to check that $\mathbb{E}[L_{y_1,\ldots,y_r}] = m/p$, where the expectation is taken over the choices of the random hash functions. The theorem establishes conditions under which the maximum load of all servers is $\leq (1+\delta)m/p$. To explain it, we establish a necessary condition. If every server has load $L_{\mathbf{y}} \leq (1+\delta)m/p$, then, for any subset of attributes $U \subseteq [r]$ and any sub-cube $\mathbf{z} \in \prod_{d \in U}[p_d]$, the accumulated load at the sub-cube $\mathbf{z}$ is $L_{\mathbf{z}} \stackrel{\text{def}}{=} \sum_{\mathbf{y}:\bigwedge_{d \in U} y_d = z_d} L_{\mathbf{y}} \leq (1+\delta)m/\prod_{d \in U} p_d$. This implies that, for any U-tuple J, it is necessary for its degree to be bounded by $d_J(R) \leq (1+\delta)m/\prod_{d \in U} p_d$; otherwise, the J tuple will cause at least one sub-cube $\mathbf{z}$ to have load $L_{\mathbf{z}} > (1+\delta)m/\prod_{d \in U} p_d$. The theorem establishes stricter conditions that are sufficient. Item 1 says, essentially, that if all degrees are bonded by a constant, then the probability that the maximum load exceeds $(1+\delta)m/p$ decreases exponentially with $m/(p\Delta)$. Item 2 essentially allows the degrees to increase up to $\beta^{|U|} m/\prod_{d \in U} p_d$, where $\beta = O(\delta/\ln p) < 1$: In that case, the probability that the maximum load exceeds $(1+\delta)m/p$ decreases polynomially in p; notice that here there is a lower bound how small we can choose δ. Finally, item 3 allows the degrees to approach their upper bound $\frac{m}{\prod_{d \in U} p_d}$ by requiring δ to be larger, $\delta = \Omega(\ln p/\ln \ln p)$.

The condition $p_i \geq 2$ is without loss of generality: If some $p_i = 1$, then we can simply project away the attribute x_i of R and apply the theorem to the remaining relation. If R is a set (has no duplicate tuples), then after projecting away x_i it may become a bag; this is the reason why Theorem 3.2 is stated in terms of bags.

This theorem immediately implies the following statement on the behavior of the HC algorithm:

Corollary 3.3. *Let* $\mathbf{p} = (p_1, \ldots, p_k)$ *be the shares of the HC algorithm. Fix* $0 < \delta < 1$ *and suppose that every relation* $S_j(x_{i_1}, \ldots, x_{i_r})$ *satisfies the condition of Theorem 3.2 with respect to the integers* $p_{i_1}, \ldots, p_{i_r}$ *and the constants* δ, c. *Then, the probability over the choices of the random hash functions that the maximum load per server of the HC algorithm exceeds*

$$(1+\delta) \max_j \frac{m_j}{\prod_{i:i \in S_j} p_i}$$

is exponentially small when all degrees are bounded by some constant or polynomially small when the degrees are bounded as in Theorem 3.2 items 2 or 3.

Theorem 3.2 follows from repeated application of a Chernoff-like inequality, Lemma 3.4, which, together with its corollary, is of independent interest. Let $D(q'||q) \stackrel{\text{def}}{=} q' \ln(\frac{q'}{q}) + (1-q') \ln(\frac{1-q'}{1-q})$ denote the relative entropy (also known as the KL-divergence) of Bernoulli indicator variables with probabilities q' and q.

Lemma 3.4. *Let* $\mathbf{w} = (w_1, \ldots, w_n)$ *be a sequence of non-negative numbers, and* m, k *be two numbers such that* $||\mathbf{w}||_1 \leq m$ *and* $||\mathbf{w}||_\infty \leq m/k$. *Let* $X_1, \ldots, X_n$ *be a sequence of i.i.d.*[6] *random variables,*

[6] As in Impagliazzo and Kabanets (2010), we can relax the i.i.d. requirement and allow the random variables to be identical, and negatively correlated, that is, for any set $S \subseteq [n]$, $\Pr(\bigwedge_{i \in S} X_i) \leq \mu^{|S|}$.

$X_i \in \{0,1\}$, *and* $\mu = \mathbb{E}[X_i]$. *Then, for all* γ *such that* $\mu < \gamma < 1$, *the following holds:*

$$\mathbb{P}\left(\sum_{i\in[n]} w_i X_i > \gamma \cdot m\right) \le \exp\left(-k \cdot D(\gamma||\mu)\right). \tag{9}$$

Proof. The proof follows along similar lines to constructive proofs of Chernoff bounds in Impagliazzo and Kabanets (2010). We may assume w.l.o.g. that $||\mathbf{w}||_1 = m$. Indeed, if $||\mathbf{w}||_1 < m$, then we extend $\mathbf{w}$ to a longer sequence $\mathbf{w}'$ such that $||\mathbf{w}'||_1 = m$ and $||\mathbf{w}'||_\infty \le m/k$, by defining $w'_{n+1} = w'_{n+2} = \cdots = m/k$, possibly setting the last value $w'_{n'}$ to $< m/k$, to ensure that $\sum_i w'_i = m$; then $\mathbb{P}(\sum_{i=1,n} w_i X_i > \gamma m) \le \mathbb{P}(\sum_{i=1,n'} w'_i X_i > \gamma m)$ (where $X_{n+1}, X_{n+2}, \ldots$ are independent random variables identical to X_1), and the claim for $\mathbf{w}$ follows from the similar claim for $\mathbf{w}'$. Thus, we will assume $||\mathbf{w}||_1 = m$.

Choose a random $S \subseteq [n]$ by including each element i independently with probability q_i:

$$\mathbb{P}(i \in S) = q_i \stackrel{\text{def}}{=} 1 - (1-q)^{kw_i/m}.$$

Let $\mathcal{E}$ denote the event that $\sum_{i\in[n]} w_i X_i \ge \gamma \cdot m$. Then

$$\mathbb{E}\left[\bigwedge_{i\in S} X_i = 1\right] \ge \mathbb{E}\left[\bigwedge_{i\in S} X_i = 1 \mid \mathcal{E}\right] \cdot \mathbb{P}(\mathcal{E}).$$

We bound both expectations. First, we see that

$$\begin{aligned}
\mathbb{E}\left[\bigwedge_{i\in S} X_i = 1\right] &= \sum_{S\subseteq[n]} \mu^{|S|} \prod_{i\in S} q_i \prod_{i\notin S}(1-q_i) = \sum_{S\subseteq[n]} \prod_{i\in S}(\mu q_i) \prod_{i\notin S}(1-q_i) \\
&= \prod_{i\in[n]} (\mu q_i + (1-q_i)) = \prod_{i\in[n]} (\mu + (1-\mu)(1-q_i)) \qquad (10) \\
&= \prod_{i\in[n]} \left(\mu + (1-\mu)(1-q)^{kw_i/m}\right) \\
&\le \prod_{i\in[k]} (\mu + (1-\mu)(1-q)) \qquad (11) \\
&= (1-q(1-\mu))^k. \qquad (12)
\end{aligned}$$

We prove inequality (11). The function $f(w) \stackrel{\text{def}}{=} \mu + (1-\mu)(1-q)^{kw/m}$ is log-convex,[7] therefore the multi-variate function $F(\mathbf{w}) \stackrel{\text{def}}{=} \prod_i f(w_i) = \prod_{i\in[n]}(\mu + (1-\mu)(1-q)^{kw_i/m})$ is also log-convex. Its maximum value on the polytope $\{\mathbf{w} \mid 0 \le w_i \le m/k, \sum_i w_i = m\}$ is obtained at some vertex of the polytope. Every vertex of this polytope consists of $\lfloor k \rfloor$ values $w_i = m/k$, one value $w_i = r \cdot m/k$, where $r \stackrel{\text{def}}{=} k - \lfloor k \rfloor < 1$, and all other values $w_i = 0$. At each such vertex, the value of $F(\mathbf{w})$ is $(\mu + (1-\mu)(1-q))^{\lfloor k \rfloor}(\mu + (1-\mu)(1-q)^r)$ and inequality (11) follows from $\mu + (1-\mu)(1-q)^r \le (\mu + (1-\mu)(1-q))^r$ (by the concavity of the function x^r).

[7] Write $g(w) = \log f(w)$. Then $g(w) = \log(a + be^{-cw})$ for some $a, b, c > 0$; hence $g'(w) = -bce^{-cw}/(a + be^{-cw}) = -bc/(ae^{cw} + b)$ is increasing and so g is convex.

Second, for any outcome of $X_1, \ldots, X_n$ that satisfies $\mathcal{E}$, the probability that S misses all indices i such that $X_i = 0$ is

$$\mathbb{E}\left[\bigwedge_{i \in S} X_i = 1 \mid \mathcal{E}\right] = \prod_{i:X_i=0} (1-q)^{\frac{kw_i}{m}} = (1-q)^{\frac{k\sum_{i:X_i=0} w_i}{m}} \geq (1-q)^{\frac{k(m-\gamma m)}{m}} = (1-q)^{k(1-\gamma)}, \quad (13)$$

since $\mathcal{E}$ implies that $\sum_{i:X_i=0} w_i = m - \sum_{i:X_i=1} w_i \leq m - \gamma m$. Combining Equations (12) and (13), we obtain the follows:

$$\mathbb{P}(\mathcal{E}) \leq \left(\frac{1-q(1-\mu)}{(1-q)^{1-\gamma}}\right)^k.$$

As noted in Impagliazzo and Kabanets (2010), by looking at its first derivative one can show that the function $F(q) = \frac{1-q(1-\mu)}{(1-q)^{1-\gamma}}$ takes its minimum at $q = q^* = \frac{\gamma-\mu}{\gamma(1-\mu)}$, where it has value $e^{-D(\gamma||\mu)}$, and we obtain the following:

$$\mathbb{P}(\mathcal{E}) \leq e^{-k \cdot D(\gamma||\mu)},$$

as required. □

Before we prove Theorem 3.2, we apply this lemma to weighted balls-in-bins.

Corollary 3.5 (Weighted Balls in Bins). *Suppose that n balls of weights $w_1, \ldots, w_n$ with total weight $m = \sum_{i=1}^n w_i$ are each uniformly and independently sent to one of p bins. Let $0 < \delta < p - 1$. If all weights w_i are at most $\Delta = \beta m/p$ for some $\beta > 0$, then*

$$\mathbb{P}\left(\textit{some bin has weight} \geq (1+\delta)m/p\right) \leq p \cdot e^{-D(\frac{1+\delta}{p}||\frac{1}{p}) \cdot p/\beta} \leq p \cdot e^{-h(\delta)/\beta},$$

where $h(x) = (1+x)\ln(1+x) - x$.

Proof. The bound of the corollary using KL divergence follows by applying Lemma 3.4 with $\mu = 1/p$, $\gamma = (1+\delta)/p$, and $k = p/\beta$ and then taking a union bound over the p bins. Now, by definition,

$$\begin{aligned}
D\left(\frac{1+\delta}{p}||\frac{1}{p}\right) &= \frac{(1+\delta)}{p}\ln(1+\delta) + \frac{(p-1-\delta)}{p}\ln\left(1 - \frac{\delta}{p-1}\right) \\
&= \frac{1}{p}\left[(1+\delta)\ln(1+\delta) + (p-1-\delta)\ln\left(1-\frac{\delta}{p-1}\right)\right] \\
&= \frac{1}{p}\left[(1+\delta)\ln(1+\delta) - (p-1-\delta)\left(\frac{\delta}{(p-1)} + \frac{\delta^2}{2(p-1)^2} + \frac{\delta^3}{3(p-1)^3} + \cdots\right)\right] \\
&= \frac{1}{p}\left[(1+\delta)\ln(1+\delta) - \left(\delta - \frac{\delta^2}{2(p-1)} - \frac{\delta^3}{6(p-1)^2} - \cdots\right)\right] \\
&\geq \frac{1}{p}[(1+\delta)\ln(1+\delta) - \delta] = h(\delta)/p,
\end{aligned}$$

which yields the bound in terms of the function h. □

This corollary naturally bounds the result of applying a uniformly random hash function from $[n]$ to $[p]$ to a bag of m elements where w_i is the multiplicity of item i.

Remark 3.6. We observe that the failure bound in Corollary 3.5 expressed in terms of $h(\delta)$ also trivially holds even when $(1+\delta)/p \geq 1$ since the failure probability is 0 in that case. In general, that failure probability has the following qualitative properties:

(1) When β is $O(p/m)$ (equivalently, Δ is constant), the failure bound is similar to the usual Chernoff bounds, decaying exponentially in m/p for every constant $\delta > 0$.

(2) When β is $O(1/\ln p)$, there is a constant $\delta > 0$ for which the failure bound is $< 1/p$.
(3) When β is a constant, there is some δ that is $\Theta(\ln p/\ln\ln p)$ such that the failure bound is $< 1/p$.

We generalize now Lemma 3.4 from a sequence (w_i) to a multi-dimensional array. If U is a finite set, then we call $I \in [n]^U$ a *U-tuple* and for $d \in U$, we write the corresponding lower-case i_d for the component of I indexed by d. A *multi-dimensional array with dimensions* U, or U-dimensional array, is a sequence $\mathbf{w} = (w_I)_{I\in[n]^U}$ of non-negative numbers. If I and J are U-tuples and V-tuples, respectively, where U and V are disjoint, then I, J is the $(U \cup V)$-tuple that combines both tuples. If $\mathbf{w}$ is a U-dimensional array, $V \subseteq U$ and $J \in [n]^V$, then $w_J \in [n]^{U-V}$ denotes the $(U - V)$-dimensional array $(w_J)_K \stackrel{\text{def}}{=} w_{J,K}$ for $K \in [n]^{V_U}$. We identify $[n]^r$ with $[n]^{[r]}$ and use the standard norms $||\mathbf{w}||_1 = \sum_{I\in[n]^U} w_I$ and $||\mathbf{w}||_\infty = \max_{I\in[n]^U} w_I$.

Theorem 3.7. *Let $r > 0$ and $\mathbf{w} = (w_I)_{I\in[n]^r}$ be an r-dimensional array of non-negative weights. Let $m, k_1, \ldots, k_r > 0$ be numbers such that $\mathbf{w}$ satisfies the following* degree constraints*:*

$$\forall\, U \subseteq [r]\ \ \forall\, J \in [n]^U, \quad ||w_J||_1 \le \frac{m}{\prod_{d\in U} k_d}. \tag{14}$$

For each $d \in [r]$, let $\mathbf{X}^{(d)} = (X_1^{(d)}, \ldots, X_n^{(d)})$ be a sequence of i.i.d. random variables $X_i^{(d)} \in \{0, 1\}$, let $\mu_d = \mathbb{E}[X_i^{(d)}]$, and let γ_d satisfy $\mu_d < \gamma_d < 1$. Further assume that all variables $(X_i^{(d)})_{i\in[n],d\in[r]}$ are independent, and for $I \in [n]^U$, write $X_I \stackrel{\text{def}}{=} \prod_{d\in U} X_{i_d}^{(d)}$.

Then

$$\mathbb{P}\left(\sum_{I\in[n]^r} w_I X_I > \left(\prod_{d\in[r]} \gamma_d\right)\cdot m\right) \le 2\sum_{\ell\in[r]}\prod_{d\in[\ell-1]} (k_d+1)\cdot\exp\left(-k_\ell\cdot D(\gamma_\ell||\mu_\ell)\right).$$

Proof. We prove the theorem by induction on r. Note that, in particular, Equation (14) implies that $||\mathbf{w}||_1 \le m$ and $||\mathbf{w}||_\infty \le \frac{m}{\prod_{d\in[r]} k_d}$. The case when $r = 1$ follows immediately from Lemma 3.4, so we assume that $r > 1$. By definition,

$$\sum_{I\in[n]^r} w_I X_I = \sum_{J\in[n]^{r-1}}\sum_{i_r\in[n]} w_{J,i_r} X_J X_{i_r}^{(r)} = \sum_{J\in[n]^{r-1}} w_J^{\mathbf{X}^{(r)}} X_J,$$

where we write J, i_r for the r-tuple whose first $r - 1$ components are given by J and whose last component is i_r and where

$$w_J^{\mathbf{X}^{(r)}} \stackrel{\text{def}}{=} \sum_{i_r\in[n]} w_{J,i_r} X_{i_r}^{(r)},$$

yielding an $(r - 1)$-dimensional array of weights, denoted $\mathbf{w}^{\mathbf{X}^{(r)}}$, that depends on the outcomes of the random variables $\mathbf{X}^{(r)}$.

Thus, if we write $m' \stackrel{\text{def}}{=} \gamma_r \cdot m$, then we have

$$\mathbb{P}\left(\sum_{I\in[n]^r} w_I X_I > \left(\prod_{d\in[r]}\gamma_d\right)\cdot m\right) = \mathbb{P}\left(\sum_{J\in[n]^{r-1}} w_J^{\mathbf{X}^{(r)}} X_J > \left(\prod_{d\in[r-1]}\gamma_d\right)\cdot m'\right).$$

We will show that, except for a small failure probability, the $(r - 1)$-dimensional array $\mathbf{w}^{\mathbf{X}^{(r)}}$ satisfies the analog of Equation (14) with respect to m' and $r - 1$, and hence we can apply the inductive hypothesis to this $(r - 1)$-dimensional subproblem to achieve the final bound on the failure probability.

Consider some $U \subseteq [r-1]$. The probability that the analog of Equation (14) with respect to $r-1$ and m' fails for U is

$$\mathbb{P}\left(\exists J \in [n]^U.\ ||w_J^{X^{(r)}}||_1 > \frac{m'}{\prod_{d\in U} k_d}\right) = \mathbb{P}\left(\exists J \in [n]^U.\ ||\sum_{i_r\in[n]} w_{J,i_r} X_{i_r}^{(r)}||_1 > \gamma_r \frac{m}{\prod_{d\in U} k_d}\right)$$
$$= \mathbb{P}\left(\exists J \in [n]^U.\ \sum_{i_r\in[n]} ||w_{J,i_r}||_1 \cdot X_{i_r}^{(r)} > \gamma_r \frac{m}{\prod_{d\in U} k_d}\right).$$

For simplicity of notation, for each $J \in [n]^U$, define $\mathbf{v}^{(J)} \in \mathbb{R}^n$ by $v_{i_r}^{(J)} \stackrel{\text{def}}{=} ||w_{J,i_r}||_1$ for $i_r \in [n]$. With this notation, the probability that the analog of Equation (14) fails for U is

$$\mathbb{P}\left(\exists J \in [n]^U.\ \sum_{i_r\in[n]} v_{i_r}^{(J)} \cdot X_{i_r}^{(r)} > \gamma_r \frac{m}{\prod_{d\in U} k_d}\right). \tag{15}$$

By Equation (14), for each J,

$$||\mathbf{v}^{(J)}||_1 = ||w_J||_1 \le \frac{m}{\prod_{d\in U} k_d}, \text{ and}$$
$$||\mathbf{v}^{(J)}||_\infty = \max_{i_r\in[n]} ||w_{J,i_r}||_1 \le \frac{m}{\prod_{d\in U\cup\{r\}} k_d}.$$

We could apply Lemma 3.4 to the one-dimensional array $\mathbf{v}^{(J)}$ to obtain a bound on $\mathbb{P}(\sum_{i_r\in[n]} v_{i_r}^{(J)} \cdot X_{i_r}^{(r)} > \gamma_r \frac{m}{\prod_{d\in U} k_d})$, but Equation (15) is the union of $n^{|U|}$ such events. Instead, we will show that we can partition $[n]^U$ into at most $K = 2\prod_{d\in U} k_d$ sets $B_1, \ldots, B_K$, and bound Equation (15) as a union of only K events, and then apply Lemma 3.4 to each. To perform this partition we use the fact that:

$$\sum_{J\in[n]^U} ||\mathbf{v}^{(J)}||_1 = \sum_{J\in[n]^U} ||w_J||_1 = ||\mathbf{w}||_1 \le m.$$

We partition the set $[n]^U$ into "bins" $B_1, \ldots, B_K$ by applying a greedy bin-packing algorithm using the $||\cdot||_1$ norm of the $\mathbf{v}^{(J)}$ vectors with bins of capacity $C = m/\prod_{d\in U} k_d$. That is, we consider the tuples J in decreasing order of $||\mathbf{v}^{(J)}||_1$ value and continue to add J to the current bin while the total of the $||\mathbf{v}^{(J)}||_1$ values is at most $m/\prod_{d\in U} k_d$. Since the J are considered in decreasing order of $||\mathbf{v}^{(J)}||_1$, for each bin B_j we have $\sum_{J\in B_j} ||\mathbf{v}^{(J)}||_1 > C/2$. From this and the fact that $\sum_{J\in[n]^U} ||\mathbf{v}^{(J)}||_1 \le m$ the greedy packing uses at most $2m/C = 2\prod_{d\in U} k_d = K$ bins as claimed. (An algorithm like first-fit decreasing might improve the packing, but it will not improve the worst-case analysis.) For each bin B_j, define vector $\mathbf{u}^{(j)} \in \mathbb{R}^n$ by

$$u_{i_r}^{(j)} \stackrel{\text{def}}{=} \max_{J\in B_j} v_{i_r}^{(J)}$$

and observe that

$$||\mathbf{u}^{(j)}||_1 \le \sum_{J\in B_j} ||\mathbf{v}^{(J)}||_1 \le \frac{m}{\prod_{d\in U} k_d}$$
$$||\mathbf{u}^{(j)}||_\infty \le \max_{J\in B_j} ||v^{(J)}||_\infty \le \frac{m}{\prod_{d\in U\cup\{r\}} k_d}, \text{ and}$$
$$\sum_{i_r\in[n]} u_{i_r}^{(j)} \cdot X_{i_r}^{(r)} \ge \sum_{i_r\in[n]} v_{i_r}^{(J)} \cdot X_{i_r}^{(r)} \text{ for all } J \in B_j.$$

For each $j \in [K]$, applying Lemma 3.4 with $\mathbf{w}$ replaced by $\mathbf{u}^{(j)}$, m replaced by $\frac{m}{\prod_{d\in U} k_d}$, γ replaced by γ_r, μ replaced by μ_r, and k replaced by k_r, we obtain that the total probability in Equation (15) is at most

$$2\prod_{d\in U} k_d \cdot \exp(-k_r \cdot D(\gamma_r||\mu_r)).$$

Thus the probability that there is some $U \subseteq [r-1]$ for which the analog of Equation (14) for $r-1$ and m' does not hold is at most

$$\sum_{U\subseteq[r-1]} 2\prod_{d\in U} k_d \cdot \exp(-k_r \cdot D(\gamma_r||\mu_r)) = 2\prod_{d\in[r-1]} (k_d+1) \cdot \exp(-k_r \cdot D(\gamma_r||\mu_r)).$$

In the remaining case when Equation (14) does hold for $r-1$ and m', we can apply the inductive hypothesis to say that the failure probability is at most

$$2\sum_{\ell\in[r-1]} \prod_{d\in[\ell-1]} (k_d+1) \cdot \exp(-k_\ell \cdot D(\gamma_\ell||\mu_\ell)),$$

and hence the total failure probability is at most

$$2\sum_{\ell\in[r]} \prod_{d\in[\ell-1]} (k_d+1) \cdot \exp(-k_\ell \cdot D(\gamma_\ell||\mu_\ell))$$

as required. □

We now apply this theorem to prove our bounds for multidimensional hashing.

Proof of Theorem 3.2. The value of the expectation follows immediately by the independence and uniformity of the individual hash functions.

Suppose that for every tuple J over $U \subseteq [r]$ we have

$$d_J(R) \le \beta^{|U|} \frac{m}{\prod_{d\in U} p_d}$$

for some $\beta > 0$. For every $(i_1,\ldots,i_r) \in [n]^{[r]}$, define $w_{i_1,\ldots,i_r}$ to be the multiplicity of $(i_1,\ldots,i_r)$ in R; these together comprise an $[r]$-dimensional non-negative array $\mathbf{w}$. If we set $k_d = p_d/\beta$ for all $d \in [r]$, then the bounds on $d_J(R)$ for all $J \subseteq [r]$ imply that property (14) holds for $\mathbf{w}$. Fix one of the p servers and, for all $d \in [r]$, $i_d \in [n]$, define the random indicator variable $X_{i_d}^{(d)} = \begin{cases} 1 & \text{if } h_d(i_d) = y_d \\ 0 & \text{otherwise} \end{cases}$. Since the hash functions are drawn from a set of independent and uniformly random hash functions, $X_{i_d}^{(d)}$ are independent random variables; moreover, $\mu_d \stackrel{\text{def}}{=} \mathbb{E}[X_{i_d}^{(d)}] = 1/p_d$ and for $\delta' > 0$ define $\gamma_d = (1+\delta')\mu_d = (1+\delta')/p_d$ for each $d \in [r]$.

Suppose, first, that $\gamma_d < 1$ for all $d \in [r]$. Then we can apply Theorem 3.7 to say that the probability that this server receives more than $(1+\delta')^r m/p$ tuples is at most

$$2\sum_{\ell\in[r]} \prod_{d\in[\ell-1]} (p_d/\beta+1) \cdot \exp\left(-p_d \cdot D\left(\frac{1+\delta'}{p_d}||\frac{1}{p_d}\right)/\beta\right)$$

and, by Corollary 3.5, this is at most

$$2\sum_{\ell\in[r]} \prod_{d\in[\ell-1]} (p_d/\beta+1) \cdot \exp(-h(\delta')/\beta). \tag{16}$$

As noted in Remark 3.6, this bound also holds when $\gamma_d = (1+\delta')/p_d \ge 1$ for some choices of d, because the failure probabilities of the corresponding terms in the induction of Theorem 3.7 are actually 0. Therefore, we can avoid the hypothesis that $\gamma_d < 1$, and conclude that the probability that any server receives more than $(1+\delta')^r m/p$ tuples is at most p times Equation (16).

We now consider constraints on β and δ' that are sufficient to prove the various parts of the theorem. For $\beta = 1$, which corresponds to item (3), we have $p_d/\beta + 1 = p_d + 1 < p_d^{5/3}$ since $p_d \geq 2$. Then

$$2 \sum_{\ell \in [r]} \prod_{d \in [\ell-1]} (p_d/\beta + 1) \leq 2 \sum_{\ell \in [r]} \prod_{d \in [\ell-1]} p_d^{5/3} \leq 2 \sum_{\ell \in [r]} p^{5/3}/3^{r-\ell+1} \leq p^{5/3},$$

and the total failure probability is at most $p^{8/3} e^{-h(\delta')}$. Plugging in $1 + \delta' = (c + 8/3) \ln p / \ln \ln p$ we have that the probability that any server receives more than $(1 + \delta')^r m/p$ tuples is at most p^{-c}.

Next consider the case that $\delta' = \delta/(2r)$ where $\delta \leq 2$. Then $(1 + \delta')^r \leq e^{\delta/2} \leq 1 + \delta$ and the Taylor series expansion for $h(x)$ in this range yields that $h(\delta') \geq \frac{\delta^2}{3r^2}$. If we now set $\beta = \frac{\delta^2}{c' r^2 \ln p}$, then $e^{-h(\delta')/\beta} \leq p^{-c'/12}$. In this case,

$$p_d/\beta + 1 = \frac{c' r^2 p_d \ln p}{\delta^2} + 1 < \frac{c' r^2 \ln p}{\delta^2} \cdot (p_d + 1).$$

Then

$$2 \sum_{\ell \in [r]} \prod_{d \in [\ell-1]} (p_d/\beta + 1) < \left(\frac{c' r^2 \ln p}{\delta^2} \right)^{r-1} p^{5/3}$$

by the same reasoning as in item (3). Using the lower bound on δ in terms of p and r and choosing $c' = 24cr$ for $c \geq 2$ yields that the probability that any server receives more than $(1 + \delta)m/p$ tuples is bounded above by p^{-cr}. This proves item (2).

Finally, suppose that for every tuple J we have that $d_J(R)$ is at most Δ. In this case we choose β such that for every J on subset $U \subseteq [r]$, $\beta^{|U|} m / \prod_{d \in U} p_d \geq \Delta$; that is,

$$\beta = \max_{U \subseteq [r]} \left(\frac{\Delta \prod_{d \in U} p_d}{m} \right)^{1/|U|} = (\Delta p/m)^{1/r},$$

since the bound is clearly monotone increasing in $|U|$. Again choosing $\delta' = \delta/(2r)$, we obtain that the probability that any server has load more than $(1 + \delta)m/p$ is at most $\beta^{-(r-1)} p^{8/3} e^{-h(\delta')/\beta}$ which is exponentially small in $1/\beta = (m/(p\Delta))^{1/r}$, yielding item (1). □

3.1.2 Choosing the Shares. Here we discuss how to compute the shares p_i to optimize the expected load per server. Afrati and Ullman (2010) compute the shares by optimizing the total load $\sum_j m_j / \prod_{i: i \in S_j} p_i$ subject to the constraint $\prod_i p_i = p$, which is a non-linear system that can be solved using Lagrange multipliers. Our approach is to optimize the maximum load *per relation*, $\max_j m_j / \prod_{i: i \in S_j} p_i$; the total load per server is at most ℓ times larger. This leads to a linear optimization problem, as follows. We express here the load in terms of the derived statistics $M_j = a_j m_j \log n$, as $L = \max_j M_j / \prod_{i: i \in S_j} p_i$, in to facilitate the connection to the lower bounds in Section 3.2. First, write the shares as $p_i = p^{e_i}$ where $e_i \in [0, 1]$ is called the *share exponent* for x_i, denote $\lambda = \log_p L$ and $\mu_j = \log_p M_j$ (we will assume w.l.o.g. that $M_j \geq p$, hence $\mu_j \geq 1$ for all j). Then, we optimize the LP:

$$\begin{aligned} \text{minimize} \quad & \lambda \\ \text{subject to} \quad & \sum_{i \in [k]} -e_i \geq -1 \\ & \forall j \in [\ell] : \sum_{i \in S_j} e_i + \lambda \geq \mu_j \\ & \forall i \in [k] : e_i \geq 0, \quad \lambda \geq 0. \end{aligned} \tag{17}$$

Let e^* be the optimal value of the solution to Equation (17), and denote $L^{upper} = p^{e^*}$.

Theorem 3.8 (Upper Bound). *Fix a query q over a database with relations $S_1, \ldots, S_\ell$, cardinalities $\mathbf{m} = (m_1, \ldots, m_\ell)$, and derived statistics $\mathbf{M} = (M_1, \ldots, M_\ell)$, and let p be a number of servers. Let $\mathbf{e} = (e_1, \ldots, e_k)$ be the optimal solution to Equation (17) and denote $p_i = p^{e_i}$. Let $\beta > 0$ be some constant such that, for every relation S_j and every tuple J over $U \subseteq [a_j]$, its degree is bounded by $d_J(S_j) \leq \frac{\beta^{|U|} m_j}{\prod_{i \in U} p_i}$. Then the HC algorithm with shares p_i achieves $O(\ell L^{upper})$ maximum load with high probability.*

A special case of interest is when all sizes M_j are equal, therefore $\mu_1 = \cdots = \mu_\ell = \mu$. In that case, the optimal solution to Equation (17) can be obtained from an optimal fractional vertex cover $\mathbf{v}^* = (v_1^*, \ldots, v_k^*)$ by setting $e_i = v_i^*/\tau^*$ (where $\tau^* = \sum_i v_i^*$). To see this, we note that any feasible solution $(\lambda, e_1, \ldots, e_k)$ to Equation (17) defines the vertex cover $v_i = e_i/(\mu - \lambda)$, and in the opposite direction every vertex cover defines the feasible solution $e_i = v_i/(\sum_i v_i)$, $\lambda = \mu - 1/(\sum_i v_i)$; furthermore, minimizing λ is equivalent to minimizing $\sum_i v_i$. Thus, when all relations have the same size M, at optimality $\lambda^* = \mu - 1/\tau^*$, and $L^{upper} = M/p^{1/\tau^*}$. We illustrate more examples in Section 3.5.

3.2 The Lower Bound

In this section, we prove a lower bound on the maximum load per server over databases with sizes $\mathbf{M}$ (expressed in bits); our bounds apply immediately to the special case when $\mathbf{M}$ are derived from cardinalities $\mathbf{m}$, $M_j = a_j m_j \log n$, but hold in more general cases when M_j is much smaller than $a_j m_j \log n$; with some abuse, we call $\mathbf{M}$ derived statistics in all cases.

Fix a query q and a fractional edge packing $\mathbf{u}$ of q. Denote:

$$\mathcal{L}(\mathbf{u}, \mathbf{M}, p) = \left(\frac{\prod_{j=1}^{\ell} M_j^{u_j}}{p} \right)^{1/\sum_j u_j} . \tag{18}$$

Further denote $L^{lower} = \max_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \mathbf{M}, p)$, where $\mathbf{u}$ ranges over all edge packings for q. In the next two sections we prove (more detailed versions of) the following theorem, which shows that L^{lower} yields a lower bound on the load required by any p-processor one-round MPC algorithm computing q over a database with derived statistics $\mathbf{M}$.

Theorem 3.9. *Let $\mathbf{M}$ be the vector of derived statistics for the input database to a query q. Let p be the number of processors.*

- *(a) There is a (natural) probability distribution $\mathcal{I}$ on databases with statistics $\mathbf{M}$ such that for any one-round p-processor deterministic MPC algorithm A with load $L < L^{lower}/4$, the expected number of outputs produced by A on input I chosen from $\mathcal{I}$ is strictly less than the expected number of outputs required to compute q on input I.*
- *(b) There is a constant $\delta > 0$ such that the worst-case load L for any one-round p-processor randomized MPC algorithm to compute a conjunctive query q having c components with probability at least $1/2^c$ requires $L \geq c\delta \cdot L^{lower}$.*

To gain some intuition behind the formula (18) and L^{lower}, consider the case when all derived statistics are equal, $M_1 = \cdots = M_\ell = M$. Then $L^{lower} = M/p^{1/\sum_j u_j}$, and this quantity is maximized when $\mathbf{u}$ is a maximum fractional edge packing, whose value is τ^*, the fractional vertex covering number for q. Thus, $L^{lower} = M/p^{1/\tau^*}$, which is the same expression as L^{upper}. In fact, this equivalence holds in general:

Theorem 3.10 (Equivalence). *For any vector of derived statistics* $\mathbf{M}$ *and number of processors* p*, we have the following:*

$$L^{lower} = L^{upper} = \max_{\mathbf{u} \in pk(q)} \mathcal{L}(\mathbf{u}, \mathbf{M}, p),$$

where $pk(q)$ *is the set of extreme points of the convex polytope defined by the fractional edge packing constraints in Equation (2).*

If additionally $M_1 = \cdots = M_\ell = M$*, then*

$$L^{lower} = L^{upper} = M/p^{1/\tau^*},$$

where τ^* *is the fractional vertex covering number for* q*.*

Together, Theorems 3.9 and 3.10 immediately imply the following theorem.

Corollary 3.11 (Optimality Without Skew). *Under the conditions of Theorem 3.8, the HC algorithm with shares chosen as in that theorem, achieves a load that is within a constant factor of the optimal load for any one-round MPC algorithm where the constant depends only linearly on the ratio of the number of relations to the number of connected components in the query* q*.*

In the remainder of this section, we prove Theorem 3.10 and then prove Theorem 3.9 in the next two sections.

Proof of Theorem 3.10. The open segment defined by two points $x, y \in \mathbb{R}^n$ is the set $\{(1-t)x + ty \mid t \in (0,1)\}$. A vertex of a polytope $P \subseteq \mathbb{R}^n$ is a point $z \in P$ that that does not belong to any open segment defined by two points $x, y \in P$. Let $V(P)$ denote the set of vertices of P. When P is given by a set of linear inequalities, each vertex is obtained by converting a maximal set of inequalities into equalities. If P is the fractional edge packing polytope given by Equation (2), then $V(P) = pk(q)$ and $|pk(q)| \leq \binom{k+\ell}{\ell}$, because each vertex is obtained by converting ℓ of the $k + \ell$ inequalities into equalitiesand then solving for the ℓ variables $\mathbf{u}$.

By definition, $L^{upper} = p^{e^*}$, where e^* is the optimal solution to the *primal* LP problem (17). Consider its *dual* LP:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j \in [\ell]} \mu_j f_j - f_0 \\
\text{subject to} \quad & \sum_{j \in [\ell]} f_j \leq 1 \\
\forall i \in [k] : & \sum_{j : i \in S_j} f_j - f_0 \leq 0 \\
\forall j \in [\ell] : & f_j \geq 0, \quad f_0 \geq 0.
\end{aligned} \tag{19}$$

Let $A \subseteq \mathbb{R}^{\ell+1}$ denote the polytope of feasible solutions, $\varphi : A \to \mathbb{R}$ denote the objective function $\varphi(f_0, f_1, \ldots, f_\ell) = \sum_j \mu_j f_j - f$, and $f^* = \max \varphi(A)$ denote the optimal value. Let $A_0 \subset A$ be the subset of feasible solutions where $f_0 > 0$. (A_0 is no longer a polytope.) We claim that $f^* = \max \varphi(A_0)$. Indeed, by the primal-dual theorem, we have $f^* = e^*$, and, assuming $\mu_j > 1$ for all j, we have $e^* > 0$, because $\lambda = 0$ is not part of any feasible solution of the primal LP. Since $f^* > 0$, for any optimal solution $\mathbf{f}$ of the dual LP at least one $f_j > 0$, and thus $\mathbf{f} \in A_0$.

Consider now the following non-linear optimization problem:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{u_0} \cdot \left(\sum_{j \in [\ell]} \mu_j u_j - 1 \right) \\ \text{subject to} \quad & \sum_{j \in [\ell]} u_j \le u_0 \\ & \forall i \in [k] : \sum_{j : i \in S_j} u_j \le 1 \\ & \forall j \in [\ell] : u_j \ge 0, \quad u_0 > 0. \end{aligned} \tag{20}$$

Let $B_0 \subseteq \mathbb{R}^{\ell+1}$ denote the set of feasible solutions, and let $B \supset B_0$ denote the polytope obtained by relaxing $u_0 > 0$ to $u_0 \ge 0$. Let $\psi : B_0 \to \mathbb{R}$ denote the objective function, $\psi(u_0, u_1, \ldots, u_\ell) = (\sum_j \mu_j u_j - 1)/u_0$, and $u^* = \sup \psi(B_0)$ the optimal value. First, we prove that $u^* = f^*$ and, moreover, ψ reaches the maximum in B_0, that is, $\sup \psi(B_0) = \max \psi(B_0)$. Let $F : \mathbb{R}^{\ell+1} \to \mathbb{R}^{\ell+1}$ be the function $F(u_0, u_1, \ldots, u_\ell) = (1/u_0, u_1/u_0, \ldots, u_\ell/u_0)$. Then it is easily verified that F maps B_0 to A_0, $F(B_0) \subseteq A_0$, meaning that for any feasible solution $(u_0, u_1, \ldots, u_\ell) \in B_0$, its image is a feasible solution $(f_0, f_1, \ldots, f_\ell) \in A_0$. We also check that $F(B_0) = A_0$ by noting that F has an inverse, $F^{-1}(f_0, f_1, \ldots, f_\ell) = (1/f_0, f_1/f_0, \ldots, f_\ell/f_0)$, and $F^{-1}(A_0) \subseteq B_0$. Moreover, $\varphi \circ F = \psi$, because $\varphi(F(u_0, u_1, \ldots, u_\ell)) = \sum_j \mu_j u_j/u_0 - 1/u_0 = (\sum_j \mu_j u_j - 1)/u_0 = \psi(u_0, \ldots, u_\ell)$. This implies that $f^* = \max \varphi(A_0) = \max \varphi(F(B_0)) = \max \psi(B_0) = u^*$.

Second, we prove that the optimal value u^* is attained at a vertex of the polytope B, that is, at some $(u_0, u_1, \ldots, u_\ell) \in B_0 \cap V(B)$. The objective of Equation (19) is maximized at a vertex of the polytope A, hence it is obtained at some vertex in $A_0 \cap V(A)$. Thus, it suffices to show that F maps the vertices $B_0 \cap V(B)$ one-to-one to $A_0 \cap V(A)$, or, equivalently, that it maps a non-vertex to a non-vertex. Consider a non-vertex $t\mathbf{u} + t'\mathbf{u}' \in B_0$, where $\mathbf{u} = (u_0, u_1, \ldots, u_\ell) \in B_0$, $\mathbf{u}' = (u_0', u_1', \ldots, u_\ell') \in B_0$, $t, t' \in (0, 1)$ and $t + t' = 1$. It follows by direct calculation that $F(t\mathbf{u} + t'\mathbf{u}') = \frac{tu_0}{tu_0 + t'u_0'} F(\mathbf{u}) + \frac{t'u_0'}{tu_0 + t'u_0'} F(\mathbf{u}')$, proving that it is also a non-vertex.

Third, we prove that we can add the constraint $u_0 = \sum_{j=1,\ell} u_j$ to Equation (20) without affecting the optimal value u^*. Indeed, if $(u_0, u_1, \ldots, u_\ell)$ is any optimal solution for Equation (20), then $u^* = (\sum_j \mu_j u_j - 1)/u_0 \le (\sum_j \mu_j u_j - 1)/(\sum_{j=1,\ell} u_j)$, because $u_0 \ge \sum_{j=1,\ell} u_j$, and $\sum_j \mu_j u_j - 1 > 0$ (since $u^* = f^* > 0$ when $\mu_j > 1$ for all j, as we have shown earlier).

Finally, we observe that a vector $(\sum_j u_j, u_1, \ldots, u_\ell)$ is a feasible solution of Equation (20) iff $\mathbf{u} = (u_1, \ldots, u_\ell)$ is a fractional edge packing of the hypergraph of the query q. This implies that $u^* = \max_{\mathbf{u} \in pk(q)} (\sum_j \mu_j u_j - 1)/(\sum_j u_j)$, and the first claim of the theorem follows from

$$L^{upper} = p^{e^*} = p^{u^*} = \max_{\mathbf{u} \in pk(q)} p^{(\sum_j \mu_j u_j - 1)/(\sum_j u_j)} = \max_{\mathbf{u} \in pk(q)} \left(\frac{\prod_j^\ell M_j^{u_j}}{p} \right)^{1/\sum_j u_j} = L^{lower}.$$

The second claim follows immediately by noting that, when $M_1 = \cdots = M_\ell = M$, then $\mathcal{L}(\mathbf{u}, \mathbf{M}, p) = M/p^{1/\sum_j u_j}$ and therefore $\max_{\mathbf{u} \in pk(q)} \mathcal{L}(\mathbf{u}, \mathbf{M}, p) = M/p^{1/\max_{\mathbf{u} \in pk(q)} \sum_j u_j} = M/p^{1/\tau^*}$. □

3.3 Lower Bound in Expectation

In this section, we prove part (a) of Theorem 3.9. The analysis in this section will also be used in the following section to extend the lower bound to randomized algorithms and prove part (b) of Theorem 3.9.

To prove the lower bound, we will define a probability space from which the input databases are drawn. Given the cardinalities of the ℓ relations of q, $\mathbf{m}$, we first choose a domain size $n \ge \max_j m_j$,

to be specified later, and choose independently and uniformly each relation S_j from all matchings of $[n]^{a_j}$ with exactly m_j tuples. This will determine the derived statistics $\mathbf{M}$. We call this the *matching probability space*. Observe that the probability space contains only databases with relations without skew (in fact, all degrees are exactly 1). We write $\mathbb{E}[|q(I)|]$ for the expected number of answers to q under the above probability space.

Theorem 3.12 (Lower Bound in Expectation). *Fix cardinalities* $\mathbf{m}$*, and consider any deterministic MPC algorithm that runs in one communication round on p servers, and outputs a subset of the answers of a query q on an input database with statistics* $\mathbf{m}$*. Let* $\mathbf{u}$ *be any fractional edge packing of q. If s is any server and L_s is its load, then server s reports at most*

$$\frac{L_s^{\sum_j u_j}}{(\sum_j u_j/4)^{\sum_j u_j} \prod_{j=1}^{\ell} M_j^{u_j}} \cdot \mathbb{E}[|q(I)|]$$

answers in expectation, where I is randomly chosen from the matching probability space with cardinalities $\mathbf{m}$ *and domain size $n = (\max_j m_j)^2$, where $\mathbf{M} = (M_1, \ldots, M_\ell)$ are the derived statistics. Therefore, the p servers of the algorithm report at most*

$$\left(\frac{4L}{(\sum_j u_j) \cdot \mathcal{L}(\mathbf{u}, \mathbf{M}, p)}\right)^{\sum_j u_j} \cdot \mathbb{E}[|q(I)|]$$

answers in expectation, where $L = \max_{s \in [p]} L_s$ is the maximum load of all servers. The same bounds also hold if all relations have equal cardinalities $m_1 = \cdots = m_\ell = m$, arity $a_j \geq 2$, and $n = m$.

Before we prove Theorem 3.12, we show how it implies part (a) of Theorem 3.9.

Proof of Theorem 3.9(a). Let $\mathbf{u}^*$ denote a fractional edge packing in $pk(q)$ that achieves $\mathcal{L}(\mathbf{u}^*, \mathbf{M}, p) = L^{lower}$, which must exist by Theorem 3.10. Since $\mathbf{u}^* = (u_1^*, \ldots, u_\ell^*)$ is an optimal vertex of $pk(q)$, we have $\sum_j u_j^* \geq 1$. Therefore, by Theorem 3.12, the expected number of output tuples that the algorithm produces on input I is strictly less than the number $\mathbb{E}[|q(I)|]$ of tuples required in expectation, unless $L \geq (\sum_j u_j^*) \cdot \mathcal{L}(\mathbf{u}^*, \mathbf{M}, p)/4 \geq L^{lower}/4$. In the case where all relations have equal size and arity at least 2, we obtain $L \geq L^{lower}$. □

To prove Theorem 3.12 we first analyze $\mathbb{E}[|q(I)|]$, which is a particularly nice function of the characteristic of the query when the relations have equal sizes.

Lemma 3.13. *The expected number of answers to q is $\mathbb{E}[|q(I)|] = n^{k-a} \prod_{j=1}^{\ell} m_j$. In particular, if $n = m_1 = \cdots = m_\ell$, then $\mathbb{E}[|q(I)|] = n^{c-\chi(q)}$, where c is the number of connected components of q.*

Proof. For any relation S_j, and any tuple $\mathbf{a}_j \in [n]^{a_j}$, the probability that S_j contains $\mathbf{a}_j$ is $P(\mathbf{a_j} \in S_j) = m_j/n^{a_j}$. Given a tuple $\mathbf{a} \in [n]^k$ of the same arity as the query answer, let $\mathbf{a}_j$ denote its projection on the variables in S_j. Then:

$$\begin{aligned}\mathbb{E}[|q(I)|] &= \sum_{\mathbf{a} \in [n]^k} P\left(\bigwedge_{j=1}^{\ell} (\mathbf{a}_j \in S_j)\right) = \sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} P(\mathbf{a}_j \in S_j) \\ &= \sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} m_j n^{-a_j} = n^{k-a} \prod_{j=1}^{\ell} m_j.\end{aligned}$$

□

The key idea for the rest of the argument is that, since the algorithm is not allowed to report any false positives, before a server can output any tuple, it must be *certain* that the tuple is in the answer, based on the messages that it has received during the single communication round. We

show how the number of bits received imposes on upper bound on the number of certain tuples in the answer. We first show that a bound on the number of bits received about one input relation imposes an upper bound on the expected number (over the random choice of the input database) of the tuples that the server is certain belong to that relation. Second, we apply Friedgut's inequality to bound the expected number of certain tuples in the query's answer as a function of the expected number of certain tuples in each input relation.

Let us fix some server $s \in [p]$, and let $\mathrm{msg}(I)$ denote the function specifying the message the server receives on input I. Recall that, in the input-sever model, each input relation S_j is stored at a separate input server, and therefore the message received by s consists of ℓ separate messages $\mathrm{msg}_j = \mathrm{msg}_j(S_j)$, for each $j = 1, \ldots, \ell$. One should think of msg_j is a bit string. Once the server s receives msg_j it "knows" that the input relation S_j is in the set $\{S_j \mid \mathrm{msg}_j(S_j) = \mathrm{msg}_j\}$. This justifies the following definition: Given a message msg_j, *the set of tuples known by the server* is as follows:

$$K_{\mathrm{msg}_j}(S_j) = \{t \in [n]^{a_j} \mid \text{ for all instances } S_j \subseteq [n]^{a_j}, \mathrm{msg}_j(S_j) = \mathrm{msg}_j \Rightarrow t \in S_j\},$$

where a_j is the arity of S_j. We also define $K_{\mathrm{msg}}(q)$ to be the set of output tuples known by the server on receiving a message msg.

Clearly, an algorithm A may output a tuple $\mathbf{a} \in [n]^k$ as answer to the query q iff, for every j, $\mathbf{a}_j \in K_{\mathrm{msg}_j}(S_j)$ for all $j = 1, \ldots, \ell$, where $\mathbf{a}_j$ denotes the projection of $\mathbf{a}$ on the variables in the atom S_j.

We will first prove an upper bound for each $|K_{\mathrm{msg}_j}(S_j)|$ in Section 3.3.1. Then, in Section 3.3.2, we use this bound, along with Friedgut's inequality, to establish an upper bound for $|K_{\mathrm{msg}}(q)|$ and hence prove Theorem 3.12.

3.3.1 Bounding the Knowledge of Each Relation. Let us fix a server s, and an input relation S_j. Recall that $M_j = a_j m_j \log n$ denotes the number of bits using a naive encoding of S_j. An algorithm A may use fewer bits, $\mathcal{M}_j$, by exploiting the fact that S_j is a uniformly chosen a_j-dimensional matching. There are precisely $\binom{n}{m_j}^{a_j}(m_j!)^{a_j-1}$ different a_j-dimensional matchings of arity a_j and size m_j and thus the number of bits N necessary to represent the relation is given by the entropy:

$$\mathcal{M}_j = H(S_j) = a_j \log \binom{n}{m_j} + (a_j - 1)\log(m_j!). \tag{21}$$

We will prove later that $\mathcal{M}_j$ is $\Omega(M_j)$. The following lemma provides a bound on the expected knowledge $K_{m_j}(S_j)$ the server may obtain from S_j:

Lemma 3.14. *Suppose that the size of S_j is m_j and define*

$$\gamma_j = \begin{cases} 1 & \text{if } m_j = n \\ 2 & \text{if } m_j \le n/2 \end{cases}$$

and that the message $msg_j(S_j)$ has at most $f_j \cdot \mathcal{M}_j$ bits, for some $f_j < 1$. Then $\mathbb{E}[|K_{msg_j}(S_j)|] \le \gamma_j \cdot f_j \cdot m_j$, where the expectation is taken over random choices of the matching S_j.

It says that, if the message msg_j has only a fraction f_j of the bits needed to encode S_j, then a server receiving this message knows, in expectation, only a fraction $\gamma_j f_j$ of the m_j tuples in S_j. Notice that the bound holds only in expectation: A specialized encoding may choose to use very few bits to represent a particular matching $S_j \subseteq [n]^{a_j}$. When a server receives that message, then it knows all tuples in S_j; however, then there will be fewer bit combinations left to encode the other matchings S_j.

Proof. The entropy $H(S_j)$ in Equation (21) has two parts, corresponding to the two parts needed to encode S_j: For each attribute of S_j we need to encode a subset $\subseteq [n]$ of size m_j, and for each

attribute except one we need to encode a permutation over $[m_j]$. Fix a value msg_j of the message received by the server from the input S_j, and let $k = |K_{\text{msg}_j}(S_j)|$. Because S_j is uniformly distributed, the conditional entropy $H(S_j|\text{msg}_j(S_j) = \text{msg}_j)$ is as follows:

$$\log|\{S_j \mid \text{msg}_j(S_j) = \text{msg}_j\}| \le a_j \log\binom{n-k}{m_j-k} + (a_j - 1)\log((m_j - k))!, \tag{22}$$

since msg_j fixes k tuples of S_j. We will next show that

$$\log|\{S_j \mid \text{msg}_j(S_j) = \text{msg}_j\}| \le \left(1 - \frac{k}{\gamma_j m_j}\right).\mathcal{M}_j. \tag{23}$$

In other words, we claim that the entropy has decreased by at least a $1 - k/(\gamma_j m_j)$ factor. We show this by proving that each of the two parts of the entropy in Equation (21) is decreased by at least that factor. We tackle the second term first.

Since $\log(x)$ is an increasing function, it follows that $\sum_{i=1}^{m_j-k}(\log(i)/(m_j - k)) \le \sum_{i=1}^{m_j}(\log(i)/m_j)$, which is equivalent to

$$\frac{\log((m_j - k)!)}{\log(m_j!)} \le \frac{m_j - k}{m_j},$$

and hence $\log((m_j - k)!) \le (1 - \frac{k}{m_j})\log(m_j!)$.

In the case that $m_j = n$, the first term of both Equations (21) and (22) is 0, and $\gamma_j = 1$, so this suffices to prove Equation (23). It remains to prove Equation (23) when $m_j \le n/2$ for which $\gamma_j = 2$, and for that we show

$$\log\binom{n-k}{m_j-k} \le \left(1 - \frac{k}{2m_j}\right)\log\binom{n}{m_j}. \tag{24}$$

Since

$$\frac{\binom{n-k}{m_j-k}}{\binom{n}{m_j}} = \frac{m_j \cdot (m_j - 1)\cdots(m_j - k + 1)}{n \cdot (n-1)\cdots(n-k+1)} \le \left(\frac{m_j}{n}\right)^k,$$

we have

$$\log\binom{n-k}{m_j-k} \le \log\binom{n}{m_j} - k\log(n/m_j) = \left(1 - \frac{k\log(n/m_j)}{\log\binom{n}{m_j}}\right)\log\binom{n}{m_j}.$$

To show Equation (24), it there suffices to show that when $k \le m_j \le n/2, \log(\binom{n}{m_j}) \le 2m_j\log(n/m_j)$. For this, we use the bound $(\binom{n}{m_j}) \le nH_2(m_j/n)$, where $H_2(x) = -x\log(x) - (1-x)\log(1-x)$ is the *binary entropy function*. Then $H_2(x) = f(x) + f(1-x)$, where $f(x) = -x\log(x)$. Since f is a decreasing function on $\mathbb{R}^+$, $f(x) \ge f(1-x)$ for $x \in (0, 1/2]$ and hence since $m_j \le n/2$,

$$\log\binom{n}{m_j} \le nH_2(m_j/n) \le 2nf(m_j/n) = 2m_j\log(n/m_j),$$

which implies Equation (24) and hence Equation (23).

Finally, we use Equation (23) to prove the lemma. For that we apply the chain rule for entropy, $H(S_j, \text{msg}_j(S_j)) = H(\text{msg}_j(S_j)) + H(S_j|\text{msg}_j(S_j))$, and then use the fact that

$H(S_j, \text{msg}_j(S_j)) = H(S_j)$ (since S_j completely determines $\text{msg}_j(S_j)$) and apply the definition of $H(S_j|\text{msg}_j(S_j))$ together with Equation (23):

$$\begin{aligned} H(S_j) &= H(\text{msg}_j(S_j)) + \sum_{\text{msg}_j} P(\text{msg}_j(S_j) = \text{msg}_j) \cdot H(S_j|\text{msg}_j(S_j) = \text{msg}_j) \\ &\le f_j \cdot H(S_j) + \sum_{\text{msg}_j} P(\text{msg}_j(S_j) = \text{msg}_j) \cdot H(S_j|\text{msg}_j(S_j) = \text{msg}_j) \\ &\le f_j \cdot H(S_j) + \sum_{\text{msg}_j} P(\text{msg}_j(S_j) = \text{msg}_j) \cdot \left(1 - \frac{|K_{\text{msg}_j}(S_j)|}{\gamma_j m_j}\right) H(S_j) \\ &= f_j \cdot H(S_j) + \left(1 - \sum_{\text{msg}_j} P(\text{msg}_j(S_j) = \text{msg}_j) \frac{|K_{\text{msg}_j}(S_j)|}{\gamma_j m_j}\right) H(S_j) \\ &= f_j \cdot H(S_j) + \left(1 - \frac{\mathbb{E}[|K_{\text{msg}_j(S_j)}(S_j)|]}{\gamma_j m_j}\right) H(S_j), \end{aligned} \tag{25}$$

where the first inequality follows from the assumed upper bound on $|\text{msg}_j(S_j)|$, the second inequality follows from Equation (23), and the last two lines follow by definition. Dividing both sides of Equation (25) by $H(S_j)$ since $H(S_j)$ is not zero and rearranging, we obtain the required statement. □

3.3.2 Bounding the Knowledge of the Query. We now use Lemma 3.14 to derive an upper bound on the number of answers to $q(I)$ that a server s can report, which will complete the proof of Theorem 3.12. Recall that Lemma 3.14 assumed that the message $\text{msg}_j(S_j)$ is at most a fraction f_j of the entropy of S_j. We do not know the values of f_j, instead we know that the entire $\text{msg}(I)$ received by the server s (the concatenation of all ℓ messages $\text{msg}_j(S_j)$) has at most L bits. For each relation S_j, define

$$f_j = \frac{\max_{S_j \subseteq [n]^{a_j}} |\text{msg}_j(S_j)|}{\mathcal{M}_j}.$$

Thus, f_j is the largest fraction of bits of S_j that the server receives, over all choices of the matching S_j. We immediately derive an upper bound on the f_j's. We have $\sum_{j=1}^{\ell} \max_{S_j} |\text{msg}_j(S_j)| \le L$, because each relation S_j can be chosen independently, which implies $\sum_{j=1}^{\ell} f_j \mathcal{M}_j \le L$.

For $\mathbf{a}_j \in [n]^{a_j}$, let $w_j(\mathbf{a}_j)$ denote the probability that the server knows the tuple $\mathbf{a}_j$. In other words $w_j(\mathbf{a}_j) = P(\mathbf{a}_j \in K_{\text{msg}_j(S_j)}(S_j))$, where the probability is over the random choices of S_j.

Lemma 3.15. *For any relation S_j:*

(a) $\forall \mathbf{a}_j \in [n]^{a_j} : w_j(\mathbf{a}_j) \le m_j/n^{a_j}$, *and*
(b) $\sum_{\mathbf{a}_j \in [n]^{a_j}} w_j(\mathbf{a}_j) \le \gamma_j \cdot f_j \cdot m_j$ *where γ_j is defined as in Lemma 3.14.*

Proof. To show (a), notice that $w_j(\mathbf{a}_j) \le P(\mathbf{a}_j \in S_j) = m_j/n^{a_j}$, while (b) follows from the fact $\sum_{\mathbf{a}_j \in [n]^{a_j}} w_j(\mathbf{a}_j) = \mathbf{E}[|K_{\text{msg}_j(S_j)}(S_j)|] \le \gamma_j \cdot f_j \cdot m_j$ by Lemma 3.14. □

Since the server receives a separate message for each relation S_j, from a distinct input server, the events $\mathbf{a}_1 \in K_{\text{msg}_1}(S_1), \ldots, \mathbf{a}_\ell \in K_{\text{msg}_\ell}(S_\ell)$ are independent, hence:

$$\mathbf{E}[|K_{\text{msg}(I)}(q)|] = \sum_{\mathbf{a} \in [n]^k} P(\mathbf{a} \in K_{\text{msg}(I)}(q)) = \sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j).$$

We now use Friedgut's inequality. Recall that to apply the inequality, we need to find a fractional edge cover. Let us pick any fractional edge packing $\mathbf{u} = (u_1, \ldots, u_\ell)$. Given q, defined as in Equation (1), consider the *extended query*, which has a new unary atom for each variable x_i:

$$q'(x_1, \ldots, x_k) = S_1(\bar{x}_1), \ldots, S_\ell(\bar{x}_\ell), T_1(x_1), \ldots, T_k(x_k).$$

For each new symbol T_i, define $u'_i = 1 - \sum_{j: x_i \in \text{vars}(S_j)} u_j$. Since $\mathbf{u}$ is a packing, $u'_i \geq 0$. Let us define $\mathbf{u}' = (u'_1, \ldots, u'_k)$.

Lemma 3.16. *(a) The assignment* $(\mathbf{u}, \mathbf{u}')$ *is both a tight fractional edge packing and a tight fractional edge cover for* q'*. (b)* $\sum_{j=1}^{\ell} a_j u_j + \sum_{i=1}^{k} u'_i = k$.

Proof. (a) is straightforward, since for every variable x_i we have $u'_i + \sum_{j: x_i \in \text{vars}(S_j)} u_j = 1$. Summing up:

$$k = \sum_{i=1}^{k} \left(u'_i + \sum_{j: x_i \in \text{vars}(S_j)} u_j \right) = \sum_{i=1}^{k} u'_i + \sum_{j=1}^{\ell} a_j u_j,$$

which proves (b). □

We will apply Friedgut's inequality to the extended query q'. Set the variables $w(-)$ used in Friedgut's inequality as follows:

$$\begin{aligned} w_j(\mathbf{a}_j) &= \mathbb{P}(\mathbf{a}_j \in K_{\text{msg}_j(S_j)}(S_j)) \text{ for } S_j, \text{ tuple } \mathbf{a}_j \in [n]^{a_j} \\ w'_i(\alpha) &= 1 \qquad\qquad\qquad\qquad \text{for } T_i, \text{ value } \alpha \in [n]. \end{aligned}$$

Recall that, for a tuple $\mathbf{a} \in [n]^k$ we use $\mathbf{a}_j \in [n]^{a_j}$ for its projection on the variables in S_j; with some abuse, we write $\mathbf{a}_i \in [n]$ for the projection on the variable x_i. Assume first that $u_j > 0$, for $j = 1, \ldots, \ell$. Then:

$$\begin{aligned} \mathbf{E}[|K_{\text{msg}}(q)|] &= \sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \\ &= \sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \prod_{i=1}^{k} w'_i(\mathbf{a}_i) \\ &\leq \prod_{j=1}^{\ell} \left(\sum_{\mathbf{a} \in [n]^{a_j}} w_j(\mathbf{a})^{1/u_j} \right)^{u_j} \prod_{i=1}^{k} \left(\sum_{\alpha \in [n]} w'_i(\alpha)^{1/u'_i} \right)^{u'_i} \\ &= \prod_{j=1}^{\ell} \left(\sum_{\mathbf{a} \in [n]^{a_j}} w_j(\mathbf{a})^{1/u_j} \right)^{u_j} \prod_{i=1}^{k} n^{u'_i}. \end{aligned}$$

Note that, since $w'_i(\alpha) = 1$ we have $w'_i(\alpha)^{1/u'_i} = 1$ even if $u'_i = 0$. Write $w_j(\mathbf{a})^{1/u_j} = w_j(\mathbf{a})^{1/u_j - 1} w_j(\mathbf{a})$, and use Lemma 3.15 to obtain

$$\begin{aligned} \sum_{\mathbf{a} \in [n]^{a_j}} w_j(\mathbf{a})^{1/u_j} &\leq (m_j / n^{a_j})^{1/u_j - 1} \sum_{\mathbf{a} \in [n]^{a_j}} w_j(\mathbf{a}) \\ &\leq (m_j n^{-a_j})^{1/u_j - 1} \gamma_j f_j \cdot m_j \\ &= \gamma_j f_j \cdot m_j^{1/u_j} \cdot n^{(a_j - a_j/u_j)}. \end{aligned}$$

Plugging this in the bound, we have shown that

$$\begin{aligned}
\mathbb{E}[|K_{\text{msg}}(q)|] &\le \prod_{j=1}^{\ell} \left(\gamma_j f_j \cdot m_j^{1/u_j} \cdot n^{(a_j - a_j/u_j)}\right)^{u_j} \cdot \prod_{i=1}^{k} n^{u'_i} \\
&= \prod_{j=1}^{\ell} (\gamma_j f_j)^{u_j} \cdot \prod_{j=1}^{\ell} m_j \cdot n^{(\sum_{j=1}^{\ell} a_j u_j - a)} \cdot n^{\sum_{i=1}^{k} u'_i} \\
&= \prod_{j=1}^{\ell} (\gamma_j f_j)^{u_j} \cdot \prod_{j=1}^{\ell} m_j \cdot n^{-a + (\sum_{j=1}^{\ell} a_j u_j + \sum_{i=1}^{k} u'_i)} \\
&= \prod_{j=1}^{\ell} (\gamma_j f_j)^{u_j} \cdot \prod_{j=1}^{\ell} m_j \cdot n^{k-a} \\
&= \prod_{j=1}^{\ell} (\gamma_j f_j)^{u_j} \cdot \mathbb{E}[|q(I)|]. \qquad (26)
\end{aligned}$$

Here, the third equality follows from Lemma 3.16(b), and the last equality follows from Lemma 3.13. If some $u_j = 0$, then we can derive the same lower bound as follows: We can replace each u_j with $u_j + \delta$ for any $\delta > 0$ still yielding an edge cover. Then we have $\sum_j a_j u_j + \sum_i u'_i = k + a\delta$, and hence an extra factor $n^{a\delta}$ multiplying the term $n^{\ell+k-a}$ in Equation (26); however, we obtain the same upper bound since, in the limit as δ approaches 0, this extra factor approaches 1.

Given Equation (26), the final step is to upper bound the quantity $\prod_{j=1}^{\ell} (\gamma_j f_j)^{u_j}$; using the fact that $\sum_{j=1}^{\ell} f_j \mathcal{M}_j \le L$. Recall that $u = \sum_j u_j$, then:

$$\begin{aligned}
\prod_{j=1}^{\ell} (\gamma_j f_j)^{u_j} &= \prod_{j=1}^{\ell} \left(\frac{f_j \mathcal{M}_j}{u_j}\right)^{u_j} \prod_{j=1}^{\ell} \left(\frac{\gamma_j u_j}{\mathcal{M}_j}\right)^{u_j} \\
&\le \left(\frac{\sum_{j=1}^{\ell} f_j \mathcal{M}_j}{\sum_j u_j}\right)^{\sum_j u_j} \prod_{j=1}^{\ell} \left(\frac{\gamma_j u_j}{\mathcal{M}_j}\right)^{u_j} \\
&\le \left(\frac{L}{\sum_j u_j}\right)^{\sum_j u_j} \prod_{j=1}^{\ell} \left(\frac{\gamma_j u_j}{\mathcal{M}_j}\right)^{u_j} \\
&= \prod_{j=1}^{\ell} \left(\frac{\gamma_j L}{u \cdot \mathcal{M}_j}\right)^{u_j} \prod_{j=1}^{\ell} (u_j)^{u_j} \\
&\le \prod_{j=1}^{\ell} \left(\frac{\gamma_j L}{u \cdot \mathcal{M}_j}\right)^{u_j}.
\end{aligned}$$

Here, the first inequality comes from the weighted version of the Arithmetic Mean-Geometric Mean inequality. The last inequality holds since $u_j \le 1$ for any j.

To prove Theorem 3.12 it suffices to prove that $\gamma_j / \mathcal{M}_j \le 4/M_j$ in each of the two cases. To do so, we need a lower bound on the number of bits $\mathcal{M}_j$ needed to represent relation S_j.

Proposition 3.17. *The number of bits $\mathcal{M}_j$ needed to represent S_j is given by the following:*

(a) If $n \ge m_j^2$, then $\mathcal{M}_j \ge M_j/2$.
(b) If $n = m_j$ and $a_j \ge 2$, then $\mathcal{M}_j \ge M_j/4$.

Proof. For the first item, we have:

$$\mathcal{M}_j \geq a_j \log \binom{n}{m_j} \geq a_j m_j \log(n/m_j) \geq (1/2) a_j m_j \log(n) = M_j/2.$$

For the second item, we have:

$$\mathcal{M}_j \geq (a_j - 1)\log(m_j!) \geq \frac{a_j - 1}{2} m_j \log(m_j) \geq \frac{(a_j - 1)}{2a_j} M_j \geq M_j/4,$$

where the last inequality comes from the assumption that $a_j \geq 2$. □

Since $\gamma_j = 2$ in the first case and $\gamma_j = 1$ in the second case, we obtain that $\gamma_j/\mathcal{M}_j \leq 4/M_j$ and hence complete the proof of Theorem 3.12. (Recall that L in this subsection denotes the load of an arbitrary server, which was denoted L_i in the statement of the theorem.)

3.4 Lower Bound for Randomized Algorithms

In this section, we prove the lower bound on the load of randomized algorithms given in Theorem 3.9(b). The general idea of the argument is to apply Yao's lemma (Yao 1977), which implies that for any probability space $\mathcal{I}$ of database instances I, if every deterministic algorithm (with given resource constraints) fails to compute $q(I)$ with probability $\geq 1 - \delta$, taken over the random choices of $\mathcal{I}$, then there exists an instance I in the support of $\mathcal{I}$ on which every randomized algorithm A (with the same resource constraints) fails with probability $\geq 1 - \delta$, taken over the random choices of A.

While Theorem 3.12 does prove a bound for deterministic algorithms given a distribution on instances, it does not directly yield a lower bound of the form we need to apply Yao's lemma since it is phrased in terms of the expected size of the output rather than the probability of success. Indeed, the space $\mathcal{I}$ of uniformly chosen random matchings is not useful for this purpose. For example, for a connected query with a large characteristic $\chi(q)$, $\mathbb{E}[|q(I)|]$ is $O(1/n)$ and therefore $P(q(I) \neq \emptyset)$ is $O(1/n)$, which means that a naive deterministic algorithm that always returns the empty answer will fail only with a very small probably, $O(1/n)$.

Instead, we will define an event $\mathcal{E}_\alpha$ for some constant α with $0 \leq \alpha < 1$ which, in the case that q is connected, is the event that $|q(I)| > \alpha\mu$ for $\mu = \mathbb{E}[|q(I)|]$. We will then apply Yao's lemma using the probability distribution $\mathcal{I}|\mathcal{E}_\alpha$ of random matchings conditioned on $\mathcal{E}_\alpha$.

First, using the second moment method, we prove for connected queries q that the event $\mathcal{E}_\alpha$ occurs with significant probability.

Lemma 3.18. *Let I be a random matching database for a connected conjunctive query q, and let $\mu = \mathbb{E}[|q(I)|]$. Then, for any $\alpha \in [0, 1)$ we have:*

$$P(|q(I)| > \alpha\mu) \geq (1 - \alpha)^2 \frac{\mu}{\mu + 1}.$$

Proof. To prove the bound, we will use a version of the second moment method, the Paley-Zygmund inequality, for the random variable $|q(I)|$:

$$P(|q(I)| > \alpha\mu) \geq (1 - \alpha)^2 \frac{\mu^2}{\mathbb{E}[|q(I)|^2]}.$$

To bound $\mathbb{E}[|q(I)|^2]$, we construct a new query q' that consists of q plus a copy of q with a disjoint set of new variables; q' has two connected components. For example, if $q = R(x, y), S(y, z)$, we define $q' = R(x, y), S(y, z), R(x', y'), S(y', z')$. Since the two copies of q in q' have disjoint variables,

they are independent and we have:

$$\mathbb{E}[|q(I)|^2] = \mathbb{E}[|q'(I)|] = \sum_{\mathbf{a},\mathbf{a}' \in [n]^k} \prod_{j=1}^{\ell} \mathbb{P}(\mathbf{a}_j \in S_j \wedge \mathbf{a}'_j \in S_j).$$

Consider two tuples $\mathbf{a}, \mathbf{a}' \in [n]^k$. If $\mathbf{a} = \mathbf{a}'$, then $\prod_{j=1}^{\ell} \mathbb{P}(\mathbf{a}_j \in S_j \wedge \mathbf{a}'_j \in S_j) = \prod_{j=1}^{\ell} \mathbb{P}(\mathbf{a}_j \in S_j)$. If $\mathbf{a} \neq \mathbf{a}'$, but $\mathbf{a}, \mathbf{a}'$ agree on at least one attribute position, then $\prod_{j=1}^{\ell} \mathbb{P}(\mathbf{a}_j \in S_j \wedge \mathbf{a}'_j \in S_j) = 0$, because every S_j is a matching and the query is connected. If $\mathbf{a} \neq \mathbf{a}'$ and they differ on all attribute positions, then for each j, $\mathbb{P}(\mathbf{a}_j \in S_j \wedge \mathbf{a}'_j \in S_j) = \mathbb{P}(\mathbf{a}_j \in S_j)\mathbb{P}(\mathbf{a}'_j \in S_j)$, because the random matching S_j selects the tuples $\mathbf{a}_j$ and $\mathbf{a}'_j$ independently. This implies:

$$\begin{aligned}
\mathbb{E}[|q(I)|^2] &\leq \sum_{\mathbf{a}\neq\mathbf{a}' \in [n]^k} \prod_{j=1}^{\ell} P(\mathbf{a}_j \in S_j) P(\mathbf{a}'_j \in S_j) + \sum_{\mathbf{a} \in [n]^k} \prod_{j=1}^{\ell} P(\mathbf{a}_j \in S_j) \\
&= (n^{2k} - n^k) \prod_j (m_j/n^{a_j})^2 + n^k \prod_j m_j/n^{a_j} \\
&= (1 - n^{-k})\mu^2 + \mu \leq \mu^2 + \mu,
\end{aligned}$$

where the last equality follows from Lemma 3.13. Plugging in this bound yields the statement of the lemma. □

We now generalize the definition of the event $\mathcal{E}_\alpha$ to arbitrary queries. If the connected components of query q are $q_1, \ldots, q_c$, then we define $\mathcal{E}_\alpha$ to be the event that for each $i \in [c]$. $q_i(I) > \alpha\mu_i$ where $\mu_i = \mathbb{E}[|q_i(I)|]$.

Let $\mathbf{u}^*$ be an edge packing that maximizes $\mathcal{L}(\mathbf{u}, \mathbf{M}, p)$, and write

$$\theta = \left(\frac{4L}{(\sum_j u_j^*) \cdot L^{lower}} \right)^{\sum_j u_j^*}.$$

Theorem 3.12 implies that, for any one-round deterministic algorithm with load $\leq L$, $\mathbb{E}[|A(I)|] \leq \theta \cdot \mathbb{E}[|q(I)|]$.

For a deterministic algorithm A that computes the answers to a query q over a randomized instance I, the event that $|q(I) \setminus A(I)| > 0$ is the event that the algorithm A fails to return all the required output tuples. The next lemma shows how we can use our upper bound on the expected number of tuples returned to obtain a lower bound on the probability of failure. (The choice of $\alpha = 1/3$ is a simple value that nearly optimizes the bound.)

Lemma 3.19. *Let I be a random matching database for a query q with connected components $q_1, \ldots, q_c$. Let A be a deterministic algorithm such that $\mathbb{E}[|A(I)|] \leq \theta \cdot \mathbb{E}[|q(I)|]$, for some $\theta \leq 1$. Let $\mu_i = \mathbb{E}[|q_i(I)|]$ and let $\mathcal{E}_\alpha$ denote the event that $|q_i(I)| > \alpha\mu_i$ for all $i \in [c]$. Then,*

$$P(|q(I) \setminus A(I)| > 0 \mid \mathcal{E}_{1/3}) \geq 1 - 9^c\theta.$$

Proof. Define $T_\alpha = \prod_{i=1}^{c} (\lfloor \alpha\mu_i \rfloor + 1)$. Note that, since $|q(I)| = |q_1(I)| \cdots |q_c(I)|$, if $\mathcal{E}_\alpha$ is true for I, then $|q(I)| \geq T_\alpha$. Moreover, since the distribution $\mathcal{I}$ is independent between components, we have $\mu = \prod_{i=1}^{c} \mu_i$. Now

$$\begin{aligned}
P(|q(I) \setminus A(I)| > 0 \mid \mathcal{E}_\alpha) &= P(|A(I)| < |q(I)| \mid \mathcal{E}_\alpha) \\
&\geq P(|A(I)| < T_\alpha \mid \mathcal{E}_\alpha) \qquad \text{by definitions of } T_\alpha \text{ and } \mathcal{E}_\alpha \\
&= 1 - P(|A(I)| \geq T_\alpha \mid \mathcal{E}_\alpha).
\end{aligned}$$

Additionally, we have

$$\begin{aligned}\mathbb{E}[|A(I)|] &\geq T_\alpha \cdot P(|A(I)| \geq T_\alpha) \\ &\geq T_\alpha \cdot P(|A(I)| \geq T_\alpha \wedge \mathcal{E}_\alpha) \\ &= T_\alpha \cdot P(\mathcal{E}_\alpha) P(|A(I)| \geq T_\alpha \mid \mathcal{E}_\alpha).\end{aligned}$$

Combining the above two inequalities and the assumption that $\mathbb{E}[|A(I)|] \leq \theta\mu$, we can now write

$$P(|q(I) \setminus A(I)| > 0 \mid \mathcal{E}_\alpha) \geq 1 - \frac{\mathbb{E}[|A(I)|]}{T_\alpha \cdot P(\mathcal{E}_\alpha)} \geq 1 - \frac{\theta\mu}{T_\alpha \cdot P(\mathcal{E}_\alpha)}.$$

Since the properties of $q_i(I)$ for different i are independent, we can write

$$P(\mathcal{E}_\alpha) = \prod_{i=1}^{c} P(|q_i(I)| > \alpha\mu_i) \geq \prod_{i=1}^{c} (1-\alpha)^2 \mu_i/(\mu_i + 1)$$

by applying Lemma 3.18 separately to each connected component q_i. Therefore

$$\begin{aligned}P(|q(I) \setminus A(I)| > 0 \mid \mathcal{E}_\alpha) &\geq 1 - \frac{\theta\mu}{T_\alpha} \cdot \prod_{i=1}^{c} \frac{\mu_i + 1}{\mu_i (1-\alpha)^2} = 1 - \frac{\theta}{(1-\alpha)^{2c}} \cdot \frac{\prod_{i=1}^{c}(\mu_i+1)}{T_\alpha} \\ &= 1 - \frac{\theta}{(1-\alpha)^{2c}} \cdot \prod_{i=1}^{c} \frac{\mu_i + 1}{\lfloor \alpha\mu_i \rfloor + 1}\end{aligned}$$

using the definition of T_α. We now distinguish two cases for each term in the product, depending on the value of μ_i:

- If $\mu_i \geq 1/\alpha$, then since $\alpha\mu_i \leq \lfloor \alpha\mu_i \rfloor + 1$ we also have $\frac{\mu_i+1}{\lfloor \alpha\mu_i \rfloor+1} \leq (\mu_i+1)/(\alpha\mu_i) \leq (1 + 1/\mu_i)/\alpha \leq (1+\alpha)/\alpha$.
- If $\mu_i < 1/\alpha$, then $\lfloor \alpha\mu_i \rfloor = 0$. Thus, $\frac{\mu_i+1}{\lfloor \alpha\mu_i \rfloor+1} = \mu_i + 1 < 1/\alpha + 1 = (1+\alpha)/\alpha$.

Choosing $\alpha = 1/3$ (for which $\frac{(1+\alpha)}{\alpha \cdot (1-\alpha)^2} = 9$ and is nearly minimal) we obtain that

$$P(|q(I) \setminus A(I)| > 0 \mid \mathcal{E}_{1/3}) \geq 1 - 9^c\theta$$

as required. □

We are now ready to apply Yao's Lemma to the distribution $\mathcal{I} \mid \mathcal{E}_{1/3}$ to obtain the lower bound for randomized algorithms.

Theorem 3.20. *Let q be a conjunctive query having c connected components. Fix any vector of cardinalities $\mathbf{m}$ for q and let A be any one round, randomized MPC algorithm A for q. Then for any domain size $n \geq (\max_j m_j)^2$, if A has maximum load $L \leq \delta \cdot L^{lower}$ for some constant $\delta < 1/36$, where $\mathbf{M}$ is the set of derived statistics for q, then there exists an instance I with cardinalities $\mathbf{m}$ and derived statistics $\mathbf{M}$ such that the randomized algorithm A fails to compute $q(I)$ correctly with probability $\geq 1 - (36\delta/c)^c$.*

Proof. Let q be a conjunctive query with c connected components. By Theorem 3.12, for the edge packing $\mathbf{u}^*$ that achieves the maximum value L^{lower} of $L(\mathbf{u}^*, \mathbf{M}, p)$, we have that for any

deterministic algorithm A, $\mathbb{E}[|A(I)|] \leq \theta \cdot \mathbb{E}[|q(I)|]$ for

$$\begin{aligned}\theta &= \left(\frac{4L}{(\sum_j u_j^*) \cdot L^{lower}}\right)^{\sum_j u_j^*} \\ &\leq \left(\frac{4\delta}{\sum_j u_j^*}\right)^{\sum_j u_j^*} \qquad \text{by our assumption on } L, \\ &\leq (4\delta/c)^c,\end{aligned}$$

because $4\delta \leq 1$ and because an optimal $\mathbf{u}^*$ has $\sum_j u_j^* \geq c$ since q has c connected components. Hence by Lemma 3.19, the probability of failure under distribution $\mathcal{I}|\mathcal{E}_{1/3}$,

$$P(|q(I) \setminus A(I)| > 0 \mid \mathcal{E}_{1/3}) \geq 1 - 9^c\theta \geq 1 - 9^c(4\delta/c)^c = 1 - (36\delta/c)^c.$$

Applying Yao's Lemma using the distribution $\mathcal{I}|\mathcal{E}_{1/3}$ we obtain the claimed lower bound for randomized algorithms. □

We now can derive part (b) of Theorem 3.9.

Proof of Theorem 3.9(b). We apply Theorem 3.20 with $\delta = 1/72$ and see that any randomized algorithm with load at most $\delta \cdot L^{lower}$ has a probability of failure at least $1 - 1/2^c$. □

3.5 Discussion

We present here examples and applications of the theorems proved in this section. Table 2 presents the computation of the optimal share exponents and τ^* for several classes of queries.

The Speedup of the HyperCube. Denote $\mathbf{u}^*$ the fractional edge packing that maximizes $\mathcal{L}(\mathbf{u}, \mathbf{M}, p)$ (18). When the number of servers increases, the load decreases at a rate of $1/p^{1/\sum_j u_j^*}$, which we call the *speedup* of the HyperCube algorithm. We call the quantity $1/\sum_j u_j^*$ the *speedup exponent*. We have seen that, when all cardinalities are equal, then the speedup exponent is $1/\tau^*$, but when the cardinalities are unequal then the speedup exponent may be better.

Example 3.21. Consider the triangle query

$$C_3 = S_1(x_1, x_2), S_2(x_2, x_3), S_3(x_3, x_1)$$

and assume that the relation sizes are M_1, M_2, M_3. Then, $pk(C_3)$ has five vertices, and each gives a different value for $\mathcal{L}(\mathbf{u}, \mathbf{M}, p) = (M_1^{u_1} M_2^{u_2} M_3^{u_3}/p)^{1/(u_1+u_2+u_3)}$:

$\mathbf{u}$	$\mathcal{L}(\mathbf{u}, \mathbf{M}, p)$
$(1/2, 1/2, 1/2)$	$(M_1M_2M_3)^{1/3}/p^{2/3}$
$(1, 0, 0)$	M_1/p
$(0, 1, 0)$	M_2/p
$(0, 0, 1)$	M_3/p
$(0, 0, 0)$	0

(The last row is justified by the fact that $\mathcal{L}(\mathbf{u}, \mathbf{M}, p) \leq \max(M_1, M_2, M_3)/p^{1/(u_1+u_2+u_3)} \to 0$ when $u_1 + u_2 + u_3 \to 0$.) The load of the HC algorithm is given by the largest of these quantities, in other words, the optimal solution to the LP (17) that gives the load of the HC algorithm can be given in closed form, as the maximum over these five expressions. To compute the speedup, suppose $M_1 < M_2 = M_3 = M$. Then there are two cases. When $p \leq M/M_1$, the optimal packing is $(0, 1, 0)$ (or $(0, 0, 1)$) and the load is M/p. HyperCube achieves linear speedup by computing a standard join of $S_2 \bowtie S_3$ and broadcasting the smaller relation S_1; it does this by allocating shares $p_1 = p_2 = 1$,

Table 2. Query Examples: C_k = Cycle Query, L_k = Linear Query, T_k = Star Query, and $B_{k,m}$ = Query with $\binom{k}{m}$ Relations, Where Each Relation Contains a Distinct Set of m of the k Head Variables

Conjunctive Query	Share Exponents	Value $\tau^*(q)$	Lower Bound for Space Exponent
$C_k(x_1,\ldots,x_k) = \bigwedge_{j=1}^{k} S_j(x_j, x_{(j \bmod k)+1})$	$\frac{1}{k},\ldots,\frac{1}{k}$	$k/2$	$1-2/k$
$T_k(z,x_1,\ldots,x_k) = \bigwedge_{j=1}^{k} S_j(z,x_j)$	$1,0,\ldots,0$	1	0
$L_k(x_0,x_1,\ldots,x_k) = \bigwedge_{j=1}^{k} S_j(x_{j-1},x_j)$	$0,\frac{1}{\lceil k/2\rceil},0,\frac{1}{\lceil k/2\rceil},\ldots$	$\lceil k/2\rceil$	$1-1/\lceil k/2\rceil$
$B_{k,m}(x_1,\ldots,x_k) = \bigwedge_{I\subseteq[k],\lvert I\rvert=m} S_I(\bar{x}_I)$	$\frac{1}{k},\ldots,\frac{1}{k}$	k/m	$1-m/k$

The share exponents presented are for the case where the relation sizes are equal.

$p_3 = p$. When $p > M/M_1$ then the optimal packing is $(1/2,1/2,1/2)$ the load is $(M_1M_2M_3)^{1/3}/p^{2/3}$, and the speedup decreases to $1/p^{2/3}$.

The following lemma sheds some light into how the HyperCube algorithm exploits unequal cardinalities.

Lemma 3.22. *Let q be a query, over a database with derived statistics $\mathbf{M}$, and let $\mathbf{u}^* = argmax_{\mathbf{u}}\mathcal{L}(\mathbf{u},\mathbf{M},p)$, and $L = \mathcal{L}(\mathbf{u}^*,\mathbf{M},p)$. Then:*

(1) If for some j, $M_j < L$, then $u_j^* = 0$.
(2) Let $M = \max_k M_k$. If for some j, $M_j < M/p$, then $u_j^* = 0$.
(3) When p increases, the speedup exponent remains constant or decreases, eventually reaching $1/\tau^*$.

Proof. We prove the three items of the lemma.
(1) If we modify a fractional edge packing $\mathbf{u}$ by setting $u_j = 0$, we still obtain a fractional edge packing. We claim that the function $f(u_j) = \mathcal{L}(\mathbf{u},\mathbf{M},p)$ is strictly decreasing in u_j on $(0,\infty)$: The claim implies the lemma, because $f(0) > f(u_j)$ for any $u_j > 0$. The claim follows by noticing that $f(u_j) = p^{(u_j \log_p M_j + b)/(u_j + c)}$ where a, b, c are positive constants, hence f is monotone on $u_j \in (0,\infty)$, and $f(u_j) = L > M_j = f(\infty)$, implying that it is monotonically decreasing.
(2) This follows immediately from the previous item by noticing that $M/p \le L$; to see the latter, let k be such that $M_k = M$, and let $\mathbf{u}$ be the packing $u_k = 1, u_j = 0$ for $j \ne k$. Then $M/p = \mathcal{L}(\mathbf{u},\mathbf{M},p) \le \mathcal{L}(\mathbf{u}^*,\mathbf{M},p) = L$.
(3) Consider two edge packings $\mathbf{u},\mathbf{u}'$, denote $u = \sum_j u_j$, $u' = \sum_j u_j'$, and assume $u < u'$. Let $f(p) = \mathcal{L}(\mathbf{u},\mathbf{M},p)$ and $g(p) = \mathcal{L}(\mathbf{u}',\mathbf{M},p)$. We have $f(p) = c/p^{1/u}$ and $g(p) = c'/p^{1/u'}$, where c, c' are constants independent of p. Then $f(p) < g(p)$ if and only if $p > (c/c')^{1/(1/u-1/u')}$, since $1/u - 1/u' > 0$. Thus, as p increases from 1 to ∞, initially we have $f(p) < g(p)$, then $f(p) > g(p)$, and the crossover point is $(c/c')^{1/(1/u-1/u')}$. Therefore, the value $\sum_j u_j^*$ can never decrease, proving the claim. To see that the speedup exponent reaches $1/\tau^*$, denote $\mathbf{u}^*$ the optimal vertex packing (maximizing $\sum_j u_j$) and let $\mathbf{u}$ be any edge packing s.t. $u = \sum_j u_j < \tau^*$. Then, when $p^{1/u-1/\tau^*} > (\prod_j M_j^{u_j^*})^{1/\tau^*}/(\prod_j M_j^{u_j})^{1/u}$, we have $\mathcal{L}(\mathbf{u}^*,\mathbf{M},p) > \mathcal{L}(\mathbf{u},\mathbf{M},p)$. □

The first two items in the lemma say that, if M is the size of the largest relation, then the only relations S_j that matter to the HC algorithm are those for which $M_j \ge M/p$; any smaller relation will be broadcast by the HC algorithm. The last item says that the HC algorithm can take advantage of unequal cardinalities and achieve speedup better than $1/p^{1/\tau^*}$, for example, by allocating fewer

shares to the smaller relations, or even broadcasting them. As p increases, the speedup decreases until it reaches $1/p^{1/\tau^*}$.

Space Exponent. Let $|I| = \sum_j M_j$ denote the size of the input database. Sometimes it is convenient to study algorithms whose maximum load per server is given as $L = O(|I|/p^{1-\varepsilon})$, where $0 \le \varepsilon < 1$ is a constant parameter ε called the *space exponent* of the algorithm. The lower bound given by Theorem 3.12 can be interpreted as a lower bound on the space exponent. To see this, consider the special case, when all relations have equal size $M_1 = \cdots = M_\ell = M$; then the load can also be written as $L = O(M/p^{1-\varepsilon})$, and, denoting $\mathbf{u}^*$ the optimal fractional edge packing, we have $\sum_j u_j^* = \tau^*$ and $\mathcal{L}(\mathbf{u}^*, \mathbf{M}, p) = M/p^{1/\tau^*}$. Table 2 shows the space exponents for several classes of queries. Theorem 3.12 implies that any algorithm with a fixed space exponent ε will report at most as many answers:

$$O\left(\left(\frac{L}{\mathcal{L}(\mathbf{u}^*, \mathbf{M}, p)}\right)^{\tau^*}\right) \cdot \mathbb{E}[|q(I)|] = O(p^{\tau^*[\varepsilon-(1-1/\tau^*)]}) \cdot \mathbb{E}[|q(I)|]. \tag{27}$$

Therefore, if the algorithm has a space exponent $\varepsilon < 1 - 1/\tau^*$, then, as p increases, it will return a smaller fraction of the expected number of answers. This supports the intuition that achieving parallelism becomes harder when p increases: an algorithm with a small space exponent may be able to compute the query correctly when p is small, but will eventually fail, when p becomes large enough.

Replication Rate. Given an algorithm that computes a conjunctive query q, let L_s be the load of server s, where $s = 1, \ldots, p$. The *replication rate* r of the algorithm, defined in Afrati et al. (2012), is $r = \sum_{s=1}^{p} L_i/|I|$. In other words, the replication rate computes how many times on average each input bit is communicated. The authors in Afrati et al. (2012) discuss the tradeoff between r and the maximum load in the case where the number of servers is not given, but can be chosen optimally. We show next how we can apply our lower bounds to obtain a lower bound for the tradeoff between the replication rate and the maximum load.

Corollary 3.22. *Let q be a conjunctive query with derived statistics $\mathbf{M}$. Any algorithm that computes q with maximum load L, where $L \le M_j$ for every S_j,[8] must have replication rate*

$$r \ge \frac{cL}{\sum_j M_j} \max_{\mathbf{u}} \prod_{j=1}^{\ell} \left(\frac{M_j}{L}\right)^{u_j},$$

where $\mathbf{u}$ ranges over all fractional edge packings of q and $c = \max_{\mathbf{u}}(\sum_j u_j/4)^{\sum_j u_j}$.

Proof. Let f_s be the fraction of answers returned by server s, in expectation, where I is a randomly chosen matching database with statistics $\mathbf{M}$. Let $\mathbf{u}$ be an edge packing for q and $c(\mathbf{u}) = (\sum_j u_j/4)^{\sum_j u_j}$; by Theorem 3.12, $f_s \le \frac{L_s^{\sum_j u_j}}{c(\mathbf{u}) \prod_j M_j^{u_j}}$. Since we assume all answers are returned,

$$1 \le \sum_{s=1}^{p} f_s = \sum_{s=1}^{p} \frac{L_s^{\sum_j u_j}}{c(\mathbf{u}) \prod_j M_j^{u_j}} \le \frac{L^{\sum_j u_j - 1} \sum_{s=1}^{p} L_s}{c(\mathbf{u}) \prod_j M_j^{u_j}} = \frac{L^{\sum_j u_j - 1} r|I|}{c(\mathbf{u}) \prod_j M_j^{u_j}},$$

where we used the fact that $\sum_j u_j \ge 1$ for the optimal $\mathbf{u}$. The claim follows by noting that $|I| = \sum_j M_j$. □

[8] If $L > M_j$, then we can send the whole relation to any processor without cost.

In the specific case where the relation sizes are all equal to M, the above corollary tells us that the replication rate must be $r = \Omega((M/L)^{\tau^*-1})$. Hence, the ideal case where $r = o(1)$ is achieved only when the maximum vertex cover number τ^* is equal to 1 (which happens if and only if a variable occurs in every atom of the query).

Example 3.24. Consider again the triangle query C_3 and assume that all sizes are equal to M. In this case, the edge packing that maximizes the lower bound is $(1/2, 1/2, 1/2)$, and $\tau^* = 3/2$. Thus, we obtain an $\Omega(\sqrt{M/L})$ bound for the replication rate for the triangle query.

4 HANDLING DATA SKEW IN ONE COMMUNICATION STEP

In this section, we discuss how to compute queries in the MPC model in the presence of skew. We first start by presenting an example where the HC algorithm that uses the optimal shares from Equation (17) fails to work when the data have skewed, even though it is asymptotically optimal when the relations are of low degree.

Example 4.1. Let $q(x, y, z) = S_1(x, z), S_2(y, z)$ be a simple join query, where both relations have cardinality m (and size in bits M). The optimal shares are $p_1 = p_2 = 1$, and $p_3 = p$. This allocation of shares corresponds to a standard parallel hash-join algorithm, where both relations are hashed on the join variable z. When the data have no skew, the maximum load is $O(M/p)$ with high probability.

However, if the relation has skew, the maximum load can be as large as $O(M)$. This occurs in the case where all tuples from S_1 and S_2 have the same value for variable z.

As we can see from the above example, the problem occurs when the input data contain values with high frequency of occurrence, which we call *outliers*, or *heavy hitters*. We will consider two different scenarios when handling data skew. In the first scenario, in Section 4.1, we assume that the algorithm has no information about the data apart from the size of the relations.

In the second scenario, presented in Section 4.2, we assume that the algorithm knows about the outliers in our data. All the results in this section are limited to single-round algorithms.

4.1 The HyperCube Algorithm with Skew

We answer the following question: What are the optimal shares for the HC algorithm such that the maximum load is minimized over all possible distributions of input data? In other words, we limit our treatment to the HyperCube algorithm, but we consider data that can heavily skewed, as in Example 4.1. Notice that the HC algorithm is oblivious of the values that are skewed, so it cannot be modified to handle these cases separately. Our analysis is based on the following lemma about hashing.

Lemma 4.2. *Let $R(A_1, \ldots, A_r)$ be a relation of arity $r > 1$ with m tuples. Let $p_1, \ldots, p_r$ be integers, $p = \prod_{i\in[r]} p_i$ and suppose that $m \geq (1+\delta)^r \min_i p_i$. Suppose that we hash each tuple $(a_1, \ldots, a_r)$ to the bin $(h_1(a_1), \ldots, h_r(a_r))$, where $h_1, \ldots, h_r$ are independent and uniformly random hash functions. Let $\delta > 0$. The probability that there is a bin that receives $> (2+\delta)m/\min_i p_i$ tuples from R is at most $2pr \cdot \exp(-(m/\min_i p_i)^{1/r} h(\delta)/(1+\delta))$.*

Proof. Assume without loss of generality that $p_1 = \min_i p_i$. Define $k = a \cdot p_1$, where $a = (m/p_1)^{1/r}/(1+\delta) = (m/\min_i p_i)^{1/r}/(1+\delta)$. We partition R into $r+1$ relations $R^1, \ldots, R^r, C$ as follows: Begin with $S = R$. While there is an $d \in [r]$ such that S_d, the projection of S on coordinate d, has $|S_d| \geq k$, for each $j \in S_d$ choose one tuple $t \in S$ with $t_d = j$ and move t from S to R^d. Let C be the set of tuples of R remaining in S.

Observe that by definition, the projections $C_1, \ldots, C_r$ of C satisfy $C_d < k$ for all $d \in [r]$. Write $m_d = |R^d|$. Since at most 1 of the $\geq k$ tuples added to R^d in each step had any fixed dth coordinate, by construction, there are at most m_d/k tuples t in R^d such that $t_d = j$.

The event that some bin receives $> (2+\delta)m/\min_i p_i = (2+\delta)m/p_1$ tuples is contained in the union of the following $r+1$ events: The r events that some bin receives $> (1+\delta)m_d/p_1$ tuples from R^d for $d \in [r]$ and the event that some bin receives $> m/p_1$ tuples from C.

Since each $p_d \geq p_1$, we can upper bound the probability for the dth event by an analysis for a hash function h_d that uses p_1 bins rather than p_d bins. So, by Corollary 3.5 with $p = p_1$ and $\beta = p_1/k = 1/a$, for each $d \in [r]$ we have

$$\mathbb{P}(\text{some bin receives} > (1+\delta)m_d/p_1 \text{ tuples from } R^d) \leq p \cdot e^{-a \cdot h(\delta)}.$$

We upper bound the probability of the last event by the probability that for some bin b more than m/p_1 tuples from $C_1 \times \cdots \times C_r$ map to b. The distribution of the number of such tuples mapping to b is given by the product of independent binomial distributions $B(k, 1/p_d)$ for $d \in [r]$. If each binomial $B(k, 1/p_d)$ is $\leq (m/p_1)^{1/r}$, then the number of tuples is at most m/p_1; therefore if $> m/p_1$ tuples map to b, then one of the binomials $B(k, 1/p_d)$ must be $> (m/p_1)^{1/r}$. This is upper bounded by the probability that $B(k, 1/p_1)$ is $> (m/p_1)^{1/r}$.

Since $k/p_1 = a = (m/p_1)^{1/r}/(1+\delta)$, by standard Chernoff bounds (or Corollary 3.5 with $p = p_1$ and $\beta = 1/a$), the probability that $B(k, 1/p_1)$ is $> (m/p_1)^{1/r}$ is at most $e^{-a \cdot h(\delta)}$. Since there are r coordinates and p bins,

$$\mathbb{P}(\text{some bin receives} > (1+\delta)m_d/p_1 \text{ tuples from } C) \leq pr \cdot e^{-a \cdot h(\delta)}.$$

Adding the bounds for all events together and plugging in the value of a yields the claimed bound. □

Since $h(\delta)/(1+\delta) > 1$ is an unbounded function of δ, we immediately obtain the following corollary.

Corollary 4.3. *Let $p_1, \ldots, p_r$ be integers and $p = \prod_{i \in [r]} p_i$. Let $R(A_1, \ldots, A_r)$ be a relation of arity r with $m \geq \ln^r(2pr) \cdot \min_i p_i$ tuples. Suppose that we hash each tuple $(a_1, \ldots, a_r)$ to the bin $(h_1(a_1), \ldots, h_r(a_r))$, where $h_1, \ldots, h_r$ are independent and uniformly random hash functions. Then, the probability that the maximum load exceeds $O(m/(\min_i p_i))$ is exponentially small in $(m/\min_i p_i)^{1/r}$.*

Corollary 4.4. *Let $\mathbf{p} = (p_1, \ldots, p_k)$ be the shares of the HC algorithm. For any relations, with high probability the maximum load per server is*

$$O\left(\max_j \frac{M_j}{\min_{i: i \in S_j} p_i}\right).$$

The above bound is tight: We can always construct an instance for given shares such that the maximum load is at least as above. Indeed, for a relation S_j with $i = \arg\min_{i \in S_j} p_i$, we can construct an instance with a single value for any attribute other than x_i and M_j values for x_i. In this case, the hashing will be across only one dimension with p_i servers, and so the maximum load has to be at least M_j/p_i for the relation S_j.

As in the previous section, if L denotes the maximum load per server, then we must have that $M_j / \min_{i \in S_j} p_i \leq L$. Denoting $\lambda = \log_p L$ and $\mu_j = \log_p M_j$, the load is optimized by the following LP:

$$\begin{aligned} \text{minimize} \quad & \lambda \\ \text{subject to} \quad & \sum_{i \in [k]} -e_i \geq -1 \\ \forall j \in [\ell] : & h_j + \lambda \geq \mu_j \\ \forall j \in [\ell], i \in S_j : & e_i - h_j \geq 0 \\ \forall i \in [k] : & e_i \geq 0, \quad \forall j \in [\ell] : h_j \geq 0 \quad \lambda \geq 0. \end{aligned} \tag{28}$$

Following the same process as in the previous section, we can obtain the dual of the above LP, and after transformations obtain the following non-linear program with the same optimal objective function:

$$\begin{aligned} \text{maximize} \quad & \frac{\sum_{j \in [\ell]} \mu_j u_j - 1}{\sum_{j \in [\ell]} u_j} \\ \text{subject to} \quad \forall i \in [k] : & \sum_{j : i \in S_j} w_{ij} \leq 1 \\ \forall j \in [\ell] : & u_j \leq \sum_{i \in S_j} w_{ij} \\ \forall j \in [\ell] : & u_j \geq 0 \\ \forall i \in [k], j \in [\ell] : & w_{ij} \geq 0. \end{aligned} \tag{29}$$

4.2 Skew with Information

We discuss here the case where there is additional information known about skew in the input database. We will present a general lower bound for arbitrary conjunctive queries and show an algorithm that matches the bound for *star queries*

$$q = S_1(z, x_1), S_2(z, x_2), \ldots, S_\ell(z, x_\ell),$$

which are a generalization of the join query and for the triangle query as well. In Beame et al. (2014), we show how our algorithmic techniques (the BinHC algorithm) can be used to compute arbitrary conjunctive queries; however, there is a substantial gap between the upper and lower bounds in the general case.

We first introduce some necessary notation for the star query. For each relation S_j with $|S_j| = m_j$, and each assignment $h \in [n]$ for a variable z, we define its *frequency* as $m_j(h) = |\sigma_{z=h}(S_j)|$. We will be interested in assignments that have high frequency, which we call *heavy hitters*. To design algorithms that take skew into account, we will assume that every input server knows the assignments with frequency $\geq m_j/p$ for every relation S_j, along with their frequency. Because each relation can contain at most p heavy hitters, the total number over all relations will be $O(p)$. Since we are considering cases where the number of servers is much smaller than the data, an $O(p)$ amount of information can be easily stored in the input server.

To prove the lower bound, we will make a stronger assumption about the information available to the input servers. Given a conjunctive query q, fix a set of variables $\mathbf{x}$ and let $d = |\mathbf{x}|$. Also, let $\mathbf{x}_j = \mathbf{x} \cap \text{vars}(S_j)$ for every relation S_j, and $d_j = |\mathbf{x}_j|$. A *statistics of type* $\mathbf{x}$*, or* $\mathbf{x}$*-statistics* is a vector $m = (m_1, \ldots, m_\ell)$, where m_j is a function $m_j : [n]^{\mathbf{x}_j} \to \mathbb{N}$. We associate with m the function $m : [n]^{\mathbf{x}} \to (\mathbb{N})^\ell$, where $m(\mathbf{h}) = (m_1(\mathbf{h}_1), \ldots, m_\ell(\mathbf{h}_\ell))$, and $\mathbf{h}_j$ denotes the restriction of the tuple $\mathbf{h}$ to

the variables in $\mathbf{x}_j$. We say that an instance of S_j satisfies the statistics if for any tuple $\mathbf{h}_j \in [n]^{\mathbf{x}_j}$, its frequency is precisely $m_j(\mathbf{h}_j)$. When $\mathbf{x} = \emptyset$, then m simply consists of ℓ numbers, each representing the cardinality of a relation; thus, a $\mathbf{x}$-statistics generalizes the cardinality statistics. Recall that we use upper case $\mathbf{M} = (M_1, \ldots, M_\ell)$ to denote the corresponding derived statistics expressed in bits, that is, $M_j(\mathbf{h}) = a_j m_j(\mathbf{h}) \log(n)$.

In the particular case of the star query, we will assume that the input servers know the z-statistics; in other words, for every assignment $h \in [n]$ of variable z, we know that its frequency in relation $S_j(z, x_j)$ is precisely $m_j(h)$. Observe that in this case the cardinality of S_j is $|S_j| = \sum_{h\in[n]} m_j(h)$.

4.2.1 Algorithm for Star Queries. The algorithm uses the same principle popular in virtually all parallel join implementations to date: Identify the heavy hitters and treat them differently when distributing the data. However, the analysis and optimality proof is new, to the best of our knowledge.

Let H denote the set of heavy hitters in all relations. Note that $|H| \le \ell p$. The algorithm will deal with the tuples that have no heavy hitter values (*light tuples*) by running the vanilla HC algorithm, which runs with shares $p_z = p$ and $p_{x_j} = 1$ for every $j = 1, \ldots, \ell$. For this case, the load analysis of Corollary 3.3 will give us a maximum load of $\tilde{O}(\max_j M_j/p)$ with high probability, where $\tilde{O}$ hides a polylogarithmic factor that depends on p. For heavy hitters, we will have to adapt its function as follows.

To compute q, the algorithm must compute for each $h \in H$ the subquery

$$q[h/z] = S_1(h, x_1), \ldots S_k(h, x_k),$$

which is equivalent to computing the Cartesian product $q_z = S'_1(x_1), \ldots, S'_k(x_k)$, where $S'_1(x_1) = S_1(h, x_1)$ and $S'_2(x_2) = S_2(h, x_2)$, and each relation S'_j has cardinality $m_j(h)$ (and size in bits $M_j(h)$). We call q_z the *residual query*. The algorithm will allocate p_h servers to compute $q[h/z]$ for each $h \in H$, such that $\sum_{h\in H} p_h = \Theta(p)$. Since the unary relations have no skew, they will be of low degree and thus the maximum load L_h for each h is given by

$$L_h = O\left(\max_{\mathbf{u}\in pk(q_z)} L(\mathbf{u}, \mathbf{M}(h), p_h)\right).$$

For the star query, we have $pk(q_z) = \{0, 1\}^\ell \setminus (0, 0, \ldots, 0)$. At this point, since p_h is not specified, it is not clear which edge packing in $pk(q_z)$ maximizes the above quantity for each h. To overcome this problem, we further refine the assignment of servers to heavy hitters: We allocate $p_{h,\mathbf{u}}$ servers to each h and each $\mathbf{u} \in pk(q_z)$, such that $p_h = \sum_{\mathbf{u}} p_{h,\mathbf{u}}$. Now, for a given $\mathbf{u} \in pk(q_z)$, we can evenly distribute the load among the heavy hitters by allocating servers proportionally to the "heaviness" of executing the residual query. In other words we want $p_{h,\mathbf{u}} \sim \prod_j M_j(h)^{u_j}$ for every $h \in H$. Hence, we will choose:

$$p_{h,\mathbf{u}} = \left\lceil p \cdot \frac{\prod_j M_j(h)^{u_j})}{\sum_{h'\in H} \prod_j M_j(h')^{u_j}} \right\rceil.$$

Since $\lceil x \rceil \le x + 1$, and $|H| \le \ell p$, we can compute that the total number of servers we need is at most $(\ell + 1) \cdot |pk(q_z)| \cdot p$, which is $\Theta(p)$. Since we have that $L(\mathbf{u}, \mathbf{M}(h), p_h) = (\prod_j M_j(h)^{u_j}/p_h)^{1/\sum_j u_j}$, the maximum load L_h for every $h \in H$ will be

$$L_h = O\left(\max_{\mathbf{u}\in pk(q_z)} \left(\frac{\sum_{h\in H} \prod_j M_j(h)^{u_j}}{p}\right)^{1/(\sum_j u_j)}\right).$$

Plugging in the values of $pk(q_z)$, we obtain the following upper bound on the algorithm for the heavy hitter case:

$$O\left(\max_{I\subseteq[\ell]}\left(\frac{\sum_{h\in H}\prod_{j\in I}M_j(h)}{p}\right)^{1/|I|}\right). \tag{30}$$

Observe that the terms depend on the frequencies of the heavy hitters and can be much larger than the bound $O(\max_j M_j/p)$ we obtain from the light hitter case. In the extreme, a single heavy hitter h with $m_j(h) = m_j$ for $j = 1, \ldots, \ell$ will demand maximum load equal to $O((\prod_j M_j/p)^{1/\ell})$.

4.2.2 Algorithm for Triangle Query. We show here how to compute the triangle query $C_3 = R(x, y), S(y, z), T(z, x)$ when all relation sizes are equal to m (and have M bits). As with the star query, the algorithm will deal with the tuples that have no heavy hitter values, that is, the frequency is less than $m/p^{1/3}$, by running the vanilla HC algorithm. For this case, the load analysis of Corollary 3.3 will give us a maximum load of $\tilde{O}(M/p^{2/3})$ with high probability.

Next, we show how to handle the heavy hitters. We distinguish two cases.

Case 1. In this case, we handle the tuples that have values with frequency $\geq m/p$ in at least two variables. Observe that we did not set the heaviness threshold to $m/p^{1/3}$ for reasons that we will explain in the next case.

Without loss of generality, suppose that both x, y are heavy in at least one of the two relations they belong to. The observation is that there at most p such heavy values for each variable, and hence we can send all tuples of $R(x, y)$ with both x, y heavy (at most p^2) to all servers. Then, we essentially have to compute the query $S'(y, z), T'(z, x)$, where x and y can take only p values. We can do this by computing the join on z; since the frequency of z will be at most p for each relation (because every value of z can relate to at most p values of x or y), the maximum load from the join computation will be $O(M/p)$.

Case 2. In this case, we handle the remaining output: This includes the tuples where one variable has frequency $\geq m/p^{1/3}$, and the other variables are light, that is, have frequency $\leq m/p$. Without loss of generality, assume that we want to compute the query q for the values of x that are heavy in either R or T. Observe that there are at most $2p^{1/3}$ of such heavy hitters. If H_x denotes the set of heavy hitter values for variable x, then the residual query $q[h/x]$ for each $h \in H$ is

$$q[h/x] = R(h, y), S(y, z), T(z, h),$$

which is equivalent to computing the query $q_x = R'(y), S(y, z), T'(z)$ with cardinalities $m_R(h), m, m_T(h)$, respectively. As before, we allocate p_h servers to compute $q[h/x]$ for each $h \in H$. If there is no skew, then the maximum load L_h is given by the following formula:

$$L_h = O\left(\max\left(\frac{M}{p_h}, \sqrt{\frac{M_R(h)M_T(h)}{p_h}}\right)\right).$$

The above formula holds, because there only two edge packings that will give the optimal load: $u_S = 1, u_{R'} = u_{T'} = 0$ and $u_S = 0, u_{R'} = u_{T'} = 1$. Notice now that the only cause of skew for q_x may be that y or z are heavy in $S(y, z)$. However, we assumed that the frequencies for both y, z are $\leq m/p$, so there will be no skew (this is why we set the heaviness threshold for Case 1 to m/p instead of $m/p^{1/3}$).

We can now set $p_h = p_{h,1} + p_{h,2}$ (for each of the quantities in the max expression), and choose the allocated servers similarly to how we chose for the star queries:

$$p_{h,1} = \left\lceil p \cdot \frac{M_S(h)}{M} \right\rceil \qquad p_{h,2} = \left\lceil p \cdot \frac{M_R(h)M_T(h)}{\sum_{h \in H_x} M_R(h)M_T(h)} \right\rceil.$$

We now get a load of

$$L = O\left(\max\left(\frac{M}{p}, \sqrt{\frac{\sum_h M_R(h)M_T(h)}{p}}\right)\right).$$

Summing up all the cases, we obtain that the load of the 1-round algorithm for computing triangles is

$$L = \tilde{O}\left(\max\left(\frac{M}{p^{2/3}}, \sqrt{\frac{\sum_h M_R(h)M_T(h)}{p}}, \sqrt{\frac{\sum_h M_R(h)M_S(h)}{p}}, \sqrt{\frac{\sum_h M_S(h)M_T(h)}{p}}\right)\right).$$

In the next section, we show that this bound is tight for 1-round algorithms.

4.2.3 Lower Bound. The lower bound we present here holds for any conjunctive query, and generalizes the lower bound in Theorem 3.12, which was over databases with simple cardinality statistics $\mathbf{m} = (m_1, \ldots, m_\ell)$, to databases with a fixed degree sequence. If the degree sequence is skewed, then the new bounds can be stronger, proving that skew in the input data makes query evaluation harder.

Let us fix statistics $\mathbf{m}$ of type $\mathbf{x}$. We define $q_{\mathbf{x}}$ as the *residual query*, obtained by removing all variables $\mathbf{x}$, and decreasing the arities of S_j as necessary (the new arity of relation S_j is $a_j - d_j$). Clearly, every fractional edge packing of q is also a fractional edge packing of $q_{\mathbf{x}}$, but the converse does not hold in general. If $\mathbf{u}$ is a fractional edge packing of $q_{\mathbf{x}}$, then we say that $\mathbf{u}$ *saturates* a variable $x_i \in \mathbf{x}$, if $\sum_{j: x_i \in vars(S_j)} u_j \geq 1$; we say that $\mathbf{u}$ saturates $\mathbf{x}$ if it saturates all variables in $\mathbf{x}$. For a given $\mathbf{x}$ and $\mathbf{u}$ that saturates $\mathbf{x}$, define

$$L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p) = \left(\frac{\sum_{\mathbf{h} \in [n]^{\mathbf{x}}} \prod_j M_j(\mathbf{h}_j)^{u_j})}{p}\right)^{1/\sum_j u_j}, \tag{31}$$

where $\mathbf{M}$ is the derived statistics of type $\mathbf{x}$ based on $\mathbf{m}$.

Theorem 4.5. *Fix statistics $\mathbf{m}$ of type $\mathbf{x}$ such that $a_j > d_j$ for every relation S_j. Consider any deterministic MPC algorithm that runs in one communication round on p servers and has maximum load L in bits. Then, for any fractional edge packing $\mathbf{u}$ of q that saturates $\mathbf{x}$, we must have*

$$L \geq \min_j \frac{(a_j - d_j)}{4a_j} \cdot L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p),$$

where $\mathbf{M}$ are the derived statistics of type $\mathbf{x}$.

Note that, when $\mathbf{x} = \emptyset$ then $L_{\mathbf{x}}(\mathbf{u}, \mathbf{M}, p) = L(\mathbf{u}, \mathbf{M}, p)$, as defined in Equation (18). However, our theorem does not imply Theorem 3.12, since it does not give a lower bound on the expected size of the algorithm output as a fraction of the expected output size.

Proof. For $\mathbf{h} \in [n]^{\mathbf{x}}$ and $\mathbf{a}_j \in S_j$, we write $\mathbf{a}_j \parallel \mathbf{h}$ to denote that the tuple $\mathbf{a}_j$ from S_j matches with $\mathbf{h}$ at their common variables $\mathbf{x}_j$ and denote $(S_j)_{\mathbf{h}}$ the subset of tuples $\mathbf{a}_j$ that match $\mathbf{h}$: $(S_j)_{\mathbf{h}} = \{\mathbf{a}_j \mid \mathbf{a}_j \in S_j, \mathbf{a}_j \parallel \mathbf{h}\}$. Let $I_{\mathbf{h}}$ denote the restriction of I to $\mathbf{h}$, in other words $I_{\mathbf{h}} = ((S_1)_{\mathbf{h}}, \ldots, (S_\ell)_{\mathbf{h}})$.

We pick the domain n such that $n = (\max_j\{m_j\})^2$ and construct a probability space for instances I defined by the derived $\mathbf{x}$-statistics $\mathbf{M}$ as follows. For a fixed tuple $\mathbf{h} \in [n]^{\mathbf{x}}$, the restriction $I_{\mathbf{h}}$ is

a uniformly chosen instance over all matching databases with cardinality vector $\mathbf{M}(\mathbf{h})$, which is precisely the probability space that we used in the proof of Theorem 3.10. In particular, for every $\mathbf{a}_j \in [n]^{\mathbf{x}_j}$, the probability that S_j contains $\mathbf{a}_j$ is $P(\mathbf{a}_j \in S_j) = m_j(\mathbf{h}_j)/n^{a_j-d_j}$. Lemma 3.13 immediately gives

$$\mathbb{E}[|q(I_{\mathbf{h}})|] = n^{k-d} \prod_{j=1}^{\ell} \frac{m_j(\mathbf{h}_j)}{n^{a_j-d_j}}. \tag{32}$$

Let us fix some server and let $\mathrm{msg}(I)$ be the message the server receives on input I. As in the previous section, let $K_{\mathrm{msg}_j}(S_j)$ denote the set of tuples from relation S_j *known by the server*. Let $w_j(\mathbf{a}_j) = P(\mathbf{a}_j \in K_{\mathrm{msg}_j(S_j)}(S_j))$, where the probability is over the random choices of S_j. This is upper bounded by $P(\mathbf{a}_j \in S_j)$:

$$w_j(\mathbf{a}_j) \le m_j(\mathbf{h}_j)/n^{a_j-d_j}, \quad \text{if } \mathbf{a}_j \parallel \mathbf{h}. \tag{33}$$

We derive a second upper bound by exploiting the fact that the server receives a limited number of bits, in analogy with Lemma 3.14:

Lemma 4.6. *Let S_j a relation with $a_j > d_j$. Suppose that the size of S_j is $m_j \le n/2$ (or $m_j = n$) and that the message $msg_j(S_j)$ has at most L bits. Then, we have $\mathbb{E}[|K_{msg_j}(S_j)|] \le \frac{4L}{(a_j-d_j)\log(n)}$.*

Observe that in the case where $a_j = d_j$ for some relation S_j, the $\mathbf{x}$-statistics fix all the tuples of the instance for this particular relation, and hence $\mathbb{E}[|K_{\mathrm{msg}_j}(S_j)|] = m_j$.

Proof. We can express the entropy $H(S_j)$ as follows:

$$\begin{aligned} H(S_j) &= H(\mathrm{msg}_j(S_j)) + \sum_{\mathrm{msg}_j} P(\mathrm{msg}_j(S_j) = \mathrm{msg}_j) \cdot H(S_j \mid \mathrm{msg}_j(S_j) = \mathrm{msg}_j) \\ &\le L + \sum_{\mathrm{msg}_j} P(\mathrm{msg}_j(S_j) = \mathrm{msg}_j) \cdot H(S_j \mid \mathrm{msg}_j(S_j) = \mathrm{msg}_j). \end{aligned} \tag{34}$$

For every $\mathbf{h} \in [n]^{\mathbf{x}}$, let $K_{\mathrm{msg}_j}((S_j)_{\mathbf{h}})$ denote the known tuples that belong in the restriction of S_j to $\mathbf{h}$. Following the proof of Lemma 3.14, and denoting by $\mathcal{M}_j(\mathbf{h}_j)$ the number of bits necessary to represent $(S_j)_{\mathbf{h}}$, we have

$$\begin{aligned} H(S_j \mid \mathrm{msg}_j(S_j) = \mathrm{msg}_j) &\le \sum_{\mathbf{h}\in[n]^{\mathbf{x}}} \left(1 - \frac{|K_{\mathrm{msg}_j}((S_j)_{\mathbf{h}})|}{2m_j(\mathbf{h}_j)}\right) \mathcal{M}_j(\mathbf{h}_j) \\ &= H(S_j) - \sum_{\mathbf{h}\in[n]^{\mathbf{x}}} \frac{|K_{\mathrm{msg}_j}((S_j)_{\mathbf{h}})|}{2m_j(\mathbf{h}_j)} \mathcal{M}_j(\mathbf{h}_j) \\ &\le H(S_j) - \sum_{\mathbf{h}\in[n]^{\mathbf{x}}} \frac{|K_{\mathrm{msg}_j}((S_j)_{\mathbf{h}})|}{2m_j(\mathbf{h}_j)} m_j(\mathbf{h}_j) \frac{a_j - d_j}{2} \log(n) \\ &= H(S_j) - (1/4) \cdot |K_{\mathrm{msg}_j}(S_j)|(a_j - d_j)\log(n), \end{aligned}$$

where the last inequality comes from Proposition 3.17. Plugging this in Equation (34), and solving for $\mathbb{E}[|K_m(S_j)|]$,

$$\mathbb{E}[|K_{\mathrm{msg}_j}(S_j)|] \le \frac{4L}{(a_j - d_j)\log(n)}.$$

This concludes our proof. □

Let $q_{\mathbf{x}}$ be the residual query, and recall that $\mathbf{u}$ is a fractional edge packing that saturates $\mathbf{x}$. Define the *extended query* $q_{\mathbf{x}}'$ to consist of $q_{\mathbf{x}}$, where we add a new atom $S_i'(x_i)$ for every variable $x_i \in vars(q_{\mathbf{x}})$. Define $u_i' = 1 - \sum_{j:i\in S_j} u_j$. In other words, u_i' is defined to be the slack at the variable x_i of the packing $\mathbf{u}$. The new edge packing $(\mathbf{u}, \mathbf{u}')$ for the extended query $q_{\mathbf{x}}'$ has no more slack, and hence it is both a tight fractional edge packing and a tight fractional edge cover for $q_{\mathbf{x}}$. By adding all equalities of the tight packing we obtain the following:

$$\sum_{j=1}^{\ell}(a_j - d_j)u_j + \sum_{i=1}^{k-d} u_i' = k - d. \tag{35}$$

We next compute how many output tuples from $q(I_{\mathbf{h}})$ will be known in expectation by the server. Note that $q(I_{\mathbf{h}}) = q_{\mathbf{x}}(I_{\mathbf{h}})$, and thus,

$$\begin{aligned}
\mathbb{E}[|K_{\text{msg}}(q(I_{\mathbf{h}}))|] &= \mathbb{E}[|K_{\text{msg}}(q_{\mathbf{x}}(I_{\mathbf{h}}))|] \\
&= \sum_{\mathbf{a}\|\mathbf{h}} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \\
&= \sum_{\mathbf{a}\|\mathbf{h}} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \prod_{i=1}^{k-d} w_i'(\mathbf{a}_i) \\
&\le \prod_{i=1}^{k-d} n^{u_i'} \cdot \prod_{j=1}^{\ell} \left(\sum_{\mathbf{a}_j\|\mathbf{h}} w_j(\mathbf{a}_j)^{1/u_j} \right)^{u_j}.
\end{aligned}$$

By writing $w_j(\mathbf{a}_j)^{1/u_j} = w_j(\mathbf{a}_j)^{1/u_j - 1} w_j(\mathbf{a}_j)$ for $\mathbf{a}_j \parallel \mathbf{h}$, we can bound the sum in the above quantity as follows using Equation (33),

$$\sum_{\mathbf{a}_j\|\mathbf{h}} w_j(\mathbf{a}_j)^{1/u_j} \le \left(\frac{m_j(\mathbf{h}_j)}{n^{a_j - d_j}}\right)^{1/u_j - 1} \sum_{\mathbf{a}_j\|\mathbf{h}} w_j(\mathbf{a}_j) \quad = (m_j(\mathbf{h}_j) n^{d_j - a_j})^{1/u_j - 1} L_j(\mathbf{h}),$$

where $L_j(\mathbf{h}) = \sum_{\mathbf{a}_j\|\mathbf{h}} w_j(\mathbf{a}_j)$. Notice that for every relation S_j, we have $\sum_{\mathbf{h}_j \in [n]^{\mathbf{x}_j}} L_j(\mathbf{h}_j) = \sum_{\mathbf{a}_j \in [n]^{a_j}} w_j(\mathbf{a}_j)$. We can now write

$$\begin{aligned}
\mathbb{E}[|K_{\text{msg}}(q(I_{\mathbf{h}}))|] &\le n^{\sum_{i=1}^{k-d} u_i'} \prod_{j=1}^{\ell} \left(L_j(\mathbf{h}) m_j(\mathbf{h}_j)^{1/u_j - 1} n^{(d_j - a_j)(1/u_j - 1)} \right)^{u_j} \\
&= \prod_{j=1}^{\ell} L_j(\mathbf{h})^{u_j} \cdot \prod_{j=1}^{\ell} m_j(\mathbf{h}_j)^{-u_j} \cdot \mathbb{E}[|q(I_{\mathbf{h}})|],
\end{aligned} \tag{36}$$

where the equality follows from Equation (35) and Equation (32).

Summing over all p servers, we obtain that the expected number of answers that can be output for $q(I_{\mathbf{h}})$ is at most $p \cdot \mathbb{E}[|K_{\text{msg}}(q(I_{\mathbf{h}}))|]$. If for some $\mathbf{h} \in [n]^{\mathbf{x}}$ this number is not at least $\mathbb{E}[|q(I_{\mathbf{h}})|]$, then the algorithm will fail to compute $q(I)$. Consequently, for every $\mathbf{h}$, we must have that

$\prod_{j=1}^{\ell} L_j(\mathbf{h}_j)^{u_j} \geq (1/p) \cdot \prod_{j=1}^{\ell} m_j(\mathbf{h}_j)^{u_j}$. Summing the inequalities for every $\mathbf{h} \in [n]^{\mathbf{x}}$:

$$\begin{aligned}
\frac{1}{p} \cdot \sum_{\mathbf{h}\in[n]^{\mathbf{x}}} \prod_{j=1}^{\ell} m_j(\mathbf{h}_j)^{u_j} &\leq \sum_{\mathbf{h}\in[n]^{\mathbf{x}}} \prod_{j=1}^{\ell} L_j(\mathbf{h}_j)^{u_j} \\
&\leq \prod_{j=1}^{\ell} \left(\sum_{\mathbf{h}_j \in [n]^{\mathbf{x}_j}} L_j(\mathbf{h}_j) \right)^{u_j} && \text{by Friedgut's inequality} \\
&\leq \prod_{j=1}^{\ell} \left(\frac{4L}{(a_j - d_j)\log(n)} \right)^{u_j} && \text{by Lemma 4.6.}
\end{aligned}$$

Solving for L and using the fact that $M_j = a_j m_j \log(n)$, we obtain that for any edge packing $\mathbf{u}$ that saturates $\mathbf{x}$,

$$L \geq \left(\min_j \frac{a_j - d_j}{4a_j} \right) \cdot \left(\frac{\sum_{\mathbf{h}\in[n]^{\mathbf{x}}} \prod_j M_j(\mathbf{h}_j)^{u_j}}{p} \right)^{1/\sum_j u_j},$$

which concludes the proof. □

To see how Theorem 4.5 applies to the star query, we assume that the input servers know z-statistics $\mathbf{M}$; in other words, for every assignment $h \in [n]$ of variable z, we know that its frequency in relation S_j is $m_j(h)$. Then, for any edge packing $\mathbf{u}$ that saturates z, we obtain a lower bound of

$$L \geq (1/8) \cdot \left(\frac{\sum_{h\in[n]} \prod_j M_j(h)^{u_j}}{p} \right)^{1/\sum_j u_j}.$$

Observe that the set of edge packings that saturate z and maximize the above quantity is $\{0,1\}^{\ell} \setminus (0,\ldots,0)$. Hence, we obtain a lower bound

$$L \geq (1/8) \cdot \max_{I\subseteq[\ell]} \left(\frac{\sum_{h\in[n]} \prod_{j\in I} M_j(h)}{p} \right)^{1/|I|}.$$

It is not hard to see that our upper and lower bounds for the star query are tight up to constant factors: First, note that the set of heavy hitters H is a subset of $[n]$; hence, the upper bound (30) for the heavy hitter case matches the lower bound. The upper bound $O(\max_j M_j/p)$ for the light hitter case also matches the lower bound by setting I to be a singleton set, that is, $|I| = 1$.

By similarly varying over the choices of subsets of vertices saturated by $\mathbf{u}$, one obtains a lower bound matching the upper bound for triangle queries in Section 4.2.2.

5 MULTIPLE COMMUNICATION STEPS

In this section, we discuss the computation of queries in the MPC model in the case of multiple steps. We will establish both upper and lower bounds on the number of rounds needed to compute a query q.

To prove our results, we restrict both the structure of the input and the type of computation in the MPC model. In particular, our multi-round algorithms will process only queries where the relations are *of equal size* and the data have *no skew*. Additionally, our lower bounds are proven for a restricted version of the MPC model, called the the *tuple-based MPC model*, which limits the way communication is performed.

5.1 An Algorithm for Multiple Rounds

In Section 3, we showed that in the case where all relations have size equal to M and are matching databases (i.e., the degree of any value is exactly one), we can compute a conjunctive query q in one round with maximum load

$$L = O(M/p^{1/\tau^*(q)}),$$

where $\tau^*(q)$ denotes the fractional vertex covering number of q. Hence, for any $\varepsilon \geq 0$, a conjunctive query q with $\tau^*(q) \leq 1/(1-\varepsilon)$ can be computed in one round in the MPC model with load $L = O(M/p^{1-\varepsilon})$; recall from Section 3.5 that we call the parameter ε the *space exponent*.

We define now the class of queries Γ^r_ε using induction on r. For $r = 1$, we define

$$\Gamma^1_\varepsilon = \{q \mid \tau^*(q) \leq 1/(1-\varepsilon)\}.$$

For $r > 1$, we define Γ^r_ε to be the set of all conjunctive queries q constructed as follows. Let $q_1, \ldots, q_m \in \Gamma^{r-1}_\varepsilon$ be m queries, and let $q_0 \in \Gamma^1_\varepsilon$ be a query over a different vocabulary $V_1, \ldots, V_m$, such that $|\text{vars}(q_j)| = \text{arity}(V_j)$ for all $j \in [m]$. Then, the query $q = q_0[q_1/V_1, \ldots, q_m/V_m]$, obtained by substituting each view V_j in q_0 with its definition q_j, is in Γ^r_ε. In other words, Γ^r_ε consists of queries that have a *query plan* of depth r, where each operator is a query computable in one step with maximum load $O(M/p^{1-\varepsilon})$.

Proposition 5.1. *Every conjunctive query $q \in \Gamma^r_\varepsilon$ with input a matching database where each relation has size M can be computed by an MPC algorithm in r rounds with maximum load $L = O(M/p^{1-\varepsilon})$.*

Proof. The proof follows by induction on the number of rounds r. For the base case, by definition a query in Γ^1_ε can be computed in one round with load $L = O(M/p^{1-\varepsilon})$. For the inductive step, consider a query $q \in \Gamma^r_\varepsilon$. Then, it can be expressed as $q = q_0[q_1/V_1, \ldots, q_m/V_m]$. By the inductive hypothesis, each view V_j can be computed in $r-1$ rounds with load $O(M/p^{1-\varepsilon})$. Additionally, since the database is matching, the resulting size of each view will be at most M. Since $q_0 \in \Gamma^1_\varepsilon$, we can compute the final result from q by an additional round with load $O(M/p^{1-\varepsilon})$. □

We next present two examples that provide some intuition on the structure of the queries in the class Γ^r_ε.

Example 5.2. Consider the query L_k in Table 2 with $k = 16$; we can construct a query plan of depth $r = 2$ and load $L = O(M/p^{1/2})$ (thus with space exponent $\varepsilon = 1/2$). For ease of notation, let us write $x_{i:j} = x_i, x_{i+1}, \ldots, x_j$. The first step computes in parallel four queries,

$$\begin{aligned}
V_1 : q_1 &= S_1(x_{0:1}), S_2(x_{1:2}), S_3(x_{2:3}), S_4(x_{3:4}),\\
V_2 : q_2 &= S_5(x_{4:5}), S_6(x_{5:6}), S_7(x_{6:7}), S_8(x_{7:8}),\\
V_3 : q_3 &= S_9(x_{8:9}), S_{10}(x_{9:10}), S_{11}(x_{10:11}), S_{12}(x_{11:12}),\\
V_4 : q_4 &= S_{13}(x_{12:13}), S_{14}(x_{13:14}), S_{15}(x_{14:15}), S_{16}(x_{15:16}).
\end{aligned}$$

Each q_i is isomorphic to the query L_4, and therefore $\tau^*(q_1) = \cdots = \tau^*(q_4) = 2$. Hence, $q_i \in \Gamma^1_{1/2}$ and each q_i can be computed in one round with load $L = O(M/p^{1/2})$. Now we can express L_{16} by using the views $V_1, \ldots, V_4$ as the query

$$q_0 = V_1(x_{0:4}), V_2(x_{4:8}), V_3(x_{8:12}), V_4(x_{12:16}).$$

It is easy to see that $\tau^*(q_0) = 2$ as well, and hence $q_0 \in \Gamma^1_{1/2}$. Since $L_{16} = q_0[q_1/V_1, \ldots, q_4/V_4]$, and $q_0, \ldots, q_4 \in \Gamma^1_{1/2}$, we conclude that $L_{16} \in \Gamma^2_{1/2}$

We can generalize the above approach for any L_k. For any $\varepsilon \geq 0$, let k_ε be the largest integer such that $L_{k_\varepsilon} \in \Gamma^1_\varepsilon$. In other words, $\tau^*(L_{k_\varepsilon}) \leq 1/(1-\varepsilon)$, and so we choose $k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$. Then,

for any $k \geq k_\varepsilon$, L_k can be computed using L_{k_ε} as a building block at each round: The plan will have a depth of $\lceil \log_{k_\varepsilon}(k) \rceil$ and will achieve a load of $L = O(M/p^{1-\varepsilon})$.

Example 5.3. Consider the query $SP_k = \bigwedge_{i=1}^{k} R_i(z, x_i), S_i(x_i, y_i)$. Since $\tau^*(SP_k) = k$, the one round algorithm can achieve a load of $O(M/p^{1/k})$.

When we consider multiple rounds, we can show that for every k, $SP_k \in \Gamma_0^2$; hence there exists a query plan with two rounds and load $O(M/p)$. To achieve this, we first compute the views $V_i : q_i = R_i(z, x_i), S_i(x_i, y_i)$ for every $i = 1, \ldots, k$. Each query q_i is in Γ_0^1. Now we can write SP_k as $q_0[q_1/V_1, \ldots, q_k/V_k]$, where $q_0 = \bigwedge_{i=1}^{k} V_i(z, x_i, y_i)$. The query q_0 has $\tau^*(q_0) = 1$, hence $q_0 \in \Gamma_0^1$.

We should note here that there is no connection between the above query plans and plans formed by query decomposition methods (e.g., General Hypertree Decompositions (GHDs) (Gottlob et al. 2009)). First, the notion of width in a decomposition is different from the space exponent ε. Second, since we want to construct plans that can be executed within few rounds, minimizing the depth of a query plan is an additional objective to minimizing the space exponent.

We next present an upper bound on the number of rounds needed to compute any query if we want to achieve a given load $L = O(M/p^{1-\varepsilon})$; in other words, we ask what is the minimum number of rounds for which we can achieve a space exponent ε.

Let $\text{rad}(q) = \min_u \max_v d(u, v)$ denote the *radius* of a query q, where $d(u, v)$ denotes the distance between two nodes in the hypergraph of q. For example, $\text{rad}(L_k) = \lceil k/2 \rceil$ and $\text{rad}(C_k) = \lfloor k/2 \rfloor$.

Lemma 5.4. *Fix $\varepsilon \geq 0$, let $k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$, and let q be any connected query. Define*

$$r(q) = \begin{cases} \lceil \log_{k_\varepsilon}(rad(q)) \rceil + 1 & \text{if } q \text{ is treelike,} \\ \lfloor \log_{k_\varepsilon}(rad(q)) \rfloor + 2 & \text{otherwise.} \end{cases}$$

Then, q can be computed in $r(q)$ rounds on any matching database input with relations of size M with maximum load $L = O(M/p^{1-\varepsilon})$.

Proof. By definition of $\text{rad}(q)$, there exists some node $v \in \text{vars}(q)$, such that the maximum distance of v to any other node in the hypergraph of q is at most $\text{rad}(q)$. If q is treelike, then we can decompose q into a set of at most $|\text{atoms}(q)|^{\text{rad}(q)}$ (possibly overlapping) paths $\mathcal{P}$ of length $\leq \text{rad}(q)$, each having v as one endpoint. Since it is essentially isomorphic to L_ℓ, a path of length $\ell \leq \text{rad}(q)$ can be computed in at most $\lceil \log_{k_\varepsilon}(\text{rad}(q)) \rceil$ rounds using the query plan from Proposition 5.1 together with repeated use of the one-round HyperCube algorithm for paths of length k_ε. Moreover, all the paths in $\mathcal{P}$ can be computed in parallel, because $|\mathcal{P}|$ is a constant depending only on q. Since every path will contain variable v, we can compute the join of all the paths in one final round with load $O(M/p)$.

The only difference for general connected queries is that q may also contain atoms that join vertices at distance $\text{rad}(q)$ from v that are not on any of the paths of length $\text{rad}(q)$ from v: These can be covered using paths of length $\text{rad}(q) + 1$ from v. To get the final formula, we apply the equality $\lceil \log_a(b+1) \rceil = \lfloor \log_a(b) \rfloor + 1$, which holds for positive integers a, b. □

As an application of the above lemma, Table 3 shows the number of rounds required by different types of queries.

5.2 Lower Bounds for Multiple Rounds

To show lower bounds for the case of multiple rounds, we will need to restrict the communication in the MPC model; to do this, we define a restriction of the MPC model that we call the *tuple-based MPC model.*

Table 3. The Tradeoff between Space and Communication Rounds for Several Queries

q query	ε space exponent for 1 round	r rounds to achieve load $O(M/p)$	$r = f(\varepsilon)$ rounds/space tradeoff
C_k	$1 - 2/k$	$\lceil \log k \rceil$	$\sim \frac{\log k}{\log(2/(1-\varepsilon))}$
L_k	$1 - \frac{1}{\lceil k/2 \rceil}$	$\lceil \log k \rceil$	$\sim \frac{\log k}{\log(2/(1-\varepsilon))}$
T_k	0	1	NA
SP_k	$1 - 1/k$	2	NA

5.2.1 Tuple-Based MPC. In the general MPC model, we did not have any restrictions on the messages sent between servers at any round. In the tuple-based MPC model, we will impose some structure on how we can communicate messages.

Let I be the input database instance, q be the query we want to compute, and $\mathcal{A}$ an algorithm. For a server $s \in [p]$, we denote by $\text{msg}^1_{j\to s}(\mathcal{A}, I)$ the message sent during round 1 by the input server for S_j to the server s, and by $\text{msg}^k_{s\to s'}(\mathcal{A}, I)$ the message sent to server s' from server s at round $k \geq 2$. Let $\text{msg}^1_s(\mathcal{A}, I) = (\text{msg}^1_{1\to s}(\mathcal{A}, I), \ldots, \text{msg}^1_{\ell\to s}(\mathcal{A}, I))$ and $\text{msg}^k_s(\mathcal{A}, I) = (\text{msg}^k_{1\to s}(\mathcal{A}, I), \ldots, \text{msg}^k_{p\to s}(\mathcal{A}, I))$ for any round $k \geq 2$.

Further, we define $\text{msg}^{\leq k}_s(\mathcal{A}, i)$ to be the vector of messages received by server s during the first k rounds, and $\text{msg}^{\leq k}(\mathcal{A}, i) = (\text{msg}^{\leq k}_1(\mathcal{A}, i), \ldots, \text{msg}^{\leq k}_p(\mathcal{A}, i))$.

Define a *join tuple* to be any tuple in $q'(I)$, where q' is any connected subquery of q. An algorithm $\mathcal{A}$ in the *tuple-based MPC model* has the following two restrictions on communication during rounds $k \geq 2$, for every server s

- —the message $\text{msg}^k_{s\to s'}(\mathcal{A}, I)$ is a set of join tuples.
- —for every join tuple t, the server s decides whether to include t in $\text{msg}^k_{s\to s'}(\mathcal{A}, I)$ based only on the parameters t, s, s', r, and the messages $\text{msg}^1_{j\to s}(\mathcal{A}, I)$ for all j such that t contains a tuple in S_j.

The restricted model still allows unrestricted communication during the first round; the information $\text{msg}^1_s(\mathcal{A}, I)$ received by server s in the first round is available throughout the computation. However, during the following rounds, server s can only send messages consisting of join tuples, and, moreover, the destination of these join tuples can depend only on the tuple itself and on $\text{msg}^1_s(\mathcal{A}, I)$.

The restriction of communication to join tuples (except for the first round during which arbitrary, for example, statistical, information can be sent) is natural and the tuple-based MPC model captures a wide variety of algorithms for multiway join, including the most natural MapReduce algorithms. Since the servers can perform arbitrary inferences based on the messages that they receive, even a limitation to messages that are join tuples starting in the second round, without a restriction on how they are routed, would still essentially have been equivalent to the fully general MPC model. For example, any server wishing to send a sequence of bits to another server can encode the bits by sending a sequence of tuples that the two exchanged in previous rounds, or (with slight loss in efficiency) using the understanding that the tuples themselves are not important, but some arbitrary fixed Boolean function of those tuples is the true message being communicated. This explains the need for the condition on routing tuples that the tuple-based MPC model imposes.

5.2.2 A Lower Bound. We present here a general lower bound for connected conjunctive queries in the tuple-based MPC model.

We first introduce a combinatorial object associated with every query q, called the (ε, r)-plan, which is central to the construction of the multi-round lower bound. We next define this notion and also discuss how we can construct such plans for various classes of queries.

Given a query q and a set $M \subseteq \text{atoms}(q)$, recall that q/M is the query that results from contracting the edges M in the hypergraph of q. Also, we define $\overline{M} = \text{atoms}(q) \backslash M$. Recall that $\Gamma^1_\varepsilon = \{q \mid \tau^*(q) \leq 1/(1-\varepsilon)\}$.

Definition 5.5. Let q be a connected conjunctive query. A set $M \subseteq \text{atoms}(q)$ is *ε-good for q* if it satisfies the following two properties:

(1) Every connected subquery of q that is in Γ^1_ε contains at most one atom in M.
(2) $\chi(\overline{M}) = 0$ (and thus $\chi(q/\overline{M}) = \chi(q)$ by Lemma 2.1).

For $\varepsilon \in [0, 1)$ and integer $r \geq 0$, an *(ε, r)-plan* $\mathcal{M}$ is a sequence $M_1, \ldots, M_r$, with $M_0 = \text{atoms}(q) \supset M_1 \supset \cdots M_r$ such that (a) for $j = 0, \ldots, r-1$, M_{j+1} is ε-good for $q/\overline{M}_j$, and (b) $q/\overline{M}_r \notin \Gamma^1_\varepsilon$.

We provide some intuition about the above definition with the next two lemmas, which show how we can obtain such plans for the queries L_k and C_k, respectively.

Lemma 5.6. *The query L_k admits an $(\varepsilon, \lceil \log_{k_\varepsilon}(k) \rceil - 2)$-plan for any integer $k > k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$.*

Proof. We will prove using induction in the number of rounds r that for every integer $r \geq 0$, if $k \geq k_\varepsilon^{r+1} + 1$, then L_k admits an (ε, r)-plan. This proves the lemma, because then for a given k the smallest integer r we can choose for the plan is $r = \lfloor \log_{k_\varepsilon}(k-1) \rfloor - 1 = \lceil \log_{k_\varepsilon}(k) \rceil - 2$.

For the base case $r = 0$, we need to show that if $k \geq k_\varepsilon + 1$, then L_k admits an $(\varepsilon, 0)$-plan. Since $r = 0$, it suffices to show that condition (b) holds, that is, $q/\overline{M}_0 \notin \Gamma^1_\varepsilon$. But $M_0 = \text{atoms}(q)$, hence $L_k/\overline{M}_0 = L_k$. In other words, for $r = 0$ a query admits an $(\varepsilon, 0)$-plan if and only if it is not possible to compute the query in one round with space exponent ε. Since $L_k \notin \Gamma^1_\varepsilon$, the claim follows.

For the induction step, let $k_0 \geq k_\varepsilon^{r+1} + 1$; then from the inductive hypothesis we have that for every $k \geq k_\varepsilon^r + 1$ the query L_k has an $(\varepsilon, r-1)$-plan. Define M to be the set of atoms where we include every k_εth atom L_{k_0}, starting with S_1; in other words, $M = \{S_1, S_{k_\varepsilon+1}, S_{2k_\varepsilon+1}, \ldots\}$. Observe now that after the contraction we have:

$$L_{k_0}/\overline{M} = S_1(x_0, x_1), S_{k_\varepsilon+1}(x_1, x_{k_\varepsilon+1}), S_{2k_\varepsilon+1}(x_{k_\varepsilon+1}, x_{2k_\varepsilon+1}), \ldots$$

Observe that the query $L_{k_0}/\overline{M}$ is isomorphic to $L_{\lceil k_0/k_\varepsilon \rceil}$. Since $\lceil k_0/k_\varepsilon \rceil \geq \lceil k_\varepsilon^r + 1/k_\varepsilon \rceil = k_\varepsilon^r + 1$, from the inductive hypothesis the query $L_{k_0}/\overline{M}$ admits an $(\varepsilon, r-1)$-plan. By definition, this implies a sequence $M_1, \ldots, M_{r-1}$; we will show that the extended sequence $M, M_1, \ldots, M_{r-1}$ is an (ε, r)-plan for L_{k_0}. To show this, it is sufficient to prove that M is ε-good for L_{k_0}.

Indeed, we have $\chi(L_{k_0}/\overline{M}) = \chi(L_{\lceil k_0/k_\varepsilon \rceil}) = \chi(L_{k_0})$ and thus property (2) is satisfied. Additionally, recall that Γ^1_ε consists of queries for which $\tau^*(q) \leq 1/(1-\varepsilon)$; thus the connected subqueries of L_{k_0} that are in Γ^1_ε are precisely queries of the form $S_j(x_{j-1}, x_j), S_{j+1}(x_j, x_{j+1}), \ldots, S_{j+k-1}(x_{j+k-2}, x_{j+k-1})$, where $k \leq k_\varepsilon$. By choosing M to contain every k_εth atom, no such subquery in Γ^1_ε will contain more than one atom from M and thus property (1) is satisfied as well. Intuitively, this property guarantees that no algorithm can find which tuples join from two different atoms of M with space exponent ε. □

Lemma 5.7. *The query C_k admits an $(\varepsilon, \lfloor \log_{k_\varepsilon}(k/(m_\varepsilon + 1)) \rfloor)$-plan, for every integer $k > m_\varepsilon = \lfloor 2/(1-\varepsilon) \rfloor$.*

Proof. The proof is similar to the proof for the query L_k. Indeed, observe that any set M of atoms that are (at least) k_ε positions apart along the cycle C_k is a ε-good set for C_k. Further, the contracted query $C_k/\overline{M}$ is isomorphic to $C_{\lfloor k/k_\varepsilon \rfloor}$. The only difference is that the base case for $r = 0$ is that $k \geq m_\varepsilon + 1$. Thus, the inductive step is that for every integer $r \geq 0$, if $k \geq k_\varepsilon^r(m_\varepsilon + 1)$ then C_k admits an (ε, r)-plan. □

The above examples of queries show how we can construct (ε, r)-plans. We next present the main theorem of this section, which tells us how we can use such plans to obtain lower bounds on the number of communication rounds needed to compute a conjunctive query.

Theorem 5.8 (Lower Bound for Multiple Rounds). *Let q be a conjunctive query that admits an (ε, r)-plan. For every randomized algorithm in the tuple-based MPC model that computes q in $r + 1$ rounds and with load $L \leq cM/p^{1-\varepsilon}$ for a sufficiently small constant c, there exists an instance I with relations of size M where the algorithm fails to compute q with probability $\Omega(1)$.*

The constant c in the above theorem depends on the query q and the parameter ε. To state the precise expression the constant c, we need some additional definitions.

Definition 5.9. Let q be a conjunctive query and $\mathcal{M}$ be an (ε, r)-plan for q. We define $\tau^*(\mathcal{M})$ to be the minimum of $\tau^*(q/\overline{M}_r)$ and $\tau^*(q')$, where q' ranges over all connected subqueries of $q/\overline{M}_{j-1}$, $j \in [r]$, such that $q' \notin \Gamma_\varepsilon^1$.

Proposition 5.10. *Let q be a conjunctive query and $\mathcal{M}$ be an (ε, r)-plan for q. Then, $\tau^*(\mathcal{M}) > 1/(1-\varepsilon)$.*

Proof. For every $q' \notin \Gamma_\varepsilon^1$, we have by definition that $\tau^*(q') > 1/(1-\varepsilon)$. Additionally, by the definition of an (ε, r)-plan, we have that $\tau^*(q/\overline{M}_r) > 1/(1-\varepsilon)$. □

Further, for a given query q, let us define the following sets:

$$C(q) = \{q' \mid q' \text{ is a connected subquery of } q\}$$
$$C_\varepsilon(q) = \{q' \mid q' \notin \Gamma_\varepsilon^1,\ q' \text{ is a connected subquery of } q\}$$
$$\mathcal{S}_\varepsilon(q) = \{q' \mid q' \notin \Gamma_\varepsilon^1,\ q' \text{ is a minimal connected subquery of } q\}$$

and let

$$\beta(q, \mathcal{M}) = \left(\frac{1}{\tau^*(q/\overline{M}_r)}\right)^{\tau^*(\mathcal{M})} + \sum_{k=1}^{r} \sum_{q' \in \mathcal{S}_\varepsilon(q/\overline{M}_{k-1})} \left(\frac{1}{\tau^*(q')}\right)^{\tau^*(\mathcal{M})}.$$

We can now present the precise statement.

Theorem 5.11. *If q has an (ε, r)-plan $\mathcal{M}$, then any deterministic tuple-based MPC algorithm running in $r + 1$ rounds with maximum load L reports at most*

$$\beta(q, \mathcal{M}) \cdot \left(\frac{(r+1)L}{M}\right)^{\tau^*(\mathcal{M})} p \cdot \mathbb{E}[|q(I)|]$$

correct answers in expectation over a uniformly at random chosen matching database I where each relation has size M.

The above theorem implies Theorem 5.8 by following the same proof as in Theorem 3.20. Indeed, for $L \leq cM/p^{1-\varepsilon}$, we obtain that the output tuples will be at most $\theta \cdot \mathbb{E}[|q(I)|]$, where $\theta = \beta(q, \mathcal{M}) \cdot ((r+1)c)^{\tau^*(\mathcal{M})}$. If we choose the constant c such that $\theta < 1/9$, then we can apply Lemma 3.19 to show that for any randomized algorithm we can find an instance I where it will fail to produce the output with probability $\Omega(1)$.

In the rest of this section, we present the proof of Theorem 5.11. Let $\mathcal{A}$ be an algorithm that computes q in $r+1$ rounds. The intuition is as follows. Consider an ε-good set M; then any matching database i consists of two parts, $i = (i_M, i_{\overline{M}})$,[9] where i_M are the relations for atoms in M, and $i_{\overline{M}}$ are all the other relations. We show that, for a fixed instance $i_{\overline{M}}$, the algorithm can be used to compute $q/\overline{M}(i_M)$ in $r+1$ rounds; however, the first round is almost useless, because the algorithm can discover only a tiny number of join tuples with two or more atoms $S_j \in M$ (since every subquery q' of q that has two atoms in M is not in Γ^1_ε). This shows that the algorithm can compute most of the answers in $q/\overline{M}(i_M)$ in only r rounds, and we repeat the argument until a one-round algorithm remains.

To formalize this intuition, we need some notation. For two relations A, B we write $A \ltimes B$, called the *semijoin*, to denote the set of tuples in A for which there is a tuple in B that has equal values on their common variables. We also write $A \triangleright B$, called the *antijoin*, to denote the set of tuples in A for which no tuple in B has equal values on their common variables.

Let $\mathcal{A}$ be a deterministic algorithm with $r+1$ rounds, $k \in [r+1]$ a round number, s a server, and q' a subquery of q. We define:

$$K^{\mathcal{A},k,s}_{\text{msg}}(q') = \{\mathbf{a}' \in [n]^{\text{vars}(q')} \mid \forall \text{ matching database } i, \text{msg}^{\le k}_s(\mathcal{A}, i) = \text{msg} \Rightarrow \mathbf{a}' \in q'(i)\}$$

$$K^{\mathcal{A},k}_{\text{msg}}(q') = \bigcup_{s=1}^{p} K^{\mathcal{A},k,s}_{\text{msg}_s}(q').$$

Using the above notation, $K^{\mathcal{A},k,s}_{\text{msg}^{\le k}_s(\mathcal{A},i)}(q')$ and $K^{\mathcal{A},k}_{\text{msg}^{\le k}(\mathcal{A},i)}(q')$ denote the set of join tuples from q' known at round k by server s, and by all servers, respectively, on input i. Further, $\mathcal{A}(i) = K^{\mathcal{A},r+1}_{\text{msg}^{\le r+1}(\mathcal{A},i)}(q)$ is w.l.o.g. the final answer of the algorithm $\mathcal{A}$ on input i. Finally, let us define

$$J^{\mathcal{A},q}(i) = \bigcup_{q' \in C(q)} K^{\mathcal{A},1}_{\text{msg}^{\le 1}(\mathcal{A},i)}(q')$$

$$J^{\mathcal{A},q}_\varepsilon(i) = \bigcup_{q' \in C_\varepsilon(q)} K^{\mathcal{A},1}_{\text{msg}^{\le 1}(\mathcal{A},i)}(q').$$

$J^{\mathcal{A},q}_\varepsilon(i)$ is precisely the set of join tuples known after the first round but restricted to those that correspond to subqueries that are not computable in one round. Since these queries are not computable in one round, Equation (27) implies that the join tuples we can infer is a vanishing fraction (w.r.t. p) of the total number of join tuples; thus, the number of tuples in $J^{\mathcal{A},q}_\varepsilon(i)$ will be small.

We can now state the two lemmas we need as building blocks to prove Theorem 5.11.

Lemma 5.12. *Let q be a query, and M be any ε-good set for q. If $\mathcal{A}$ is an algorithm with $r+1$ rounds for q, then for any matching database $i_{\overline{M}}$ over the atoms of $\overline{M}$, there exists an algorithm $\mathcal{A}'$ with r rounds for $q/\overline{M}$ using the same number of processors and the same total number of bits of communication received per processor such that, for every matching database i_M defined over the atoms of M:*

$$|\mathcal{A}(i_M, i_{\overline{M}})| \le |q(i_M, i_{\overline{M}}) \ltimes J^{\mathcal{A},q}_\varepsilon(i_M, i_{\overline{M}})| + |\mathcal{A}'(i_M)|.$$

In other words, the algorithm returns no more answers than the (very few) tuples in $J^{\mathcal{A},q}_\varepsilon$, plus what another algorithm $\mathcal{A}'$ that we define next computes for $q/\overline{M}$ using *one fewer* round.

[9] We will use i to denote a fixed matching instance, as opposed to I which denotes a random instance.

Proof. We call $q/\overline{M}$ the *contracted* query. While the original query q takes as input the complete database $i = (i_M, i_{\overline{M}})$, the input to the contracted query is only i_M. Observe also that for different matching databases $i_{\overline{M}}$, the lemma produces different algorithms $\mathcal{A}'$. We fix now a matching database $i_{\overline{M}}$.

The construction of $\mathcal{A}'$ is based on the following two constructions, which we call *contraction* and *retraction*.

Contraction. We first show how to use the algorithm $\mathcal{A}$ to derive an algorithm $\mathcal{A}^c$ for $q/\overline{M}$ that uses the same number of rounds as $\mathcal{A}$.

For each connected component q_c of $\overline{M}$, we choose a representative variable $z_c \in \text{vars}(q_c)$. The query answer $q_c(i_{\overline{M}})$ is a matching instance, since q_c is treelike (because $\chi(\overline{M}) = 0$). Denote $\boldsymbol{\sigma} = \{\sigma_x \mid x \in \text{vars}(q)\}$, where, for every variable $x \in \text{vars}(q)$, $\sigma_x : [n] \to [n]$ is the following permutation. If $x \notin \text{vars}(\overline{M})$, then σ_x is defined as the identity, that is, $\sigma_x(a) = a$ for every $a \in [n]$. Otherwise, if q_c is the unique connected component such that $x \in \text{vars}(q_c)$ and $\mathbf{a} \in q_c(i_{\overline{M}})$ is the unique tuple such that $\mathbf{a}_x = a$, we define $\sigma_x(a) = \mathbf{a}_{z_c}$. In other words, we think of $\boldsymbol{\sigma}$ as permuting the domain of each variable $x \in \text{vars}(q)$. Observe that $\boldsymbol{\sigma}$ is known to all servers, since $i_{\overline{M}}$ is a fixed instance.

It holds that $\boldsymbol{\sigma}(q(i)) = q(\boldsymbol{\sigma}(i))$, and $\boldsymbol{\sigma}(i_{\overline{M}}) = \mathbf{id}_{\overline{M}}$, where $\mathbf{id}_{\overline{M}}$ is the identity matching database (where each relation in $\overline{M}$ is $\{(1, 1, \ldots), (2, 2, \ldots), \ldots\}$). Therefore,[10] if Π denotes the projection operator, then

$$q/\overline{M}(i_M) = \boldsymbol{\sigma}^{-1}(\Pi_{\text{vars}(q/\overline{M})}(q(\boldsymbol{\sigma}(i_M), \mathbf{id}_{\overline{M}}))).$$

Using the above equation, we can now define the algorithm $\mathcal{A}^c$ that computes the query $q/\overline{M}(i_M)$. First, each input server for $S_j \in M$ replaces S_j with $\boldsymbol{\sigma}(S_j)$. Second, we run $\mathcal{A}$ unchanged, substituting all relations $S_j \in \overline{M}$ with the identity. Finally, we apply $\boldsymbol{\sigma}^{-1}$ to the answers and return the output. Hence, we have:

$$\mathcal{A}^c(i_M) = \boldsymbol{\sigma}^{-1}(\Pi_{\text{vars}(q/\overline{M})}(\mathcal{A}(\boldsymbol{\sigma}(i_M), \mathbf{id}_{\overline{M}}))). \tag{37}$$

Retraction. Next, we transform $\mathcal{A}^c$ into a new algorithm $\mathcal{A}^r$, called the *retraction* of A^c, that takes as input i_M as follows.

- In round 1, each input server for S_j sends (in addition to the messages sent by $\mathcal{A}^c$) every tuple in $\mathbf{a}_j \in S_j$ to all servers s that eventually receive $\mathbf{a}_j$. In other words, the input server sends t to every s for which there exists $k \in [r+1]$ such that $\mathbf{a}_j \in K^{\mathcal{A}^c, k, s}_{\text{msg}^{\leq k}_s(\mathcal{A}^c, i_M)}(S_j)$. This is possible because of the restrictions in the tuple-based MPC model: All destinations of $\mathbf{a}_j$ depend only on S_j, and hence can be computed by the input server. Note that this does not increase the total number of bits received by any processor, though it does mean that more communication, possibly all of it, will be performed during the first round. Therefore, over the course of the entire inductive argument, the load is never more than the number of rounds r times the original load.
- In round 2, $\mathcal{A}^r$ sends *no tuples*.
- In rounds $k \geq 3$, $\mathcal{A}^r$ sends a join tuple t from server s to server s' if server s *knows* t at round k, and also algorithm $\mathcal{A}^c$ sends t from s to s' at round k.

Observe first that the algorithm $\mathcal{A}^r$ is correct, in the sense that the output $\mathcal{A}^r(i_M)$ will be a subset of $q/\overline{M}(i_M)$. We now need to quantify how many tuples $\mathcal{A}^r$ misses compared to the

[10] We assume $\text{vars}(q/\overline{M}) \subseteq \text{vars}(q)$; for that, when we contract a set of nodes of the hypergraph, we replace them with one of the nodes in the set.

contracted algorithm $\mathcal{A}^c$. Let $Q_M = \{q' \mid q' \text{ subquery of } q/\overline{M}, |q'| \geq 2\}$, and define

$$J_+^{\mathcal{A}^c}(i_M) = \bigcup_{q' \in Q_M} K^{\mathcal{A}^c,1}_{\text{msg}^1(\mathcal{A}^c,i)}(q').$$

The set $J_+^{\mathcal{A}^c}(i_M)$ is exactly the set of non-atomic tuples known by $\mathcal{A}^c$ right after round 1: These are also the tuples that the new algorithm $\mathcal{A}^r$ will choose not to send during round 2.

LEMMA 5.13. $(\mathcal{A}^c(i_M) \triangleright J_+^{\mathcal{A}^c}(i_M)) \subseteq \mathcal{A}^r(i_M)$.

PROOF. We will prove the statement by induction on the number of rounds: For any subquery q' of $q/\overline{M}$, if server s knows $t \in (q'(i_M) \triangleright J_+^{\mathcal{A}^c}(i_M))$ at round k for algorithm $\mathcal{A}^c$, then server s knows t at round k for algorithm $\mathcal{A}^r$ as well.

For the induction base, in round 1 we have by construction that $K^{\mathcal{A}^c,1,s}_{\text{msg}^1_s(\mathcal{A}^c,i_M)}(S_j) \subseteq K^{\mathcal{A}^r,1,s}_{\text{msg}^1_s(\mathcal{A}^r,i_M)}(S_j)$ for every $S_j \in M$, and thus any tuple t (join or atomic) that is known by server s for algorithm $\mathcal{A}^c$ will be also known for algorithm $\mathcal{A}^r$.

Consider now some round $k+1$ and a tuple $t \in (q'(i_M) \triangleright J_+^{\mathcal{A}^c}(i_M))$ known by server s for algorithm $\mathcal{A}^c$. If q' is a single relation, then the statement is correct since by construction all atomic tuples are known at round 1 for algorithm $\mathcal{A}^r$. Otherwise, $q' \in Q_M$. Let $t_1, \ldots, t_m$ be the subtuples at server s from which tuple t is constructed, where $t_j \in q_j(i_M)$ for every $j = 1, \ldots, m$. Observe that $t_j \in (q_j(i_M) \triangleright J_+^{\mathcal{A}^c}(i_M))$. Thus, if t_i was known at round k by some server s' for algorithm $\mathcal{A}^c$, by the induction hypothesis it would be known by server s' for algorithm $\mathcal{A}^r$ as well, and thus it would have been communicated to server s at round $k+1$. □

From the above lemma it follows that

$$\mathcal{A}^c(i_M) \subseteq \mathcal{A}^r(i_M) \cup (q/\overline{M}(i_M) \ltimes J_+^{\mathcal{A}^c}(i_M)). \tag{38}$$

Additionally, by the definition of ε-goodness, if a subquery q' of q has two atoms in M, then $q' \notin \Gamma^1_\varepsilon$. Hence, we also have

$$J_+^{\mathcal{A}^c}(i_M) \subseteq \sigma^{-1}(\Pi_{\text{vars}(q/\overline{M})}(J_\varepsilon^{\mathcal{A},q}(\sigma(i)))). \tag{39}$$

Since $\mathcal{A}^r$ send no information during the second round, we can compress it to an algorithm $\mathcal{A}'$ that uses only r rounds. Finally, since M is ε-good, we have $\chi(q/\overline{M}) = \chi(q)$ and thus $|\mathcal{A}^c(i_M)| = |\mathcal{A}(i_M, i_{\overline{M}})|$. Combining everything together,

$$\begin{aligned}
|\mathcal{A}(i_M, i_{\overline{M}})| &= |\mathcal{A}^c(i_M)| \\
&\leq |\mathcal{A}^r(i_M)| + |(q/\overline{M}(i_M) \ltimes J_+^{\mathcal{A}^c}(i_M))| \\
&\leq |\mathcal{A}'(i_M)| + |(q/\overline{M}(i_M) \ltimes \sigma^{-1}(\Pi_{\text{vars}(q/\overline{M})}(J_\varepsilon^{\mathcal{A},q}(\sigma(i)))))| \\
&\leq |\mathcal{A}'(i_M)| + |\Pi_{\text{vars}(q/\overline{M})}(q(i) \ltimes J_\varepsilon^{\mathcal{A},q}(i))| \\
&\leq \mathcal{A}'(i_M)| + |q(i) \ltimes J_\varepsilon^{\mathcal{A},q}(i)|.
\end{aligned}$$

This concludes the proof. □

LEMMA 5.14. *Let q be a conjunctive query and q' a subquery of q. Let $\mathcal{B}$ be any algorithm that outputs a subset of answers to q' (i.e., for every database i, $\mathcal{B}(i) \subseteq q'(i)$). Let I be a uniformly at random chosen matching database for q, and $I' = I_{atoms(q')}$ its restriction over the atoms in q'.*

If $\mathbb{E}[|\mathcal{B}(I')|] \leq \gamma \cdot \mathbb{E}[|q'(I')|]$, then $\mathbb{E}[|q(I) \ltimes \mathcal{B}(I')|] \leq \gamma \cdot \mathbb{E}[|q(I)|]$.

Proof. Let $\mathbf{y} = \text{vars}(q')$ and $d = |\mathbf{y}|$. By symmetry, the quantity $\mathbb{E}[|\sigma_{\mathbf{y}=\mathbf{a}}(q(I))|]$ is independent of $\mathbf{a}$ and therefore equals $\mathbb{E}[|q(I)|]/n^d$. Notice that by construction $\sigma_{\mathbf{y}=\mathbf{a}}(\mathcal{B}(i')) \subseteq \{\mathbf{a}\}$. We now have

$$\begin{aligned}
\mathbb{E}[|q(I) \ltimes B(I')|] &= \sum_{\mathbf{a}\in[n]^d} \mathbb{E}[|\sigma_{\mathbf{y}=\mathbf{a}}(q(I)) \ltimes \sigma_{\mathbf{y}=\mathbf{a}}(\mathcal{B}(I'))|] \\
&= \sum_{\mathbf{a}\in[n]^d} \mathbb{E}[|\sigma_{\mathbf{y}=\mathbf{a}}(q(I))|] \cdot P(\mathbf{a} \in \mathcal{B}(I')) \\
&= \left(\mathbb{E}[|q(I)|]/n^d\right) \cdot \sum_{\mathbf{a}\in[n]^d} \mathbb{P}(\mathbf{a} \in \mathcal{B}(I')) \\
&= \mathbb{E}[|q(I)|] \cdot \mathbb{E}[|\mathcal{B}(I')|]/n^d \\
&\le \mathbb{E}[|q(I)|] \cdot (\gamma \cdot \mathbb{E}[|q'(I')|])/n^d \\
&\le \gamma \cdot \mathbb{E}[|q(I)|].
\end{aligned}$$

where the last inequality follows from the fact that $\mathbb{E}[|q'(I')|] \le n^d$ (since for every database i', we have $|q'(i')| \le n^d$). □

Proof of Theorem 5.11. Given an (ε, r)-plan $\text{atoms}(q) = M_0 \supset M_1 \supset \ldots \supset M_r$, we define $\hat{M}_k = \overline{M}_k - \overline{M}_{k-1}$ for $k \ge 1$. Let $\mathcal{A}$ be an algorithm for q that uses $(r+1)$ rounds.

We start by applying Lemma 5.12 for algorithm $\mathcal{A}$ and the ε-good set M_1. Then, for every matching database $i_{\widehat{M}_1} = i_{\widehat{M}_1}$, there exists an algorithm $\mathcal{A}^{(1)}_{i_{\widehat{M}_1}}$ for $q/\widehat{M}_1$ that runs in r rounds such that for every matching database i_{M_1} we have

$$|\mathcal{A}(i)| \le |q(i) \ltimes J_\varepsilon^{\mathcal{A},q}(i)| + |\mathcal{A}^{(1)}_{i_{\widehat{M}_1}}(i_{M_1})|.$$

We can iteratively apply the same argument. For $k = 1, \ldots, r-1$, let us denote $\mathcal{B}^k = \mathcal{A}^{(k)}_{i_{\overline{M}_k}}$ the inductively defined algorithm for query $q/\overline{M}_k$, and consider the ε-good set M_{k+1}. Then, for every matching database $i_{\widehat{M}_{k+1}}$ there exists an algorithm $\mathcal{B}^{k+1} = \mathcal{A}^{(k+1)}_{i_{\overline{M}_{k+1}}}$ for $q/\overline{M}_{k+1}$ such that for every matching database $i_{M_{k+1}}$, we have

$$|\mathcal{B}^k(i_{M_k})| \le |q/\overline{M}_k(i_{M_k}) \ltimes J_\varepsilon^{\mathcal{B}^k, q/\overline{M}_k}(i_{M_k})| + |\mathcal{B}^{k+1}(i_{M_{k+1}})|.$$

We can now combine all the above inequalities for $k = 0, \ldots, r-1$ to obtain

$$\begin{aligned}
|\mathcal{A}(i)| \le\ & |q(i) \ltimes J_\varepsilon^{\mathcal{A},q}(i_{M_r}, i_{\widehat{M}_1}, \ldots, i_{\widehat{M}_r})| \\
&+ |q/\overline{M}_1(i_{M_1}) \ltimes J_\varepsilon^{\mathcal{B}^1, q/\overline{M}_1}(i_{M_r}, i_{\widehat{M}_2}, \ldots, i_{\widehat{M}_r})| \\
&+ \ldots \\
&+ |q/\overline{M}_{r-1}(i_{M_{r-1}}) \ltimes J_\varepsilon^{\mathcal{B}^{r-1}, q/\overline{M}_{r-1}}(i_{M_r}, i_{\widehat{M}_r})| \\
&+ |\mathcal{B}^r(i_{M_r})|. \qquad (40)
\end{aligned}$$

We now take the expectation of Equation (40) over a uniformly chosen matching database I and upper bound each of the resulting terms. Observe, first, that for all $k = 0, \ldots, r$, we have $\chi(q/\overline{M}_k) = \chi(q)$, and, hence, by Lemma 3.13, we have $\mathbb{E}[|q(I)|] = \mathbb{E}[|(q/\overline{M}_k)(I_{M_k})|]$.

We start by analyzing the last term of the equation, which is the expected output of an algorithm $\mathcal{B}^r$ that uses one round to compute $q/\overline{M}_r$. By the definition of $\tau^*(\mathcal{M})$, we have $\tau^*(q/\overline{M}_r) \ge \tau^*(\mathcal{M})$. Since the number of bits received by each processor in the first round of algorithm $\mathcal{B}^r$ is at most

the total for the original algorithm $\mathcal{A}$, it therefore is at most $(r+1)L$, so we can apply Theorem 3.12 to obtain that:

$$\mathbb{E}[\mathcal{B}^r(I_{M_r})] \le p\left(\frac{(r+1)L}{\tau^*(q/\overline{M}_r)M}\right)^{\tau^*(q/\overline{M}_r)} \mathbb{E}[|(q/\overline{M}_r)(I_{M_r})|$$
$$\le p\left(\frac{(r+1)L}{\tau^*(q/\overline{M}_r)M}\right)^{\tau^*(\mathcal{M})} \mathbb{E}[|q(I)||.$$

We next bound the remaining terms. Note that $I_{M_{k-1}} = (I_{M_r}, I_{\widetilde{M}_k}, \ldots, I_{\widetilde{M}_r})$ and consider the expected number of tuples in $J = J_\varepsilon^{\mathcal{B}^{k-1}, q/\overline{M}_{k-1}}(I_{M_{k-1}})$. The algorithm $\mathcal{B}^{k-1} = \mathcal{A}^{(k-1)}_{I_{\overline{M}_{k-1}}}$ itself depends on the choice of $I_{\overline{M}_{k-1}}$; still, we show that J has a small number of tuples. Every subquery q' of $q/\overline{M}_{k-1}$ that is not in Γ^1_ε (and hence contributes to J) has $\tau^*(q') \ge \tau^*(\mathcal{M})$. For each fixing $I_{\overline{M}_{k-1}} = i_{\overline{M}_{k-1}}$, the expected number of tuples produced for subquery q' by $B_{q'}$, where $B_{q'}$ is the portion of the first round of $\mathcal{A}^{(k-1)}_{i_{\overline{M}_{k-1}}}$ that produces tuples for q', satisfies $\mathbb{E}[|B_{q'}(I_{M_{k-1}})|] \le \gamma(q') \cdot \mathbb{E}[|q'(I_{M_{k-1}})|]$, where

$$\gamma(q') = p\left(\frac{(r+1)L}{\tau^*(q')M}\right)^{\tau^*(\mathcal{M})},$$

since each processor in a round of $\mathcal{A}^{(k-1)}_{i_{\overline{M}_{k-1}}}$ (and hence $B_{q'}$) receives at most $r+1$ times the communication bound for a round of A. We now apply Lemma 5.14 to derive

$$\mathbb{E}[|q(I) \ltimes B_{q'}(I_{M_{k-1}})|] = \mathbb{E}[|(q/\overline{M}_{k-1})(I_{M_{k-1}}) \ltimes B_{q'}(I_{M_{k-1}})|]$$
$$\le \gamma(q') \cdot \mathbb{E}[|(q/\overline{M}_{k-1})(I_{M_{k-1}})|]$$
$$= \gamma(q') \cdot \mathbb{E}[|q(I)|].$$

Averaging over all choices of $I_{\overline{M}_{k-1}} = i_{\overline{M}_{k-1}}$ and summing over the number of different queries $q' \in \mathcal{S}(q/\overline{M}_{k-1})$, where we recall that $\mathcal{S}_\varepsilon(q/\overline{M}_{k-1})$ is the set of all minimal connected subqueries q' of $q/\overline{M}_{k-1}$ that are not in Γ^1_ε, we obtain

$$\mathbb{E}[|q(I) \ltimes J_\varepsilon^{A^{k-1}, q/\overline{M}_{k-1}}(I_{M_{k-1}})|] \le \sum_{q' \in \mathcal{S}_\varepsilon(q/\overline{M}_{k-1})} \gamma(q') \cdot \mathbb{E}[|q(I)|].$$

Combining the bounds obtained for the $r+1$ terms in Equation (40), we conclude that

$$\mathbb{E}[|A(I)|] \le \left(\left(\frac{1}{\tau^*(q/\overline{M}_r)}\right)^{\tau^*(\mathcal{M})} + \sum_{k=1}^{r} \sum_{q' \in \mathcal{S}_\varepsilon(q/\overline{M}_{k-1})} \left(\frac{1}{\tau^*(q')}\right)^{\tau^*(\mathcal{M})}\right)$$
$$\times \left(\frac{(r+1)L}{M}\right)^{\tau^*(\mathcal{M})} p \cdot \mathbb{E}[|q(I)|]$$
$$= \beta(q, \mathcal{M}) \cdot \left(\frac{(r+1)L}{M}\right)^{\tau^*(\mathcal{M})} p \cdot \mathbb{E}[|q(I)|],$$

which proves Theorem 5.11. □

5.3 Application of the Lower Bound

We show now how to apply Theorem 5.8 to obtain lower bounds for several query classes and compare the lower bounds with the upper bounds.

The first class is the queries L_k, where the following corollary is a straightforward application of Theorem 5.8 and Lemma 5.6.

Corollary 5.15. *Any tuple-based MPC algorithm that computes L_k with load $L = O(M/p^{1-\varepsilon})$ requires at least $\lceil \log_{k_\varepsilon} k \rceil$ rounds of computation.*

Observe that this gives a tight lower bound for L_k, since in the previous section we showed that there exists a query plan with depth $\lceil \log_{k_\varepsilon} k \rceil$ and load $O(M/p^{1-\varepsilon})$.

Second, we give a lower bound for treelike queries and for that we use a simple observation:

Proposition 5.16. *Let q be a treelike query, and q' be any connected subquery of q. If there exists a tuple-based algorithm that computes q' with load L in r rounds, then any tuple-based algorithm that computes q with load L needs at least r rounds.*

Proof. Given any tuple-based MPC algorithm A for computing q in r rounds with maximum load L, we construct a tuple-based MPC algorithm A' that computes q' in r rounds with at most load L. A' will interpret each instance over q' as part of an instance for q by using the relations in q' and using the identity permutation $(S_j = \{(1, 1, \ldots), (2, 2, \ldots), \ldots\})$ for each relation in $q \setminus q'$. Then A' runs exactly as A for r rounds; after the final round, A' projects out for every tuple all the variables not in q'. The correctness of A' follows from the fact that q is treelike. □

Define $\mathrm{diam}(q)$, the *diameter* of a query q, to be the longest distance between any two nodes in the hypergraph of q. In general, $\mathrm{rad}(q) \le \mathrm{diam}(q) \le 2\,\mathrm{rad}(q)$. For example, $\mathrm{rad}(L_k) = \lfloor k/2 \rfloor$, $\mathrm{diam}(L_k) = k$, and $\mathrm{rad}(C_k) = \mathrm{diam}(C_k) = \lfloor k/2 \rfloor$.

Corollary 5.17. *Any tuple-based MPC algorithm that computes a treelike query q with load $L = O(M/p^{1-\varepsilon})$ needs at least $\lceil \log_{k_\varepsilon}(\mathrm{diam}(q)) \rceil$ rounds.*

Proof. Let q' be the subquery of q that corresponds to the diameter of q. Notice that q' is a connected query, and, moreover, it behaves exactly like $L_{\mathrm{diam}(q)}$. Hence, by Lemma 5.15 any algorithm needs at least $\lceil \log_{k_\varepsilon}(\mathrm{diam}(q)) \rceil$ to compute q'. By applying Proposition 5.16, we have that q needs at least that many rounds as well. □

Let us compare the lower bound $r_{\mathrm{low}} = \lceil \log_{k_\varepsilon}(\mathrm{diam}(q)) \rceil$ and the upper bound $r_{\mathrm{up}} = \lceil \log_{k_\varepsilon}(\mathrm{rad}(q)) \rceil + 1$ from Lemma 5.4. Since $\mathrm{diam}(q) \le 2\mathrm{rad}(q)$, we have that $r_{\mathrm{low}} \le r_{\mathrm{up}}$. Additionally, $\mathrm{rad}(q) \le \mathrm{diam}(q)$ implies $r_{\mathrm{up}} \le r_{\mathrm{low}} + 1$. Thus, the gap between the lower bound and the upper bound on the number of rounds is at most 1 for treelike queries. When $\varepsilon < 1/2$, these bounds are matching, since $k_\varepsilon = 2$ and $2\mathrm{rad}(q) - 1 \le \mathrm{diam}(q)$ for treelike queries.

Third, we study one instance of a non treelike query, namely the cycle query C_k. The lemma is a direct application of Lemma 5.7.

Lemma 5.18. *Any tuple-based MPC algorithm that computes the query C_k with load $L = O(M/p^{1-\varepsilon})$ requires at least $\lfloor \log_{k_\varepsilon}(k/(m_\varepsilon + 1)) \rfloor + 2$ rounds, where $m_\varepsilon = \lfloor 2/(1-\varepsilon) \rfloor$.*

For cycle queries we also have a gap of at most 1 between this lower bound and the upper bound in Lemma 5.4.

Example 5.19. Let $\varepsilon = 0$ and consider two queries, C_5 and C_6. In this case, we have $k_\varepsilon = m_\varepsilon = 2$ and $\mathrm{rad}(C_5) = \mathrm{rad}(C_6) = 2$.

For query C_6, the lower bound is then $\lfloor \log_2(6/3) \rfloor + 2 = 3$ rounds, while the upper bound is $\lceil \log_2(3) \rceil + 1 = 3$ rounds. Hence, in the case of C_6 we have tight upper and lower bounds. For query

C_5, the upper bound is again $\lceil \log_2(3) \rceil + 1 = 3$ rounds, but the lower bound becomes $\lfloor \log_2(5/3) \rfloor + 2 = 2$ rounds. The exact number of rounds necessary to compute C_5 is thus open.

Finally, we show how to apply Lemma 5.15 to show that transitive closure even on some very simple graphs requires many rounds. In particular, we consider the problem Connected-Components, for which, given an undirected graph $G = (V, E)$ with input a set of edges, the requirement is to label the nodes of each connected component with the same label, unique to that component. Our very simple graphs will be disjoint unions of paths, and the lower bound holds when the input edges are distributed for the algorithm in a structured fashion that is natural for such graphs. For this class of inputs, the multiway join view naturally captures the problem. Moreover, the tuple-based restriction of these algorithms captures a natural restricted class of MapReduce algorithms for general connectivity in which the grouping by key only gathers together portions of the graph that are known to be connected, which for our restricted class of graphs means being part of a join tuple. The routing by key of MapReduce yields the restriction on the routing decisions for join tuples in our model.

Theorem 5.20. *Let G be an input graph of size M. For any $0 \le \varepsilon < 1$, there is no algorithm in the tuple-based MPC model that computes* Connected-Components *on unions of disjoint paths using p processors and load $L = O(M/p^{1-\varepsilon})$ in $o(\log p)$ rounds.*

The idea of the proof is to construct input graphs for Connected-Components whose components correspond to the output tuples for L_k for $k = p^\delta$ for some small constant δ depending on ε and use the round lower bound for solving L_k. Notice that the size of the query L_k is not fixed but depends on the number of processors p.

Proof. Since larger ε implies a more powerful algorithm, we assume without loss of generality that $\varepsilon = 1 - 1/t$ for some integer constant $t > 1$. Let $\delta = 1/(2t(t+2))$. The family of input graphs and the initial distribution of the edges to servers will look like an input to L_k, where $k = \lfloor p^\delta \rfloor$. In particular, the vertices of the input graph G will be partitioned into $k+1$ sets $P_1, \ldots, P_{k+1}$, each partition containing m/k vertices. The edges of G will form permutations (matchings) between adjacent partitions, P_i, P_{i+1}, for $i = 1, \ldots, k$. Thus, G will contain exactly $k \cdot (m/k) = m$ edges. This construction creates essentially k binary relations, each with m/k tuples and size $M_k = (m/k)\log(m/k)$.

Since $k < p$, we can assume that the adversary initially places the edges of the graph so each server is given edges only from one relation. It is now easy to see that any tuple-based algorithm in MPC that solves Connected-Components for an arbitrary graph G of the above family in r rounds with load L implies an $(r+1)$-round tuple-based algorithm with the same load that solves L_k when each relation has size M. Indeed, the new algorithm runs the algorithm for connected components for the first r rounds and then executes a join on the labels of each node. Since each tuple in L_k corresponds exactly to a connected component in G, the join will recover all the tuples of L_k.

Since the query size is not independent of the number of servers p, we have to carefully compute the constants for our lower bounds. Consider an algorithm for L_k with load $L \le cM/p^{1-\varepsilon}$, where $M = m\log(m)$. Let $r = \lceil \log_{k_\varepsilon} k \rceil - 2$. Observe also that $k_\varepsilon = 2t$ since $\varepsilon = 1 - 1/t$.

We will use the (ε, r)-plan $\mathcal{M}$ for L_k presented in the proof of Lemma 5.6, apply Theorem 5.11, and compute the factor $\beta(L_k, \mathcal{M})$. First, notice that each query $L_k/\overline{M}_j$ for $j = 0, \ldots, r$ is isomorphic to L_{k/k_ε^j}. Then, the set $\mathcal{S}_\varepsilon(L_{k/k_\varepsilon^j})$ consists of at most k/k_ε^j paths q' of length $k_\varepsilon + 1$. By the choice of r, $L_k/\overline{M}_r$ is isomorphic to L_ℓ, where $k_\varepsilon + 1 \le \ell < k_\varepsilon^2$. Further, we have that $\tau^*(\mathcal{M}) = \tau^*(L_{k_\varepsilon+1}) = \lceil (k_\varepsilon+1)/2 \rceil = t+1$, since $k_\varepsilon = 2t$.

Thus, we have

$$\begin{aligned}\beta(L_k, \mathcal{M}) &= \left(\frac{1}{\tau^*(L_k/\overline{M}_r)}\right)^{\tau^*(\mathcal{M})} + \sum_{j=1}^{r} \sum_{q' \in \mathcal{S}_\varepsilon(q/\overline{M}_{k-1})} \left(\frac{1}{\tau^*(q')}\right)^{\tau^*(\mathcal{M})} \\ &\leq (1-\varepsilon)^{\tau^*(\mathcal{M})} \left(1 + \sum_{j=1}^{r} \frac{k}{k_\varepsilon^{j-1}}\right) \\ &\leq (2k+1)(1-\varepsilon)^{\tau^*(\mathcal{M})}.\end{aligned}$$

Observe now that $M/M_k = 1/(1/k - \log(k)/(k\log(m))) \leq 2k$, assuming that $m \geq p^{2\delta}$. Consequently, Theorem 5.11 implies that any tuple-based MPC algorithm using at most $\lceil \log_{k_\varepsilon} k \rceil - 1$ rounds reports at most the following fraction of the required output tuples for the L_k query:

$$\begin{aligned}\beta(L_k, \mathcal{M}) \cdot p \left(\frac{(r+1)L}{M_k}\right)^{\tau^*(\mathcal{M})} &\leq (2k+1)(2ck(r+1)/t)^{\tau^*(\mathcal{M})} \cdot p^{1-\tau^*(\mathcal{M})(1-\varepsilon)} \\ &= (2k+1)(2ck(r+1)/t)^{1+t} \cdot p^{1-(1+t)(1-\varepsilon)} \\ &\leq c_1 k^{t+2} (c_2 r)^{1+t} \cdot p^{1-(1+t)(1-\varepsilon)} \\ &\leq c_3 k^{t+2} (\log_2 k)^{c_4} \cdot p^{1-(1+t)(1-\varepsilon)} \\ &\leq c_3 (\delta \log_2 p)^{c_4} \cdot p^{\delta(t+2)+1-(1+t)(1-\varepsilon)} \\ &= c_3 (\delta \log_2 p)^{c_4} \cdot p^{\delta(t+2)-1/t} \\ &= c_3 (\delta \log_2 p)^{c_4} \cdot p^{-1/2t},\end{aligned}$$

where c_3, c_4 are constants. Since $t > 1$, the fraction of the output tuples is $o(1)$ as a function of the number of processors p. This implies that any algorithm that computes Connected-Components on G requires at least $\lceil \log_{k_\varepsilon} \lfloor p^\delta \rfloor \rceil - 2 = \Omega(\log p)$ rounds. □

6 RELATED WORK

MapReduce-Related Models. Several computation models have been proposed to understand the power of MapReduce and other related massively parallel programming frameworks (Feldman et al. 2010; Karloff et al. 2010; Koutris and Suciu 2011; Afrati et al. 2012; Pietracaprina et al. 2012). These all identify the number of communication rounds as a main complexity parameter but differ in their treatment of the communication.

The first of these models was the Massive, Unordered, Distributed (MUD) model of Feldman et al. (2010). It takes as input a sequence of elements and applies a binary merge operation repeatedly, until obtaining a final result, similarly to a User Defined Aggregate in database systems. The article compares MUD with streaming algorithms: A streaming algorithm can trivially simulate MUD, and the converse is also possible if the merge operators are computationally powerful (beyond PTIME).

Karloff et al. (2010) define $\mathcal{MRC}$, a class of multi-round algorithms based on using the MapReduce primitive as the sole building block, and fixing specific parameters for balanced processing. The number of processors p is $\Theta(N^{1-\epsilon})$, and each can exchange MapReduce outputs expressible in $\Theta(N^{1-\epsilon})$ bits per step, resulting in $\Theta(N^{2-2\epsilon})$ total storage among the processors on a problem of size N. Their focus was algorithmic, showing simulations of other parallel models by $\mathcal{MRC}$, as well as the power of two round algorithms for specific problems.

Pietracaprina et al. (2012) propose a MapReduce model where the parameters are the number of rounds, the maximum size m of the memory of each reducer, and the total amount of memory

M. The $\mathcal{MRC}$ model is a special case where $m = O(N^{1-\epsilon})$ and $M = O(N^{2-2\epsilon})$, but there is no restriction on the number of processors. Another MapReduce model is introduced by Goodrich et al. (2011), where again the maximum size of each reducer is a parameter.

Lower bounds for the single round MapReduce model are first discussed by Afrati et al. (2012), who derive an interesting tradeoff between reducer size and replication rate. This is nicely illustrated by Ullman's drug interaction example (Ullman 2012). There are n (= 6,500) drugs, each consisting of about 1MB of data about patients who took that drug, and one has to find all drug interactions by applying a user defined function (UDF) to all pairs of drugs. To see the tradeoffs, it helps to simplify the example, by assuming we are given *two* sets, each of size n, and we have to apply a UDF to every pair of items, one from each set, in effect computing their Cartesian product. There are two extreme ways to solve this. One can use n^2 reducers, one for each pair of items; while each reducer has size 2, this approach is impractical because all of the data are replicated n times. At the other extreme one can use a single reducer that handles the entire data; the replication rate is 1, but the size of the reducer is $2n$, which is also impractical. As a tradeoff, partition each set into g groups of size n/g, and use one reducer for each of the g^2 pairs of groups: The size of a reducer is $2n/g$, while the replication rate is g. Thus, there is a tradeoff between the replication rate and the reducer size, which was also shown to hold for several other classes of problems (Afrati et al. 2012).

There are two significant limitations of this prior work: (1) As powerful and as convenient as the MapReduce framework is, the operations it provides may not be able to take full advantage of the resource constraints of modern systems. The lower bounds say nothing about alternative ways of structuring the computation that send and receive the same amount of data per step. (2) Even within the MapReduce framework, the only lower bounds apply to a single communication round, and say nothing about the limitations of multi-round MapReduce algorithms.

While it is convenient that MapReduce hides the number of servers from the programmer, when considering the most efficient way to use resources to solve problems it is natural to expose information about those resources to the programmer. In this article, we take the view that the number of servers p should be an explicit parameter of the model, which allows us to focus on the tradeoff between the amount of communication and the number of rounds. For example, going back to our Cartesian product problem, if the number of servers p is known, there is one optimal way to solve the problem: partition each of the two sets into $g = \sqrt{p}$ groups, and let each server handle one pair of groups.

A model with p as explicit parameter was proposed by Koutris and Suciu (2011), who showed both lower and upper bounds for one round of communication. In this model only tuples are sent and they must be routed independent of each other. For example, Koutris and Suciu (2011) proves that multi-joins on the same attribute can be computed in one round, while multi-joins on different attributes, like $R(x), S(x, y), T(y)$ require strictly more than one round. The study was mostly focused on understanding data skew, the model was limited, and the results do not apply to more than one round.

The MPC model we introduce in this article is more general than the above models, allowing arbitrary bits to represent communicated data, rather than just tuples, and unbounded computing power of servers so the lower bounds we show for it apply more broadly. Moreover, we establish lower bounds that hold even in the absence of skew.

Other Parallel Models. The prior parallel model that is closest to the MPC model is Valiant's Bulk Synchronous Parallel (BSP) model (Valiant 1990). The BSP model operates in synchronous rounds of supersteps consisting of possibly asynchronous steps. In addition to the number of processors, there is a superstep size, L, there is the notion of an h-relation, a mapping in which each processor sends and receives at most h bits, as well as an architecture-dependent bandwidth parameter g

that says that a superstep has to have at least gh steps in order for the processors to deliver an h-relation during a superstep. In the MPC model, the notion of load L largely parallels the notion of h-relation (though we technically only need to bound the number of bits each processor receives to obtain our lower bounds) but the other parameters are irrelevant, because we strengthen the model to allow unbounded local computation and hence the only notion of time in the model is the number of synchronous rounds (supersteps).

The finer-grained LogP model (Culler et al. 1996) does away with the synchronization barriers and the notion of h-relations inherent in the BSP model and has a more continuous notion of relaxed synchrony based on a latency bound and bound on processor overhead for setting up communication rather than based on supersteps. While its finer grain computation and relaxed asynchrony allowed tighter modeling of a number of parallel architectures, it seems less well matched to system architectures for MapReduce-style computations than either the BSP or MPC models.

Communication Complexity. The results we show belong to the study of communication complexity, for which there is a very large body of existing research (Kushilevitz and Nisan 1997). Communication complexity considers the number of bits that need to be communicated between cooperating agents to solve computational problems when the agents have unlimited computational power. Our model is related to the so-called number-in-hand multi-party communication complexity, in which there are multiple agents and no shared information at the start of communication. This has already been shown to be important to understanding the processing of massive data: Analysis of number-in-hand (NIH) communication complexity has been the main method for obtaining lower bounds on the space required for datastream algorithms (e.g., Alon et al. (1999)).

However, there is something very different about the results that we prove here. In almost all prior lower bounds, there is at least one agent that has access to all communication between agents.[11] (Typically, this is either via a shared blackboard to which all agents have access or a referee who receives all communication.) In this case, no problem on N bits whose answer is M bits long can be shown to require more than $N + M$ bits of communication. In our MPC model, all communication between servers is *private* and we restrict the communication per processor per step, rather than the total communication. Indeed, the privacy of communication is essential to our lower bounds, since we prove lower bounds that apply when the total communication is much larger than $N + M$. (Our lower bounds for some problems apply when the total communication is as large as $N^{1+\delta}$.)

Follow-up Work. Several recent works follow up on the results in this article. In Koutris et al. (2016) and Ketsman and Suciu (2017) the authors construct parallel algorithms in the MPC model that are *worst-case optimal*, that is, match the maximum load of the worst behaved data distribution. They show that for every full conjunctive query q, there exists a data distribution of input size M for which every MPC algorithm has load $\Omega(M/p^{1/\rho^*})$, where ρ^* is the minimum fractional edge cover of the query q. Moreover, for binary conjunctive queries where the relations have equal size it is shown that there exists an algorithm that achieves load $\tilde{O}(M/p^{1/\rho^*})$. In Afrati et al. (2017), the authors propose a multi-round algorithm for acyclic conjunctive queries where the load depends not only on the input but also the output size.

[11] Though private-messages models have been defined before, we are aware of only two lines of work where lower bounds make use of the fact that no single agent has access to all communication: (1) Results of Gál and Gopalan (2007) and Guha and Huang (2009) use the assumption that communication is both private and (multi-pass) one-way, but unlike the bounds we prove here, their lower bounds are smaller than the total input size; (2) Tiwari (1987) defined a distributed model of communication complexity in networks in which in input is given to two processors that communicate privately using other helper processors. However, this model is equivalent to ordinary public two-party communication when the network allows direct private communication between any two processors, as our model does.

The Hypercube algorithm was also implemented as part of massively parallel systems in Chu et al. (2015) and Vitorovic et al. (2016) and compared against more traditional parallel query plans.

7 CONCLUSION

In this article, we introduce a simple but powerful model, the MPC model, that allows us to analyze query processing in massively parallel systems. The MPC model captures two important parameters: the number of communication rounds and the maximum load that a server receives during the computation. We prove the first tight upper and lower bounds for the maximum load in the case of one communication round and input data without skew. Then, we show how to handle skew for several classes of queries. Finally, we analyze the precise tradeoff between the number of rounds and maximum load for the case of multiple rounds.

Our work leaves open many interesting questions. The analysis for multiple rounds works for a limited class of inputs, since we have to assume that all relations have the same size. Further, our lower bounds are (almost) tight only for a specific class of queries (treelike queries), so it remains open how we can obtain lower bounds for any conjunctive query, where relations have different size.

The effect of data skew in parallel computation is another exciting research direction. Although we have some understanding on how to handle skew in a single round, it is an open how skew influences computation when we have multiple rounds and what are the tradeoffs we can obtain in such cases.

REFERENCES

Foto N. Afrati, Manas R. Joglekar, Christopher Ré, Semih Salihoglu, and Jeffrey D. Ullman. 2017. GYM: A multiround distributed join algorithm. In *Proceedings of the 20th International Conference on Database Theory (ICDT'17)*. 4:1–4:18. DOI: http://dx.doi.org/10.4230/LIPIcs.ICDT.2017.4

F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. 2012. Upper and lower bounds on the cost of a map-reduce computation. *CoRR* abs/1206.4377 (2012).

F. N. Afrati and J. D. Ullman. 2010. Optimizing joins in a map-reduce environment. In *EDBT*. 99–110.

N. Alon, Y. Matias, and M. Szegedy. 1999. The space complexity of approximating the frequency moments. *JCSS* 58, 1 (1999), 137–147.

A. Atserias, M. Grohe, and D. Marx. 2008. Size bounds and query plans for relational joins. In *FOCS*. 739–748.

Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'13)*, Richard Hull and Wenfei Fan (Eds.). ACM, 273–284. DOI: http://dx.doi.org/10.1145/2463664.2465224

Paul Beame, Paraschos Koutris, and Dan Suciu. 2014. Skew in parallel query processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'14)*, Richard Hulland Martin Grohe (Eds.). ACM, 212–223. DOI: http://dx.doi.org/10.1145/2594538.2594558

Johann Brault-Baron. 2016. Hypergraph acyclicity revisited. *ACM Comput. Surv.* 49, 3 (2016), 54:1–54:26. http://doi.acm.org/10.1145/2983573

S. Chaudhuri. 2012. What next?: A half-dozen data management research goals for big data and the cloud. In *PODS*. 1–4.

Shumo Chu, Magdalena Balazinska, and Dan Suciu. 2015. From theory to practice: Efficient join query evaluation in a parallel database system. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 63–78. DOI: http://dx.doi.org/10.1145/2723372.2750545

David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Eunice E. Santos, Klaus E. Schauser, Ramesh Subramonian, and Thorsten von Eicken. 1996. LogP: A practical model of parallel computation. *Commun. ACM* 39, 11 (1996), 78–85. DOI: http://dx.doi.org/10.1145/240455.240477

J. Dean and S. Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *OSDI*. 137–150.

EMC Corporation. Data Science Revealed: A Data-Driven Glimpse into the Burgeoning New Field. Retrieved from http://www.emc.com/collateral/about/news/emc-data-science-study-wp.pdf.

J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. 2010. On distributing symmetric streaming computations. *ACM Transactions on Algorithms* 6, 4 (2010), 66:1–66:19.

E. Friedgut. 2004. Hypergraphs, entropy, and inequalities. *Am. Math. Monthly* 111, 9 (2004), 749–760.

A. Gál and P. Gopalan. 2007. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *FOCS*. 294–304.

S. Ganguly, A. Silberschatz, and S. Tsur. 1992. Parallel bottom-up processing of datalog queries. *J. Log. Program.* 14, 1&2 (1992), 101–126.

Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, searching, and simulation in the mapreduce framework. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC'11)*. 374–383. DOI: http://dx.doi.org/10.1007/978-3-642-25591-5_39

Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. 2009. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM* 56, 6, Article 30 (Sept. 2009), 32 pages. DOI: http://dx.doi.org/10.1145/1568318.1568320

M. Grohe and D. Marx. 2006. Constraint solving via fractional edge covers. In *SODA*. 289–298.

Sudipto Guha and Zhiyi Huang. 2009. Revisiting the direct sum theorem and space lower bounds in random order streams. In *ICALP (LNCS)*, Vol. 5555. Springer, 513–524.

Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. 2014. Demonstration of the myria big data management service. In *Proceedings of the International Conference on Management of Data (SIGMOD'14)*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 881–884. DOI: http://dx.doi.org/10.1145/2588555.2594530

Russell Impagliazzo and Valentine Kabanets. 2010. Constructive proofs of concentration bounds. In *Proceedings, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 14th International Workshop, RANDOM (Lecture Notes in Computer Science)*, Vol. 6302. Springer, 617–631.

H. J. Karloff, S. Suri, and S. Vassilvitskii. 2010. A model of computation for mapreduce. In *SODA*. 938–948.

Bas Ketsman and Dan Suciu. 2017. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'17)*. 417–428. DOI: http://dx.doi.org/10.1145/3034786.3034788

Paraschos Koutris, Paul Beame, and Dan Suciu. 2016. Worst-case optimal algorithms for parallel query processing. In *Proceedings of the 19th International Conference on Database Theory (ICDT'16)*. 8:1–8:18. DOI: http://dx.doi.org/10.4230/LIPIcs.ICDT.2016.8

P. Koutris and D. Suciu. 2011. Parallel evaluation of conjunctive queries. In *PODS*. 223–234.

E. Kushilevitz and N. Nisan. 1997. *Communication Complexity*. Cambridge University Press, Cambridge, England; New York.

YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome A. Rolia. 2012. SkewTune: Mitigating skew in mapreduce applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*. 25–36. DOI: http://dx.doi.org/10.1145/2213836.2213840

S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. 2010. Dremel: Interactive analysis of web-scale datasets. *PVLDB* 3, 1 (2010), 330–339.

H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. 2012. Worst-case optimal join algorithms: (Extended abstract). In *PODS*. 37–48.

C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. 2008. Pig latin: A not-so-foreign language for data processing. In *SIGMOD Conference*. 1099–1110.

Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. 2012. Space-round tradeoffs for mapreduce computations. In *Proceedings of the 26th ACM International Conference on Supercomputing (ICS'12)*. ACM, New York, NY, 235–244. DOI: http://dx.doi.org/10.1145/2304576.2304607

Spark. Apache Spark. http://spark.apache.org/.

Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *WWW*. 607–614.

A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. 2009. Hive - A warehousing solution over a map-reduce framework. *PVLDB* 2, 2 (2009), 1626–1629.

P. Tiwari. 1987. Lower bounds on communication complexity in distributed computer networks. *JACM* 34, 4 (Oct. 1987), 921–938.

J. D. Ullman. 2012. Designing good mapreduce algorithms. *ACM Crossroads* 19, 1 (2012), 30–34.

Leslie G. Valiant. 1990. A bridging model for parallel computation. *Commun. ACM* 33, 8 (1990), 103–111. DOI: http://dx.doi.org/10.1145/79173.79181

Aleksandar Vitorovic, Mohammed Elseidy, Khayyam Guliyev, Khue Vu Minh, Daniel Espino, Mohammad Dashti, Yannis Klonatos, and Christoph Koch. 2016. Squall: Scalable real-time analytics. *PVLDB* 9, 13 (2016), 1553–1556.

A. C. Yao. 1977. Probabilistic computations: Toward a unified measure of complexity. In *FOCS*. 222–227.

Received December 2015; revised June 2017; accepted July 2017