On Derandomization and Average-Case Complexity of Monotone Functions *

George Karakostas †
McMaster University
karakos@mcmaster.ca

Jeff Kinne ‡
Indiana State University
jkinne@indstate.edu

Dieter van Melkebeek §
University of Wisconsin-Madison
dieter@cs.wisc.edu

October 8, 2011

Abstract

We investigate whether circuit lower bounds for monotone circuits can be used to derandomize randomized monotone circuits. We show that, in fact, any derandomization of randomized monotone computations would derandomize all randomized computations, whether monotone or not. We prove similar results in the settings of pseudorandom generators and average-case hard functions – that a pseudorandom generator secure against monotone circuits is also secure with somewhat weaker parameters against general circuits, and that an average-case hard function for monotone circuits is also hard with somewhat weaker parameters for general circuits.

1 Introduction

One of the central topics in the theory of computing deals with the power of randomness – can randomized procedures be efficiently simulated by deterministic ones? A long line of research (see [Mil01] for an introduction) has shown how to construct pseudorandom generators sufficient to derandomize all time-efficient randomized algorithms under a very reasonable complexity-theoretic hardness assumption. The latter states that there exists a problem decidable in time $2^{\Theta(n)}$ that cannot be solved by a family of circuits that uses only $2^{o(n)}$ gates for inputs of length n. We call such a hardness condition a "circuit lower bound for an explicit function" and the pseudorandom generators constructed using the hard function "hardness-based". Given the hardness assumption, all problems solvable by polynomial-time bounded-error randomized algorithms can be solved in polynomial time on deterministic machines, i.e., BPP = P.

*Parts of this work appeared in the 20th International Symposium on Algorithms and Computation (ISAAC) [Kar09] and in the second author's PhD dissertation [Kin10].
†Research supported by an NSERC Discovery grant.
‡Partially supported by NSF award 0728809, a Cisco Systems Distinguished Graduate Fellowship, and Indiana State University Research Council award 11-07. A significant portion of this work was completed while a graduate student at the University of Wisconsin-Madison.
§Partially supported by NSF awards 0728809 and 1017597, and by the Humboldt Foundation.

Though the hardness assumption is plausible and widely believed in the community, circuit lower bounds have been notoriously difficult to prove and recent work [KI04] has shown that in fact *any* non-trivial derandomization of BPP implies circuit lower bounds that have been elusive for a long time.

A natural question then is, for which algorithmic settings do we have hard functions that yield derandomization? Monotone functions are one of the notable settings where hard functions are known. In this paper we consider whether these hardness results imply derandomization; and we ask the broader question of how derandomization of monotone computations relates to the derandomization of general non-monotone computations.

1.1 Background

The hope in studying monotone Boolean functions is that the property of monotonicity can be exploited to prove stronger results than in the general case. This hope has come to fruition in the area of circuit lower bounds. A long line of research has proved that various explicit monotone functions require monotone circuits of super-polynomial size (perfect matchings) or even exponential size (clique) (see [BS91] and [Kor03] for surveys).

An immediate question is whether the exponential *worst-case* lower bounds for monotone circuits can be used in hardness-based pseudorandom generators to derandomize bounded-error randomized monotone circuits. The latter are monotone circuits C that take two inputs: the input x to the problem, and the coin flip sequence r. C should satisfy the promise that for every x, $\Pr_R[C(x,R) = 1] \geq \frac{2}{3}$ or $\Pr_R[C(x,R) = 1] \leq \frac{1}{3}$, where R is chosen uniformly at random. Including a uniformity condition – that there is a deterministic polynomial-time machine that on input 1^n outputs the circuit C – gives a natural monotone version of the complexity class BPP.

Consider the requirements needed to apply a hardness-based pseudorandom generator to derandomize monotone circuits. The proofs for hardness-based pseudorandom generators typically argue that if the generator can be broken by small circuits, then we can use those small circuits to approximately compute the presumed hard function. In the setting of monotone circuits, we would assume a small monotone circuit that distinguishes the output of the generator from uniform, and with this monotone distinguisher we should construct a small monotone circuit that approximately computes the presumed hard function. For this to work, the reduction from the distinguisher to the circuit approximating the hard function should preserve monotonicity.

Let us consider two different generator constructions that have been developed to derandomize time-bounded computations – the Nisan-Wigderson generator [NW94] and the Shaltiel-Umans generator [SU05, Uma03]. The latter generator uses ingredients such as list-decodable codes and finite field arithmetic that perform non-monotone operations such as parity, and it is unclear if these elements can be made monotone.

On the other hand, an examination of the reduction for the Nisan-Wigderson generator reveals that only a single negation is needed, and a monotone function hard for both monotone circuits and their negations could be used in this generator to derandomize monotone circuits. To derandomize a circuit of size n^k, the known proof requires a function that is $(\frac{1}{2} - \frac{1}{n^k})$-hard for small circuits. But can such lower bounds be proved for monotone functions? A negative answer comes from work in learning theory. [KLV94] showed that for any monotone function f, f is within distance $\frac{1}{2} - \Omega(\frac{1}{n})$ of the constant 0 function, the constant 1 function, or one of the dictator functions (which

are equal to the i-th bit of the input for some i); the results of [KKL88] improved the distance to $\frac{1}{2} - \Omega(\frac{\log n}{n})$. In particular, no monotone function has hardness greater than this amount even for constant-depth constant size circuits – and the approach of using the existing proof of the Nisan-Wigderson generator to derandomize randomized monotone computations fails.

1.2 Our Results

From the discussion above, we know that there can be no monotone function with high enough average-case hardness to be used in known hardness-based pseudorandom generators to derandomize monotone circuits. But can we easily prove high average case hardness for some explicit *non-monotone* function against *monotone* circuits? We consider this goal and other questions related to derandomizing monotone circuits.

Hard on Average Functions First, we show that a function that is hard on average for monotone circuits is hard on average for general circuits with somewhat weaker parameters. We prove the contrapositive – that a general circuit approximating any function can be converted into a monotone circuit without too much loss in parameters. In the following, an anti-monotone circuit is the negation of a monotone circuit, and a function is to within ϵ of some other function on the same domain if they differ on a fraction at most ϵ of the domain.

Theorem 1. *Let f be any function. If there is a general circuit C with s gates that computes f to within $\frac{1}{2} - \epsilon$, then there is either a monotone or anti-monotone circuit with $2s + O(n \log^2 n)$ gates that computes f to within $\frac{1}{2} - \epsilon'$ for $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$ for $c > 0$ an absolute constant that does not depend on n, s, or ϵ.*

We observe that Theorem 1 is tight to within a constant factor for the parity function. We also observe that monotone functions exist with hardness close to the barrier implied by results from learning theory. Subsequent to [KLV94] and [KKL88], [OW09] show that any monotone function is within distance $\frac{1}{2} - \Omega(\log n/\sqrt{n})$ of the constant 0 function, constant 1 function, a dictator, or majority. We observe that there exist monotone functions that are not within $(\frac{1}{2} - 1/n^{1/2-\eta})$ from functions computed by general circuits of size $2^{n^{\Omega(1)}}$, for every positive constant η and all but finitely many input lengths n.

Pseudorandom Generators Theorem 1 shows that one particular method of constructing a pseudorandom generator secure against monotone circuits – namely constructing a hard function for use in the Nisan-Wigderson generator – would also yield results for general non-monotone circuits. We show that in fact *any* method for constructing a pseudorandom generator secure against monotone circuits also implies a generator secure against general circuits with somewhat weaker parameters.

Theorem 2. *Let C be a circuit of size s that ϵ-distinguishes some distribution $\mathcal{D}$ from uniform. Then there is a monotone circuit C' of size $2s + O(n \log^2 n)$ that ϵ'-distinguishes $\mathcal{D}$ from uniform for $\epsilon' = \max(\frac{\epsilon}{2(n+1)}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$ for $c > 0$ an absolute constant that does not depend on n, s, or ϵ.*

In particular, Theorem 2 shows that if $\mathcal{D}$ is the output distribution of a pseudorandom generator G, then a distinguisher for G can be converted into a monotone distinguisher without too much

loss in the distinguishing probability. We observe that Theorem 2 is nearly tight for pseudorandom generators with small stretch, in particular for the generator that outputs its seed and the parity of the seed.

Derandomization in General Constructing pseudorandom generators is one method to derandomize (monotone) randomized circuits. We show that *any* method of derandomizing monotone randomized circuits can also be used to derandomize general non-monotone randomized computations.

Theorem 3. *Let L be any language computable by polynomial-time bounded-error randomized machines. There is a language L_{mon} computable by uniform monotone bounded-error polynomial-size randomized circuits such that L poly-time mapping reduces to L_{mon}. In particular, if $L_{mon} \in \mathrm{P}$ then $L \in \mathrm{P}$.*

1.3 Techniques

Each of our results entails reducing computation by general circuits to computation by monotone circuits. The key concept involved in the transformation is that of a *slice function.* A function f is called a slice function if there is a value k such that $f(x) = 1$ for all x with Hamming weight greater than k, $f(x) = 0$ for all x with Hamming weight less than k, and $f(x)$ can take arbitrary values for x with Hamming weight equal to k. We refer to the set of inputs having Hamming weight exactly k as the *k-th slice* of the Boolean n-cube. Beyond the fact that slice functions are monotone, our proofs use the following two properties.

> *Monotone Complexity of Slice Functions.* The monotone and general circuit complexity of slice functions are polynomially related.
>
> *Embedding Functions Within Slices.* The truth table of any Boolean function f on n bits can be embedded within the middle slice of another function f' on m bits, where $m = n + O(\log n)$.

The formal statement of these properties is contained in the preliminaries. We now discuss how the properties are used to prove our results.

Theorem 1 states that a general circuit C that approximates a function f can be converted into a monotone approximating circuit C_{mon} without much loss in parameters. The basic idea is to find a slice k on which C computes f well and let C_{mon} be a monotone circuit that computes the monotone slice function that agrees with C on the k-th slice. C_{mon} has a monotone circuit of size polynomially related to the size of C by the first property above.

Theorem 2 states that a circuit C that distinguishes some distribution (e.g., the output of a pseudorandom generator) from uniform can be made monotone without much loss in parameters. The main idea is similar to that of Theorem 1 but for the setting of a distinguisher rather than computing a function.

Theorem 3 states that for any BPP language L there is a language L_{mon} computed by a monotone bounded-error randomized circuit such that L poly-time many-one reduces to L_{mon}. The main idea is to use the second property above to convert the BPP machine into a monotone circuit and then use the first property to show the resulting monotone computation has polynomial-size monotone circuits.

1.4 Organization

In Section 2 we give preliminaries, in particular reviewing the properties of monotone slice functions used in the main results. Section 3 contains our results on average-case hard functions, including Theorem 1. Section 4 contains our results on distinguishers, including Theorem 2. Section 5 contains our results on derandomization, including Theorem 3.

2 Preliminaries

We introduce our notation and terminology, and we state relevant properties of monotone functions. For a more thorough treatment of the general concepts within computational complexity, see [AB09]. See [Kor03] for a survey on monotone functions and circuits.

For a binary string x, we use the notation $\overline{x}$ to denote the string resulting from negating each bit of x. We use the notation $|x|$ to refer to the Hamming weight of a string x, so $|x|$ is equal to the number of ones in x. For an index i between 1 and n, we let x_i refer to the i-th bit of x.

A *Boolean circuit* is a rooted directed acyclic graph with each internal node labeled as an AND, OR, or NOT gate and with each leaf node labeled as either some input bit x_i or one of the constants 0 or 1. The root produces the output of the circuit, and this output is computed in the natural way. We measure the *size* of a circuit by the number of gates.

Monotone Boolean Functions A *monotone Boolean function* is one such that flipping an input bit from 0 to 1 cannot change the output of the function from 1 to 0. Monotone functions can be computed by *monotone circuits* – circuits consisting of AND and OR gates but with no NOT gates. There are many examples of natural monotone Boolean functions based on graph properties – such as clique, connectivity, or perfect matchings – where adding edges can only make the property easier to satisfy.

An *anti-monotone* function is the negation or complement of a monotone function. Anti-monotone circuits can equivalently be viewed as either negations of monotone circuits or monotone circuits that are given $\overline{x}$ as input rather than x. When we speak of anti-monotone circuits we refer to the former by default.

Slices and Slice Functions We use the terminology *k-th slice* of the Boolean n-cube to refer to the set of $\binom{n}{k}$ n-bit strings that have Hamming weight exactly k. The *middle slice* refers to the $\lfloor n/2 \rfloor$-th slice. Using Stirling's formula, the middle slice contains $\binom{n}{\lfloor n/2 \rfloor} = \Theta(\frac{1}{\sqrt{n}} 2^n)$ many strings.

A *monotone slice function for the k-th slice* is a monotone function that can take arbitrary values for inputs on the k-th slice, evaluates to 1 for inputs x with $|x| > k$, and evaluates to 0 for inputs x with $|x| < k$. An *anti-monotone slice function* for the k-th slice is the negation of a monotone slice function for the k-th slice. We will say simply "slice function" when it is clear from context whether we refer to a monotone slice function or an anti-monotone slice function.

Monotone Complexity of Slice Functions Berkowitz [Ber82] observed that the monotone and general circuit complexity of slice functions are polynomially related, as follows. Let f be a monotone slice function for the k-th slice. Note that for x with Hamming weight exactly k, $\overline{x_i} = 1$ if and only if the $n-1$ remaining bits in x have Hamming weight at least k. Then given a circuit for

computing f, we first push all negations to the inputs (this at most doubles the size of the circuit) and then replace any instance of $\overline{x_i}$ by a threshold circuit over $n-1$ bits. As thresholds can be computed by $O(n \log n)$ size monotone circuits [AKS83], the resulting monotone circuit is of size $2s + O(n^2 \log n)$.

The construction can also be used to produce an anti-monotone circuit that agrees with f on the k-th slice – produce a monotone circuit computing the monotone slice function that is the complement of f on the k-th slice and then negate this circuit. Similarly, if f is an anti-monotone slice function, the process can be used to produce either a monotone or anti-monotone circuit agreeing with f on the k-th slice.

[Val86] gives a slightly more efficient construction that computes the threshold circuits for each x_i simultaneously with $O(n \log^2 n)$ many gates, which implies that if a slice function f has general circuits with s gates then f has monotone circuits with $2s + O(n \log^2 n)$ gates. Further, the construction is poly-time uniform.

Lemma 1 ([Val86]). *Let f be any slice function and let C be a circuit with at most s gates that computes f. There is a monotone circuit C_{mon} and an anti-monotone circuit $C_{anti-mon}$ such that both agree with f on the slice in question, compute slice functions, are of size $2s + O(n \log^2 n)$, and can be constructed in time polynomial in n and the size of C.*

Embedding Functions Within Slices Given a function f on n bits, the truth table for f can be embedded within a slice of a function f' on m bits for some m not too much larger than n. One method is to let f' take $m = 2n$ bits as input and set f' to be a monotone slice function such that for each n-bit string x, $f'(x, \overline{x}) = f(x)$. Note that all inputs of the form $(x, \overline{x})$ fall within the middle slice, so f' exists. The monotone circuit complexity of f' is polynomially related to its general circuit complexity by Lemma 1. However, only a very small fraction of the middle slice of f' is used in the embedding, namely $\frac{2^n}{\binom{2n}{n}} = \Theta(\frac{\sqrt{n}}{2^n})$.

For our application in Theorem 3, we need an embedding that uses a larger fraction of the input space of f'. If we let f' take m-bit inputs, then it is possible to embed the truth table of f into the middle slice of f' provided $\binom{m}{\lfloor m/2 \rfloor} \geq 2^n$. Because the binomial coefficient $\binom{m}{\lfloor m/2 \rfloor}$ grows by less than a factor of 2 for each increment of m, m can also be chosen so that $\binom{m}{\lfloor m/2 \rfloor} \leq 2 \cdot 2^n$, so the embedding occupies a constant fraction of the slice. To achieve the embedding efficiently, we let $k = \lfloor m/2 \rfloor$ and associate x with its representation in the so-called "combinatorial number system of degree k", which represents x uniquely in the form

$$x = \binom{a_k}{k} + \binom{a_{k-1}}{k-1} + ... + \binom{a_1}{1}$$

where $m > a_k > a_{k-1} > ... > a_1 \geq 0$ and with the definition that $\binom{a_i}{i} = 0$ if $a_i < i$. Given x, we can find the combinatorial representation of x efficiently using a greedy approach; determining x given its combinatorial representation is easy as well. See [Knu06, section 7.2.1.3] for further details.

To embed f within the k-th slice of the m-cube of a function f', we associate x with the m-bit string x' that has ones precisely in positions $a_k + 1$, $a_{k-1} + 1$, ..., $a_1 + 1$. We set f' to be the slice function that has $f'(x') = f(x)$. We summarize the relevant properties of this embedding in the next lemma.

Lemma 2. *For any positive integers n, m, k such that $\binom{m}{k} \geq 2^n$, there is a one-to-one mapping ϕ from $\{0,1\}^n$ into the set of m-bit strings with Hamming weight exactly k; the mapping is computable and invertible in* $\mathrm{poly}(m)$ *time.*

Concentration of Slices The number of n-bit strings having exactly k ones is equal to $\binom{n}{k}$, a value which is largest when k is close to $n/2$ and smaller when k is far from $n/2$. In fact, the weight of a uniformly at random chosen n-bit string is strongly concentrated around $n/2$. The following lemma, known as a Chernoff bound, quantifies this phenomenon. A proof can be found, e.g., in [AS00, Corollary A.1.2].

Lemma 3 (Chernoff Bound). *Let n be a positive integer and $0 \leq j \leq \lfloor \frac{n}{2} \rfloor$. The number of n-bit strings that have Hamming weight either less than $\lfloor \frac{n}{2} \rfloor - j$ or greater than $\lceil \frac{n}{2} \rceil + j$ is at most $2e^{-2j^2/n} \cdot 2^n$.*

Average-Case Hardness We say that a language L is *within* δ of another language L' if their characteristic functions are within relative Hamming distance δ, i.e., they differ on at most a δ fraction of the inputs for each input length n. At any input length n, the distance between a language L and a class of languages is defined as the minimum distance between L and some language in the class. We call a language L hard for a class if for any language in the class, L is far from the language on almost all input lengths n.

Definition 1 (hardness on average). *A language L is* $\delta(\cdot)$-hard *for a class of languages $\mathcal{C}$ if for every language $L' \in \mathcal{C}$, L' is within relative Hamming distance $\delta(n)$ of L for only finitely many input lengths n.*

Notice that worst-case hardness corresponds to setting $\delta(n) = \frac{1}{2^n}$.

3 Average-Case Hardness

In this section we prove our results concerning average-case hardness. In section 3.1, we show that functions which are hard on average for monotone circuits are hard on average for general circuits with somewhat weaker parameters (Theorem 1). In section 3.2, we observe that there exist monotone functions with average-case hardness approaching the barrier discussed in the introduction.

3.1 Reduction to Monotone Circuits

We establish Theorem 1 by showing that a circuit which approximates a given function can be made monotone without too much loss in accuracy.

Theorem 1 (restated). *Let f be any function. If there is a general circuit C with s gates that computes f to within $\frac{1}{2} - \epsilon$, then there is either a monotone or anti-monotone circuit with $2s + O(n \log^2 n)$ gates that computes f to within $\frac{1}{2} - \epsilon'$ for $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$ for $c > 0$ an absolute constant that does not depend on n, s, or ϵ.*

Proof. The main idea is that there must be some slice on which C computes f well and contains a large fraction of all inputs. Once this is proven, we show that either the monotone or anti-monotone circuit that agrees with C on the slice in question must compute f on at least a $\frac{1}{2} + \epsilon'$ fraction of the inputs. The choice between the monotone or anti-monotone circuit is made to ensure the circuit computes f with probability at least $\frac{1}{2}$ on inputs outside of the slice of interest.

We begin by considering for each slice i, the value A_i that the i-th slice contributes to the advantage C has in computing f. A_i is the difference between the number of x with $|x| = i$ and $C(x) = f(x)$ and the number of x with $|x| = i$ and $C(x) \neq f(x)$. We have by assumption that

$$\sum_{i=0}^{n} A_i \geq 2^n(2\epsilon).$$

By an averaging argument, there exists an index i such that $A_i \geq 2^n \frac{2\epsilon}{n+1}$. Lemma 1 gives us both a monotone circuit C_{mon} and an anti-monotone circuit $C_{anti-mon}$ of size $2s + O(n \log^2 n)$ that agree with C on the i-th slice. C_{mon} and $C_{anti-mon}$ thus have advantage at least $2^n \frac{2\epsilon}{n+1}$ in computing f on the i-th slice. Because C_{mon} and $C_{anti-mon}$ are complements outside of the i-th slice, at least one of them agrees with f on at least $\frac{1}{2}$ of all inputs outside of the i-th slice. Altogether, we have that either C_{mon} or $C_{anti-mon}$ has total advantage at least $2^n \frac{2\epsilon}{n+1}$ in computing f; equivalently at least one of the circuits computes f to within $\frac{1}{2} - \frac{\epsilon}{n+1}$.

The alternate value for ϵ' comes by only considering $\Theta(\sqrt{n \log(1/\epsilon)})$ slices around the middle which together contain $1 - \frac{\epsilon}{2}$ fraction of all strings. The Chernoff Bound of Lemma 3 tells us that if we pick an n-bit string at random, the probability that the Hamming weight deviates from $\lfloor \frac{n}{2} \rfloor$ by at least j is at most $\frac{\epsilon}{2}$ if we set j such that $2e^{-2j^2/n} \leq \frac{\epsilon}{2}$, so $j = \Theta(\sqrt{n \log(1/\epsilon)})$. Thus we remove from consideration at most $\frac{\epsilon}{2} 2^n$ strings by restricting to the $\Theta(\sqrt{n \log(1/\epsilon)})$ many slices closest to the middle, and therefore C must compute f correctly on at least $2^n(\frac{1}{2} + \frac{\epsilon}{2})$ of the remaining strings. We can now carry out an argument similar to the above – where instead of $n + 1$ many slices we consider $\Theta(\sqrt{n \log(1/\epsilon)})$ many and start from a circuit that is correct on at least $2^n(\frac{1}{2} + \frac{\epsilon}{2})$ of the strings in these slices rather than $2^n(\frac{1}{2} + \epsilon)$ – to obtain the alternate value of ϵ'. ∎

Tightness of Theorem 1 We observe that Theorem 1 is within a constant factor of being tight for large ϵ when applied to the parity function. The parity function can easily be computed by a small circuit, and applying Theorem 1 to this circuit, with $\epsilon = \frac{1}{2}$, gives either a monotone or anti-monotone circuit computing parity to within $\frac{1}{2} - \frac{c}{\sqrt{n}}$ for some constant c. On the other hand, it is well-known that no monotone or anti-monotone function can compute parity to within more than $\frac{1}{2} - O(\frac{1}{\sqrt{n}})$.

The hardness of parity for monotone functions can be seen, for example, by considering the average sensitivity of parity and monotone functions. The average sensitivity of a function f is $\sum_{i=1}^{n} \Pr_{x \in \{0,1\}^n}[f(x) \neq f(x \oplus e_i)]$ where $x \oplus e_i$ is equal to x but with the i-th bit flipped. Parity has average sensitivity equal to n, while it is known that every monotone (and anti-monotone) function has average sensitivity $O(\sqrt{n})$ (see, for example, [BT96]). We can pick an input x' uniformly at random by first picking x uniformly at random, picking i at random, and letting $x' = x$ with probability 1/2 and letting $x' = x \oplus e_i$ with probability 1/2. If f is monotone or anti-monotone, the bound on the average sensitivity implies that with probability $1 - \frac{1}{\Omega(\sqrt{n})}$, an input x and bit

position i are chosen so that $f(x) = f(x \oplus e_i)$. When this occurs, we obtain an x' such that $f(x') \neq parity(x')$ with probability 1/2. Overall, f differs from parity on a fraction $\frac{1}{2} - O(\frac{1}{\sqrt{n}})$ of the inputs.

3.2 Monotone Hard Functions

As mentioned in Section 1, results from learning theory tell us that no monotone function can be more than $(\frac{1}{2} - \Omega(\frac{\log n}{\sqrt{n}}))$-hard for circuits large enough to compute majority – linear-size general circuits or $O(n \log n)$-size monotone circuits. In this subsection, we observe that there do exist monotone functions whose hardness approaches this barrier.

First, [ACR97] prove a result which implies the existence of a mildly average-case hard monotone function. They establish an asymptotic characterization of how inapproximable a function can be on any subset of its inputs. In particular, there exist monotone slice functions which are hard to approximate on the middle slice; the precise parameters are stated in the next lemma. Recall that a Boolean function is balanced if it outputs 0 and 1 with the same frequency.

Lemma 4 (follows from [ACR97]). *There exist constants $c_1, c_2 > 0$ such that for sufficiently large n, the following holds. There is a balanced monotone slice function f such that no circuit with $s = \frac{c_1 2^n}{n^{3/2}}$ gates computes f to within $\frac{3}{4}$ on the middle slice and therefore to within $1 - \delta(n)$ overall at length n for $\delta(n) = \frac{c_2}{\sqrt{n}}$.*

In many settings, it has been shown that average-case hardness can be amplified by applying a hard function to many independent inputs and then taking the parity. Results of this form are called XOR lemmas. This fails as a method to amplify hardness for monotone functions because the resulting hard function would not be monotone. A similar issue arises when attempting to amplify the hardness of NP functions. To obtain a hardness amplification lemma for NP, O'Donnell [O'D04] showed that a monotone combining function can be used in place of parity with some loss in parameters. The technique can also be used in the setting of monotone functions, giving Lemma 5.

Lemma 5 (follows from [O'D04]). *Let H be a monotone function that is balanced and $\frac{1}{n^c}$-hard for circuits of size s, for some positive constant c. There is a polynomial p and a polynomial-time computable monotone function C such that $H' : \{0,1\}^{n \cdot p(n)} \to \{0,1\}$ defined as*

$$H'(x_1, x_2, ..., x_{p(n)}) = C(H(x_1), H(x_2), ..., H(x_{p(n)}))$$

is $(\frac{1}{2} - \frac{1}{(n \cdot p(n))^{1/2-\eta}})$-hard for circuits of size $\frac{s}{n^d}$ on inputs of length $n \cdot p(n)$, where d is a constant that depends on c.

By applying Lemma 5 to the hard function of Lemma 4, we obtain the following.

Theorem 4. *For every constant $\eta > 0$ there exists a constant $c(\eta) > 0$ and a monotone function f such that for sufficiently large n, f at length n is δ-hard for circuits of size $2^{n^{c(\eta)}}$, where $\delta = \frac{1}{2} - \frac{1}{n^{1/2-\eta}}$.*

We point out that the hard function of Theorem 4 is computable in $\mathrm{E}^{\Sigma_2^p}$, exponential time with an oracle to the second level of the polynomial hierarchy, using the same techniques that show $\mathrm{E}^{\Sigma_2^p}$ contains a language with maximal general circuit complexity (see [MVW99] for a discussion of those techniques).

4 Pseudorandom Generators

In this section we prove our results concerning distinguishers of (pseudorandom) distributions.

Reduction to Monotone Adversaries Theorem 2 states that a circuit that distinguishes a distribution from uniform can be converted into a monotone distinguisher with somewhat weaker parameters.

Theorem 2 (restated). *Let C be a circuit of size s that ϵ-distinguishes some distribution $\mathcal{D}$ from uniform. Then there is a monotone circuit C' of size $2s + O(n \log^2 n)$ that ϵ'-distinguishes $\mathcal{D}$ from uniform for $\epsilon' = \max(\frac{\epsilon}{2(n+1)}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$ for $c > 0$ an absolute constant that does not depend on n, s, or ϵ.*

Proof. The proof is essentially identical to that of Theorem 1 except in the setting of distinguishers rather than computing a Boolean function. The main idea is to find a slice i on which C $O(\epsilon')$-distinguishes and let C' compute a monotone slice function agreeing with C on that slice. A simple calculation then shows that either C' or the threshold function outputting 1 iff $|x| > i$ distinguishes with probability ϵ' over all inputs.

Let C be an ϵ-distinguisher of size s for $\mathcal{D}$. By definition, $|\Pr_{X \leftarrow U_n}[C(X) = 1] - \Pr_{Y \leftarrow \mathcal{D}}[C(Y) = 1]| \geq \epsilon$. Without loss of generality, we assume that the inequality holds without the absolute value signs. By breaking the probability space into disjoint events, we have that

$$\sum_{i=0}^{n} (\Pr_{X \leftarrow U_n}[C(X) = 1 \text{ and } |X| = i] - \Pr_{Y \leftarrow \mathcal{D}}[C(Y) = 1 \text{ and } |Y| = i]) \geq \epsilon.$$

By an averaging argument, there exists an index i such that $\Pr_{X \leftarrow U_n}[C(X) = 1 \text{ and } |X| = i] - \Pr_{Y \leftarrow \mathcal{D}}[C(Y) = 1 \text{ and } |Y| = i] \geq \frac{\epsilon}{n+1}$. By Lemma 1, there is a monotone circuit C_{mon} that agrees with C on the i-th slice and uses at most $2s + O(n \log^2 n)$ gates. The overall distinguishing probability of C_{mon} can be expressed as

$$\begin{aligned}
&(\Pr_{X \leftarrow U_n}[C_{mon} = 1 \text{ and } |X| = i] - \Pr_{Y \leftarrow \mathcal{D}}[C_{mon} = 1 \text{ and } |Y| = i]) \\
&+(\Pr_{X \leftarrow U_n}[C_{mon} = 1 \text{ and } |X| > i] - \Pr_{Y \leftarrow \mathcal{D}}[C_{mon} = 1 \text{ and } |Y| > i]) \\
&+(\Pr_{X \leftarrow U_n}[C_{mon} = 1 \text{ and } |X| < i] - \Pr_{Y \leftarrow \mathcal{D}}[C_{mon} = 1 \text{ and } |Y| < i]).
\end{aligned}$$

The last term is 0 because C_{mon} outputs 0 on strings of weight less than i. The middle term is $\Pr_{X \leftarrow U_n}[|X| > i] - \Pr_{Y \leftarrow \mathcal{D}}[|Y| > i]$ because C_{mon} outputs 1 on strings of weight greater than i. If the absolute value of this term is greater than $\frac{\epsilon}{2(n+1)}$, then the threshold function that outputs 1 iff $|X| > i$ – computable by $O(n \log n)$ size monotone circuits [AKS83] – is an $\frac{\epsilon}{2(n+1)}$-distinguisher. Otherwise, the distinguishing probability of C_{mon} is at least

$$(\Pr_{X \leftarrow U_n}[C_{mon}(X) = 1 \text{ and } |X| = i] - \Pr_{Y \leftarrow \mathcal{D}}[C_{mon}(Y) = 1 \text{ and } |Y| = i]) - \frac{\epsilon}{2(n+1)} \geq \frac{\epsilon}{2(n+1)}.$$

The alternate value for ϵ' comes by only considering $\Theta(\sqrt{n \log(n/\epsilon)})$ layers around the middle, which together contain a fraction $1 - \frac{\epsilon}{2}$ of all strings. These layers collectively distinguish with $\frac{\epsilon}{2}$ advantage, so one of them must distinguish with $\Omega(\frac{\epsilon}{\sqrt{\log(n/\epsilon)}})$ advantage. The analysis for this case is the same as for the corresponding case of Theorem 1. ∎

Remark In the setting of general circuits, it is known that the existence of explicit pseudorandom generators is equivalent to the existence of explicit functions that are hard on average. A natural question is whether this remains true in the setting of monotone circuits; if so then Theorem 2 for the case of pseudorandom distributions would follow as a corollary to Theorem 1. A simple argument shows that the language L defined as the set of strings output by a pseudorandom generator secure against certain adversaries must be *worst-case* hard for those same adversaries. The argument carries through for monotone circuits, but worst-case hardness is not enough to apply Theorem 1. For general circuits and pseudorandom generators computable in exponential time in the seed length, [NW94] observe that L must be *average-case* hard by appealing to the known worst-case to average-case reductions for languages computable in exponential time. These reductions do not seem to preserve monotonicity and therefore do not prove a connection between pseudorandom generators secure against monotone circuits and average-case hard functions for monotone circuits.

Tightness of Theorem 2 One question is whether the parameters in Theorem 2 can be tightened further. It is well-known that a hard function f can be used to create a generator G^f, defined by $G^f(x) = (x, f(x))$, that is a pseudorandom generator with 1 bit of stretch. When dealing with monotone circuits as the distinguishers, we can use parity as the hard function to obtain the following theorem.

Theorem 5. *Define a generator $G^{\oplus}$ as follows: $G^{\oplus}(x) = (x, \oplus(x))$. Then $G^{\oplus} : \{0,1\}^n \to \{0,1\}^{n+1}$ is a ϵ-pseudorandom generator secure against monotone and anti-monotone circuits of any size, where $\epsilon = \frac{c}{n^{1/2}}$ for some absolute constant $c > 0$.*

The proof for the setting of general circuits is standard, and it can be verified that the standard proof preserves monotonicity. For completeness we provide the proof for the setting of Theorem 5 in the appendix.

Theorem 5 shows that Theorem 2 is tight to within a constant factor for large ϵ: $G^{\oplus}$ is easily distinguishable with $\epsilon = \frac{1}{2}$ by a small general circuit, and applying Theorem 2 to this circuit produces a monotone circuit that $\frac{\gamma}{\sqrt{n}}$-distinguishes $G^{\oplus}$ from uniform for some constant γ – a monotone distinguisher within a constant factor of optimal.

5 Derandomization

In this section we show that *any* method of derandomizing monotone randomized circuits can also be used to derandomize general non-monotone circuits.

Monotone Randomized Computations One natural definition for the class of monotone randomized computations is the set of BPP languages that are also monotone. However, one can easily reduce any BPP language L to this class by simply embedding the truth table of L within the middle slice of a monotone function. Thus, with this definition of monotone BPP, derandomizing monotone BPP trivially implies derandomizing all of BPP.

We instead consider another natural, more restrictive, definition of monotone randomized computations, namely the set of languages that can be solved by uniform bounded-error monotone randomized circuits. The uniformity requirement is that on input 1^n, the circuit can be output in

poly(n) time. The resulting circuit should be monotone in both the input and random bits and should have bounded error on every input.

We point out that there exist monotone languages in BPP that are not computable by uniform bounded-error monotone randomized circuits. This follows from two facts. First, randomness can be removed from bounded-error monotone randomized computations by reducing the error to be less than 2^{-n} (which only uses majority and thus preserves monotonicity) and then fixing a random string that is correct for all inputs; thus bounded-error randomized monotone circuits can be simulated efficiently by non-uniform deterministic monotone circuits. Second, [Raz85] and [Tar87] demonstrate monotone languages in P, and thus also BPP, that require non-uniform monotone circuits of super-polynomial size (exponential size for the result of [Tar87]).

Despite the more restrictive character of our notion of monotone randomized computation, Theorem 3 establishes an efficient reduction from any BPP language L to languages solvable by this weaker model of randomized monotone computations.

Theorem 3 (restated). *Let L be any language computable by polynomial-time bounded-error randomized machines. There is a language L_{mon} computable by uniform monotone bounded-error polynomial-size randomized circuits such that L poly-time mapping reduces to L_{mon}. In particular, if $L_{mon} \in \mathrm{P}$ then $L \in \mathrm{P}$.*

Proof. Let M be a bounded-error randomized machine running in time n^k computing a BPP language L, for some constant k. The basic idea is to take the function computed by the deterministic machine underlying M and embed it within a monotone slice function. Viewing this monotone slice function as a randomized monotone circuit, we must ensure the following.

(i) The circuit has bounded error on all inputs.

(ii) L polynomial-time many-one reduces to the language computed by the circuit.

Let $f : \{0,1\}^n \times \{0,1\}^{n^k} \to \{0,1\}$ be the function computed by M given an n-bit input x and random string r of length n^k. To produce a randomized monotone circuit, we separately embed both the input and the random string into the middle slice of larger Boolean cubes. To embed the input we can use the simple embedding associating x with the $2n$-bit string $(x, \overline{x})$. We must take more care with the embedding of the random bits because the circuit must have error bounded away from one half on each input. We achieve this by using the embedding of Lemma 2.

Let m be the smallest even integer such that $\binom{m}{m/2} \geq 2^{n^k}$. Because $\binom{m}{\lfloor m/2 \rfloor}$ grows by less than a factor of two for each increment of m, we also have that $\binom{m}{m/2} \leq 4 \cdot 2^{n^k}$. Consider the function f_{mon} that takes an input x' of $2n$ bits and a random string r' of m bits and behaves as follows.

1. *Slice function of x'*
   If $|x'| > n$, set $f_{mon}(x', r') = 1$. If $|x'| < n$, set $f_{mon}(x', r') = 0$.

2. *Slice function of r' for x' on middle slice*
   If $|x'| = n$ and $|r'| > m/2$, set $f_{mon}(x', r') = 1$.
   If $|x'| = n$ and $|r'| < m/2$, set $f_{mon}(x', r') = 0$.

3. *Embed f within middle slice of f_{mon}*
   If $x' = (x, \overline{x})$ for some x of length n and $|r'| = m/2$, do the following. If r' is among the 2^{n^k} strings matched with $\{0,1\}^{n^k}$ by the embedding of Lemma 2, let r be the associated value and set $f_{mon}(x', r') = f(x, r)$. For r' that do not have a match within $\{0,1\}^{n^k}$, set $f_{mon}(x', r')$ to 0 on half of these and 1 on the other half.

4. *Other x' on the middle slice*
   If $|x'| = n$, x' is not of the form $(x, \overline{x})$, and $|r'| = m/2$, set $f_{mon}(x', r') = 0$.

We first argue that $f_{mon}(x', \cdot)$ has error bounded away from $1/2$ by at least $1/\text{poly}(n)$ on each input x'. For x' of the form $(x, \overline{x})$, the construction ensures $\Pr_{r'}[f_{mon}(x', r') = 1] = \frac{1}{2} \cdot (1 - \rho) + \rho \cdot \Pr_r[f(x, r) = 1]$, where ρ is the fraction of strings used by the embedding of n^k-bit random strings into the middle slice of the m-cube. By our choice of m, $m = n^k + O(\log n)$ and $\rho = \Theta(\frac{1}{\sqrt{m}})$. Thus the majority value of $f_{mon}(x', \cdot)$ agrees with the majority value of $f(x, \cdot)$, and the error is bounded away from one half by $1/\text{poly}(n)$.

For x' with $|x'| = n$ that is not of the form $(x, \overline{x})$, steps 2 and 4 ensure error bounded away from one half as well – for such x', $\Pr_{r'}[f_{mon}(x', r') = 0] \geq \frac{1}{2} + \frac{1}{\text{poly}}$. For x' with $|x'| \neq n$, $f_{mon}(x', \cdot)$ is either the constant 0 or constant 1 by the first step.

Let us see that f_{mon} can be computed by a uniform polynomial-size circuit. Let C be a uniform polynomial-size circuit for f_{mon}; we wish to remove the negations from this circuit without increasing the size too much. We first push the negations to the inputs, at most doubling the circuit size. Because f_{mon} is a monotone slice function of x', as noted in the discussion before Lemma 1, we can replace the negations of those variables by a monotone circuit of size $O(n \log^2 n)$. For x' on the non-trivial slice of f_{mon}, f_{mon} is a monotone slice function of r', so we can replace the negations of those variables by a monotone circuit of size $O(m \log^2 m)$. We conclude that f_{mon} has a uniform polynomial-size circuit.

We can reduce the error of the circuit from $\frac{1}{2} - 1/\text{poly}$ to $\frac{1}{3}$ using standard error reduction consisting of taking multiple trials and majority voting. This can be implemented by a uniform monotone circuit of polynomial size [AKS83]. The result is a uniform polynomial-size monotone circuit C_{mon} such that for every x, if $\Pr_R[M(x, R) = 1] \geq \frac{2}{3}$ then $\Pr_{R'}[C_{mon}((x, \overline{x}), R') = 1] \geq \frac{2}{3}$, and if $\Pr_R[M(x, R) = 1] \leq \frac{1}{3}$ then $\Pr_{R'}[C_{mon}((x, \overline{x}), R') = 1] \leq \frac{1}{3}$. ■

Acknowledgments

We thank Tassos Viglas, Iannis Tourlakis, and Nicola Galesi for helpful discussions, and Valentine Kabanets for pointing out useful references. We also thank Matt Anderson and Scott Diehl for helpful discussions and comments.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[ACR97] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. Optimal bounds for the approximation of Boolean functions and some applications. *Theoretical Computer Science*, 180(1-2):243–268, 1997.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An 0(n log n) sorting network. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 1–9, 1983.

[AS00] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley-Interscience, second edition, 2000.

[Ber82] S.J. Berkowitz. On some relationships between monotone and non-monotone circuit complexity. Technical report, University of Toronto, 1982.

[BS91] Ravi B. Boppana and Michael Sipser. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*, chapter The Complexity of Finite Functions, pages 757–804. MIT Press, 1991.

[BT96] Nader H. Bshouty and Christino Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43:747–770, 1996.

[Kar09] George Karakostas. General pseudo-random generators from weaker models of computation. In *Proceedings of the International Symposium on Algorithms and Computation*, pages 1094–1103, 2009.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1/2):1–46, 2004.

[Kin10] Jeff Kinne. *Deterministic Simulations and Hierarchy Theorems for Randomized Algorithms*. PhD thesis, University of Wisconsin-Madison, 2010.

[KKL88] Jeff Kahn, Gil Kalai, and Nathan Linial. The influence of variables on Boolean functions. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 68–80, 1988.

[KLV94] Michael Kearns, Ming Li, and Leslie Valiant. Learning Boolean formulas. *Journal of the ACM*, 41(6):1298–1328, 1994.

[Knu06] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3, Generating All Combinations and Partitions*. Addison-Wesley Professional, 2006.

[Kor03] Aleksej D. Korshunov. Monotone Boolean functions. *Russian Math. Surveys*, 58(5):929–1001, 2003.

[Mil01] Peter Bro Miltersen. Derandomizing complexity classes. In *Handbook of Randomized Computing*, pages 843–941. Kluwer Academic Publishers, 2001.

[MVW99] Peter Bro Miltersen, N. Variyam Vinodchandran, and Osamu Watanabe. Superpolynomial versus half-exponential circuit size in the exponential hierarchy. In *Proceedings of the Annual International Computing and Combinatorics Conference*, pages 210–220, 1999.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[O'D04] Ryan O'Donnell. Hardness amplification within NP. *Journal of Computer and System Sciences*, 69(1):68–94, 2004.

[OW09] Ryan O'Donnell and Karl Wimmer. KKL, Kruskal-Katona, and monotone nets. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 2009.

[Raz85] Alexandor Razborov. A lower bound on the monotone network complexity of the logical permanent. *Matematicheskie Zametki*, 37(6):887–900 (in Russian), 1985. English translation in *Mathematical Notes of the Academy of Sciences of the USSR* 37:6, 485–493.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[Tar87] Eva Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 7(4):141–142, 1987.

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[Val86] Leslie G. Valiant. Negation is powerless for Boolean slice functions. *SIAM Jounral on Computing*, 15(2):531–535, 1986.

Proof of Theorem 5

We follow the standard proof from the general setting and keep track of monotonicity to verify the final circuit is monotone or anti-monotone. We assume a monotone or anti-monotone circuit C that ϵ-distinguishes the output of $G^{\oplus}$ from uniform. We would like to use C to compute parity on some n-bit string x. If C were a perfect distinguisher then for any x, $C(x, \oplus(x)) = 1$ and $C(x, \overline{\oplus(x)}) = 0$. C is not a perfect distinguisher, but we treat it as if it were and analyze the probability that we are correct. Namely, we choose a random bit b and query the value $C(x, b)$. If $C(x, b) = 1$ we assume $\oplus(x) = b$; if $C(x, b) = 0$ we assume $\oplus(x) = \overline{b}$. A random bit b is equal to $\oplus(x)$ with probability $\frac{1}{2}$ and is equal to $\overline{\oplus x}$ with probability $\frac{1}{2}$, so the probability we output the correct value for $\oplus(x)$ is

$$\frac{1}{2}(\Pr_{X \in U_n}[C(X, \oplus(X)) = 1] + \Pr_{X \in U_n}[C(X, \overline{\oplus(X)}) = 0]). \tag{1}$$

We use the fact that C is an ϵ-distinguisher to lower bound (1). We have that

$$|\Pr_{X \in U_n}[C(X, \oplus(X)) = 1] - \Pr_{X \in U_n, \beta \in U_1}[C(X, \beta) = 1]| \geq \epsilon.$$

By expressing the second term as a sum depending on whether β is $\oplus(X)$ or $\overline{\oplus(X)}$, we have

$$\begin{array}{ll} \frac{1}{2}|\Pr_{X \in U_n}[C(X, \oplus(X)) = 1] - \Pr_{X \in U_n}[C(X, \overline{\oplus(X)}) = 1]| & \geq \epsilon, \text{ and therefore} \\ \frac{1}{2}|\Pr_{X \in U_n}[C(X, \oplus(X)) = 1] + \Pr_{X \in U_n}[C(X, \overline{\oplus(X)}) = 0] - 1| & \geq \epsilon. \end{array}$$

If the sign on the absolute value is positive, we have that (1) is at least $\frac{1}{2} + \epsilon$. Otherwise we have that (1) is at most $\frac{1}{2} - \epsilon$; in that case the negation of our strategy is correct with probability at least $\frac{1}{2} + \epsilon$.

Let us verify that this strategy produces a monotone or anti-monotone circuit. First, there is a value for b that preserves the probability of success, and we can fix this value into the circuit. If b is fixed to 1, then our strategy outputs $C(x, 1)$; if b is set to 0, our strategy outputs $\overline{C(x, 0)}$. Due to the sign on the absolute value, we may need to place an additional negation at the top of the final circuit. We have that if C is an ϵ-distinguisher for $G^{\oplus}$ then one of $C(x, 1), \overline{C(x, 1)}, \overline{C(x, 0)}, C(x, 0)$ computes parity to within $\frac{1}{2} - \epsilon$. If C is monotone or anti-monotone, then so are each of these circuits, but we know that parity cannot be computed to within $\frac{1}{2} - \epsilon$ by monotone or anti-monotone circuits for $\epsilon \geq c\frac{1}{\sqrt{n}}$ for an appropriate constant c, as mentioned at the end of Section 3.1.