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 would separate \( P \) from \( NC^1 \).

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

---

*A preliminary version of this paper appeared in the 28th ACM Symposium on Parallelism in Algorithms and Architectures, July 2016.
†Stanford University. Supported in part by NSF Grant CCF-1524062.
‡Google, Inc.
§Stanford University. Supported in part by a Stanford Graduate Fellowship and NSF Grant CCF-1524062.
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 extract 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 every computing framework, present 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, 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 space \( s = \frac{n}{\epsilon} \) per machine requires \( \frac{1}{\epsilon} \) rounds in our model. Similarly, if \( s \) is only polylogarithmic, then \( \Omega \left( \log n / \log \log n \right) \) rounds are needed. Our lower bounds also extend, with a constant-factor degradation, to randomized computations.

Are there 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 \( \lceil \log n \rceil \) rounds. 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 \( \Theta(\log \log n) \) for a function in \( P \) would separate \( NC^1 \) from \( P \), a major open question in circuit complexity (Theorem 7.1).
Fourth, by relating MapReduce-type computations to polynomials, we can apply the sophisticated tool-
box 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 focus on 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 different 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, but also not too large, by restricting the total number of
machines to be sublinear in the input as well. They showed how to simulate a subclass of EREW algorithms
in MRC, but left open the question of whether all NC languages can be simulated. Goodrich et al. [16]
进一步 extended the simulation results to CRCW PRAMs as well as BSP algorithms [38], and gave MRC
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 focus on the total input to each machine, the key parameter of the model of the present work.
This measure was subsequently adopted by several authors in proving upper bounds on the space-round
trade-offs in these computations, and this parameter has been variously called memory, space, and key-
complexity [2, 5, 15, 26, 32]. The algorithmic tools and techniques developed for efficient computation
in this model 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 models of massively parallel computation concern
either communication (with a fixed number of rounds) or restricted classes of algorithms (for round lower
bounds). 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 [40] 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.

We next describe several previously studied computational models that are closely related to the model studied in this paper.

**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 for this model was 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 \( s = \omega(w) \)). 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.

---

\(^1\)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 and resolve \( NC^1 \) 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.
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.³

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 which 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:

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

³More broadly, variations of “the polynomial method” have been used previously to prove lower bounds in many fields of complexity theory, including circuit complexity [33, 37] and quantum query complexity [4].
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.

We now 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 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 at most 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, “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 easily 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 com-
putations 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 computa-
tions? 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 con-
ected 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 \( \bigcirc_{i=1}^{m} a_i \).

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.\(^4\)

\(^4\)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 [39] 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 $\alpha_{uv}$ from $\{0, 1, \bot\}^s$ to $\{0, 1, \bot\}^s$.

We think of each machine as having $s$ “ports,” where each port can accept at most 1 bit. The interpretation of a function $\alpha_{uv}$ is: given that machine $u$ received $z = z_1, \ldots, z_s$ on its $s$ input ports (where each $z_i \in \{0, 1, \bot\}$), it sends the message $\alpha_{uv}(z) \in \{0, 1, \bot\}^s$ to the $s$ input ports of $v$.\footnote{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.} Thus, machine $u$ explicitly communicates with machine $v$ (on at least one port) if and only if at least one coordinate of $\alpha_{uv}(z)$ is not $\bot$. 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 in a circuit, with the important constraint that, on every input, each input port should receive a bit (i.e., a non-$\bot$) from at most one machine.

**Definition 2.4** (Result of an $s$-SHUFFLE Computation) The result of an $s$-SHUFFLE computation assigns a value $g(v) \in \{0, 1, \bot\}^s$ 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 $(x_i, \bot, \bot, \ldots, \bot)$.
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 $\bot$-sum, over all machines $u$ with $r(u) < r(v)$, of the message $\alpha_{uv}(g(u))$ sent to $v$ by $u$:

$$g(v) := \bigcirc_{u : r(u) < r(v)} \alpha_{uv}(g(u)).$$

(1)

The result of an $s$-SHUFFLE computation is valid if every $\bot$-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 $(0, \bot, \bot, \ldots, \bot)$ or $(1, \bot, \bot, \ldots, \bot)$. 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.
**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.\(^6\) We say that such a computation computes a function \(f\) if, for every input \(x\), the output of the computation equals \(f(x)\) 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 translate MapReduce computations to *s*-SHUFFLE computations. See Section 4 for details.

### 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 on 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 \(f : \{0, 1\}^n \rightarrow \{0, 1\}^k\), \(f\) 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 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 (Unbounded Width Upper Bound)** For every \(s \geq 2\), every function \(f : \{0, 1\}^n \rightarrow \{0, 1\}^k\) can be computed by an *s*-SHUFFLE computation in \(\lceil \log_s n \rceil\) rounds.

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

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

**Theorem 3.1 (Round-Efficient Shuffles Are Low-Degree Polynomials)** Suppose that an *s*-SHUFFLE computation correctly computes the function \(f : \{0, 1\}^n \rightarrow \{0, 1\}^k\) 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 \(p_i(x) = f(x)\) for all \(i \in \{1, 2, \ldots, k\}\) and \(x \in \{0, 1\}^n\).

\(^6\)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.

9
The following corollary is immediate.

**Corollary 3.2 (Round Lower Bound)** If some output bit of the function \( f : \{0, 1\}^n \rightarrow \{0, 1\}^k \) 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 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 \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) if \( |p(x) - f(x)| \leq \frac{1}{3} \) for every \( x \in \{0, 1\}^n \). The \textit{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 (Round-Efficient Randomized Shuffles Are Low-Degree Approximate Polynomials)

Suppose that a randomized \( s \)-SHUFFLE computation computes the function \( f : \{0, 1\}^n \rightarrow \{0, 1\}^k \) 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 \( |p_i(x) - f(x)| \leq \frac{1}{3} \) for all \( i \in \{1, 2, \ldots, k\} \) and \( x \in \{0, 1\}^n \).

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 \( x \in \{0, 1\}^n \), \( p_i(x) \) equals the probability that the randomized \( s \)-SHUFFLE computation outputs a 1 on the input \( x \). Since the randomized computation computes \( f \) in the sense of Remark 2.6, these polynomials satisfy the conclusion of the theorem. □

Corollary 3.4 (Round Lower Bound (Randomized)) If some output bit of the function \( f : \{0, 1\}^n \rightarrow \{0, 1\}^k \) 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

We now extend Definitions 2.3 and 2.4 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 \((\text{key}; \text{value})\) 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 \( \Sigma \), the input belongs to \( \Sigma^n \) and the output to \( \Sigma^k \). Each machine \( u \) receives a value \( g(u) \) and sends messages \( \alpha_{uv}(g(u)) \) belonging to \( \Sigma \cup \{\bot\}^s \). A \( \bot \)-sum of \( m \) values from \( \Sigma \cup \{\bot\} \) is defined in the obvious way whenever at most \( 1 \) of the values is not \( \bot \) (and is undefined otherwise).

To incorporate unordered input ports, the functions \( \alpha_{uv} \) are redefined to have domain and range equal to the multi-sets of \( \Sigma \) of cardinality at most \( s \). A \( \bot \)-sum of such multi-sets is their union (with multiplicities adding), and is undefined if the sum of the multiplicities exceeds \( s \).
We now outline how to modify Theorem 3.1 so that it applies to the extended model in Section 4. To represent a function \( f : \Sigma^n \rightarrow \Sigma^k \), 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 \( x \in \Sigma^n \) 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 \( i \)th output bit of \( f \) is \( \sigma \).

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

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

In general, enlarging the alphabet of an \( s \)-SHUFFLE computation beyond the domain and range of the function \( f \) to be computed cannot lead to lower-degree polynomial representations. For example, we have the following corollary for Boolean functions and arbitrary alphabets \( \Sigma \).

**Corollary 4.2 (Round Lower Bound (Non-Binary Alphabets))** If some output bit \( j \) of the function \( f : \{0,1\}^n \rightarrow \{0,1\}^k \) cannot be represented by a polynomial with degree less than \( d \), then every \( s \)-SHUFFLE computation over the alphabet \( \Sigma \) that computes \( f \) uses at least \( \lceil \log_s d \rceil \) rounds.

**Proof:** Suppose that some \( s \)-SHUFFLE computation over the alphabet \( \Sigma \) computes \( f \) in less than \( \lceil \log_s d \rceil \) rounds. By Theorem 4.1, there is a polynomial \( p \), with degree less than \( d \) over the variable set \( \{x_{i\sigma}\}_{i\in[n],\sigma\in\Sigma} \), that represents the event when the \( j \)th output bit is 1. 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 \( d \) that represents the \( j \)th output bit of \( f \), contradicting the assumption. We conclude that every \( s \)-SHUFFLE computation over \( \Sigma \) that computes \( f \) requires at least \( \lceil \log_s d \rceil \) rounds. \( \blacksquare \)

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

For example, since the \( \text{AND}_n \) function is represented (uniquely) by the polynomial \( \prod_{i=1}^{n} x_i \), it has the maximum-possible degree. Similarly, the \( \text{OR}_n \) 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 for \( s \)-SHUFFLE computations with a binary alphabet.

**Corollary 5.1 (Lower Bound for \( \text{AND}_n \) and \( \text{OR}_n \))** Every \( s \)-SHUFFLE computation that computes the \( \text{AND}_n \) or the \( \text{OR}_n \) function uses at least \( \lceil \log_s n \rceil \) rounds.

Corollary 4.2 implies the same lower bound for \( s \)-SHUFFLE computations with an arbitrary alphabet \( \Sigma \).

### 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 \( n^2/3 - o(n^2) \) [20, 23, 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.

**Theorem 5.2 (Lower Bound for Monotone Graph Properties)** 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 \( \frac{1}{2} \log_s n - \frac{1}{3} \log_s 6 \) 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: \( \deg(f) \leq D(f) \leq 216 \cdot \deg(f)^6 \), where \( \deg \) denotes the approximate degree [4, 30].

We therefore have the following analog of Theorem 5.2 for randomized computations.

**Theorem 5.3 (Lower Bound for Monotone Graph Properties (Randomized))** 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 \( \frac{1}{3} \log_s n - \frac{1}{6} \log_s 648 \) rounds.

In general, round lower bounds for deterministic \( s \)-SHUFFLE computations proved via Corollaries 3.2 and 4.2 extend automatically, with a small constant-factor loss, to randomized computations.

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

Lemma 6.2 (Properties of Parity Differences) Fix a set $S \subseteq \{0, 1\}^n$.

(a) If $S_1, S_2$ form a partition of $S$, then

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

(b) If $S$ is 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

$$\sum_{x \in S} \prod_{i} (-1)^{x_i} = \sum_{y \in S_1} \prod_{i} (-1)^{y_i} + \sum_{z \in S_2} \prod_{i} (-1)^{z_i}.$$ 

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

$$\sum_{x \in S} \prod_{i} (-1)^{x_i} = \sum_{y \in S_1} \sum_{z \in S_2} \prod_{i} (-1)^{y_i} \prod_{i} (-1)^{z_i}$$

$$= \sum_{y \in S_1} \prod_{i} (-1)^{y_i} \sum_{z \in S_2} \prod_{i} (-1)^{z_i}.$$
6.2 Graph Connectivity Problems

We express graph problems as Boolean function using a $n^2$-variable input to represent an undirected graph on $n$ vertices (or a $2n^2$-variable input for directed graphs), where each variable indicates the presence or absence of a given edge. This is effectively the adjacency matrix representation of the graph.

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 is: given a graph $G = (V, E)$ and two vertices $s, t \in V$, is there a path from $s$ to $t$? We pinpoint the degree of both of these problems for both undirected and directed graphs. The proof strategy is similar in 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.

6.2.1 Undirected Connectivity

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

**Theorem 6.3 (Undirected Connectivity)** The degree of the Boolean function for undirected Connectivity on graphs with $n$ vertices is $\binom{n}{2}$.

**Proof:** Let $f_n$ denote the Boolean function representing undirected Connectivity for graphs with $n$ vertices. We prove that
\[
\Phi(f_n) = (-1)^{n-1}(n-1)!,
\]
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 \((n - 1)!\) of permutations, which completes the inductive hypothesis and the proof. ■

**Theorem 6.4 (Undirected s-t Connectivity)** The degree of the Boolean function representing undirected ST-Connectivity on graphs with \(n\) vertices is \((\binom{n}{2})\).

**Proof:** Let \(f_n\) denote the Boolean function representing undirected ST-Connectivity for graphs with \(n\) vertices. By convention, we 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 \((n-2)!\) 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 \((n-3)\) 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. ■

### 6.2.2 Directed Connectivity

We now consider the more complicated case of directed graphs.

**Theorem 6.5 (Directed Connectivity)** The degree of the Boolean function for directed Connectivity on graphs with \(n\) vertices is \(2\binom{n}{2}\).

**Proof:** The proof is similar but somewhat trickier than that for undirected Connectivity. Letting \(f_n\) denote the Boolean function for directed Connectivity on graphs with \(n\) vertices, we show by induction
on \( n \) that \( \Phi(f_n) = (n - 1)! \). The base case of \( n = 1 \) is trivial, as the graph is always connected and \( \Phi(f_1) = 1 \).

For the inductive step, fix \( n > 1 \) and consider the first vertex of a strongly connected graph on \( n \) vertices. If we remove this vertex, then the remaining \( n - 1 \) vertices are partitioned into strongly connected components. We can split the set \( S \) of all strongly connected graphs into sets \( S_P \) that yield the same partition \( P \) after the first vertex is removed. By Lemma 6.2(a), \( \Phi(S) = \sum_P \Phi(S_P) \).

We claim that total parity difference contributions \( \Phi(S_P) \) of the partitions \( P \) of the last \( n - 1 \) vertices in which there is at least one edge between two blocks is zero. To see this, order the edges that span different blocks arbitrarily, and bucket the strongly connected graphs that induce \( P \) according to the first such edge \((v, w)\) that it includes. Now fix \((v, w)\), let \( v \) be in block \( A \), \( w \) in block \( B \), and let \( u \) denote the first vertex of the graph. For each graph \( G \) in this bucket that includes the edge \((u, w)\), the graph \( G \setminus \{(u, w)\} \) is also in this bucket — since all graphs in the bucket include edge \((v, w)\) and contain a path from \( u \) to all vertices in \( A \), \( G \setminus \{(u, w)\} \) is also strongly connected. Thus, we can uniquely pair up the even and odd solutions of this bucket. Since the buckets are disjoint, we can pair up the even and odd strongly connected graphs that include at least one edge that spans two blocks.

We now consider only strongly connected graphs with no edges between the blocks of the partition \( P \). Fixing \( P \), we can view these graphs as a product of the set of directed edges between the first vertex and the first block, the set of ways to strongly connect the first block, the set of directed edges between the first vertex and the second block, the set of ways to strongly connect the second block, and so on. By Lemma 6.2(b), we can compute the parity difference of each of these sets separately, and take their product.

First, we consider the number of ways to connect the first vertex to a block with \( k \) vertices. There are \( k \) possible edges to the block, and \( k \) possible edges from the block. We count the parity differences separately and invoke Lemma 6.2(b) to compute the total parity difference by multiplying. For the \( k \) edges to the block, the only forbidden choice is the empty set. Hence there are \( 2^{k-1} \) odd choices but only \( 2^{k-1} - 1 \) even choices. The parity difference here is \(-1\). But the same reasoning applies to the \( k \) edges from the block, so the total parity difference is \( 1 \).

By induction, the parity difference of the number of ways to strongly connect a block of size \( k \) is \((k-1)!\). If we include the number of ways that the first vertex can be connected to the block, the net contribution to the partition from a block of size \( k \) is \((k-1)!\).

Finally, we need to sum the parity difference over all partitions, which we claim is \((n-1)!\). As in the proof of Theorem 6.3, partitions with blocks weighted by \((k-1)!\) correspond to permutations of \( n - 1 \) items, and the total parity difference is \((n-1)!\). This proves the inductive step and completes the proof. \( \blacksquare \)

Finally, we consider the directed \textsc{st-Connectivity} problem. In contrast to the other three cases, the degree of the corresponding Boolean function is only half of the maximum possible. The proof is correspondingly more intricate, as we cannot simply prove that the parity difference of the function is non-zero.

**Theorem 6.6 (Directed \( s\)-\textit{t} Connectivity)** The degree of the Boolean function for directed \textsc{st-Connectivity} on graphs with \( n \) vertices is \( \binom{n}{2} \).

**Proof:** Let \( f_n \) denote the Boolean function for directed \textsc{st-Connectivity} on graphs with \( n \) vertices. First, we show that \( \deg(f_n) \geq \binom{n}{2} \). Suppose that the polynomial for directed \textsc{st-Connectivity} is \( p(x) \), and index \( x \) by possible edges in our graph, i.e. \( x_{(i,j)} \) denotes whether edge \((i, j)\) is in the graph. Suppose we plug in both \( x_{(i,j)} = y_{(i,j)} \) for all \( i < j \) and \( x_{(j,i)} = y_{(i,j)} \) for all \( i < j \) and simplify. In other words, we replace variables for each pair of directed edges with a single variable for an undirected edge. The
polynomial now computes undirected ST-CONNECTIVITY. Its degree can only be lower than that of the original polynomial for directed ST-CONNECTIVITY, because simplifying consists only of applying the rule $x^2 = x$ and combining like terms. By Theorem 6.4, the degree for undirected ST-CONNECTIVITY is $\binom{n}{2}$, so the degree for directed ST-CONNECTIVITY is at least that.

In the rest of the proof, we show that $\text{deg}(f_n) \leq \binom{n}{2}$. We continue to use $p(x)$ to represent the polynomial for directed ST-CONNECTIVITY. We use $\mathcal{E}$ to denote the set of all possible edge sets over $V$. Then written as a sum of monomials, $p(x)$ looks like:

$$p(x) = \sum_{E \in \mathcal{E}} \alpha_E \prod_{e \in E} x_e,$$

where the $\alpha_E$’s are constant coefficients.

We will want to evaluate $p(x)$ for particular choices of $x$. Let $x_E$ be the vector where $x_e = 1$ if and only if $e \in E$. Evaluating $p(x_E)$ then yields whether there is an $s$-$t$ path using only edges in $E$. We also know that $p(x_E) = \sum_{E' \subseteq E} \alpha_{E'}$. 

Our first claim is that $\alpha_E = 0$ for any edge set $E \in \mathcal{E}$ that contains an edge $e \in E$ which is not used by any simple $s$-$t$ path that only goes through edges in $E$. We will prove this claim by induction on the size of $E$.

The base case of our induction, when $|E| = 0$, is trivially true because $E$ does not have any edges. For the inductive step, consider an edge set $E$ with such an edge $e$. Let $\bar{E} = E \setminus e$. Because $e$ is not used in any simple $s$-$t$ paths, removing it does not change whether there is an $s$-$t$ path. In other words, $p(x_E) = p(x_{\bar{E}})$. Hence

$$p(x_E) = \sum_{E' \subseteq E, e \in E'} \alpha_{E'} + \sum_{E' \subseteq E, e \notin E'} \alpha_{E'} = \sum_{E' \subseteq E, e \in E'} \alpha_{E'} + p(x_{\bar{E}})$$

and $\sum_{E' \subseteq E, e \in E'} \alpha_{E'} = 0$. Our inductive hypothesis states that for every edge set $E'$ with fewer edges than $E$ and which contains an edge $e \in E'$ which is not used in any simple $s$-$t$ path, we have that $\alpha_{E'} = 0$. Since $e$ was not used in any simple $s$-$t$ path containing only edges from $E$, $e$ is also not used in any simple $s$-$t$ path containing only edges from any $E'$. Hence $\alpha_{E'} = 0$ for every term in this summation with $E' \neq E$, which in turn implies that $\alpha_E = 0$ as well. This completes the proof of our first claim.

Our second claim is that for every edge set $E \in \mathcal{E}$ which contains both an edge $e = (i, j)$ and its reverse edge $e' = (j, i)$, we have $\alpha_E = 0$. This second claim is enough to complete the proof, since it implies that every edge set $E \in \mathcal{E}$ with a nonzero $\alpha_E$ has at most $\binom{n}{2}$ edges, and hence the degree of the polynomial is at most $\binom{n}{2}$. We also prove our second claim by induction on the size of $E$.

The base case $|E| = 0$ is again trivial. For the inductive step, consider an edge set $E$ with such a pair of edges $e = (i, j)$ and $e' = (j, i)$.

Due to our first claim, $\alpha_E$ is zero if either of $e$ or $e'$ is not used in a simple $s$-$t$ path. Hence we need only consider the case where there are simple $s$-$t$ paths using both of these edges. This implies that there must be $s$-$i$, $s$-$j$, $i$-$t$, and $j$-$t$ paths, none of which use $e$ or $e'$. Let $E_1 = E \setminus \{e\}$, $E_2 = E \setminus \{e'\}$, and $E_3 = E \setminus \{e, e'\}$. We know that all of $E$, $E_1$, $E_2$, and $E_3$ still have $s$-$t$ paths, i.e. $p(x_E) = p(x_{E_1}) = p(x_{E_2}) = p(x_{E_3}) = 1$. 

18
As in the proof of our first claim, we can manipulate as follows:

\[ p(x_E) = \sum_{E' \subseteq E} \alpha_{E'} + \sum_{e \in E'} \alpha_{E'} + \sum_{e \notin E'} \alpha_{E'} + \sum_{e' \in E'} \alpha_{E'} \]

\[ = \sum_{E' \subseteq E} \alpha_{E'} + [p(x_{E_2}) - p(x_{E_3})] + [p(x_{E_3}) - p(x_{E_4})] + [p(x_{E_4})] \]

and so

\[ 1 = \sum_{E' \subseteq E} \alpha_{E'} + [1 - 1] + [1 - 1] + [1] \]

and

\[ \sum_{E' \subseteq E} \alpha_{E'} = 0. \]

Our inductive hypothesis states that for every edge set \( E' \) with fewer edges than \( E \) and which contains edges \( e = (i, j) \) and \( e' = (j, i) \), \( \alpha_{E'} = 0 \). Again, this implies that every term in the summation other than \( \alpha_E \) must be zero, and hence \( \alpha_E = 0 \) as well. This completes the proof the second claim and the theorem. ■

6.2.3 Connectivity with Large Alphabets

Similar ideas can be used to prove degree lower bounds for connectivity problems that are formulated using non-binary alphabets. For example, we can represent graphs with \( m \) edges using \( m \) inputs from an alphabet of size \( \binom{n}{2} \). This corresponds to using an adjacency list representation, rather than an adjacency matrix representation.

We next present a degree lower bound for undirected \( ST\text{-CONNECTIVITY} \), for graphs provided in this adjacency list representation. To make our lower bound stronger, we disallow repeated edges (otherwise the lower bound follows easily from that for the \( OR \) function). We also assume a weak upper bound on the density of the graph (with high enough density, the graph is guaranteed to be connected and the problem is trivial).

**Theorem 6.7 (Lower Bound for Adjacency List Representation)** Let \( f \) be the function associated with the undirected \( ST\text{-CONNECTIVITY} \) problem for graphs with \( m \) edges and \( n \) vertices in adjacency list representation. If \( m \leq \frac{n^2}{8} \), then every polynomial over variables \( \{x_{i\sigma}\}_{i \in [m], \sigma \in \Sigma} \) that represents this function has degree at least \( m \).

**Proof:** We construct a set of \( 2^m \) different inputs, where each edge takes on one of two possible values, and exactly one of the inputs fails to contain an \( s-t \) path. Assume that \( s \) and \( t \) are the vertices 0 and 1. The first edge is either \((0, 1)\) or \((0, 2)\), the second either \((1, 2)\) or \((1, 3)\), the third either \((0, 3)\) or \((0, 4)\), the fourth either \((2, 3)\) or \((2, 4)\), and so on. In general, each edge is drawn from a pair of the form \( \{(a, b), (a, b+1)\} \), where \( a < b \) and \( a, b \) have opposite parities, and pairs are considered in nondecreasing order of \( b \), with ties broken in favor of smaller values of \( a \). There are at least \( n^2/8 \) pairs of this form. Since \( m \leq n^2/8 \), we can consider the first \( m \) such pairs and the corresponding \( 2^m \) different inputs. The key observation is that such
an input fails to contain an $s$-$t$ path if and only if, for each edge, the second option is chosen. As long as the second option is chosen, the graph has two connected components, one spanning the even-parity vertices considered so far (including 0) and the other the odd-parity vertices considered so far (including 1). If the first option is ever chosen, this edge connects the two components, and in particular 0 and 1.

Restricted to these $2^n$ inputs, the ST-CONNECTIVITY function specializes to the $OR_m$ function (formally, by plugging in variables as in the proof of Corollary 4.2). Since there is no degree-$m$ polynomial representation of the $OR_m$ function (Corollary 5.1), there is no such representation of the ST-CONNECTIVITY function.

\section{Prospects for Stronger Lower Bounds}

Our lower bounds in Section 3–5 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 new circuit lower bounds.

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

\begin{theorem}[Circuit Complexity Barrier to Stronger Lower Bounds] Consider any model of parallel computation with the following properties:

\begin{enumerate}
    \item The number of machines is polynomial in the input size $n$.
    \item Computation proceeds in time steps. In each time step, a machine can read $s(n) \geq 2$ bits from the input or from previously completed computations.
    \item Each machine has enough power to evaluate a Boolean circuit with size at most $s(n)$ and depth at most $\log_2 s(n)$ in one time step.
\end{enumerate}

If some problem in $P$ cannot be solved in $O(\log[s] n)$ time steps in such a model, then $NC^1 \subsetneq P$.
\end{theorem}

Property (1) states that an arbitrarily large polynomial number of machines is 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 (where the third property may not hold), and some of the previous work reviewed in Section 1.2 is of this type.

\textit{Proof of Theorem 7.1:} We simply need to show that every model with these three properties can efficiently simulate $NC^1$ circuits. Fix some such model of parallel computation. 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 $\text{poly}(n)$ gates
and $O(\log n)$ depth (where $n$ denotes the number of inputs). We will transform each circuit into a parallel computation that uses only $O(\log s(n))$ time steps (see Figure 2 for an example). This will show that no $NC^1$ family computes our hypothesized problem, and hence $NC^1 \subseteq P$, as desired.

By the first assumed property, we can associate each gate of the circuit with a machine, whose sole responsibility is to compute the output of the gate. Each time step of the parallel computation will correspond to $\log_2 s(n)$ layers of the circuit. For example, in the first time step, we will have a machine for every gate at depth at most $\log_2 s(n)$. Since the circuit has fan-in 2, computing the output of this gate requires reading at most $s(n)$ input bits and evaluating a Boolean circuit with depth at most $\log_2 s(n)$ and size at most $s(n)$.

By the second and third assumed properties, the corresponding machine can perform this computation in the first time step of the parallel computation. In general, in the $i^{th}$ time step, we have a machine for every gate at depth between $(i-1)\log_2 s(n)$ and $i\log_2 s(n)$. Again, the output of each of these gates can be computed by reading at most $s(n)$ bits (from the input or the outputs of gates corresponding to machines in previous time steps) and evaluating a depth-$i\log_2 s(n)$ and size-$s(n)$ Boolean circuit. We reach the output gates and complete the computation in $O(\frac{\log n}{\log s(n)}) = O(\log s(n))$ time steps of parallel computation, as desired. ■

The simulation argument in the proof of Theorem 7.1 is quite versatile. For example, an analogous statement holds 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 at each machine. Also, the simulation in the proof of Theorem 7.1 only blows up the width by a factor of $\log_2 s(n)$ from the 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 $\log_2 s(n)$) version of $NC$, which would still be a significant result in 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 $NC$.

Acknowledgments

We thank Noam Nisan for pointing us to several relevant references, and the SPAA referees for helpful comments.
References


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 this model, for some $\epsilon > 0$, there are at most $n^{1-\epsilon}$ machines, each with space at most $n^{1-\epsilon}$.

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 $\mu$ 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 $\rho$ is a function that takes as input a binary string, representing a key $k$, and a set of values $v_1, v_2, \ldots$ (from key-value pairs with key $k$), and outputs a set of \langle key; value \rangle pairs. We can think of the set of reducers as being indexed by the corresponding key $k$.

We first prove that, without loss of generality, a $\mathcal{MRC}$ computation uses only \langle key; value \rangle pairs with $O(\log n)$ bits.

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

**Proof (sketch):** First consider a \langle key; value \rangle pair where 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. Similarly, if the length of a value string
A $v_1 v_2 v_3 \ldots v_k$ of a $\langle \text{key}; \text{value} \rangle$ pair is more than $3 \log n$, we can break it into blocks $a_1, \ldots, a_t$ of $O(\log n)$ bits and encode each block as $\langle \text{key}; u\$i\$a_i \rangle$, where $u$ is a random $3 \log n$ bit integer (specific to $a$), and $\$ is a unique symbol. All of these blocks will be sent to the same reducer. Provided the integers $u$ generated for each value string $a$ that we break up are unique (a high probability event since there are at most $n^{2-\epsilon}$ $\langle \text{key}; \text{value} \rangle$ pairs), the reducer can use the indexes $i$ 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 $\Sigma$ 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 $\alpha_{uv}$ 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). □