Communication Steps for Parallel Query Processing*

Paul Beame, Paraschos Koutris and Dan Suciu
University of Washington, Seattle, WA
{beame,pkoutris,suciu}@cs.washington.edu

ABSTRACT

We consider the problem of computing a relational query q on a large input database of size n, using a large number p of servers. The computation is performed in *rounds*, and each server can receive only $O(n/p^{1-\varepsilon})$ bits of data, where $\varepsilon \in [0,1]$ is a parameter that controls replication. We examine how many global communication steps are needed to compute q. We establish both lower and upper bounds, in two settings. For a single round of communication, we give lower bounds in the strongest possible model, where arbitrary bits may be exchanged; we show that any algorithm requires $\varepsilon \geq 1-1/\tau^*$, where τ^* is the fractional vertex cover of the hypergraph of q. We also give an algorithm that matches the lower bound for a specific class of databases. For multiple rounds of communication, we present lower bounds in a model where routing decisions for a tuple are tuple-based. We show that for the class of *tree-like* queries there exists a tradeoff between the number of rounds and the space exponent ε. The lower bounds for multiple rounds are the first of their kind. Our results also imply that transitive closure cannot be computed in $O(1)$ rounds of communication.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Parallel Databases

Keywords

Parallel Computation, Lower Bounds

1. INTRODUCTION

Most of the time spent in 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 [5, 8]. These tasks form the main bottleneck in big data analysis, and a major challenge for the database community is improving the performance and usability of data processing tools. The motivation for this paper comes from the need to understand the complexity of query processing in big data management.

Query processing is typically performed on a shared-nothing parallel architecture. In this setting, the data is stored on a large number of independent servers interconnected by a fast network. The servers perform local computations, then exchange data in global data shuffling steps. This model of computation has been popularized by MapReduce [7] and Hadoop [15], and can be found in most big data processing systems, like PigLatin [21], Hive [23], Dremmel [19].

Unlike traditional query processing, the 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 the main memory of these servers. The new complexity bottleneck is the communication. Typical network speeds in large clusters are 1Gb/s, which is significantly lower than main memory access. In addition, any data reshuffling requires a global synchronization of all servers, which also comes at significant cost; for example, everyone needs to wait for the slowest server, and, worse, in the case of a straggler, or a local node failure, everyone must wait for the full recovery. Thus, the dominating complexity parameters in big data query processing are the number of communication steps, and the amount of data being exchanged.

MapReduce-related models.

Several computation models have been proposed in order to understand the power of MapReduce and related massively parallel programming methods [9, 16, 17, 1]. These all identify the number of communication steps/rounds as a main complexity parameter, but differ in their treatment of the communication.

The first of these models was the MUD (Massive, Unordered, Distributed) model of Feldman et al. [9]. 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 paper 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. [16] 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

*This work was partially supported by NSF IIS-1115188, IIS-0915054 and IIS-1247469.

PODS'13, June 22–27, 2013, New York, New York, USA.

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.

Lower bounds for the single round MapReduce model are first discussed by Afrati et al. [1], who derive an interesting tradeoff between reducer size and replication rate. This is nicely illustrated by Ullman's drug interaction example [25]. 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 the entire data is 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 [1].

Towards lower bound models.

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 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 paper, 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 [17], 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, [17] 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.

In this paper we develop more general models, establish lower bounds that hold even in the absence of skew, and use a bit model, rather than a tuple model, to represent data.

Our lower bound models and results.

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 allow each server to be infinitely powerful, subject only to the data to which it has access. The model requires that each server receives only $O(N/p^{1-\varepsilon})$ bits of data at any step, where N is the problem size, and $\varepsilon \in [0,1]$ is a parameter of the model. This implies that the replication factor is $O(p^{\varepsilon})$ per round. A particularly natural case is $\varepsilon = 0$, which corresponds to a replication factor of $O(1)$, or $O(N/p)$ bits per server; $\varepsilon = 1$ is degenerate, since it allows the entire data to be sent to every server.

We establish both lower and upper bounds for computing a full conjunctive query q, in two settings. First, we restrict the computation to a single communication round and examine the minimum parameter ε for which it is possible to compute q with $O(N/p^{1-\varepsilon})$ bits per processor; we call this the *space exponent*. We show that the space exponent for connected queries is always at least $1 - 1/\tau^*(q)$, where $\tau^*(q)$ is the *fractional (vertex) covering number* of the hypergraph associated with q [6], which is the optimal value of the vertex cover linear program (LP) for that hypergraph. This 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 [1, 17], which assume that the units being exchanged are tuples.

Our one round lower bound holds even in the special case of *matching databases*, when all attributes are from the same domain $[n]$ and all input relations are (hypergraph) matchings, in other words, every relation has exactly n tuples, and every attribute contains every value $1, 2, \ldots, n$ exactly once. Thus, the lower bound holds even in a case in which there is no data skew. We describe a simple tuple-independent algorithm that is easily implementable in the MapReduce framework, which, in the special case of matching databases, matches our lower bound for any conjunctive query. The algorithm uses the optimal solution for the fractional vertex cover to find an optimal split of the input data to the servers. For example, the linear query $L_2 = S_1(x,y), S_2(y,z)$ has an optimal vertex cover $0, 1, 0$ (for the variables x, y, z), hence its space exponent is $\varepsilon = 0$, whereas the cycle query $C_3 = S_1(x,y), S_2(y,z), S_3(z,x)$ has optimal vertex cover $1/2, 1/2, 1/2$ and space exponent $\varepsilon = 1/3$. We note that recent work [13, 4, 20] gives upper bounds on the query size in terms of a fractional *edge cover*, while our results are in terms of the *vertex cover*. Thus, our first result is:

THEOREM 1.1. *For every connected conjunctive query q, any p-processor randomized MPC algorithm computing q in one round requires space exponent $\epsilon \geq 1 - 1/\tau^*(q)$. This lower bound holds even over matching databases, for which it is optimal.*

Second, we establish lower bounds for multiple communication steps, 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 in-

put 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 note that any multi-step MapReduce program is tuple-based, because in any map function the key of the intermediate value depends only on the input tuple to the map function. 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 for the given ε. For example, to compute a length k chain query L_k, if $\varepsilon = 0$, 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 $\varepsilon = 1/2$, then we can use L_4 as operator (a four-way join), and the optimal number of rounds is $\log_4 k$. More generally, we can show nearly matching upper and lower bounds based on graph-theoretic properties of the query such as the following:

THEOREM 1.2. *For space exponent ε, the number of rounds required for any tuple-based MPC algorithm to compute any tree-like conjunctive query q is at least $\lceil \log_{k_\varepsilon}(diam(q)) \rceil$ where $k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$ and $diam(q)$ is the diameter of q. Moreover, for any connected conjunctive query q, this lower bound is nearly matched (up to a difference of essentially one round) by a tuple-based MPC algorithm with space exponent ε.*

These are the first lower bounds that apply to multiple rounds of MapReduce. Both lower bounds in Theorem 1.1 and Theorem 1.2 are stated in a strong form: we show that any algorithm on the MPC model retrieves only a $1/p^{\Omega(1)}$ fraction of the answers to the query in expectation, when the inputs are drawn uniformly at random (the exponent depends on the query and on ε); Yao's Lemma [26] immediately implies a lower bound for any randomized algorithm over worst-case inputs. Notice that the fraction of answers gets worse as the number of servers p increases. In other words, the more parallelism we want, the worse an algorithm performs, if the number of communication rounds is bounded.

Related work in communication complexity.

The results we show belong to the study of communication complexity, for which there is a very large body of existing research [18]. Communication complexity considers the number of bits that need to be communicated between cooperating agents in order to solve computational problems when the agents have unlimited computational power. Our model is related to the so-called number-in-hand multiparty 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 data stream algorithms (e.g. [3]).

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[1]. (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 $N^{1+\delta}$.)

2. PRELIMINARIES

2.1 Massively Parallel Communication

We fix a parameter $\varepsilon \in [0, 1]$, called the *space exponent*, and define the MPC(ε) model as follows. The computation is performed by p servers, called *workers*, connected by a complete network of private channels. The input data has size N bits, and is initially distributed evenly among the p workers. The computation proceeds in rounds, where each round consists of local computation at the workers interleaved with global communication. The complexity is measured in the number of communication rounds. The servers have unlimited computational power, but there is one important restriction: at each round, a worker may receive a total of only $O(N/p^{1-\varepsilon})$ bits of data from all other workers combined. Our goal is to find lower and upper bounds on the number of communication rounds.

The space exponent represents the degree of replication during communication; in each round, the total amount of data exchanged is $O(p^\varepsilon)$ times the size of the input data. When $\varepsilon = 0$, there is no replication, and we call this the basic MPC model. The case $\varepsilon = 1$ is degenerate because each server can receive the entire data, and any problem can be solved in a single round. Similarly, for any fixed ε, if we allow the computation to run for $\Theta(p^{1-\varepsilon})$ rounds, the entire data can be sent to every server and the model is again degenerate.

We denote M^r_{uv} the message sent by server u to server v during round r and denote $M^r_v = (M^{r-1}_v, (M^r_{1v}, \ldots, M^r_{pv}))$ the concatenation of all messages sent to v up to round r. Assuming $O(1)$ rounds, each message M^r_v holds $O(N/p^{1-\varepsilon})$ bits. For our multi-round lower bounds in Section 4, we will further restrict what the workers can encode in the messages M^r_{uv} during rounds $r \geq 2$.

2.2 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

[1]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 [11, 14] 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 [24] 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.

abort the computation if the amount of data received during a communication would exceed the $O(N/p^{1-\varepsilon})$ limit, but this will only happen with exponentially small probability.

To prove lower bounds for randomized algorithms, we use Yao's Lemma [26]. 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.

2.3 Conjunctive Queries

In this paper we consider a particular class of problems for the MPC model, namely computing answers to conjunctive queries over an input database. We fix an input vocabulary $S_1, \ldots, S_\ell$, where each relation S_j has a fixed arity r_j; we denote $r = \sum_{j=1}^{\ell} r_j$. The input data consists of one relation instance for each symbol. We denote n the largest number of tuples in any relation S_j; then, the entire database instance can be encoded using $N = O(n \log n)$ bits, because $\ell = O(1)$ and $r_j = O(1)$ for $j = 1, \ldots, \ell$.

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

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

The query is *full*, meaning that every variable in the body appears 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 *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). 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 (1). The *connected components* of q are the maximal connected subqueries of q. Table 1 illustrates example queries used throughout this paper.

We consider two query evaluation problems. In Join-Reporting, we require that all tuples in the relation defined by q be produced. In Join-Witness, we require the production of at least one tuple in the relation defined by q, if one exists; Join-Witness is the verified version of the natural decision problem Join-NonEmptiness.

Characteristic of a Query.

The *characteristic* of a conjunctive query q as in (1) is defined as $\chi(q) = k + \ell - \sum_j r_j - c$, where k is the number of variables, ℓ is the number of atoms, r_j is the arity of atom S_j, and c is the number of connected components of q.

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, for the query L_5 in Table 1, $L_5/\{S_2, S_4\} = S_1(x_1, x_2), S_3(x_2, x_4), S_5(x_4, x_6)$.

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 $M \subseteq \text{atoms}(q)$, $\chi(q/M) = \chi(q) - \chi(M)$.
(c) $\chi(q) \leq 0$.
(d) For any $M \subseteq \text{atoms}(q)$, $\chi(q) \leq \chi(q/M)$.

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 r_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 $\ell_M - r_M$ for a total reduction of $k_M + \ell_M - r_M - 1 = \chi(M)$. Thus, property (b) follows.

By property (a), it suffices to prove (c) when q is connected. If q is a single atom then $\chi(q) \leq 0$, since the number of variables is at most the arity of the atom in q. We reduce to this case by repeatedly contracting the atoms of q until only one remains and showing that $\chi(q) \leq \chi(q/S_j)$: Let $m \leq r_j$ be the number of distinct variables in atom S_j. Then, $\chi(q/S_j) = (\ell - 1) + (k - m + 1) - (r - r_j) - 1 = \chi(q) + (r_j - m) \geq \chi(q)$. Property (d) also follows by the combination of property (b) and property (c) applied to M. □

Finally, let us call a query q *tree-like* if q is connected and $\chi(q) = 0$. For example, the query L_k is tree-like, and so is any query over a binary vocabulary whose graph is a tree. Over non-binary vocabularies, any tree-like query is acyclic, but the converse does not hold: $q = S_1(x_0, x_1, x_2), S_2(x_1, x_2, x_3)$ is acyclic but not tree-like. An important property of tree-like queries is that every connected subquery will be also tree-like.

Vertex Cover and Edge Packing.

A *fractional vertex cover* of a query q is any feasible solution of the LP shown on the left of Fig. 1. The vertex cover associates a non-negative number u_i to each variable x_i s.t. every atom S_j is "covered", $\sum_{i:x_i \in \text{vars}(S_j)} v_i \geq 1$. The dual LP corresponds to a *fractional edge packing* problem (also known as a *fractional matching* problem), which associates non-negative numbers u_j to each atom S_j. The two LPs have the same optimal value of the objective function, known as the *fractional covering number* [6] of the hypergraph associated with q and denoted by $\tau^*(q)$. Thus, $\tau^*(q) = \min \sum_i v_i = \max \sum_j u_j$. Additionally, if all inequalities are satisfied as equalities by a solution to the LP, we say that the solution is *tight*.

For a simple example, a fractional vertex cover of the query[2] $L_3 = S_1(x_1, x_2), S_2(x_2, x_3), S_3(x_3, x_4)$ is any solution to $v_1 + v_2 \geq 1$, $v_2 + v_3 \geq 1$ and $v_3 + v_4 \geq 1$; the optimal

[2]We drop the head variables when clear from the context.

Vertex Covering LP	Edge Packing LP
$\forall j \in [\ell]$: $\sum_{i:x_i \in \text{vars}(S_j)} v_i \geq 1$ (2) $\forall i \in [k] : v_i \geq 0$	$\forall i \in [k]$: $\sum_{j:x_i \in \text{vars}(S_j)} u_j \leq 1$ (3) $\forall j \in [\ell] : u_j \geq 0$
minimize $\sum_{i=1}^{k} v_i$	maximize $\sum_{j=1}^{\ell} u_j$

Figure 1: The vertex covering LP of the hypergraph of a query q, and its dual edge packing LP.

is achieved by $(v_1, v_2, v_3, v_4) = (0, 1, 1, 0)$, which is not tight. An edge packing is a solution to $u_1 \leq 1$, $u_1 + u_2 \leq 1$, $u_2 + u_3 \leq 1$ and $u_3 \leq 1$, and the optimal is achieved by $(1, 0, 1)$, which is tight.

The fractional edge *packing* should not be confused with the fractional edge *cover*, which has been used recently in several papers to prove bounds on query size and the running time of a sequential algorithm for the query [4, 20]; for the results in this paper we need the fractional packing. The two notions coincide, however, when they are tight.

2.4 Input Servers

We assume that, at the beginning of the algorithm, each relation S_j is stored on a separate server, called an *input server*, which during the first round sends a message M_{ju}^1 to every worker u. After the first round, the input servers are no longer used in the computation. All lower bounds in this paper assume that the relations S_j are given on separate input servers. All upper bounds hold for either model.

The lower bounds for the model with separate input servers carry over immediately to the standard MPC model, because any algorithm in the standard model can be simulated in the model with separate input servers. Indeed, the algorithm must compute the output correctly for any initial distribution of the input data on the p servers: we simply choose to distribute the input relations $S_1, \ldots, S_\ell$ such that the first p/ℓ servers receive S_1, the next p/ℓ servers receive S_2, etc., then simulate the algorithm in the model with separate input servers (see [17, proof of Proposition 3.5] for a detailed discussion). Thus, it suffices to prove our lower bounds assuming that each input relation is stored on a separate input server. In fact, this model is even more powerful, because an input server has now access to the entire relation S_j, and can therefore perform some global computation on S_j, for example compute statistics, find outliers, etc., which are common in practice.

2.5 Input Distribution

We find it useful to consider input databases of the following form that we call a *matching* database: The domain of the input database will be $[n]$, for $n > 0$. In such a database each relation S_j is an *r_j-dimensional matching*, where r_j is its arity. In other words, S_j has exactly n tuples and each of its columns contains exactly the values $1, 2, \ldots, n$; each attribute of S_j is a key. For example, if S_j is binary, then an instance of S_j is a permutation on $[n]$; if S_j is ternary then an instance consists of n node-disjoint triangles. Moreover, the answer to a connected conjunctive query q on a matching database is a table where each attribute is a key, because we have assumed that q is full; in particular, the output to q has at most n tuples. In our lower bounds we assume that a matching database is randomly chosen with uniform probability, for a fixed n.

Matching databases are database instances *without skew*. By stating our lower bounds on matching databases we make them even stronger, because they imply that a query cannot be computed even in the absence of skew; of course, the lower bounds also hold for arbitrary instances. Our upper bounds, however, hold only on matching databases. Data skew is a known problem in parallel processing, and requires dedicated techniques. Lower and upper bounds accounting for the presence of skew are discussed in [17].

2.6 Friedgut's Inequality

Friedgut [10] introduces the following class of inequalities. Each inequality is described by a hypergraph, which in our paper corresponds to a query, so we will describe the inequality using query terminology. Fix a query q as in (1), and let $n > 0$. For every atom $S_j(\bar{x}_j)$ of arity r_j, we introduce a set of n^{r_j} variables $w_j(\mathbf{a}_j) \geq 0$, where $\mathbf{a}_j \in [n]^{r_j}$. If $\mathbf{a} \in [n]^r$, we denote by $\mathbf{a}_j$ the vector of size r_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:

$$\sum_{\mathbf{a}\in[n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \leq \prod_{j=1}^{\ell} \left(\sum_{\mathbf{a}_j \in [n]^{r_j}} w_j(\mathbf{a}_j)^{1/u_j} \right)^{u_j} \quad (4)$$

We illustrate Friedgut's inequality on C_3 and L_3:

$$C_3(x, y, z) = S_1(x, y), S_2(y, z), S_3(z, x)$$
$$L_3(x, y, z, w) = S_1(x, y), S_2(y, z), S_3(z, w) \quad (5)$$

C_3 has cover $(1/2, 1/2, 1/2)$, and L_3 has cover $(1, 0, 1)$. Thus, we obtain the following inequalities, where a, b, c stand for w_1, w_2, w_3 respectively:

$$\sum_{x,y,z\in[n]} a_{xy} \cdot b_{yz} \cdot c_{zx} \leq \sqrt{\sum_{x,y\in[n]} a_{xy}^2 \sum_{y,z\in[n]} b_{yz}^2 \sum_{z,x\in[n]} c_{zx}^2}$$

$$\sum_{x,y,z,w\in[n]} a_{xy} \cdot b_{yz} \cdot c_{zw} \leq \sum_{x,y\in[n]} a_{xy} \cdot \max_{y,z\in[n]} b_{yz} \cdot \sum_{z,w\in[n]} c_{zw}$$

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

Friedgut's inequalities immediately imply a well known result developed in a series of papers [13, 4, 20] 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 $a_{xy} = 1$ if $(x, y) \in S_1$, otherwise $a_{xy} = 0$ (and similarly for b_{yz}, c_{zx}). We obtain then $|C_3| \leq \sqrt{|S_1| \cdot |S_2| \cdot |S_3|}$. Note that all these results are expressed in terms of a fractional edge *cover*. When we apply Friedgut's inequality in Section 3.2 to a fractional edge *packing*, we ensure that the packing is tight.

3. ONE COMMUNICATION STEP

Let the *space exponent* of a query q be the smallest $\varepsilon \geq 0$ for which q can be computed using one communication step in the MPC(ε) model. In this section, we prove Theorem 1.1, which gives both a general lower bound on the space exponent for evaluating connected conjunctive queries and a precise characterization of the space exponent for evaluating them them over matching databases. The proof consists of two parts: we show the optimal algorithm in 3.1, and then present the matching lower bound in 3.2.

3.1 An Algorithm for One Round

We describe here an algorithm, which we call HyperCube (HC), that computes a conjunctive query in one step. It uses ideas that can be traced back to Ganguly [12] for parallel processing of Datalog programs, and were also used by Afrati and Ullman [2] to optimize joins in MapReduce, and by Suri and Vassilvitskii [22] to count triangles.

Let q be a query as in (1). Associate to each variable x_i a real value $e_i \geq 0$, called the *share exponent* of x_i, such that $\sum_{i=1}^k e_i = 1$. If p is the number of servers, define $p_i = p^{e_i}$: these values are called *shares* [2]. We assume that

Conjunctive Query	Expected answer size	Minimum Vertex Cover	Variable Shares	Value $\tau^*(q)$	Space Exponent
$C_k(x_1,\ldots,x_k) = \bigwedge_{j=1}^k S_j(x_j, x_{(j+1) \bmod k})$	1	$\frac{1}{2},\ldots,\frac{1}{2}$	$\frac{1}{k},\ldots,\frac{1}{k}$	$k/2$	$1-2/k$
$T_k(z,x_1,\ldots,x_k) = \bigwedge_{j=1}^k S_j(z,x_j)$	n	$1,0,\ldots,0$	$1,0,\ldots,0$	1	0
$L_k(x_0,x_1,\ldots,x_k) = \bigwedge_{j=1}^k S_j(x_{j-1},x_j)$	n	$0,1,0,1,\ldots$	$0,\frac{1}{\lceil k/2\rceil},0,\frac{1}{\lceil k/2\rceil},\ldots$	$\lceil k/2\rceil$	$1-1/\lceil k/2\rceil$
$B_{k,m}(x_1,\ldots,x_k) = \bigwedge_{I\subseteq[k],\lvert I\rvert=m} S_I(\bar{x}_I)$	$n^{k-(m-1)\binom{k}{m}}$	$\frac{1}{m},\ldots,\frac{1}{m}$	$\frac{1}{k},\ldots,\frac{1}{k}$	k/m	$1-m/k$

Table 1: Running examples in this paper: 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 out of the k head variables. Assuming the inputs are random permutation, the answer sizes represent exact values for L_k, T_k, and expected values for $C_k, B_{k,m}$.

the shares are integers. Thus, $p = \prod_{i=1}^k p_i$, and each server can be uniquely identified with a point in the k-dimensional hypercube $[p_1]\times\cdots\times[p_k]$.

The algorithm uses k independently chosen random hash functions $h_i : [n] \to [p_i]$, one for each variable x_i. During the communication step, the algorithm sends every tuple $S_j(\mathbf{a}_j) = S_j(a_{i_1},\ldots,a_{i_{r_j}})$ to all servers $\mathbf{y}\in[p_1]\times\cdots\times[p_k]$ such that $h_{i_m}(a_{i_m}) = \mathbf{y}_{i_m}$ for any $1\le m\le r_j$. In other words, the tuple $S_j(\mathbf{a}_j)$ knows the server number along the dimensions $i_1,\ldots,i_{r_j}$, but does not know the server number along the other dimensions, and there it needs to be replicated. After receiving the data, each server outputs all query answers derivable from the received data. The algorithm finds all answers, because each potential output tuple $(a_1,\ldots,a_k)$ is known by the server $\mathbf{y} = (h_1(a_1),\ldots,h_k(a_k))$.

Example 3.1. *We illustrate how to compute the 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 share exponents $e_1 = e_2 = e_3 = 1/3$. 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 $S_1(a_1,a_2)$ to all servers with index $(h_1(a_1),h_2(a_2),y_3)$, 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 $S_1(a_1,a_2)$, $S_2(a_2,a_3)$, $S_3(a_3,a_1)$ that contribute to the output tuple $C_3(a_1,a_2,a_3)$ will be seen by the server $\mathbf{y} = (h_1(a_1),h_2(a_2),h_3(a_3))$: any server that detects three matching tuples outputs them.*

Proposition 3.2. *Fix a fractional vertex cover $\mathbf{v} = (v_1,\ldots,v_k)$ for a connected conjunctive query q, and let $\tau = \sum_i v_i$. The HC algorithm with share exponents $e_i = v_i/\tau$ computes q on any matching database in one round in $MPC(\varepsilon)$, where $\varepsilon = 1-1/\tau$, with probability of failure $\eta \le exp(-O(n/p^{\varepsilon}))$.*

This proves the optimality claim of Theorem 1.1: choose a vertex cover with value $\tau^*(q)$, the fractional covering number of q. Proposition 3.2 shows that q can be computed in one round in $MPC(\varepsilon)$, with $\varepsilon = 1-1/\tau^*$.

Proof. Since $\mathbf{v}$ forms a fractional vertex cover, for every relation symbol S_j we have $\sum_{i:x_i\in\text{vars}(S_j)} e_i \ge 1/\tau$. Therefore, $\sum_{i:x_i\notin\text{vars}(S_j)} e_i \le 1-1/\tau$. Every tuple $S_j(\mathbf{a}_j)$ is replicated $\prod_{i:x_i\notin\text{vars}(S_j)} p_i \le p^{1-1/\tau}$ times. Thus, the total number of tuples that are received by all servers is $O(n\cdot p^{1-1/\tau})$. We claim that these tuples are uniformly distributed among the p servers: this proves the theorem, since then each server receives $O(n/p^{1/\tau})$ tuples.

To prove the claim, we note that for each tuple $t\in S_j$, the probability over the random choices of the hash functions $h_1,\ldots,h_k$ that the tuple is sent to server s is precisely $\prod_{i:x_i\in\text{vars}(S_j)} p_i^{-1}$. Thus, the expected number of tuples from S_j sent to s is $n/\prod_{i:x_i\in S_j} p_i \le n/p^{1-\varepsilon}$. Since S_j is an r_j-matching, different tuples are sent by the random hash functions to independent destinations, since any two tuples differ in every attribute. Using standard Chernoff bounds, we derive that the probability that the actual number of tuples per server deviates more than a constant factor from the expected number is $\eta \le \exp(-O(n/p^{1-\varepsilon}))$. □

3.2 A Lower Bound for One Round

For a fixed n, consider a probability distribution where the input I is chosen randomly, with uniform probability from all matching database instances. Let $\mathbf{E}[|q(I)|]$ denote the expected number of answers to the query q. We prove in this section:

Theorem 3.3. *Let q be a connected conjunctive query, let τ^* be the fractional covering number of q, and $\varepsilon < 1-1/\tau^*$. Then, any deterministic MPC(ε) algorithm that runs in one communication round on p servers reports $O(\mathbf{E}[|q(I)|]/p^{\tau^*(1-\varepsilon)-1})$ answers in expectation.*

In particular, the theorem implies that the space exponent of q is at least $1-1/\tau^*$. Before we prove the theorem, we show how to extend it to randomized algorithms using Yao's principle. For this, we show a lemma that we also need later.

Lemma 3.4. *The expected number of answers to connected query q is $\mathbf{E}[|q(I)|] = n^{1+\chi(q)}$, where the expectation is over a uniformly chosen matching database I.*

Proof. For any relation S_j, and any tuple $\mathbf{a}_j\in[n]^{r_j}$, the probability that S_j contains $\mathbf{a}_j$ is $\mathbf{P}(\mathbf{a_j}\in S_j) = n^{1-r_j}$. 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:

$$\mathbf{E}[|q(I)|] = \sum_{\mathbf{a}\in[n]^k} \mathbf{P}(\bigwedge_{j=1}^{\ell}(\mathbf{a}_j\in S_j))$$
$$= \sum_{\mathbf{a}\in[n]^k}\prod_{j=1}^{\ell}\mathbf{P}(\mathbf{a}_j\in S_j) = \sum_{\mathbf{a}\in[n]^k}\prod_{j=1}^{\ell} n^{1-r_j} = n^{k+\ell-r}$$

Since query q is connected, $k+\ell-r = 1+\chi(q)$ and hence $\mathbf{E}[|q(I)|] = n^{1+\chi(q)}$. □

Theorem 3.3 and Lemma 3.4, together with Yao's lemma, imply the following lower bound for randomized algorithms.

Corollary 3.5. *Let q be any connected conjunctive query. Any one round randomized MPC(ε) algorithm with $p=\omega(1)$ and $\varepsilon < 1-1/\tau^*(q)$ fails to compute q with probability $\eta = \Omega(n^{\chi(q)}) = n^{-O(1)}$.*

PROOF. Choose a matching database I input to q uniformly at random. Let $a(I)$ denote the set of correct answers returned by the algorithm on I: $a(I) \subseteq q(I)$. Observe that the algorithm fails on I iff $|q(I) - a(I)| > 0$.

Let $\gamma = 1/p^{\tau^*(q)(1-\varepsilon)-1}$. Since $p = \omega(1)$ and $\varepsilon < 1 - 1/\tau^*(q)$, it follows that $\gamma = o(1)$. By Theorem 3.3, for any deterministic one round MPC(ε) algorithm we have $\mathbf{E}[|a(I)|] = O(\gamma)\mathbf{E}[|q(I)|]$ and hence, by Lemma 3.4,

$$\mathbf{E}[|q(I) - a(I)|] = (1 - o(1))\mathbf{E}[|q(I)|] = (1 - o(1))n^{1+\chi(q)}$$

However, we also have that

$$\mathbf{E}[|q(I) - a(I)|] \leq \mathbf{P}[|q(I) - a(I)| > 0] \cdot \max_I |q(I) - a(I)|.$$

Since $|q(I) - a(I)| \leq |q(I)| \leq n$ for all I, we see that the failure probability of the algorithm for randomly chosen I, $\mathbf{P}[|q(I) - a(I)| > 0]$, is at least $\eta = (1 - o(1))n^{\chi(q)}$ which is $n^{-O(1)}$ for any q. Yao's lemma implies that every one round randomized MPC(ε) algorithm will fail to compute q with probability at least η on some matching database input. □

In the rest of the section we prove Theorem 3.3, which deals with one-round deterministic algorithms and random matching databases I. Let us fix some server and let $m(I)$ denote the function specifying the message the server receives on input I. Intuitively, this server can only report those tuples that it knows are in the input based on the value of $m(I)$. To make this notion precise, for any fixed value m of $m(I)$, define the set of tuples of a relation R of arity r *known* by the server given message m as

$$K_m(R) = \{t \in [n]^r \mid \text{ for all matching databases } I, \; m(I) = m \Rightarrow t \in R(I)\}$$

We will particularly apply this definition with $R = S_j$ and $R = q$. Clearly, an output tuple $\mathbf{a} \in K_m(q)$ iff for every j, $\mathbf{a}_j \in K_m(S_j)$, 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 $|K_m(S_j)|$ in Section 3.2.1. Then in Section 3.2.2 we use this bound, along with Friedgut's inequality, to establish an upper bound for $|K_m(q)|$ and hence prove Theorem 3.3.

3.2.1 Bounding the Knowledge of Each Relation

Fix a server, and an input relation S_j. We prove here:

LEMMA 3.6. $\mathbf{E}[|K_{m(I)}(S_j)|] = O(n/p^{1-\varepsilon})$ *for random* I.

Since S_j has exactly n tuples, the lemma says that any server knows, in expectation, only a fraction $f = O(1/p^{1-\varepsilon})$ of tuples from S_j. While $m = m(I)$ is the concatenation of ℓ messages, one for each input relation, $K_m(S_j)$ depends only on the part of the message corresponding to S_j, so we can assume w.l.o.g. that m is a function only of S_j, denoted by m_j. For convenience, we also drop the index j and write $S = S_j$,$r = r_j$, $m = m_j$; $m(S)$ is now a function computed on the single r-dimensional matching relation S.

Observe that for a randomly chosen matching database I, S is a uniformly chosen r-dimensional matching. There are precisely $(n!)^{r-1}$ different r-dimensional matchings on $[n]$ and, since q is of fixed total arity, the number of bits N necessary to represent the entire input I is $\Theta(\log(n!)) = \Theta(n \log n)$. Therefore, $m(S)$ is at most $O((n \log n)/p^{1-\varepsilon})$ bits long for all S.

LEMMA 3.7. *Suppose that for all r-dimensional matchings S, $m(S)$ is at most $f \cdot (r-1)\log(n!)$ bits long. Then $\mathbf{E}[|K_{m(S)}(S)|] \leq f \cdot n$, where the expectation is taken over random choices of the matching S.*

We observe that Lemma 3.6 is an immediate corollary of Lemma 3.7 by setting f to be $O(1/p^{1-\varepsilon})$.

PROOF. Let m be a possible value for $m(S)$. Since m fixes precisely $|K_m(S)|$ tuples of S,

$$\begin{aligned}\log|\{S \mid m(S) = m\}| &\leq (r-1)\textstyle\sum_{i=1}^{n-|K_m(S)|} \log i \\ &\leq (1 - |K_m(S)|/n)(r-1)\textstyle\sum_{i=1}^{n} \log i \\ &= (1 - |K_m(S)|/n)\log(n!)^{r-1}. \end{aligned} \quad (6)$$

We can bound the value we want by considering the binary entropy of the distribution S, $H(S) = \log(n!)^{r-1}$. By applying the chain rule for entropy, we have

$$\begin{aligned} H(S) &= H(m(S)) + \textstyle\sum_m \mathbf{P}(m(S) = m) \cdot H(S|m(S) = m) \\ &\leq f \cdot H(S) + \textstyle\sum_m \mathbf{P}(m(S) = m) \cdot H(S|m(S) = m) \\ &\leq f \cdot H(S) + \textstyle\sum_m \mathbf{P}(m(S) = m) \cdot (1 - |K_m(S)|/n)H(S) \\ &= f \cdot H(S) + (1 - \textstyle\sum_m \mathbf{P}(m(S) = m)\ |K_m(S)|/n)H(S) \\ &= f \cdot H(S) + (1 - \mathbf{E}[|K_{m(S)}(S)|]/n)H(S) \end{aligned} \quad (7)$$

where the first inequality follows from the assumed upper bound on $|m(S)|$, the second inequality follows by (6), and the last two lines follow by definition. Dividing both sides of (7) by $H(S)$ and rearranging we obtain that $\mathbf{E}|K_{m(S)}(S)|] \leq f \cdot n$, as required. □

3.2.2 Bounding the Knowledge of the Query

Here we conclude the proof of Theorem 3.3 using the results in the previous section. Let us fix some server. Lemma 3.6 implies that, for $f = c/p^{1-\varepsilon}$ for some constant c and randomly chosen matching database I, $\mathbf{E}[|K_{m_j(I)}(S_j)|] = \mathbf{E}[|K_{m_j(S_j)}(S_j)|] \leq f \cdot n$ for all $j \in [\ell]$. We prove:

LEMMA 3.8. $\mathbf{E}[|K_{m(I)}(q)|] \leq f^{\tau^*(q)} n^{1+\chi(q)}$ *for randomly chosen matching database* I.

This proves Theorem 3.3, since the total number of tuples known by all p servers is bounded by:

$$\begin{aligned} p \cdot \mathbf{E}[|K_{m(I)}(q)|] &\leq p \cdot f^{\tau^*(q)}\mathbf{E}[|q(I)|] \\ &= p \cdot c^{\tau^*(q)} \cdot \mathbf{E}[|q(I)|]/p^{(1-\varepsilon)\tau^*(q)} \end{aligned}$$

which is the upper bound in Theorem 3.3 since c and $\tau^*(q)$ are constants. In the rest of the section we prove Lemma 3.8.

We start with some notation. For $\mathbf{a}_j \in [n]^{r_j}$, let $w_j(\mathbf{a}_j)$ denote the probability that the server knows the tuple $\mathbf{a}_j$. In other words $w_j(\mathbf{a}_j) = \mathbf{P}(\mathbf{a}_j \in K_{m_j(S_j)}(S_j))$, where the probability is over the random choices of S_j.

LEMMA 3.9. *For any relation S_j:*

(a) $\forall \mathbf{a}_j \in [n]^{r_j} : w_j(\mathbf{a}_j) \leq n^{1-r_j}$, *and*

(b) $\sum_{\mathbf{a}_j \in [n]^{r_j}} w_j(\mathbf{a}_j) \leq fn$.

PROOF. To show (a), notice that $w_j(\mathbf{a}_j) \leq \mathbf{P}(\mathbf{a}_j \in S_j) = n^{1-r_j}$, while (b) follows from the fact $\sum_{\mathbf{a}_j \in [n]^{r_j}} w_j(\mathbf{a}_j) = \mathbf{E}[|K_{m_j(S_j)}(S_j)|] \leq fn$. □

Since the server receives a separate message for each relation S_j, from a distinct input server, the events $\mathbf{a}_1 \in K_{m_1}(S_1), \ldots, \mathbf{a}_\ell \in K_{m_\ell}(S_\ell)$ are independent, hence:

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

We now prove Lemma 3.8 using Friedgut's inequality. Recall that in order to apply the inequality, we need to find a fractional edge cover. Fix an optimal fractional edge packing $\mathbf{u} = (u_1, \ldots, u_\ell)$ as in Fig. 1. By duality, we have that $\sum_j u_j = \tau^*$, where τ^* is the fractional covering number (which is the value of the optimal *fractional vertex cover*, and equal to the value of the optimal *fractional edge packing*). Given q, defined as in (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 3.10. *(a) The assignment $(\mathbf{u}, \mathbf{u}')$ is both a tight fractional edge packing and a tight fractional edge cover for q'. (b) $\sum_{j=1}^{\ell} r_j u_j + \sum_{i=1}^{k} u'_i = k$*

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:

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

which proves (b). □

We will apply Friedgut's inequality to the extended query q' to prove Lemma 3.8. Set the variables $w(-)$ used in Friedgut's inequality as follows:

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

Recall that, for a tuple $\mathbf{a} \in [n]^k$ we use $\mathbf{a}_j \in [n]^{r_j}$ 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. Then, interpreting $(\sum_{\mathbf{a}} b_{\mathbf{a}}^{1/u})^u$ as $\max_{\mathbf{a}} b_{\mathbf{a}}$ for $u = 0$:

$$\mathbf{E}[|K_m(q)|] = \sum_{\mathbf{a}\in[n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) = \sum_{\mathbf{a}\in[n]^k} \prod_{j=1}^{\ell} w_j(\mathbf{a}_j) \prod_{i=1}^{k} w'_i(\mathbf{a}_i)$$

$$\leq \prod_{j=1}^{\ell} \left(\sum_{\mathbf{a}\in[n]^{r_j}} w_j(\mathbf{a})^{1/u_j}\right)^{u_j} \prod_{i=1}^{k} \left(\sum_{a\in[n]} w'_i(a)^{1/u'_i}\right)^{u'_i}$$

$$= \prod_{j=1}^{\ell} \left(\sum_{\mathbf{a}\in[n]^{r_j}} w_j(\mathbf{a})^{1/u_j}\right)^{u_j} \prod_{i=1}^{k} n^{u'_i}$$

Assume first that all $u_j > 0$. By Lemma 3.9, we obtain:

$$\begin{aligned} \sum_{\mathbf{a}\in[n]^{r_j}} w_j(\mathbf{a})^{1/u_j} &\leq (n^{1-r_j})^{1/u_j - 1} \sum_{\mathbf{a}\in[n]^{r_j}} w_j(\mathbf{a}) \\ &\leq n^{(1-r_j)(1/u_j-1)} fn = fn^{(r_j - r_j/u_j + 1/u_j)} \end{aligned}$$

Plugging this in the bound, we have shown that:

$$\begin{aligned} \mathbf{E}[|K_m(q)|] &\leq \prod_{j=1}^{\ell} (fn^{(r_j - r_j/u_j + 1/u_j)})^{u_j} \prod_{i=1}^{k} n^{u'_i} \\ &= f^{\sum_{j=1}^{\ell} u_j} n^{(\sum_{j=1}^{\ell} r_j u_j - r + \ell)} n^{\sum_{i=1}^{k} u'_i} \\ &= n^{(\ell - r)} f^{\sum_{j=1}^{\ell} u_j} n^{(\sum_{j=1}^{\ell} r_j u_j + \sum_{i=1}^{k} u'_i)} \\ &= n^{\ell + k - r} f^{\tau^*(q)} = n^{1+\chi(q)} f^{\tau^*(q)} \end{aligned} \quad (8)$$

If some $u_j = 0$, then replace each u_j with $u_j + \delta$ (still an edge cover). Now we have $\sum_j r_j u_j + \sum_i u'_i = k + r\delta$, hence an extra factor $n^{r\delta}$ in (8), which $\to 1$ when $\delta \to 0$. Lemma 3.8 follows from (8) and $\mathbf{E}[|q(I)|] = n^{1+\chi(q)}$.

3.3 Extensions

Proposition 3.2 and Theorem 3.3 imply that, over matching databases, the space exponent of a query q is $1 - 1/\tau^*$, where τ^* is its fractional covering number. Table 1 illustrates the space exponent for various families of conjunctive queries. We now discuss a few extensions and corollaries whose proofs are given in the full paper: As a corollary of Theorem 3.3 we can characterize the queries with space exponent zero, i.e. those that can be computed in a single round without any replication.

COROLLARY 3.11. *A query q has covering number $\tau^*(q) = 1$ iff there exists a variable shared by all atoms.*

Thus, a query can be computed in one round on MPC(0) iff it has a variable occurring in all atoms. The corollary should be contrasted with the results in [17], which proved that a query is computable in one round iff it is *tall-flat*. Any connected tall-flat query has a variable occurring in all atoms, but the converse is not true in general. The algorithm in [17] works for *any* input data, including skewed inputs, while here we restrict to matching databases. For example, $S_1(x, y), S_2(x, y), S_3(x, z)$ can be computed in one round if all inputs are permutations, but it is not tall-flat, and hence it cannot be computed in one round on general input data.

Theorem 3.3 tells us that a query q can report at most a $1/p^{\tau^*(q)(1-\varepsilon)-1}$ fraction of answers. We show that there is an algorithm achieving this for matching databases:

PROPOSITION 3.12. *Given q and $\varepsilon < 1 - 1/\tau^*(q)$, there exists an algorithm that reports $\Theta(\mathbf{E}[|q(I)|]/p^{\tau^*(q)(1-\varepsilon)-1})$ answers in expectation using one round in the MPC(ε) model.*

Note that the algorithm is forced to run in one round, in an MPC(ε) model strictly weaker than its space exponent, hence it cannot find all the answers: the proposition says that the algorithm can find an expected number of answers that matches Theorem 3.3.

So far, our lower bounds were for the JOIN-REPORTING problem. We can extend the lower bounds to the JOIN-WITNESS problem. For this, we choose unary relations $R(w)$ and $T(z)$ to include each element from $[n]$ independently with probability $1/\sqrt{n}$, and derive:

PROPOSITION 3.13. *For $\varepsilon < 1/2$, there exists no one-round MPC(ε) algorithm that solves* JOIN-WITNESS *for the query $q(w, x, y, z) = R(w), S_1(w, x), S_2(x, y), S_3(y, z), T(z)$.*

4. MULTIPLE COMMUNICATION STEPS

In this section we consider a restricted version of the MPC(ε) model, called the *tuple-based* MPC(ε) model, which can simulate multi-round MapReduce for database queries. We will establish both upper and lower bounds on the number of rounds needed to compute any connected query q in this tuple-based MPC(ε) model, proving Theorem 1.2.

4.1 An Algorithm for Multiple Rounds

Given an $\varepsilon \geq 0$, let Γ^1_ε denote the class of connected queries q for which $\tau^*(q) \leq 1/(1-\varepsilon)$; these are precisely the

queries that can be computed in one round in the MPC(ε) model on matching databases. We extend this definition inductively to larger numbers of rounds: Given Γ^r_ε for some $r \geq 1$, define Γ^{r+1}_ε to be the set of all connected queries q constructed as follows. Let $q_1, \ldots, q_m \in \Gamma^r_\varepsilon$ 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+1}_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. The following proposition is straightforward.

PROPOSITION 4.1. *Every query in Γ^r_ε can be computed by an MPC(ε) algorithm in r rounds on any matching database.*

EXAMPLE 4.2. *Let $\varepsilon = 1/2$. The query L_k in Table 1 for $k = 16$ has a query plan of depth $r = 2$. The first step computes in parallel four queries, $v_1 = S_1, S_2, S_3, S_4$, ..., $v_4 = S_{13}, S_{14}, S_{15}, S_{16}$. Each is isomorphic to L_4, therefore $\tau^*(q_1) = \cdots = \tau^*(q_4) = 2$ and each can be computed in one step. The second step computes the query $q_0 = V_1, V_2, V_3, V_4$, which is also isomorphic to L_4. We can generalize this approach for any L_k: for any $\varepsilon \geq 0$, let k_ε be the largest integer such that $\tau^*(L_{k_\varepsilon}) \leq 1/(1-\varepsilon)$: $k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$. 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 $\lceil \log k / \log k_\varepsilon \rceil$.*

We also 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 space exponent for one round is $1 - 1/k$. However, SP_k has a query plan of depth 2 for MPC(0), by computing the joins $q_i = R_i(z, x_i), S_i(x_i, y_i)$ in the first round and in the second round joining all q_i on the common variable z. Thus, if we insist in answering SP_k in one round, we need a huge replication $O(p^{1-1/k})$, but we can compute it in two rounds with replication $O(1)$.

We next present an upper bound on the number of rounds needed to compute any query. 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. For example, $\text{rad}(L_k) = \lceil k/2 \rceil$ and $\text{rad}(C_k) = \lfloor k/2 \rfloor$.

LEMMA 4.3. *Fix $\varepsilon \geq 0$, let $k_\varepsilon = 2\lfloor 1/(1-\varepsilon) \rfloor$, and let q be any connected query. Let $r(q) = \lceil \log(rad(q))/\log k_\varepsilon \rceil + 1$ if q is tree-like, and let $r(q) = \lceil \log(rad(q)+1)/\log k_\varepsilon \rceil + 1$ otherwise. Then, q can be computed in $r(q)$ rounds on any matching database input by repeated application of the HC algorithm in the MPC(ε) model.*

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 tree-like 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 $\lceil \log(\text{rad}(q))/\log k_\varepsilon \rceil$ rounds using the query plan from Proposition 4.1 together with repeated use of the one-round HC algorithm for paths of length k_ε as shown in Proposition 3.2 for $\tau = 1/(1-\varepsilon)$. 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 without any replication. 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. □

q query	ε space exponent	r rounds for $\varepsilon = 0$	$r = f(\varepsilon)$ raounds/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

Table 2: The tradeoff between space and communication rounds for several queries.

As an application of this proposition, Table 2 shows the number of rounds required by different types of queries.

4.2 Lower Bounds for Multiple Rounds

Our lower bound results for multiple rounds are restricted in two ways: they apply only to an MPC model where communication at rounds ≥ 2 is of a restricted form, and they match the upper bounds only for a restricted class of queries.

4.2.1 Tuple-Based MPC

Recall that $M^1_u = (M^1_{1u}, \ldots, M^1_{\ell u})$, where M^1_{ju} denotes the message sent during round 1 by the input server for S_j to the worker u. Let I be the input database instance, and q be the query we want to compute. A *join tuple* is any tuple in $q'(I)$, where q' is any connected subquery of q.

The *tuple-based* MPC(ε) model imposes the following two restrictions during rounds $r \geq 2$, for every worker u: (a) the message M^r_{uv} sent to v is a set of join tuples, and (b) for every join tuple t, the worker u decides whether to include t in M^r_{uv} based only on t, u, v, r and M^1_{ju}, for all j s.t. t contains a base tuple in S_j.

The restricted model still allows unrestricted communication during the first round; the information M^1_u received by server u in the first round is available throughout the computation. However, during the following rounds, server u 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 M^1_u. Since a join tuple is represented using $\Theta(\log n)$ bits, each server receives $O(n/p^{1-\varepsilon})$ join tuples at each round. We now describe the lower bound for multiple rounds in the tuple-based MPC model.

4.2.2 A Lower Bound

We give here a general lower bound for connected, conjunctive queries, and show how to apply it to L_k, to tree-like queries, and to C_k; these results prove Theorem 1.2. We postpone the proof to the next subsection.

DEFINITION 4.4. *Let q be a connected, conjunctive query. A set $M \subseteq atoms(q)$ is* ε-good for q *if it satisfies:*

1. *Every subquery of q that is in Γ^1_ε contains at most one atom in M. (Γ^1_ε defined in Sec. 4.2.1)*
2. *$\chi(\overline{M}) = 0$, where $\overline{M} = atoms(q) - M$. (Hence by Lemma 2.1, $\chi(q/\overline{M}) = \chi(q)$. This condition is equivalent to each connected component of $\overline{M}$ being tree-like.)*

An (ε, r)-plan $\mathcal{M}$ is a sequence $M_1, \ldots, M_r$, with $M_0 = atoms(q) \supset M_1 \supseteq \cdots M_r$ such that (a) for all $j \in [r]$, M_{j+1} is ε-good for $q/\overline{M}_j$ where $\overline{M}_j = atoms(q) - M_j$, and (b) $q/\overline{M}_r \notin \Gamma^1_\varepsilon$.

THEOREM 4.5. *If q has a (ε, r)-plan then every randomized algorithm running in $r+1$ rounds on the tuple-based MPC(ε) model with $p = \omega(1)$ processors fails to compute q with probability $\Omega(n^{\chi(q)})$.*

We prove the theorem in the next section. Here, we show how to apply it to three cases. Assume $p = \omega(1)$, and recall that $k_\varepsilon = 2\lfloor 1/(1-\varepsilon)\rfloor$ (Example 4.2). First, consider L_k.

LEMMA 4.6. *Any tuple-based MPC(ε) algorithm that computes L_k needs at least $\lceil \log k / \log k_\varepsilon \rceil$ rounds.*

PROOF. We show inductively how to produce an (ε, r)-plan for L_k with $r = \lceil \log k / \log k_\varepsilon \rceil - 1$. The subqueries that are in Γ^1_ε are precisely L_{k_0} for $k_0 \le k_\varepsilon$, hence any set of atoms M that consists of every k_ε-th atom in L_ℓ is ε-good for L_ℓ for any $\ell \ge k_\varepsilon$. Let M_1 be such a set starting with the first atom. Then $L_k/\overline{M}_1$ is isomorphic to $L_{\lceil k/k_\varepsilon \rceil}$. For $j = 2, .., r$, choose M_j to consist of every k_ε-th atom starting at the first atom in $L_k/\overline{M}_{j-1}$. Finally, $L_k/\overline{M}_{j-1}$ will be isomorphic to a path query of length L_ℓ for some $\ell \ge k_\varepsilon + 1$ and hence is not in Γ^1_ε. Thus $M_1, \ldots, M_r$ is the desired (ε, r)-plan and the lower bound follows from Theorem 4.5. □

Combined with Example 4.2, it implies that L_k requires precisely $\lceil \log k / \log k_\varepsilon \rceil$ rounds on the tuple-based MPC(ε).

Second, we give a lower bound for tree-like queries, and for that we use a simple observation:

PROPOSITION 4.7. *If q is a tree-like query, and q' is any connected subquery of q, q' needs at least as many rounds as q in the tuple-based MPC(ε) model.*

PROOF. Given any tuple-based MPC(ε) algorithm A for computing q in r rounds we construct a tuple-based MPC(ε) algorithm A' that computes q' in r rounds. 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 tree-like. □

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 $\mathrm{rad}(C_k) = \mathrm{diam}(C_k) = \lfloor k/2 \rfloor$. Lemma 4.6 and Proposition 4.7 imply:

COROLLARY 4.8. *Any tuple-based MPC(ε) algorithm that computes a tree-like query q needs at least $\lceil \log_{k_\varepsilon}(\mathit{diam}(q)) \rceil$ rounds.*

Let us compare the lower bound $r_{\mathrm{low}} = \lceil \log_{k_\varepsilon}(\mathrm{diam}(q)) \rceil$ and the upper bound $r_{\mathrm{up}} = \lceil \log_{k_\varepsilon}(\mathrm{rad}(q)) \rceil + 1$ (Lemma 4.3): $\mathrm{diam}(q) \le 2\mathrm{rad}(q)$ implies $r_{\mathrm{low}} \le r_{\mathrm{up}}$, while $\mathrm{rad}(q) \le \mathrm{diam}(q)$ implies $r_{\mathrm{up}} \le r_{\mathrm{low}} + 1$. The gap between the lower bound and the upper bound is at most 1, proving Theorem 1.2. When $\varepsilon < 1/2$, these bounds are matching, since $k_\varepsilon = 2$ and $2\mathrm{rad}(q) - 1 \le \mathrm{diam}(q)$ for tree-like queries. The tradeoff between the space exponent ε and the number of rounds r for tree-like queries is $r \cdot \log \frac{2}{1-\varepsilon} \approx \log(\mathrm{rad}(q))$.

Third, we study one instance of a non tree-like query:

LEMMA 4.9. *Any tuple-based MPC(ε) algorithm that computes C_k needs at least $\lceil \log(k/(m_\varepsilon+1)) / \log k_\varepsilon \rceil + 1$ rounds, where $m_\varepsilon = \lfloor 2/(1-\varepsilon) \rfloor$.*

PROOF. Observe that any set M of atoms that are (at least) k_ε apart along any cycle C_ℓ is ϵ-good for C_ℓ and $C_\ell/\overline{M}$ is isomorphic to $C_{\lfloor \ell/k_\varepsilon \rfloor}$. If $k \ge k^r_\varepsilon(m_\varepsilon + 1)$, we can repeatedly choose such ε-good sets to construct an (ε, r)-plan $M_1, \ldots, M_r$ such that the final contracted query $C_k/\overline{M}_r$ contains a cycle $C_{\ell'}$ with $\ell' \ge m_\varepsilon + 1$ (and therefore cannot be computed in 1 round by any MPC(ε) algorithm). The result now follows from Theorem 4.5. □

Here, too, we have a gap of 1 between this lower bound and the upper bound in Lemma 4.3. Consider C_5 and $\varepsilon = 0$; $\mathrm{rad}(C_5) = \mathrm{diam}(C_5) = 2$, $k_\varepsilon = m_\varepsilon = 2$. The lower bound is $\lfloor \log 5/3 \rfloor + 1 = 2$ rounds, the upper bound is $\lceil \log 3 \rceil + 1 = 3$ round. The exact number of rounds for C_5 is open.

As a final application, we show how to apply Lemma 4.6 to show that transitive closure requires many rounds (the proof is included in the full version of the paper).

COROLLARY 4.10. *For any fixed $\varepsilon < 1$, there is no p-server algorithm in the tuple-based MPC(ε) model that uses $o(\log p)$ rounds and computes the transitive closure of an arbitrary input graph.*

4.2.3 Proof of Theorem 4.5

Given an (ε, r)-plan $\mathcal{M}$ (Definition 4.4) for a query q, define $\tau^*(\mathcal{M})$ to be the minimum of $\tau^*(q/\overline{M}_r)$, and the minimum of $\tau^*(q')$, where q' ranges over all connected subqueries of $q/\overline{M}_{j-1}$, $j \in [r]$, such that $q' \notin \Gamma^1_\varepsilon$. Since every q' satisfies $\tau^*(q')(1-\varepsilon) > 1$ (by $q' \notin \Gamma^1_\varepsilon$), and $\tau^*(q/\overline{M}_r)(1-\varepsilon) > 1$ (by the definition of goodness), we have $\tau^*(\mathcal{M})(1-\varepsilon) > 1$.

THEOREM 4.11. *If q has an (ε, r)-plan $\mathcal{M}$ then any deterministic tuple-based MPC(ε) algorithm running in $r+1$ rounds reports $O(\mathbf{E}(|q(I)|)/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1})$ correct answers in expectation over uniformly chosen matching database I.*

The argument in Corollary 3.5 extends immediately to this case, implying that every randomized tuple-based MPC(ε) algorithm with $p = \omega(1)$ and $r+1$ rounds will fail to compute q with probability $\Omega(n^{\gamma(q)})$. This proves Theorem 4.5.

The rest of this section gives the proof of this theorem. The intuition is this. Consider a ε-good set M; then any matching database i consists of two parts, $i = (i_M, i_{\overline{M}})$, where i_M are the relations for atoms in M, and $i_{\overline{M}}$ are the other relations. We show that, for a fixed instance $i_{\overline{M}}$, the algorithm A 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 M-atoms is not in Γ^1_ε. This shows that the algorithm computes $q/\overline{M}(i_M)$ in only r rounds, and we repeat the argument until a one-round algorithm remains.

First, we need some notation. For a connected subquery q' of q, $q'(I)$ denotes as usual the answer to q' on an instance I. Whenever $\mathrm{atoms}(q') \subseteq \mathrm{atoms}(q'')$, then we say that a tuple $t'' \in q''(I)$ *contains* a tuple $t' \in q'(I)$, if t' is equal to the projection of t'' on the variables of q'; if $A \subseteq q''(I), B \subseteq q'(I)$, then $A \ltimes B$, called the *semijoin*, denotes the subset of tuples $t'' \in A$ that contain some tuple $t' \in B$.

Let A be a deterministic algorithm with $r+1$ rounds, $k \in [r+1]$ a round number, u a server, and q' a subquery of q. For a matching database input i, define $m_{A,u,k}(i)$ to be the vector of messages received by server u during the first k rounds of the execution of A on input i. Define $m_{A,k}(i) = (m_1, \ldots, m_p)$, where $m_u = m_{A,u,k}(i)$ for all $u \in [p]$, and:

$$K_m^{A,u,k}(q') = \{t' \in [n]^{\text{vars}(q')} \mid \text{ for all matching databases } i, \; m_{A,u,k}(i) = m \Rightarrow t' \in q'(i)\}$$

$$K_m^{A,k}(q') = \bigcup_u K_{m_u}^{A,u,k}(q') \qquad A(i) = K_{m_{A,r+1}(i)}^{A,r+1}(q).$$

$K_{m_{A,u,k}(i)}^{A,u,k}(q')$ and $K_{m_{A,k}(i)}^{A,k}(q')$ denote the set of join tuples from q' known at round k by server u, and by all servers, respectively, on input i. $A(i)$ is w.l.o.g. the final answer of A on input i. Define

$$J^{A,q}(i) = \bigcup\{K_{m_{A,1}(i)}^{A,1}(q') \mid q' \text{ connected subquery of } q\}$$

$$J_\varepsilon^{A,q}(i) = \bigcup\{K_{m_{A,1}(i)}^{A,1}(q') \mid q' \notin \Gamma_\varepsilon^1 \text{ connected subquery of } q\}$$

$J_\varepsilon^{A,q}(i)$ is precisely the set of join tuples known after the first round, but which correspond to subqueries that are themselves not computable in one round; thus, the number of tuples in $J_\varepsilon^{A,q}(i)$ will be small. Next, we need two lemmas.

LEMMA 4.12. *Let q be a query, and M be any ε-good set for q. If 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 A' with r rounds for $q/\overline{M}$ such that, for every matching database i_M defined over the atoms of M:*

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

In other words, the algorithm returns no more answers than the (very few) tuples in J, plus what another algorithm A' (to be defined) computes for $q/\overline{M}$ in *one less* rounds.

PROOF. The proof requires two constructions.

1. Contraction. 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. We show how to use the algorithm A for q to derive an algorithm, denoted A_M, for $q/\overline{M}$.

For each connected component C of $\overline{M}$, choose a representative variable $z_c \in \text{vars}(C)$; also denote S_C the result of applying the query C to $i_{\overline{M}}$; S_c is a matching, because C is tree-like. Denote $\bar\sigma = \{\sigma_x \mid x \in \text{vars}(q)\}$, where, for every variable $x \in \text{vars}(q)$, σ_x is the following permutation on $[n]$: if $x \notin \text{vars}(\overline{M})$ then σ_x = the identity; otherwise $\sigma_x = \Pi_{xz_c}(S_C)$, for the unique connected component s.t. $x \in \text{vars}(C)$. We think of $\bar\sigma$ as permuting the domain of each attribute $x \in \text{vars}(q)$. Then $\bar\sigma(q(i)) = q(\bar\sigma(i))$, and $\bar\sigma(i_{\overline{M}}) = \mathbf{id}_{\overline{M}}$ the identity matching database (where each relation in $\overline{M}$ is $\{(1,1,\ldots),(2,2,\ldots),\ldots\}$), and therefore:

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

(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.)

The algorithm A_M for $q/\overline{M}(i_M)$ is this. First, each input server for $S_j \in M$ replaces S_j with $\bar\sigma(S_j)$ (since $i_{\overline{M}}$ is fixed, it is known to all servers, hence, so is $\bar\sigma$); next, run A unchanged, substituting all relations $S_j \in \overline{M}$ with the identity; finally, apply $\bar\sigma^{-1}$ to the answers and return them. We have:

$$A_M(i_M) = \bar\sigma^{-1}(\Pi_{\text{vars}(q/\overline{M})}(A(\bar\sigma(i_M), \mathbf{id}_{\overline{M}}))) \qquad (9)$$

2. Retraction. Next, we transform A_M into a new algorithm R_{A_M} called the *retraction* of A_M, as follows:

(a) During round 1 of R_{A_M}, each input server for S_j sends (in addition to the messages sent by A_M) every tuple in $t \in S_j$ to all servers u that eventually receive t. In other words, the input server sends t to every u for which there exists $k \in [r+1]$ such that $t \in K_{m_{A_M,u,k}(I_M)}^{A_M,u,k}(S_j)$. This is possible because of the restrictions in the tuple-based MPC(ε) model: all destinations of t depend only on S_j, and hence can be computed by the input server. Note that this may increase the total number of bits received in the first round by a factor of r, which is $O(1)$ in our setting. R_{A_M} will not send any atomic tuples during rounds $k \ge 2$. (b) In round 2, R_{A_M} sends *no* tuples. (c) In rounds $k \ge 3$, R_{A_M} sends a tuple t from u to v if server u knows t at round k, and algorithm A_M sends t from u to v at round k.

It follows that, for each round k, and for each subquery q' of $q/\overline{M}$ with at least two atoms, $K_{m(i)}^{R_{A_M},u,k}(q') \subseteq K_{m(i)}^{A_M,u,k}(q')$: in other words, R_{A_M} knows a subset of the non-atomic tuples known by A_M. Moreover, let $J_+^{A_M}(i_M)$ be the set of non-atomic tuples known by A_M after round 1, $J_+^{A_M}(i_M) = \bigcup\{K_{m(i)}^{R_{A_M},u,1}(q') \mid q' \text{ has at least two atoms}\}$: these are the tuples that we refused to sent in round 2. Then:

$$A_M(i_M) \subseteq (q/\overline{M}(i_M) \ltimes J_+^{A_M}) \cup R_{A_M}(i_M) \qquad (10)$$

Since R_{A_M} wastes one round, we can compress it to an algorithm A' with only r rounds. To prove the lemma, we convert (10) into a statement about A. (9) already showed that $A_M(i_M)$ is related to $A(i_M, i_{\overline{M}})$. Now we show how $J_+^{A_M}$ is related to $J_\varepsilon^{A,q}(i)$: $J_+^{A_M}(i_M) \subseteq \sigma^{-1}(\Pi_{\text{vars}(q/\overline{M})}(J_\varepsilon^{A,q}(\bar\sigma(i))))$ because, by the definition of ε-goodness, if a subquery q' of q has two atoms in M, then $q' \notin \Gamma_\varepsilon^1$. (10) becomes:

$$A_M(i_M) \subseteq (q/\overline{M}(i_M) \ltimes \Pi_{\text{vars}(q/\overline{M})}(J_\varepsilon^{A,q}(i))) \cup \bar\sigma^{-1}(A'(i_M))$$

The lemma follows from $q/\overline{M}(i_M) \ltimes \Pi_{\text{vars}(q/\overline{M})}(J_\varepsilon^{A,q}(i)) \subseteq \Pi_{\text{vars}(q/\overline{M})}(q(i) \ltimes J_\varepsilon^{A,q}(i))$ and $|A_M(i_M)| = |A(i_M, i_{\overline{M}})|$, by (9). □

LEMMA 4.13. *Let q be a conjunctive query, and q' a subquery; if i is a database instance for q, we write i' for its restriction to the relations occurring in q'. Let B be any algorithm for q' (meaning that, for every matching database i', $B(i') \subseteq q'(i')$), and assume that $\mathbf{E}[|B(I')|] \le \gamma \cdot \mathbf{E}[|q'(I')|]$. Then, $\mathbf{E}[|q(I) \ltimes B(I')|] \le \gamma \mathbf{E}[|q(I)|]$ where I is a uniformly chosen matching database.*

While, in general, q' may return many more answers than q, the lemma says that, if B returns only a fraction of q', then $q \ltimes B$ returns only the same fraction of q.

PROOF. Let $\bar y = (y_1, \ldots, y_k)$ be the variables occurring in q'. For any $\bar a \in [n]^k$, let $\sigma_{\bar y = \bar a}(q(i))$ denote the subset of tuples $t \in q(i)$ whose projection on $\bar y$ equals $\bar a$. By symmetry, the quantity $\mathbf{E}[|\sigma_{\bar y=\bar a}(q(I))|]$ is independent of $\bar a$, and therefore equals $\mathbf{E}[|q(I)|]/n^k$. Notice that $\sigma_{\bar y=\bar a}(B(i'))$ is either $\emptyset$ or $\{\bar a\}$. We have:

$$\mathbf{E}[|q(I) \ltimes B(I')|] = \textstyle\sum_{\bar a \in [n]^k} \mathbf{E}[|\sigma_{\bar y=\bar a}(q(I)) \ltimes \sigma_{\bar y=\bar a}(B(I'))|]$$

$$= \textstyle\sum_{\bar a \in [n]^k} \mathbf{E}[|\sigma_{\bar y=\bar a}(q(I))|] \cdot \mathbf{P}(\bar a \in B(I'))$$

$$= \mathbf{E}[|q(I)|] \cdot \textstyle\sum_{\bar a \in [n]^k} \mathbf{P}(\bar a \in B(I'))/n^k = \mathbf{E}[|q(I)|] \cdot \mathbf{E}[|B(I')|]/n^k$$

Repeating the same calculations for q' instead of B,

$$\mathbf{E}[|q(I) \ltimes q'(I')|] = \mathbf{E}[|q(I)|]\mathbf{E}[|q'(I')|]/n^k$$

The lemma follows immediately, by using the fact that, by definition, $q(i) \ltimes q'(i') = q(i)$. □

Finally, we prove Theorem 4.11.

PROOF OF THEOREM 4.11. Given the (ε, r)-plan atoms$(q) = M_0 \supset \ldots \supset M_r$, define $\hat{M}_k = \overline{M}_k - \overline{M}_{k-1}$, for $k \geq 1$. We build up $i_{\overline{M}_r}$ by iteratively choosing matching databases $i_{\hat{M}_k} = \overline{M}_k - \overline{M}_{k-1}$ for $k = 1, \ldots, r$ and applying Lemma 4.12 with q replaced by $q/\overline{M}_{k-1}$ and M replaced by M_k to obtain algorithms $A^k = A^k_{(i_{\hat{M}_1}, \ldots, i_{\hat{M}_k})}$ for $q/\hat{M}_1 \cdots \hat{M}_k$ such that the following inequality holds for every choice of matching databases given by i_{M_r} and $i_{\overline{M}_r} = (i_{\hat{M}_1}, \ldots, i_{\hat{M}_r})$:

$$\begin{aligned}
|A(i_{M_r}, i_{\overline{M}_r})| &= |A(i_{M_r}, i_{\hat{M}_1}, \ldots, i_{\hat{M}_r})| \\
&\leq |q(i_{M_r}, i_{\overline{M}_r}) \ltimes J^{A,q}_{\varepsilon}(i_{M_r}, i_{\hat{M}_1}, \ldots, i_{\hat{M}_r})| \\
&+ |q(i_{M_r}, i_{\overline{M}_r}) \ltimes J^{A^1, q/\hat{M}_1}_{\varepsilon}(i_{M_r}, i_{\hat{M}_2}, \ldots, i_{\hat{M}_r})| \\
&+ \ldots + |q(i_{M_r}, i_{\overline{M}_r}) \ltimes J^{A^{r-1}, q/\hat{M}_1 \cdots \hat{M}_{r-1}}_{\varepsilon}(i_{M_r}, i_{\hat{M}_r})| \\
&+ |A^r(i_{M_r})| \qquad (11)
\end{aligned}$$

We now average (11) over a uniformly chosen matching database I and upper bound each of the resulting terms: For all $k \in [r]$ we have $\chi(q/\overline{M}_k) = \chi(q)$ (see Definition 4.4), and hence, by Lemma 3.4, we have $\mathbf{E}[|q(I)|] = \mathbf{E}[|(q/\overline{M}_k)(I_{M_k})|]$. By definition, we have $\tau^*(q/\overline{M}_r) \geq \tau^*(\mathcal{M})$ and hence by Theorem 3.3,

$$\begin{aligned}
\mathbf{E}[|A^r(I_{M_r})|] &= O(\mathbf{E}[|(q/\overline{M}_r)(I_{M_r})|]/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1}) \\
&= O(\mathbf{E}[|q(I)|]/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1})
\end{aligned}$$

Note that $I_{M_{k-1}} = (I_{M_r}, I_{\hat{M}_k}, \ldots, I_{\hat{M}_r})$ and consider the expected number of tuples in $J = J^{A^{k-1}, q/\hat{M}_1 \cdots \hat{M}_{k-1}}_{\varepsilon}(I_{M_{k-1}})$. The algorithm $A^{k-1} = A^{k-1}_{I_{\overline{M}_{k-1}}}$ 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 $q/\hat{M}_1 \cdots \hat{M}_{k-1}$ that is not in Γ^1_{ε} (hence contributes to J) has $\tau^*(q') \geq \tau^*(\mathcal{M})$. By Theorem 3.3, 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 $A^{k-1}_{i_{\overline{M}_{k-1}}}$ that produces tuples for q', satisfies $\mathbf{E}[|B_{q'}(I_{M_{k-1}})|] = O(\mathbf{E}[|q'(I_{M_{k-1}})|]/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1})$. We now apply Lemma 4.13 to derive

$$\begin{aligned}
\mathbf{E}[|q(I) \ltimes B_{q'}(I_{M_{k-1}})|] &= \mathbf{E}[|(q/\overline{M}_{k-1})(I_{M_{k-1}}) \ltimes B_{q'}(I_{M_{k-1}})|] \\
&= O(\mathbf{E}[|(q/\overline{M}_{k-1})(I_{M_{k-1}})|]/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1}) \\
&= O(\mathbf{E}[|q(I)|]/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1}).
\end{aligned}$$

Averaging over all choices of $I_{\overline{M}_{k-1}} = i_{\overline{M}_{k-1}}$ and summing over the constant number of different queries q' we obtain

$$\begin{aligned}
&\mathbf{E}[|q(I) \ltimes J^{A^{k-1}, q/\hat{M}_1 \cdots \hat{M}_{k-1}}_{\varepsilon}(I_{M_{k-1}})|] \\
&\qquad = O(\mathbf{E}[|q(I)|]/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1}).
\end{aligned}$$

Combining the bounds for the $r+1$ terms in (11) we obtain that $\mathbf{E}[|A(I)|] = O(r\mathbf{E}[|q(I)|]/p^{\tau^*(\mathcal{M})(1-\varepsilon)-1})$. □

5. CONCLUSION

We have introduced powerful models for capturing tradeoffs between rounds and amount of communication required for parallel computation of relational queries. For one round on the most general model we have shown that queries are characterized by τ^* which determines the space exponent $\varepsilon = 1 - 1/\tau^*$ that governs the replication rate as a function of the number of processors. For multiple rounds we derived a strong lower bound tradeoff between the number of rounds r and the replication rate of $r \cdot \log 2/(1-\varepsilon) \approx \log(\text{rad}(q))$ for more restricted tuple-based communication. For both, we showed matching or nearly matching upper bounds given by simple and natural algorithms.

6. REFERENCES

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

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

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

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

[5] S. Chaudhuri. What next?: a half-dozen data management research goals for big data and the cloud. In *PODS*, pages 1–4, 2012.

[6] F. R. K. Chung, Z. Füredi, M. R. Garey, and R. L. Graham. On the fractional covering number of hypergraphs. *SIAM J. Discrete Math.*, 1(1):45–49, 1988.

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[8] EMC Corporation. Data science revealed: A data-driven glimpse into the burgeoning new field. `http://www.emc.com/collateral/about/news/emc-data-science-study-wp.pdf`.

[9] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6(4), 2010.

[10] E. Friedgut. Hypergraphs, entropy, and inequalities. *American Mathematical Monthly*, pages 749–760, 2004.

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

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

[13] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[14] S. Guha and Z. Huang. Revisiting the direct sum theorem and space lower bounds in random order streams. In *ICALP*, volume 5555 of *LNCS*, pages 513–524. Springer, 2009.

[15] Hadoop. `http://hadoop.apache.org/`.

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

[17] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234, 2011.

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

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

[20] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms: [extended abstract]. In *PODS*, pages 37–48, 2012.

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

[22] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011.

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

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

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

[26] A. C. Yao. Lower bounds by probabilistic arguments. In *FOCS*, pages 420–428, Tucson, AZ, 1983.