Shuffles and Circuits (On Lower Bounds for Modern Parallel Computation)

Tim Roughgarden* Sergei Vassilvitskii† Joshua R. Wang*

ABSTRACT

The goal of this paper is to identify fundamental limitations on how efficiently algorithms implemented on platforms such as MapReduce and Hadoop can compute the central problems in the motivating application domains, such as graph connectivity problems.

We introduce an abstract model of massively parallel computation, where essentially the only restrictions are that the "fan-in" of each machine is limited to s bits, where s is smaller than the input size n, and that computation proceeds in synchronized rounds, with no communication between different machines within a round. Lower bounds on the round complexity of a problem in this model apply to every computing platform that shares the most basic design principles of MapReduce-type systems.

We prove that computations in our model that use few rounds can be represented as low-degree polynomials over the reals. This connection allows us to translate a lower bound on the (approximate) polynomial degree of a Boolean function to a lower bound on the round complexity of every (randomized) massively parallel computation of that function. These lower bounds apply even in the "unbounded width" version of our model, where the number of machines can be arbitrarily large. As one example of our general results, computing any non-trivial monotone graph property — such as connectivity — requires a super-constant number of rounds when every machine can accept only a sub-polynomial (in n) number of input bits s.

Finally, we prove that, in two senses, our lower bounds are the best one could hope for. For the unbounded-width model, we prove a matching upper bound. Restricting to a polynomial number of machines, we show that asymptotically better lower bounds require proving that $P \neq NC^1$.

1. INTRODUCTION

The past decade has seen a resurgence of parallel computation, and there is now an impressive array of frameworks built for working with large datasets: MapReduce, Hadoop, Pregel, Giraph, Spark, and so on. The goal of this paper is to identify fundamental limitations on how efficiently these frameworks can compute the central problems in the motivating application domains, such as graph connectivity.

Modern architectures for massively parallel computation share a number of common attributes. First, the data, or the input to the computation, is partitioned arbitrarily across all of the nodes participating in the computation. Second, computation proceeds in synchronous rounds. In every round, each machine looks at the local data available and performs some amount of computation *without communicating with other machines*. After the computation is done, a communication round begins, where each machine can send messages to other machines. Importantly, there are no restrictions on the communicating pairs — the communication pattern can be arbitrary and input-dependent. Once the communication phase is over, a new round begins. For example, for readers familiar with the MapReduce framework, the computation corresponds to the *reduce* phase, the designation of addressees to the *map* phase, and the actual communication to the *shuffle* phase.

Most previous work has focused on designing efficient algorithms for these systems. To that end, a number of models for MapReduce and its variants have been proposed; see Section 1.2 for a complete list and a detailed comparison to the present work. Since these works aspire to realizable positive results, their models aim to be faithful to a specific system, and often involve a number of system-dependent parameters and constraints that govern the algorithmic design space.

In this work, motivated by the rapidly changing landscape of the systems deployed in practice and in search of impossibility results, we take a different approach. Instead of looking for a faithful model for the system du jour, we focus on extracting the most fundamental constraints shared by all of these models and examine the consequent restrictions on feasible computations. By focusing on a single parameter, namely the *input size* for each machine, and a single benchmark, the number of *rounds* of computation, we obtain parameterized lower bounds that apply to any computing framework (past or future) that shares the same design principles.

1.1 Summary of Contributions

Our first contribution is a general model of massively parallel computation that captures the core properties common to all modern parallel processing systems. Conceptually, computation proceeds in synchronous rounds, and in each round, each machine performs an arbitrary computation on its input, and sends arbitrary information to arbitrary machines in the next round, subject only to the constraint that each machine receives at most s bits each round. In the most powerful, unbounded-width, version of our model, where the number of machines is unlimited, the only restriction is that the "fan-in" of each machine is limited to s bits,

*Stanford University. Supported in part by NSF Grant CCF-1524062.

†Google, Inc.

SPAA '16 July 11-13, 2016, Pacific Grove, CA, USA

ACM ISBN 978-1-4503-4210-0/16/07.

DOI: http://dx.doi.org/10.1145/2935764.2935799

where s is a parameter smaller than the input size n. (Unlimited extra space can be used for computations on these inputs.)

Second, we prove that computations in our model that use few rounds can be represented as low-degree polynomials over the reals. Specifically, we prove that if a function can be computed by such a computation with space s per machine in r rounds (even with unbounded width), then the function has a polynomial representation with degree at most s^r (Theorem 3.1). In particular, computing a function with a polynomial representation of degree n requires $\lceil \log_s n \rceil$ rounds. This lower bound implies, for example, that computing such a function with space $s = n^\epsilon$ per machine requires $1/\epsilon$ rounds in our model. Similarly, if s is only polylogarithmic, then $\Omega(\log n / \log\log n)$ rounds are needed. Our lower bounds also extend, with a constant-factor degradation, to randomized computations.

An obvious question is whether or not there are super-constant lower bounds on the number of rounds required to compute natural functions when the space s is polynomial in n, such as $s = \sqrt{n}$. Our third contribution is a proof that, in two senses, our lower bounds are the best one could hope for. For the unbounded-width model, our lower bound is completely tight, as every function can be computed in at most $\lceil \log_s n \rceil$ rounds in our model. But what if only a polynomial number of machines are allowed? Here, we show that better lower bounds require proving very strong circuit lower bounds. Specifically, any lower bound asymptotically larger than $\Theta(\log_s n)$ for a function in P would separate NC^1 from P, a major open question in circuit complexity (Theorem 7.1).

Fourth, by connecting MapReduce-type computations to polynomials, we can apply the sophisticated toolbox known for polynomials to reason about these computations. As one example, known results imply that every non-trivial monotone graph property can only be represented by polynomials with degree at least roughly $n^2/3$. (Here n denotes the number of vertices, and the input is given as the characteristic vector of the graph's edge set.) We therefore obtain a lower bound of approximately $\frac{1}{2}\log_s n$ on the number of rounds required for deciding any such property (Theorem 5.3), including graph connectivity.

Finally, we develop new machinery for proving lower bounds on the polynomial degree of Boolean functions, and hence on the round complexity of massively parallel computations. While polynomial degree lower bounds are of course known for many Boolean functions, we prove new (tight) lower bounds for graph problems, including undirected ST-CONNECTIVITY (Theorem 6.4). These imply an $\Omega(\log_s n)$ lower bound on the round complexity of solving these problems via any conceivable MapReduce-type system, even when the width is unbounded.

1.2 Related Work

MapReduce was introduced as a system for large scale data processing by Dean and Ghemawat [8]. As the method gained popularity, researchers began to abstract away the core features that made MapReduce successful in practice. Feldman et al. [13] introduced the *Massive Unordered Data* (MUD) model, and were the first to identify some of the restrictions imposed by the framework.

Karloff et al. [21] introduced a slightly more general model, MRC, which identified the number of synchronous rounds as the key metric for comparing any two algorithms. Furthermore, they limited the amount of parallelism allowed in the model, insisting that it be not too small, by restricting the memory of each machine to be sublinear in the input, and also not too large, by restricting the total number of machines to be sublinear in the input as well. They showed simulation results for a subclass of EREW algorithms in MRC, but left open the question of whether all NC languages can be decided in this setting. Goodrich et al. [16] further extended the simulation results to CRCW PRAMs as well as BSP algorithms [37], and gave algorithms for sorting and searching.

The MRC model in [21] only placed upper bounds on the number of machines and the space available on each, limiting both to $N^{1-\epsilon}$ for an input of size N and some fixed $\epsilon > 0$. Goodrich et al. [16] were the first to single out the total input to each machine, the key parameter of the model of the present work. This decision was later adopted by authors in proving upper bounds on the space-round trade-offs in these computations, and this parameter been variously called memory, space, and key-complexity [2, 5, 15, 26, 32].

The adoption of these models for analysis has led to a set of algorithmic tools and techniques developed for efficient computation in this setting. These include notions of filtering [26], multi-round sampling [12, 25], and coresets [3, 28], among others. Recently Fish et al. [14] introduced a uniform version of MRC, and also proved strict hierarchy theorems with respect to the computation time per processor.

The known lower bounds for explicit functions in these models concern either communication (with a fixed number of rounds) or restricted classes of algorithms (for round lower bounds). In more detail, Pietracaprina et al. [32] prove non-trivial lower bounds for matrix multiplication in a limited setting, which requires computing all elementary products (and specifically excludes Strassen-like algorithms). Similar kinds of limitations are required by [19] and [5] to prove lower bounds for a list ranking problem and relational query processing, respectively. Finally, both Beame et al. [5] and Afrati et al. [1] study, for a fixed number of rounds (usually a single round), space-communication trade-offs.

In distributed computing, the total amount of communication is often the most relevant complexity measure. For example, Woodruff and Zhang [39] and Klauck et al. [22] identify models and problems for which there is no algorithm that beats the communication benchmark of sending the entire input to a single machine. Because massively parallel systems are designed to send a potentially large amount of data in a single round, such communication lower bounds do not generally imply lower bounds for round complexity. For example, in practice, matrix multiplication is regarded as an "easy" problem in MapReduce — indeed, MapReduce was invented for precisely such computations (see e.g. [27]) — even though its solution requires total communication proportional to the size of the matrix.

1.2.1 Relation to Other Computational Models

The model proposed in this paper is closely related to several previously studied computational models.

Congested clique. Perhaps the closest model to the present work is the "congested clique" model, the special case of the CONGEST model [31] in which each pair of machines is connected by a direct link (see [11] and the references therein). The original motivation of this model was for the design and analysis of distributed algorithms, but it is also relevant to massively parallel computation. In the most commonly studied version of the model, n machines with unlimited computational power communicate in synchronized rounds. Each machine initially holds some number m of input bits, and in each round each machine can send a (different) message of w bits to every other machine. (Typically, $w = \Theta(\log n)$.) Thus, the number nw of bits that a machine can send and receive in each round automatically scales linearly with the number of machines.

In our model, we allow the number s of bits that a machine can receive in a given round to be sublinear in the total number of machines. In this sense, our model is weaker than the congested clique

model (when $s = o(n)$).[1] On the other hand, MapReduce-type systems differ from distributed settings in that there is little motivation for restricting the amount of point-to-point communication in a round. In our model, a machine can receive all s of its bits in a round from a single other machine. In this sense, our model is stronger than the congested clique model (when $s = \omega(w)$). We note that Hegeman and Pemmaraju [17] show that congested clique computations can be simulated by MapReduce computations (in the model in [21]), albeit with space-per-machine proportional to the communication-per-machine in the congested clique computation. (As already noted, the latter generally scales with the number of machines, while we are generally interested in space-per-machine smaller than this.)

Drucker et al. [11] forge an interesting connection between congested clique computations and circuits. In analogy with our "barrier" result in Section 7, they prove a simulation result implying that even slightly super-constant lower bounds on the round complexity of congested clique computations for a problem in NP would imply better circuit size-depth trade-offs for such problems than are currently known (for unbounded fan-in circuits with unweighted threshold gates or with mod-m gates, e.g. $m = 6$).[2]

Circuits with medium fan-in. Another related model is "circuits with medium fan-in" — arbitrary gates with fan-in that scales with, but is smaller than, the input size — introduced recently in [18]. Our model is stronger than the one in [18], with the most significant difference being that the communication pattern in our model can be input-dependent. Our round complexity lower bounds imply depth lower bounds for the model in [18]. The focus of [18] is on circuit size (rather than depth), so the lower bounds and barriers to lower bounds identified in [18] appear incomparable to ours.

Ideal PRAMs. The much older model of "ideal CREW PRAMs" is also highly relevant to our unbounded-width model. In this model, which was introduced in [7], there is an unbounded number of processors with unbounded computational power and a shared memory. Computation proceeds in synchronous rounds. Each round, each processor can read a single memory cell and write to a single memory cell, subject to the constraint that at each time step at most one processor can write to any given memory cell. Perhaps the most intriguing result in [7] is a parallel algorithm that computes the logical OR function (and, through reductions, many other functions) in strictly fewer than $\log_2 n$ rounds. The key idea in this result is to implicitly transmit extra information by *not* writing to a memory cell. This potential "power of staying silent" is exactly what makes our lower bounds in Section 3 interesting and non-trivial. Cook et al. [7] also prove a lower bound of $\Omega(\log n)$ on the number of rounds required to compute the OR function (and many others). This lower bound translates (via a simulation argument) to a lower bound of the form $\Omega(s^{-1} \log n)$ in our model (in [7] each processor reads only one cell in each round, while in our model each machine receives s bits per round). This implied lower bound is non-trivial only when $s = o(\log n)$, and our lower bound of $\Omega(\log_s n)$ is asymptotically superior for all super-constant s.

The work of [7] also inspired Nisan [29] to introduce the fundamental concept of the "block sensitivity" of a Boolean function (the logarithm of which characterizes the ideal PRAM round complexity, up to a constant factor), which in turn is polynomially related to the decision tree complexity, degree, and approximate degree of the function [29, 30]. Our results in Section 5 rely on this circle of ideas.

Finally, our technique of characterizing parallel computations as polynomials resembles the work of Dietzfelbinger et al. [9], who used related ideas to sharpen the lower bound in [7] by a constant factor for many Boolean functions. Using the degree of the polynomial to establish lower bounds was also used by Beals et al. [4] to establish query lower bounds for quantum networks.

[1] For a simple example, suppose the machines want to compute the logical OR of their nm input bits. In the congested clique model, this problem can be solved in 1 round. In our model, when the parameter s is less than n, more than 1 round is required.

[2] To match our barrier in Section 7 of resolving NC^1 vs. P using the techniques in [11], it seems necessary to prove logarithmic (rather than just super-constant) round lower bounds for congested clique computations.

2. THE S-SHUFFLE MODEL

Section 2.1 develops intuition for our computational model by presenting an example, and highlights some ways in which MapReduce-type computations differ from traditional circuit computations. Section 2.2 formally defines the model. Section 2.3 proves an upper bound on the round complexity of every Boolean function in the "unbounded-width" version of our model. Only Sections 2.2 and 2.3 are essential for understanding later sections.

2.1 A Warm-Up Example

Before formally defining our model, we study a specific example, adapted from Nisan and Szegedy [30].

Example 2.1 (Silence Is Golden) Consider the Boolean function $E_{12} : \{0,1\}^3 \to \{0,1\}$ on three inputs that evaluates to 1 if and only if exactly one or two inputs are 1. Define $E_{12}^2 : \{0,1\}^9 \to \{0,1\}$ as the Boolean function that takes nine inputs, applies E_{12} to each block of three inputs, and then applies E_{12} to the results of the three blocks. For instance:

- $E_{12}^2(0,0,1,0,1,0,1,0,0) = E_{12}(1,1,1) = 0$;
- $E_{12}^2(0,0,0,1,1,0,1,1,1) = E_{12}(0,1,0) = 1$.

We next describe a remarkably efficient strategy for computing E_{12}^2 in parallel, using only machines that operate on two (ordered) bits at a time. We first show how to compute E_{12} in two "rounds." Let (x_1, x_2, x_3) denote the input. The first machine reads the bits x_1 and x_2; the second machine x_2 and x_3; and the last machine x_3 and x_1. There is also a fourth machine which belongs to "the second round." Each machine in the first round sends a 1 to the second-round machine if its inputs are 0 and 1 (in this order); otherwise, it stays silent (i.e., sends nothing). The second-round machine receives either a 1 bit (if $E_{12}(x_1, x_2, x_3) = 1$) or no bits at all (if $E_{12}(x_1, x_2, x_3) = 0$), and in all cases can correctly determine and output the value of the function.

To compute the function E_{12}^2, we use a layered version of the same idea (see also Figure 1). We think of the nine input bits as three blocks of three bits each. There are nine machines in the first round, three for each block. There are three machines in the second round, each responsible for a pair of blocks. There is a single machine in the third round, responsible for the final output.

The three bits of a block are distributed to the corresponding three first-round machines (in ordered pairs) as in the computation of E_{12}. Second-round machines have two "ports" in that their inputs are also ordered; each has a "first input" (possibly empty) and a "second input" (again, possibly empty). If the inputs of a first-round machine are 0 and 1 (in this order), then the machine sends a 1 to the two second-round machines responsible for its block; otherwise, it sends nothing. The communication pattern between the three blocks of first-round machines and the second-round machines mirrors that of the distribution of bits to first-round machines: the first second-round machine receives bits (if any) from

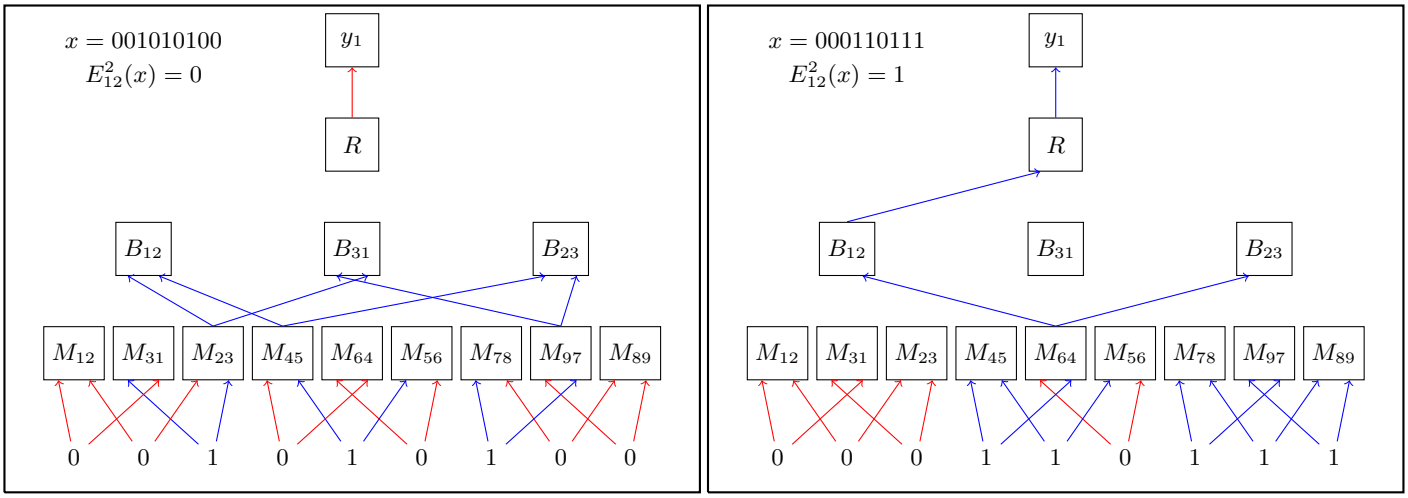

Figure 1: The parallel computation of E_{12}^2, for two different inputs. The arrows show how the machines send bits; an arrow pointing to the bottom left of a machine indicates that the bit is sent to the first port, while the bottom right indicates the second port. Notice that at most one arrow points to a given port, but the arrow that does so may change from input to input. Red arrows represent a zero bit and blue arrows represent a one bit.

the first and second blocks of first-round machines on its first and second ports, respectively; the middle second-round machine receives bits from the second and third blocks on its first and second ports, respectively; and the last second-round machine receives bits from the third and first blocks on its first and second ports, respectively. If a second-round machine receives nothing as its first input and a 1 as its second input, then it sends a 1 to the third-round machine; otherwise, it sends nothing. Finally, the third-round machine outputs 1 if it was sent a 1, and otherwise (in which case it receives nothing) it outputs 0. It is straightforward to verify that this computation correctly evaluates E_{12}^2 on all inputs. The computation uses three rounds of communication, with each machine receiving only two bits of input.

There are, unsurprisingly, many resemblances between this parallel computation and circuits — each picture in Figure 1 looks like a Boolean circuit, with machines corresponding to gates, the number of rounds corresponding to the depth, the number of machines corresponding to the size, and the number of input ports corresponding to the fan-in. We note, however, that no circuit with fan-in 2 and depth 3 (of any size) computes the function E_{12}^2, as the function depends on all nine of its inputs. This is true even for circuits with an alphabet larger than $\{0, 1\}$, such as $\{0, 1, \text{“nothing”}\}$. In general, while $\lceil \log_s n \rceil$ is an obvious lower bound on the depth of any circuit with fan-in s that computes a function that depends on all inputs, this lower bound does not necessarily hold for the number of rounds required by a MapReduce-type computation.

We emphasize that this parallel computation of E_{12}^2 can be trivially translated to a modern parallel processing infrastructure such as MapReduce (see also Appendix A). Any model that purports to capture arbitrary MapReduce-type computations, as opposed to restricted families of algorithms, should accommodate computations like the one above without significant overhead.

2.2 The Basic Model

What augmentations to standard circuit models are required to capture arbitrary MapReduce-type computations? The first two are evident from our parallel computation of E_{12}^2 in Section 2.1.

(1) The communication pattern, which plays the role of the circuit topology (i.e., which gates are connected to which), can be input-dependent. (cf., Figure 1.)

(2) Each machine has the option of staying silent and sending nothing. We refer to this as “sending a $\perp$.”

At first blush, extension (2) might seem equivalent to enlarging the alphabet by one character. This is not quite correct, since $\perp$'s combine with each other and with other bits in a particular way.

Definition 2.2 ($\perp$-sum) The *$\perp$-sum* of $z_1, z_2, \ldots, z_m \in \{0, 1, \perp\}$ is:

- 1 if exactly one z_i is 1 and the rest are $\perp$;
- 0 if exactly one z_i is 0 and the rest are $\perp$;
- $\perp$ if every z_i is $\perp$;
- undefined (or invalid) otherwise.

The $\perp$-sum of m s-tuples $a_1, \ldots, a_m$ is the entry-by-entry $\perp$-sum, denoted $\odot_{i=1}^m a_i$.

Most circuit models severely restrict the computational power of each gate. This is not appropriate in the present context, where each “gate” corresponds to a general-purpose machine embedded in a MapReduce-type infrastructure. This motivates our third extension.

(3) Each machine can perform an arbitrary computation on its inputs.[3]

[3]When the goal is to prove algorithmically meaningful upper bounds, it would be sensible to restrict machines to efficient computations. Also, the total space used by a machine, both for its input and for its computations, should be small (certainly sublinear in the total input size). Given our focus on lower bounds, our allowance of arbitrary computations and unlimited scratch space on each machine only makes our results stronger.

Our formal model extends the usual notion of a circuit (with fan-in s) to accommodate (1)–(3). Our notation follows that in Vollmer [38] for circuits.

Definition 2.3 (s-SHUFFLE Computation) An *R-round s-SHUFFLE computation with inputs $x_1, \ldots, x_n$ and outputs $y_1, \ldots, y_k$* has the following ingredients:

1. A set V of *machines*, which includes one machine for each input bit x_i and each output bit y_i.
2. An assignment of a *round* $r(v)$ to each machine $v \in V$. Machines corresponding to input bits have round 0. Machines corresponding to output bits have round $R+1$. All other machines have a round in $\{1, 2, \ldots, R\}$.
3. For each pair (u, v) of machines with $r(u) < r(v)$, a function α_{uv} from $\{0, 1, \perp\}^s$ to $\{0, 1, \perp\}^s$.

We think of each machine as having s "ports," where each port can accept at most 1 bit. The interpretation of a function α_{uv} is: given that machine u received $\mathbf{z} = z_1, \ldots, z_s$ on its s input ports (where each $z_i \in \{0, 1, \perp\}$), it sends the message $\alpha_{uv}(\mathbf{z}) \in \{0, 1, \perp\}^s$ to the s input ports of v.[4] Thus, machine u explicitly communicates with machine v (on at least one port) if and only if at least one coordinate of $\alpha_{uv}(\mathbf{z})$ is not $\perp$. We conclude that the model supports input-dependent communication patterns, as in (1). The model also supports (2) and (3), by definition.

The output of an s-SHUFFLE computation is evaluated as one would expect, as in circuits, with the important constraint that, on every input, each input port should receive a bit (i.e., a non-$\perp$) from at most one machine.

Definition 2.4 (Result of an s-SHUFFLE Computation) The *result* of an s-SHUFFLE computation assigns a value $g(v) \in \{0, 1, \perp\}^s$ to every machine $v \in V$, and is defined inductively as follows.

1. For a round-0 machine v, corresponding to an input bit x_i, the value $g(v)$ is the s-tuple $(x_i, \perp, \perp, \ldots, \perp)$.
2. Given the value $g(u)$ assigned to every machine u with $r(u) < q$, the value assigned to a machine v with $r(v) = q$ is the $\perp$-sum, over all machines u with $r(u) < r(v)$, of the message $\alpha_{uv}(g(u))$ sent to v by u:

$$g(v) := \odot_{u\,:\,r(u)<r(v)} \alpha_{uv}(g(u)). \qquad (1)$$

The result of an s-SHUFFLE computation is *valid* if every $\perp$-sum in equation (1) is well defined, and if for every machine v corresponding to an output bit y_i, the value $g(v)$ is either $(0, \perp, \perp, \ldots, \perp)$ or $(1, \perp, \perp, \ldots, \perp)$. The "0" or "1" in the first coordinate is then interpreted as the corresponding output bit y_i. Unless otherwise noted, we consider only valid s-SHUFFLE computations.

By convention, the machines corresponding to the input and output bits of the function do not contribute to the number of rounds — these are "placeholder machines" that cannot do any non-trivial computations. Translating the parallel computations in Section 2.1 of the functions E_{12} and E_{12}^2 into the formalism of Definitions 2.3 and 2.4 yields 2-round and 3-round s-SHUFFLE computations, respectively, in accordance with intuition.

The following definition is analogous to that for circuits.

[4]The model allows a machine to communicate with machines in all later rounds, not just machines in the next round. Computations of the former type can be translated to computations of the latter type by adding dummy machines, so this is not an important distinction.

Definition 2.5 (Width) The *width* of an s-SHUFFLE computation is the maximum number of machines in a round other than round 0 and the final round.

Remark 2.6 (Randomized Computations) By definition, a randomized s-SHUFFLE computation is a probability distribution over deterministic s-SHUFFLE computations.[5] We say that such a computation computes a function f if, for every input $\mathbf{x}$, the output of the computation equals $f(\mathbf{x})$ with probability at least $2/3$.

Remark 2.7 (Unordered Inputs and Larger Alphabets) Definitions 2.3 and 2.4 are easily extended to accommodate larger alphabets and unordered input ports. With these two extensions, it is straightforward to map an arbitrary r-round MapReduce computation that uses m machines with space at most s each (as formalized in [21], for example) to an $(r+1)$-round s-shuffle computation that uses $m(r+1)$ machines (not counting the machines that correspond to input and output bits). See Section 4 for formal proofs.

2.3 An Upper Bound for the Unbounded Width Model

We have argued that the s-SHUFFLE model in Definitions 2.3 and 2.4, and appropriate extensions thereof, capture arbitrary MapReduce-type computations, even those that perform exorbitant computation at each machine. What kind of lower bounds can we hope to prove for such a strong model?

To calibrate our aspirations, we next prove that, for every $s \geq 2$ and every function $f : \{0,1\}^n \to \{0,1\}^k$, f can be computed by an s-SHUFFLE computation in $\lceil \log_s n \rceil$ rounds. This upper bound requires s-SHUFFLE computations with unbounded width, and it delineates the strongest-possible lower bounds that that can be proved for the unbounded-width model. The prospects for stronger lower bounds for polynomial-width s-SHUFFLE computations are the subject of Section 7.

Proposition 2.8 For every $s \geq 2$, every function $f : \{0,1\}^n \to \{0,1\}^k$ can be computed by an s-SHUFFLE computation in $\lceil \log_s n \rceil$ rounds.

Proof (sketch): There is one machine $v_{\mathbf{x}}$ at round $\lceil \log_s n \rceil$ – the last round before the outputs – for each possible input $\mathbf{x} \in \{0,1\}^n$. Each machine $v_{\mathbf{x}}$ serves as the root of a tree of depth $\lceil \log_s n \rceil$, the machines of which are responsible for notifying $v_{\mathbf{x}}$ whether or not the input is $\mathbf{x}$. The unique machine $v_{\mathbf{x}}$ that is told that the input is indeed $\mathbf{x}$ sends the bits of the answer $f(\mathbf{x})$ to the k output machines; all other machines $v_{\mathbf{x}}$ send nothing. □

3. REPRESENTING SHUFFLES AS POLYNOMIALS

3.1 Representing the Basic Model

To prove lower bounds on the number of rounds required by an s-SHUFFLE computation, we associate each computation with a polynomial over the reals that matches (on $\{0,1\}^n$) the function it computes. We show that the number of rounds used by the computation governs the maximum degree of this polynomial, and hence

[5]This definition corresponds to "public coins," in that it allows different machines to coordinate their coin flips with no communication. One could also define a "private coins" model where each machine flips its own coins. Our lower bounds are for the stronger public-coin model, and apply to the private-coin model as a special case.

functions that correspond to polynomials of high degree cannot be computed in few rounds.

Theorem 3.1 *Suppose that an s-SHUFFLE computation computes the function $f : \{0,1\}^n \to \{0,1\}^k$ in r rounds. Then there are k polynomials $\{p_i(x_1,\ldots,x_n)\}_{i=1}^k$ of degree at most s^r such that $p_i(\mathbf{x}) = f(\mathbf{x})_i$ for all $i \in \{1,2,\ldots,k\}$ and $\mathbf{x} \in \{0,1\}^n$.*

PROOF. We proceed by induction on the number of rounds. We claim that for every non-output machine $v \in V$ and value $\mathbf{z} \in \{0,1,\perp\}^s$, there is a polynomial $p_{v,\mathbf{z}}(x_1,\ldots,x_n)$ that evaluates to 1 on points $\mathbf{x}$ for which the computation's assigned value $g(v)$ to v is $\mathbf{z}$ and to 0 on all other points $\mathbf{x} \in \{0,1\}^n$. Furthermore, $p_{v,\mathbf{z}}$ has degree at most $s^{r(v)}$.

The base case of the induction is for a machine v in round zero ($r(v) = 0$). Each such machine corresponds to an input bit x_i and its value $g(v)$ is $(x_i, \perp, \perp, \ldots, \perp)$. The (degree-1) polynomials for $\mathbf{z} = (0, \perp, \ldots, \perp)$ and $\mathbf{z} = (1, \perp, \ldots, \perp)$ are $p_{v,\mathbf{z}}(x_1,\ldots,x_n) = 1 - x_i$ and $p_{v,\mathbf{z}}(x_1,\ldots,x_n) = x_i$, respectively. All other values of $\mathbf{z}$ are impossible for such a machine, so they have polynomials $p_{v,\mathbf{z}}(x_1,\ldots,x_n) = 0$.

Consider a machine v that corresponds to neither an input bit nor an output bit. Fix some potential value $\mathbf{z} \in \{0,1,\perp\}^s$ of $g(v)$, and focus first on the i^{th} coordinate of $g(v)$. When is it equal to z_i? We first consider the case where z_i is 0 or 1 (and not $\perp$). The entry $g(v)_i$ is a $\perp$-sum of the messages that machine v receives on port i from all machines in previous rounds. Hence $g(v)_i = z_i$ when a single machine in a previous round sends z_i on port i to machine v (and the rest send $\perp$'s). Consider some machine u of a previous round. It sends z_i on port i to machine v when $\alpha_{uv}(g(u))$ has an i^{th} entry of z_i. The set of assigned values $g(u)$ to u that cause it to do this is the preimage, under α_{uv}, of the subset of $\{0,1,\perp\}^s$ containing all elements with a z_i in the ith coordinate. By our inductive hypothesis, for each value $g(u)$ in this preimage, there is a polynomial of degree at most $s^{r(v)-1}$ that indicates the inputs $\mathbf{x}$ for which u receives this value. Since u is assigned exactly one value, we can sum these polynomials together to get a polynomial (of degree at most $s^{r(v)-1}$) that indicates when u receives any value in this preimage. This polynomial indicates the inputs $\mathbf{x}$ for which u sends z_i to port i of machine v. Furthermore, since the s-SHUFFLE computation is valid, at most one machine in a previous round can send a non-$\perp$ value to v on port i. Summing over the polynomials of all machines in previous rounds yields a polynomial, still with degree at most $s^{r(v)-1}$, that indicates whether or not v received z_i on port i (from any machine).

The inputs $\mathbf{x}$ for which $g(v)_i = \perp$ are those for which no machine from a previous round sends a 0 or a 1 to v on the ith port. A polynomial for this case is just 1 minus the polynomials for the $g(v)_i = 0$ and $g(v)_i = 1$ cases. The degree of this polynomial is at most $s^{r(v)-1}$.

To obtain a polynomial that represents the inputs $\mathbf{x}$ for which $g(v) = \mathbf{z}$, we just take the product of the polynomials that represent the events $g(v)_1 = z_1$, ..., $g(v)_s = z_s$. This polynomial has degree at most $s^{r(v)}$, and this completes the inductive step.

Finally, consider a machine v that corresponds to an output bit y_i. As in the inductive step above, we can represent the first coordinate of $g(y_i)$ with a polynomial p_i of degree at most $s^{r(y_i)-1} = s^r$. Since the s-SHUFFLE computation is valid, the polynomial p_i also represents the ith output bit of the function f. □

See Section 4.1 for an extension of Theorem 3.1 to unordered inputs and larger alphabets.

The following corollary is immediate.

Corollary 3.2 *If some output bit of the function $f : \{0,1\}^n \to \{0,1\}^k$ cannot be represented by a polynomial with degree less than d, then every s-SHUFFLE computation that computes f uses at least $\lceil \log_s d \rceil$ rounds.*

Theorem 3.1 and Corollary 3.2 apply even to unbounded-width s-SHUFFLE computations. For functions f with $d = n$, the lower bound of $\lceil \log_s n \rceil$ in Corollary 3.2 matches the upper bound in Proposition 2.8 for arbitrary functions from $\{0,1\}^n$ to $\{0,1\}^k$.

There is a mature toolbox for bounding below the polynomial degree necessary to represent Boolean functions; we apply and contribute to this toolbox in Section 5.

3.2 Randomized Computations

We say that a polynomial $p(x_1,\ldots,x_n)$ *approximately represents* a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if $|p(\mathbf{x}) - f(\mathbf{x})| \leq \frac{1}{3}$ for every $\mathbf{x} \in \{0,1\}^n$. The *approximate degree* of a Boolean function is the smallest value of d such that f can be approximately represented by a degree-d polynomial. We next give an analog of Theorem 3.1 for randomized s-SHUFFLE computations (Remark 2.6), which connects the rounds required by such computations to the approximate degree of Boolean functions.

Theorem 3.3 *Suppose that a randomized s-SHUFFLE computation computes the function $f : \{0,1\}^n \to \{0,1\}^k$ in r rounds. Then there are k polynomials $\{p_i(x_1,\ldots,x_n)\}_{i=1}^k$ of degree at most s^r such that $|p_i(\mathbf{x}) - f(\mathbf{x})_i| \leq \frac{1}{3}$ for all $i \in \{1,2,\ldots,k\}$ and $\mathbf{x} \in \{0,1\}^n$.*

PROOF. A randomized s-SHUFFLE computation is a distribution over deterministic computations. By Theorem 3.1, we can represent each output bit of these deterministic computations by a polynomial of degree at most s^r. The weighted averages of these polynomials, with the weights equal to the probabilities of the corresponding deterministic computations, yields polynomials $p_1,\ldots,p_k$ such that, for every i and $\mathbf{x} \in \{0,1\}^n$, $p_i(\mathbf{x})$ equals the probability that the randomized s-SHUFFLE computation outputs a 1 on the input $\mathbf{x}$. Since the randomized computation computes f in the sense of Remark 2.6, these polynomials satisfy the conclusion of the theorem. □

Corollary 3.4 *If some output bit of the function $f : \{0,1\}^n \to \{0,1\}^k$ has approximate degree at least d, then every randomized s-SHUFFLE computation that computes f uses at least $\lceil \log_s d \rceil$ rounds.*

Section 5 applies tools for bounding from below the approximate degree of Boolean functions to derive round lower bounds for randomized s-SHUFFLE computations.

4. LARGER ALPHABETS AND UNORDERED INPUTS

Definitions 2.3 and 2.4 are easily extended to accommodate larger alphabets and unordered input ports. With these two extensions, it is straightforward to map an arbitrary r-round MapReduce computation that uses m machines with space at most s each (as formalized in [21], for example) to an $(r+1)$-round s-shuffle computation that uses $m(r+1)$ machines (not counting the machines that correspond to input and output bits). The machines in a round of the s-shuffle computation are responsible for simulating the computation and communication that occurs in the corresponding round of the MapReduce computation. MapReduce computations operate on $\langle key; value \rangle$ pairs; the id of a machine plays the role of the

key, and the alphabet in the s-shuffle computation corresponds to the set of all possible values in the MapReduce computation. See Appendix A for more details.

To formalize the extensions in more detail: with an arbitrary alphabet Σ, the input belongs to Σ^n and the output to Σ^k. Each machine u receives a value $g(u)$ and sends messages $\alpha_{uv}(g(u))$ belonging to $\{\Sigma \cup \{\perp\}\}^s$. A $\perp$-sum of m values from $\Sigma \cup \{\perp\}$ is defined in the obvious way whenever at most 1 of the values is not $\perp$ (and is undefined otherwise).

To incorporate unordered input ports, the functions α_{uv} are redefined to have domain and range equal to the multi-sets of Σ of cardinality at most s. A $\perp$-sum of such multi-sets is their union (with multiplicities adding), and is undefined if the sum of the multiplicities exceeds s.

4.1 Shuffles as Polynomials

We now outline how to modify Theorem 3.1 so that it applies to the extended model in Section 4. To represent a function $f : \Sigma^n \to \Sigma^k$, we use $k|\Sigma|$ polynomials. Every polynomial has one variable $x_{i\sigma}$ for each $i \in \{1, 2, \ldots, n\}$ and $\sigma \in \Sigma$. Inputs $\mathbf{x} \in \Sigma^n$ translate to binary $0/1$ inputs in the variable set $\{x_{i\sigma}\}$ in the obvious way. For $i \in \{1, 2, \ldots, k\}$ and $\sigma \in \Sigma$, the polynomial $p_{i\sigma}$ indicates the inputs for which the ith output bit f is σ.

In the proof of Theorem 3.1, a potential value $\mathbf{z}$ of a machine $g(v)$ is now a multi-set of Σ or size at most s, rather than an element of $\{0, 1, \perp\}^s$. We can no longer argue coordinate-by-coordinate. To extend the inductive argument in the proof, consider a machine v and suppose first that $\mathbf{z}$ is a multi-set of size exactly s. For a given partition of $\mathbf{z}$ into $\ell \leq s$ non-empty sets, and a given assignment of these sets to ℓ machines in rounds before $r(v)$, the event that v receives the value $\mathbf{z}$ in precisely this way can be inductively expressed as a polynomial with degree at most $s^{r(v)}$. (Because $\mathbf{z}$ has the maximum-allowable size s and the computation is valid, the other machines must send nothing to v.) For a multi-set $\mathbf{z}$ with size less than s, the event that v receives some superset of $\mathbf{z}$ can be likewise expressed as a polynomial with degree at most s^r. Subtracting out the events corresponding to strict supersets of $\mathbf{z}$ — formally, using downward induction on the size of $\mathbf{z}$ — completes the representation of the event that $g(v) = \mathbf{z}$ as a polynomial with degree at most s^r.

Theorem 4.1 *Suppose that an s-SHUFFLE computation computes the function $f : \Sigma^n \to \Sigma^k$ in r rounds. Then for every $i \in \{1, 2, \ldots, k\}$ and $\sigma \in \Sigma$, the event that $f(\mathbf{x})_i = \sigma$ (for $\mathbf{x} \in \Sigma^n$) can be represented as a polynomial of degree at most s^r over the variable set $\{x_{i\sigma}\}_{i=1,\ldots,n,\sigma\in\Sigma}$.*

4.2 Lower Bounds

This section extends our lower bounds to non-Boolean alphabets, in two senses. The first case is when the computational problem is still boolean, but the word size allowed for intermediate computations is larger than a single bit. The second case is when the computational problem itself is encoded using a larger alphabet.

For the first case, we show that the lower bound of Corollary 3.2 applies unchanged to larger alphabets. We explore the second case in Section 6.4, where we show that for undirected ST-CONNECTIVITY the degree of the function remains large, even in the case of richer alphabets.

Corollary 4.2 *If some output bit j of the function $f : \{0,1\}^n \to \{0,1\}^k$ cannot be represented by a polynomial with degree less than d, then every s-SHUFFLE computation over the alphabet Σ that computes f uses at least $\lceil \log_s d \rceil$ rounds.*

PROOF. Suppose that some s-SHUFFLE computation over the alphabet Σ computes f in less than $\lceil \log_s d \rceil$ rounds. By Theorem 4.1, there is a polynomial p, over the variable set $\{x_{i\sigma}\}_{i\in[n],\sigma\in\Sigma}$, that represents the event when the jth output bit is 1 with degree less than d. Plugging in $(1 - x_i)$ for x_{i0}, x_i for x_{i1}, and 0 for all other variables $x_{i\sigma}$, we obtain a polynomial p' in the variables $\{x_i\}_{i\in[n]}$ with degree less than d, contradicting the assumption. We conclude that every s$-$SHUFFLE computation over Σ that computes f requires at least $\lceil \log_s d \rceil$ rounds. □

5. LOWER BOUNDS FOR POLYNOMIAL DEGREE

Corollary 3.2 and its extensions reduce the problem of proving lower bounds on the round complexity of s-SHUFFLE computations to proving lower bounds on the degree of a polynomial that exactly or approximately represents the function to be computed. There is a sophisticated set of tools for proving the latter type of lower bounds, which we now put to use.

5.1 Warm Up

Consider a Boolean function $f : \{0,1\}^n \to \{0,1\}$. If f can be represented by a polynomial p, meaning $f(\mathbf{x}) = p(\mathbf{x})$ for all $\mathbf{x} \in \{0,1\}^n$, then it can be represented by a multilinear polynomial (since $x_i^2 = x_i$ for $x_i \in \{0,1\}$). Recall that for every such function f, there is a unique multilinear polynomial that represents it (see e.g. [10]).[6] We call the degree of this polynomial the *degree* of the Boolean function. The maximum-possible degree is n.

For example, since the AND_n function is represented (uniquely) by the polynomial $\prod_{i=1}^n x_i$, it has the maximum-possible degree. Similarly, the OR_n function has degree n because it is represented by the polynomial $1 - \prod_{i=1}^n (1 - x_i)$. Corollary 3.2 immediately implies the following.

Corollary 5.1 *Every s-SHUFFLE computation that computes the AND_n or the OR_n function uses at least $\lceil \log_s n \rceil$ rounds.*

5.2 Monotone Graph Properties

While pinning down the precise degree of a Boolean function representing a graph problem can be a difficult task, there are powerful tools for proving loose but useful lower bounds.

The first ingredient concerns the decision tree complexity of monotone graph properties. Recall that a graph property is a property of undirected graphs that is independent of the vertex labeling. It is non-trivial if it does not assign the same value to all graphs, and monotone if adding edges cannot destroy the property. The Aanderaa-Rosenberg conjecture states that, for every non-trivial monotone graph property, the decision-tree complexity is $\Omega(n^2)$ [35]. It was first proved by Rivest and Vuillemin with a lower bound of $n^2/16$ [34], and a long line of work has yielded a lower bound of $n^2/3 - o(n^2)$ [23, 20, 24, 36].

The second ingredient is the known polynomial relationship between the decision tree complexity and the degree of a Boolean function. Specifically, Nisan and Smolensky (cited in [6]) proved that, for every Boolean function f, the decision-tree complexity of f is at most $2\deg(f)^4$, where $\deg(f)$ is the degree of f.

Combining these two ingredients with Corollary 3.2 yields the following.

[6] With a non-Boolean alphabet and the expanded variable set used in Section 4.1, there can be more than one representing polynomial. This does not affect our lower bounds; see Section 4.2.

Theorem 5.2 *For every non-trivial monotone graph property $f : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ of graphs with n vertices, every s-SHUFFLE computation that computes f requires at least $\frac{1}{2}\log_s n - \frac{1}{4}\log_s(6 - o(1))$ rounds.*

5.3 Approximate Degree and Randomized Computations

Recall from Corollary 3.4 that lower bounds on the approximate degree of a Boolean function translate to lower bounds on the number of rounds required by randomized s-SHUFFLE computations. It is also known that both the degree and decision tree complexity of every Boolean function are polynomially related to its approximate degree: $\deg(f) \leq D(f) \leq 216 \cdot \widetilde{\deg}(f)^6$, where $\widetilde{\deg}$ denotes the approximate degree [4, 30].[7] We therefore have the following analog of Theorem 5.2 for randomized computations.

Theorem 5.3 *For every non-trivial monotone graph property $f : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ of graphs with n vertices, every randomized s-SHUFFLE computation that computes f requires at least $\frac{1}{3}\log_s n - \frac{1}{6}\log_s(648 - o(1))$ rounds.*

In general, round lower bounds on deterministic s-SHUFFLE computations proved via Corollary 3.2 extend automatically, with a small constant-factor loss, to randomized computations.

6. LOWER BOUNDS FOR GRAPH COMPUTATIONS

The goal of this section is develop and apply a technique for showing that many problems that are important in the field of algorithms (as opposed to in Boolean function analysis) — such as graph connectivity problems — have maximum-possible degree.

Buhrman and de Wolf [6] credit Yaoyun Shi with the observation that a Boolean function has degree n if and only if the number of even solutions (i.e., assignments with an even number of 1s for which the function evaluates to 1) is not equal to the number of odd solutions. For the reader familiar with Boolean Fourier analysis, this corresponds to computing whether $\hat{f}([n])$, the function's Fourier coefficient for the set $[n]$, is nonzero. For example, the function XOR_n has no even solutions and 2^{n-1} odd solutions, so the function has degree n and hence (by Corollary 3.2) s-SHUFFLE computations require $\lceil \log_s n \rceil$ rounds to compute it.

6.1 Parity Difference Preliminaries

We first establish some notation and lemmas that simplify the proofs.

Definition 6.1 Given a set $S \subseteq \{0,1\}^n$, define the *parity difference function* as the number of even inputs in S minus the number of odd inputs in S:

$$\Phi(S) = \sum_{x \in S} \prod_i (-1)^{x_i}.$$

Define the *parity difference* of a Boolean function f on n bits by $\Phi(f) = \Phi(\{x \mid f(x) = 1\})$.

In Fourier-analytic terms, $\Phi(f)$ is exactly $2^n \hat{f}([n])$. We have the following identities.

[7]For some specific functions, better bounds are known. For example, the majority function has approximate degree $\Omega(\sqrt{n})$ [33].

Lemma 6.2 *(a) Suppose that $S \subseteq \{0,1\}^n$ and S_1, S_2 form a partition of S. Then*

$$\Phi(S) = \Phi(S_1) + \Phi(S_2).$$

(b) Suppose that $S \subseteq \{0,1\}^n$ with S the Cartesian product $S_1 \times S_2$ of sets S_1, S_2. Then

$$\Phi(S) = \Phi(S_1) \cdot \Phi(S_2).$$

PROOF. The lemma follows immediately from basic boolean Fourier analysis, if we consider the functions that represent the characteristic vectors of S, S_1, and S_2.

It is also easy to prove the lemma directly from definitions. For part (a), using the partition assumption, we have

$$\sum_{x \in S} \prod_i (-1)^{x_i} = \sum_{x \in S_1} \prod_i (-1)^{x_i} + \sum_{x \in S_2} \prod_i (-1)^{x_i}.$$

For part (b), we can split each $x \in S$ according to the product $S_1 \times S_2$ to factorize the sum:

$$\begin{aligned}\sum_{x \in S} \prod_i (-1)^{x_i} &= \sum_{x^1 \in S_1} \sum_{x^2 \in S_2} \prod_i (-1)^{x_i^1} \prod_i (-1)^{x_i^2} \\ &= \sum_{x^1 \in S_1} \prod_i (-1)^{x_i^1} \sum_{x^2 \in S_2} \prod_i (-1)^{x_i^2}.\end{aligned}$$

□

6.2 Graph Connectivity Problems.

To apply these ideas to graph connectivity problems, we need to express graph problems as Boolean functions. One natural way is to use a $\binom{n}{2}$-variable input to represent an undirected graph on n vertices (or a $2\binom{n}{2}$-variable input for directed graphs), where each variable indicates the presence of absence of a given edge. This is effectively the adjacency matrix representation of the graph. (For lower bounds for the adjacency list representation, see Section 4.)

The CONNECTIVITY problem is: given a graph $G = (V, E)$, is there a path from every vertex to every other vertex? The related ST-CONNECTIVITY problem: given a graph $G = (V, E)$ and two vertices $s, t \in V$, is there a path from s to t? We can assume that s is the first vertex and t is the second vertex. We pinpoint the degree of both of these problems for both undirected and directed graphs. The proof strategy is similar for all four cases: we use induction on the number of vertices n, show how large problems can be decomposed into smaller problems, and use this decomposition to analyze the highest-degree Fourier coefficient.

6.3 Undirected Connectivity Functions

We begin with the undirected case (Theorems 6.3 and 6.4). Recall that to prove that a function has maximum-possible degree, it suffices to show that its parity difference is non-zero.

Theorem 6.3 *The degree of the Boolean function for undirected* CONNECTIVITY *on n vertices is $\binom{n}{2}$.*

PROOF. For a graph on n vertices, let f_n be the Boolean function for undirected CONNECTIVITY. We prove that

$$\Phi(f_n) = (-1)^{n-1}(n-1)!, \tag{2}$$

which implies the theorem.

We proceed by induction on n. The base case $n = 1$ is trivial; the graph is always connected and never has any edges, so $\Phi(f_1) = 1$.

Suppose that our inductive hypothesis is true for all smaller values of n. Consider the first vertex of our graph on n vertices. If

we begin with a connected graph and remove this vertex, we obtain a partition of the remaining $n-1$ vertices into connected components. We can split the set S of all connected graphs into sets S_P which yield the same partition P of connected components when the first vertex is removed. By Lemma 6.2(a), $\Phi(S) = \sum_P \Phi(S_P)$.

Consider a particular partition P of $n-1$ vertices. Each block of the partition must be connected, and the first vertex of the graph must be connected to each partition block with at least one edge. Thus, S_P can be thought of as a cross product between the set of ways to connect the first vertex to the first block, the set of ways to connect the first block, the set of ways to connect the first vertex to the second block, the set of ways to connect the second block, and so on. By Lemma 6.2(b), it suffices to compute the parity difference of each of these sets separately, and take their product.

First, we consider the set of ways to connect the first vertex to a block with k vertices. There are k possible edges, and the only forbidden choice is the one with no edges. Hence there are 2^{k-1} odd choices but only $2^{k-1}-1$ even choices (the missing choice is even), and the parity difference of this set is -1.

By induction, the set of ways of connecting a block of size k has parity difference $(-1)^{k-1}(k-1)!$. Multiplying this with the parity difference of the set of ways that the first vertex can be connected to the block, the factor of a partition's parity difference corresponding to a block of size k is $(-1)^k(k-1)!$. The contribution of a single partition is the product of such terms, with one term per block.

Finally, we need to sum the parity differences over all partitions. Since these are all partitions of $n-1$ vertices, they all have the same sign: $(-1)^{n-1}$. We need to show that the total magnitude is $(n-1)!$ to complete the proof.

To see this, consider all $(n-1)!$ permutations of $n-1$ items. If we write them in cycle notation, we can view the cycles of a permutation as yielding a partition of $n-1$ vertices. A block of k vertices can be created by any of $(k-1)!$ possible cycles on these vertices. Multiplying over the blocks of a partition, we see that the number of permutations that map to a partition is exactly the magnitude of the parity difference of the partition. This implies that the total parity difference magnitude is the number $(n-1)1$ of permutations, which completes the inductive hypothesis and the proof. □

Theorem 6.4 *The degree of the Boolean function for undirected* ST-CONNECTIVITY *on n vertices is $\binom{n}{2}$.*

PROOF. For a graph on n vertices, et f_n be the Boolean function for undirected ST-CONNECTIVITY. We can assume that s and t are the first and second vertices. We show that $\Phi(f_n) = (-1)^{n-1}(n-2)! \neq 0$.

It is convenient to analyze $-\Phi(f_n)$ and consider all graphs in which s and t are *not* connected. Let $S_{A,B}$ be the subset of such graphs in which the connected component of s is A and the connected component of t is B. By Lemma 6.2(a), $-\Phi(f_n)$ is the sum of the $\Phi(S_{A,B})$'s for all such A, B.

Fix A and B. Both are connected, and the remainder of the vertices of the graph can have an arbitrary subset of edges within it. We can therefore think of $S_{A,B}$ as a product of the set of ways to connect A, the set of ways to connect B, and, if there are at least two vertices outside $A \cup B$, the set of ways to pick edges in the remainder of the graph. By Lemma 6.2(b), we only need to compute the parity differences of each of these sets and take their product.

First, we observe that if $V \setminus (A \cup B)$ contains more than one vertex, then the parity difference of the last set (all subsets of edges of $V \setminus (A \cup B)$) is zero. For the rest of the proof, we focus on sets A, B whose union includes all vertices, except possibly for one.

If A has a vertices and B has b vertices, then by (2) the parity difference for the set of ways to connect A is $(-1)^{a-1}(a-1)!$ and the parity difference for the set of ways to connect B is $(-1)^{b-1}(b-1)!$.

Since $V \setminus (A \cup B)$ has zero or one vertex, there are $\frac{(n-2)!}{(a-1)!(b-1)!}$ ways to choose a set A of size a and a set B of size b. (In the latter case, there are $n-2$ choices for the missing vertex, and $\binom{n-3}{a-1}$ ways of selecting the vertices that join s in A.) Thus, the total contribution to the parity difference of sets A, B with sizes a, b is $(-1)^{n-2}(n-2)!$ (if $a+b=n$) or $(-1)^{n-3}(n-2)!$ (if $a+b=n-1$). There are $n-1$ ways to choose $a \geq 1$ and $b \geq 1$ so that $a+b=n$, and $n-2$ ways to choose $a \geq 1$ and $b \geq 1$ so that $a+b=n-1$. Our final parity difference (for $-\Phi$) is

$$(n-1)(-1)^{n-2}(n-2)! + (n-2)(-1)^{n-3}(n-2)! = (-1)^{n-2}(n-2)!.$$

This completes the proof. □

6.4 Connectivity with Large Alphabets

We consider the ST-CONNECTIVITY problem, except the graph is no longer presented as an adjacency matrix as in the boolean case. Instead, we are presented with a list of m edges. We model these as m inputs from an alphabet of size $\binom{n}{2}$. We will even assume the algorithm is promised that these edges never repeat (if they could, it would be fairly easy to prove difficulty from OR). We have a bound on degree given that there are enough vertices in the graph. Note that some such condition is necessary, since too few vertices would guarantee that nodes 0 and 1 are always connected. Our vertex requirement is only a constant factor more than the minimum n to accommodate m distinct edges.

Theorem 6.5 *Let f be the function associated with the problem of undirected* ST-CONNECTIVITY *presented as an adjacency list. Any polynomial associated with this function being true has degree at least m, given $n \geq \sqrt{8m}$.*

PROOF. Again, we will show this by plugging variables into such a polynomial. Let p be a polynomial over $x_{i\sigma}$ with degree at least m which is 1 when s and t are connected and 0 otherwise.

Each edge of the input originally has $\binom{n}{2}$ possibilities, but our plan is actually to further constrain them; we promise each edge of the input will be one of two possible values. We do this in such a way that exactly one of these 2^m possibilities results in s and t being unconnected. Equivalently, we can plug in y_i, $(1-y_i)$, or 0 for every $x_{i\sigma}$ in a way such that we get the polynomial of degree at least m (we get the polynomial for OR). But the degree of p can only decrease when we perform this operation, so this contradicts our assumption about the degree of p. Hence we will have shown that no such p exists.

As usual, we assume that s and t are vertices 0 and 1. We want to choose pairs so that only one possibility does not connect 0 and 1. The first set of inputs are chosen so that this possibility connects all odd vertices and connects all even vertices. Our first input is either $(0,1)$ or $(0,2)$, our second input is either $(1,2)$ or $(1,3)$, and our third input is either $(2,3)$ or $(2,4)$. We continue in this fashion. Notice if we ever pick the first possibility, 0 and 1 must be connected. This continues so that either 0 and 1 are connected or all even vertices are connected and all odd vertices are connected. Then the remaining inputs offer an edge between vertices of opposite parity, which will connect 0 and 1, or between vertices of the same parity. Note that we never repeat an edge in this process. This yields are least $\frac{n^2}{8}$ pairs, which is more than m if $n \geq \sqrt{8m}$. □

6.5 Directed Connectivity Functions

For the more complicated case of directed graphs, we state the theorems below, but defer the proofs to the full version of the paper.

Theorem 6.6 *For a graph of n vertices, the degree of the Boolean function for directed* CONNECTIVITY *is* $2\binom{n}{2}$.

Theorem 6.7 *For a graph of n vertices, the degree of the Boolean function for directed* ST-CONNECTIVITY *n vertices is* $\binom{n}{2}$.

7. PROSPECTS FOR STRONGER LOWER BOUNDS

Our lower bounds in Section 3 and 5 apply to general unbounded-width s-SHUFFLE computations, and by Proposition 2.8 such lower bounds cannot be larger than $\lceil \log_s n \rceil$. Realizable MapReduce-type computations translate to s-SHUFFLE computations with a reasonable number of machines, so a natural question is whether or not stronger lower bounds hold for a polynomial number of machines. For example, can we prove that there is no $O(1)$-round s-SHUFFLE computation for graph connectivity problems with $s = \sqrt{n}$ and a polynomial number of machines? In this section we show that proving such stronger impossibility results requires proving circuit lower bounds.

The following result, while not difficult to prove, has significant implications for the prospects of strong lower bounds for parallel computation.

Theorem 7.1 *Consider any model of parallel computation with the following properties:*

1. *The number of machines is polynomial in the input size n.*
2. *Computation proceeds in time steps. In each time step, a machine can read $s(n) \geq 2$ bits from the input or from previous rounds of computation.*
3. *Each machine can output at least a single bit, which is a function of the $s(n)$ bits it read. The possible functions a machine can compute need only include Boolean formulas where each variable appears once.*

Suppose some problem in P cannot be solved in $O(\log_{s(n)} n)$ time steps in such a model. Then $NC^1 \subsetneq P$.

PROOF. We simply need to show that any model with these features can still simulate NC^1 circuits. Recall that NC^1 is the family of problems that can be computed by a logspace-uniform circuit family C_n of fan-in 2 with $poly(n)$ gates and $O(\log n)$ depth. Fix some such model of parallel computation. We will transform each circuit into a computation in our model which runs in $\log_{s(n)} n$ time steps. This will show that no NC^1 family computes our hypothesized problem, and hence $NC^1 \subsetneq P$, as desired.

The computation will compute the output of every circuit gate with a machine. Each time step will correspond to $\log s(n)$ layers of the circuit. For example, in the first time step, we will have a machine for every gate at depth at most $\log s(n)$. Since the circuit had fan-in 2, each gate's output can be determined from at most $s(n)$ inputs. Furthermore, the output as a function of these inputs is a Boolean formula where each of these $s(n)$ inputs appears only once. In general, in the i^{th} time step, we have a machine for every gate at depth between $(i-1)\log s(n)$ and $i \log s(n)$. Again, these can be determined from at most $s(n)$ inputs or gates in previous time steps. We get to output gates within $O(\frac{\log n}{\log s(n)}) = O(\log_{s(n)} n)$ time steps. Each gate becomes a single machine, for $poly(n)$ machines. Figure 2 illustrates this simulation for a specific circuit of depth four. □

The barrier of $\Theta(\log_{s(n)} n)$ for lower bounds in Theorem 7.1 is the same as the lower bounds we proved, even for the unbounded-width model, in Sections 3 and 5.

Property (1) states that an arbitrarily large polynomial number of machines should be allowed. Property (2) asks for a complete communication graph and concurrent reads, and property (3) only requires fairly weak computing power per machine. s-SHUFFLE computations with a polynomial number of machines are certainly one example of a model that satisfies these assumptions. The point of Theorem 7.1 is that, more generally, the *only possible way* of proving lower bounds stronger than $\Theta(\log_s n)$ for any model of parallel computation is to either (i) prove a notoriously difficult circuit lower bound or (ii) restrict the model so much that one of the hypotheses in the theorem is not satisfied. For example, Theorem 7.1 leaves open the possibility of proving strong lower bounds for restricted classes of algorithms (this violates the third assumption); some of the previous work reviewed in Section 1.2 is of this type. Another possible approach to proving stronger lower bounds is to restrict the number of machines to be very small, for example at most constant factor larger than the minimum number n/s needed to read the input.[8]

The simulation argument in the proof of Theorem 7.1 is quite versatile. For example, it also works in the uniform setting. Since NC^1 is defined to be logspace-uniform, the lower bound barrier holds even if the model of parallel computation is restricted to logspace-uniform computations. This simulation also only blows up the width by a factor of $\log s$ from circuit to parallel computation. Hence, given a round lower bound on a width-restricted model of parallel computation, we would get a depth lower bound on a width-restricted (less so by a factor $\log s$) version of NC, which would still be an interesting result for circuit complexity. Similarly, the simulation results in the same number of machines as there were gates, so a round lower bound for a machine-restricted model of parallel computation would give a depth lower bound for a size-restricted version of NC.

8. REFERENCES

[1] Foto N. Afrati, Anish Das Sarma, Semih Salihoglu, and Jeffrey D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

[2] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583, 2014.

[3] MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab S. Mirrokni. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2591–2599, 2014.

[4] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, 2001.

[8] Limiting the number of machines to be at most a specific function of n, rather than to an arbitrary polynomial n, has various technical drawbacks, such as losing the self-reducibility of many problems.

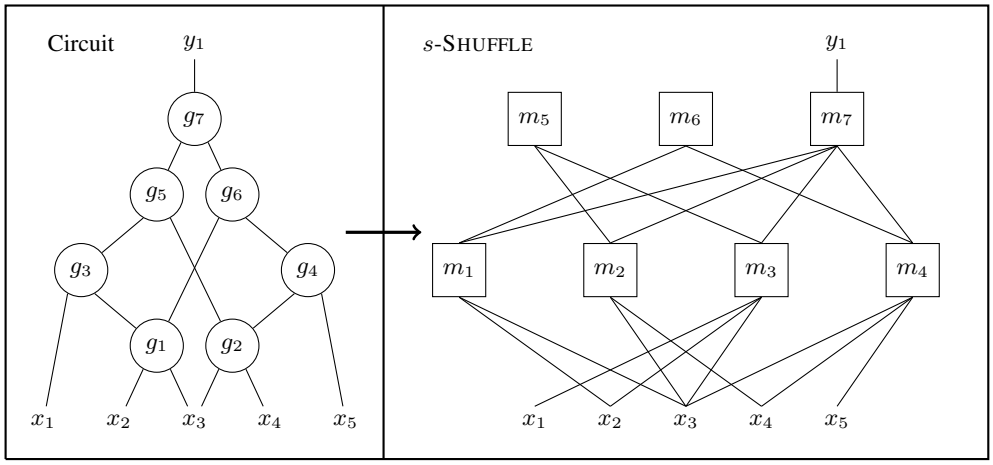

Figure 2: A depth-four circuit and its corresponding s-SHUFFLE with $s = 4$ that runs in two rounds. The number on a machine matches the number of the gate it computes.

[5] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 273–284, 2013.

[6] Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[7] Stephen Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing*, 15(1):87–97, 1986.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.

[9] Martin Dietzfelbinger, Mirosław Kutyłowski, and Rüdiger Reischuk. Exact lower time bounds for computing boolean functions on crew prams. *Journal of Computer and System Sciences*, 48(2):231–254, 1994. Preliminary version in *SPAA '90*.

[10] R. O' Donnell. *Analysis of Boolean Functions*. Cambridge, 2014.

[11] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376. ACM, 2014.

[12] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 681–689, 2011.

[13] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, and Zoya Svitkina. On distributing symmetric streaming computations. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 710–719, 2008.

[14] Benjamin Fish, Jeremy Kun, Ádám Dániel Lelkes, Lev Reyzin, and György Turán. On the computational complexity of mapreduce. *CoRR*, abs/1410.0245, 2014.

[15] Ashish Goel and Kamesh Munagala. Complexity measures for map-reduce, and comparison to parallel computing. *CoRR*, abs/1211.6526, 2012.

[16] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pages 374–383, 2011.

[17] James W. Hegeman and Sriram V. Pemmaraju. Lessons from the congested clique applied to MapReduce. *Theoretical Computer Science*, 608:268–281, 2015.

[18] Pavel Hrubes and Anup Rao. Circuits with medium fan-in. In *Proceedings of CCC*, 2015.

[19] Riko Jacob, Tobias Lieber, and Nodari Sitchinava. On the complexity of list ranking in the parallel external memory model. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 384–395, 2014.

[20] Jeff Kahn, Michael Saks, and Dean Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984.

[21] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948, 2010.

[22] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 391–410, 2015.

[23] Daniel J. Kleitman and D. J. Kwiatkowski. Further results on the aanderaa-rosenberg conjecture. *J. Comb. Theory, Ser. B*, 28(1):85–95, 1980.

[24] Torsten Korneffel and Eberhard Triesch. An asymptotic bound for the complexity of monotone graph properties. *Combinatorica*, 30(6):735–743, 2010.

[25] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pages 1–10, 2013.

[26] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94, 2011.

[27] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge, 2014. Second Edition.

[28] Vahab Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *Symposium on Theory of Computing, STOC 2015, Portland, OR, USA*, 2015.

[29] Noam Nisan. Crew prams and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991.

[30] Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational complexity*, 4(4):301–313, 1994.

[31] David Peleg. Distributed computing. *SIAM Monographs on discrete mathematics and applications*, 5, 2000.

[32] Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. Space-round tradeoffs for mapreduce computations. In *International Conference on Supercomputing, ICS'12, Venice, Italy, June 25-29, 2012*, pages 235–244, 2012.

[33] A.A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[34] Ronald L Rivest and Jean Vuillemin. A generalization and proof of the aanderaa-rosenberg conjecture. In *Proceedings of seventh annual ACM symposium on Theory of computing*, pages 6–11. ACM, 1975.

[35] Arnold L Rosenberg. On the time required to recognize properties of graphs: A problem. *ACM SIGACT News*, 5(4):15–16, 1973.

[36] Robert Scheidweiler and Eberhard Triesch. A lower bound for the complexity of monotone graph properties. *SIAM Journal on Discrete Mathematics*, 27(1):257–265, 2013.

[37] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[38] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 1999.

[39] David P. Woodruff and Qin Zhang. When distributed computation is communication expensive. In *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, pages 16–30, 2013.

APPENDIX

A. SIMULATING MAPREDUCE

To illustrate the power of the s-SHUFFLE model, this appendix shows how an s-SHUFFLE computation can be used to simulate a MapReduce computation. We focus specifically on the $\mathcal{MRC}$ class defined in [21].

Recall that in the MapReduce programming paradigm, the basic unit of information is a $\langle key; value\rangle$ pair. The computation in a given round is defined by two sets of functions, mappers and reducers. A mapper, μ is a function that takes as input some number of $\langle key; value\rangle$ pairs and, independently for each such pair, produces a finite multiset of $\langle key; value\rangle$ pairs. A reducer, ρ is a function that takes as input a binary string k, representing the key, and a set of values $v_1, v_2, \ldots$, and outputs a set of $\langle key; value\rangle$ pairs, all with key k. We can think of the set of reducers as being indexed by the corresponding key k.

For the simulation below it will be useful to have an explicit bound on the size of any $\langle key; value\rangle$ pair.

Lemma A.1 *Any randomized $\mathcal{MRC}$ computation can be simulated round per round with an $\mathcal{MRC}$ computation where every $\langle key; value\rangle$ pair has at most $w = O(\log n)$ bits.*

Proof (sketch): First suppose that in a $\langle key; value\rangle$ pair, the length of the key is larger than $3\log n$ bits. We can transform the pair into $\langle h(key); key\$value\rangle$, where h is a universal hash function onto $3\log n$ bits. Since there are at most $n^{2-2\epsilon}$ unique keys, with high probability each long key is mapped to a unique short key, thereby keeping the logic of the computation intact.

Now consider a $\langle key; value\rangle$ pair where the value string $s = v_1v_2v_3\ldots v_k$ has $k > c = w - 4\log n$ bits. We can break the string into substrings $s_1 = v_1v_2\ldots v_t$, $s_2 = v_{t+1}v_{t+2}\ldots v_{2t}$, and so on. We then transform the pair $\langle key, s\rangle$ into a set of $\lceil k/c\rceil$ pairs $\langle key, u\$i\$s_i\rangle$, where u is a random $2\log n$ bit integer, and \$ is a unique symbol. Since the length of the original string was at most $n^{1-\epsilon}$, each value now has at most w bits. Since all of these pairs have the same key they, they will be shuffled to the same reducer. Provided the integers u generated for each value string we break up are unique (a high probability event since there are at most $n^{2-\epsilon}$ pairs), the machine can use the indexes to reconstruct the original value string. □

Proposition A.2 Every r-round $\mathcal{MRC}$ computation with capacity s per mapper and reducer can be simulated by a $(r+1)$-round s-SHUFFLE computation.

Proof (sketch): We use the version of the s-SHUFFLE model with an arbitrary finite alphabet and unordered input ports (Section 4). Our alphabet Σ will be the set of all possible w-bit words. (With $w = O(\log n)$, $|\Sigma|$ is polynomial in n.)

Round $i+1$ of the s-SHUFFLE computation consists of one machine per round-i reducer of the $\mathcal{MRC}$ computation. (The space constraint of $n^{1-\epsilon}$ in the latter computation translates to a fan-in constraint of $s = n^{1-\epsilon}$ in the former computation.) Each such machine simulates the work of both the corresponding round-i reducer and the subsequent processing of the generated key-value pairs by the round-$(i+1)$ mappers. Round 1 of the s-SHUFFLE computation simulates the first mapping round of the $\mathcal{MRC}$ computation. The requisite machines corresponding to the inputs and outputs are added before the initial mappers and the final reducers. The functions α_{uv} simulate the message-passing of key-value pairs in the $\mathcal{MRC}$ computation (these are non-trivial only for pairs u, v of machines in consecutive rounds). □

Acknowledgments

We thank Noam Nisan for pointing us to several relevant references.