Meta-Algorithms versus Circuit Lower Bounds

by

Ruiwen Chen

M.Eng., Renmin University of China, 2007

B.Eng., Renmin University of China, 2004

Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2014

APPROVAL

Name: Ruiwen Chen

Degree: Doctor of Philosophy

Title of Thesis: Meta-Algorithms versus Circuit Lower Bounds

Examining Committee: Dr. Ramesh Krishnamurti, Professor
Chair

Dr. Valentine Kabanets
Senior Supervisor
Associate Professor

Dr. Binay Bhattacharya
Supervisor
Professor

Dr. Andrei Bulatov
Internal Examiner
Associate Professor

Dr. Ramamohan Paturi
External Examiner
Professor of Computer Science,
University of California, San Diego

Date Defended: July 17th, 2014

Partial Copyright Licence

Abstract

Computational complexity theory and algorithms are two major areas in theoretical computer science. Computational complexity theorists study the inherent difficulty of computational problems, and classify problems based on computational resources needed. In particular, complexity theorists intend to prove lower bounds, that is, showing certain problems are not solvable under specific resource bounds. On the other hand, algorithm designers wish to discover efficient algorithms solving particular problems. Traditionally, researchers in the two areas have different goals and use different techniques. However, several recent breakthroughs indicate fundamental connections in between. In particular, new lower bounds were proved via designing efficient algorithms, and reversely, known lower bound proofs were exploited to design efficient algorithms.

In this thesis, we focus on the connections between algorithms and lower bounds, which lead to discoveries of new algorithms and lower bounds. For small-size boolean formulas, we get new satisfiability counting algorithms. The algorithm is based on a simplified proof of the property that formula size shrinks with high probability under certain random restrictions. This approach is further adapted to get satisfiability algorithms and average-case lower bounds for small linear-size boolean circuits. We also show that circuit lower bound proofs based on the method of random restrictions yield non-trivial compression algorithms for easy boolean functions from the corresponding circuit classes. In the reverse, we show that the existence of non-trivial compression algorithms would imply circuit lower bounds for non-deterministic exponential time. In the end, we introduce the notion of weakly-uniform circuits, and generalize previous known lower bounds against uniform circuits to weakly-uniform circuits.

Keywords: satisfiability algorithm; circuit lower bound; boolean formula; boolean circuit; compression algorithm

Acknowledgments

It is my pleasure to thank many people for their support during my PhD studies. Above all, I would like to thank my advisor, Dr. Valentine Kabanets, for his guidance, inspiration and encouragement. Valentine is a knowledgeable and patient advisor. Through our weekly meetings, Valentine provided insightful discussions about the research, and shared with me his valuable experiences, ideas and intuitions. I am grateful to Valentine for bringing me to the area of computational complexity, for his inviting attitude towards research, and for his support and advice over the years.

I am also grateful to other members of my committee. I am thankful to my co-advisor Dr. Binay Bhattacharya for his encouragement and his inspiring lectures. I am indebted to my examiner Dr. Andrei Bulatov for his valuable questions and comments on my research. I am very fortunate to have Dr. Ramamohan Paturi as my external examiner; I admire his research contributions to the area. I am also thankful to Dr. Ramesh Krishnamurti for serving as the chair of my committee.

Many of the results in this thesis were done through collaborations. I am grateful to my coauthors: Valentine Kabanets, Jeff Kinne, Antonina Kolokolova, Nitin Saurabh, Ronen Shaltiel, and David Zuckerman.

I owe thanks to my advisors in my previous graduate and undergraduate studies: Dr. Xiaoyong Du, Dr. Iluju Kiringa, Dr. Yongyi Mao, Dr. Jian Pei, and Dr. Shan Wang. I am also thankful to my mentors in my previous work and study: Y. Feng, J. Leng, W. Tang, W. Zhang, and Q. Zhou. I have been very lucky to receive valuable advice and support from all of them.

Finally, I would like to express my deep gratitude to my parents and my grandparents for their unconditional support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computational complexity theory studies what computational problems are, or are not, efficiently solvable. In this thesis, we focus on the fundamental connections between proving hardness of problems and designing efficient solutions.

1.1 Algorithms and Lower Bounds

Complexity theory and algorithm design are two major areas of theoretical computer science. Algorithm designers discover efficient solutions for specific problems, and demonstrate the power of computation. Over the years, algorithm designers have made huge impact to the industry by discovering efficient solutions (algorithms) to practically interesting problems. On the other hand, complexity theorists wish to show the limits of computation by proving *lower bounds*, that is, proving certain hard problems do not have any relatively efficient solution. Although complexity theorists do not solve problems directly, they develop a foundational framework for analyzing the efficiency of computation, and use the framework to explain the impossibility results.

Algorithms explore the power of computation, while lower bounds characterize the limits of computation. Traditionally, the two areas have different goals and use different techniques. Designing algorithms is arguably easier than proving lower bounds. To solve a problem, algorithm designers only need to find one efficient solution. However, to prove a lower bound for a problem, complexity theorists must argue that no efficient solution exists.

Essentially, algorithm designers analyze the structure of the problem at hand, but complexity theorists reason more on algorithms themselves. While algorithm designers can constantly make progress by gradually refining their solutions, complexity theorists can not claim any result before they try (or have a way to rule out) all the possibilities. We have seen dramatic and versatile progress in algorithms over the years, but not much in proving lower bounds.

However, in recent years, fundamental connections between algorithms and lower bounds have been discovered in several breakthroughs. On the one hand, new lower bounds have been proved via designing relatively efficient algorithms; on the other hand, new algorithms for hard problems have been derived using techniques from lower bound proofs. In this thesis, we will study more instances of such connections.

1.1.1 Algorithms for Boolean Formulas

To better illustrate the power of computation, we will use several problems related to boolean formulas as examples.

Let $\{x_i\}_{i\in\mathbb{N}}$ be a collection of *variables* with domain $\{0,1\}$, where $\mathbb{N} = \{1,2,\ldots\}$. A *boolean formula* is an expression composed with variables in $\{x_i\}$, constants in $\{0,1\}$, and logical operators in $\{\neg,\wedge,\vee\}$. More precisely, a variable or a constant is a boolean formula, and, if E and F are boolean formulas, then so are $\neg E, (E\vee F), (E\wedge F)$. For example, the following is a boolean formula on variables $\{x_1,x_2\}$:

$$(x_1\wedge\neg x_2)\vee(\neg x_1\wedge x_2). \tag{1.1}$$

An *assignment* to n variables is a mapping from $[n] = \{1,\ldots,n\}$ to $\{0,1\}$. Given a boolean formula and an assignment to its variables, we can evaluate the formula by substituting each variable with the corresponding value in the assignment. We say an assignment *satisfies* a formula if the formula evaluates to 1 under the assignment. A boolean formula on n variables computes a boolean function from $\{0,1\}^n$ to $\{0,1\}$. For example, the formula in (1.1) outputs 1 iff $x_1\neq x_2$.

We define the *size* of a formula to be the total number of appearances of all variables. For example, the formula in (1.1) has size 4. We usually parametrize the size as a function $s(n)$ of the the number of variables n. We say a formula over n variables has *polynomial size* if the size $s(n)$ is upper-bounded by for some polynomial of n. In this subsection, we will consider only formulas of polynomial size.

Now we consider the following two computational problems:

- FormulaEval: Given a formula and an assignment, check whether the assignment satisfies the formula.
- FormulaSAT: Given a formula, check whether there exists an assignment which satisfies the formula.

The FormulaEval problem is obviously easier. We can substitute the assignment into the formula, and evaluate each sub-formula one by one. The number of steps taken in this evaluation algorithm is at most polynomial in the formula size. Thus we say that FormulaEval is *efficiently decidable*. We denote by P the class of problems which are efficiently decidable.

The FormulaSAT problem is much harder. A quick solution is to enumerate all possible assignments and evaluate the formula under each assignment to check if there is any satisfying one. Such a procedure is usually called *brute-force search*. It will take exponential number of steps to enumerate and evaluate the formula on all possible assignments. This satisfiability algorithm is not efficient, but it raises the following question: Does there exist an algorithm which solves FormulaSAT in less than exponential number of steps? Until now, we do not know the answer yet. We do not even know, for formulas on n variables, whether there is an algorithm running in $2^{0.9n}$ steps. There is a large family of problems like FormulaSAT for which we can neither give an efficient solution nor prove there are no efficient solutions.

Nevertheless, we do know that FormulaSAT is *efficiently verifiable*. That is, if a formula is satisfiable, then there is a satisfying assignment which can be verified efficiently (using the algorithm for FormulaEval). We denote by NP the class of problems which are efficiently verifiable. The famous P versus NP question asks whether efficiently verifiable problems like FormulaSAT are also efficiently decidable. Although most complexity theorists conjecture this is not true, we are still far from proving or disproving it.

1.1.2 Boolean Formulas as Computational Models

We now take a different perspective on boolean formulas. A boolean formula computes a boolean function, so it can be used as a computational device to solve problems (generating outputs from inputs). The drawback is that, each formula has a fixed number of input

variables. To overcome this, we introduce a *family of boolean formulas* $\{F_n\}_{n\in\mathbb{N}}$, where each formula F_n is over n input variables. A family of formulas can be used to solve a general computational problem, with any input size. Such models are called *non-uniform* computational models.

Now one may ask the following questions. Given a problem, does there exist a family of formulas solving the problem? If true, how can we construct such formulas? What will be the size of the formulas?

Since the truth table of a function can be encoded as a formula of exponential size, every problem is solvable by a family of exponential-size formulas. But do we have formulas of smaller size to solve specific problems like FormulaEval, FormulaSAT or even simpler ones?

We consider a very simple problem of computing the Parity of n variables, where the output is true iff the sum of the inputs is odd. Parity is easily computable by an algorithm summing through each input one by one. But how can we compute Parity using boolean formulas? In fact, the formula in (1.1) computes Parity on two inputs, and we can compound such formulas to get a formula of size n^2 which computes Parity on n inputs. Can we make even smaller formulas to compute Parity? The answer can be proved to be No.

What will be the smallest formula size required to solve FormulaEval or FormulaSAT? Till now, we do not know the answer better than the trivial exponential size. The best known formula-size lower bound is that, there is a problem in P which requires formula size almost n^3. We do not know any larger lower bounds for problems in NP.

Complexity theorists tend to favor a model which is more general than boolean formulas, that is, boolean circuits. A *boolean circuit* on n input variables of size s is a sequence of $n+s$ functions $C = (f_1, \ldots, f_{n+s})$ where (1) $f_i = x_i$ for $i = 1, \ldots n$, and (2) $f_i = \neg f_j$, $f_i = f_j \vee f_k$, or $f_i = f_j \wedge f_k$, for $i > n$ and $j, k < i$. The output of f_{n+s} is designated as the output of the circuit. A boolean circuit can also be viewed as a directed acyclic graph with n source nodes and s internal nodes which are labeled by f_i's, and the incoming arcs of f_i are from the nodes that f_i directly depends on. Boolean formulas are special cases of boolean circuits where the fan-out of each internal node is 1. Similar as boolean formulas, we define the following problems:

- CircuitEval: Given a circuit and an assignment, check whether the assignment satisfies the circuit.

- CircuitSAT: Given a circuit, check whether there is an assignment which satisfies the circuit.

It is not hard to see that CircuitEval is efficiently decidable and CircuitSAT is efficiently verifiable. Similar as FormulaSAT, we do not know if CircuitSAT is efficiently decidable, or if it is solvable by an algorithm better than the brute-force search.

Now we consider the use of boolean circuits as computational models. We also introduce a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ in order to solve computational problems. Then, what is the smallest circuit size required to compute problems like Parity, FormulaEval/CircuitEval, and FormulaSAT/CircuitSAT?

For the easy case of computing Parity, one can construct a circuit of size $3n$ following a similar approach as that for formulas. It can also be proved that circuits even slightly smaller than $3n$ can not compute Parity. For solving FormulaEval and CircuitEval, we can apply the known simulations of algorithms by circuits, which hardwires the computational process as circuit connections, and derive that any efficiently solvable problem (any problem in P including FormulaEval and CircuitEval) can also be solved by a family of polynomial-size circuits. For the harder problems like FormulaSAT and CircuitSAT, we do not know if they can (or can not) be solved by polynomial-size circuits. In fact, if one can prove that FormulaSAT or CircuitSAT can not be solved by polynomial-size circuits, that will resolve the P versus NP question. The currently best known circuit-size lower bound is that, there is a problem in P which requires circuit size almost $5n$. Again, we do not have any larger lower bounds for problems even in NP.

To summarize, we have seen two different roles of boolean circuits/formulas: on the one hand, they were used as inputs to evaluation or satisfiability algorithms; on the other hand, they were used as non-uniform computational models, which can simulate algorithms. We call algorithms like satisfiability algorithms as *meta-algorithms*, that is, algorithms on "algorithms". As introduced in the beginning of this section, several recent breakthroughs indicate fundamental connections between designing better satisfiability algorithms and prove circuit lower bounds. In this thesis, we will study more connections along this direction, and make some partial progress on the way to resolve some of the open questions raised in this section.

1.2 Contributions of This Thesis

Satisfiability algorithms and lower bounds for small-size boolean formulas. Boolean formulas have the property that, when we randomly assign values to a fraction of the variables, the formula on the remaining variables can be simplified such that the size shrinks. In Chapter 3, we show that the shrinkage happens with high probability under certain process of random restrictions, and, using this property, we get a satisfiability algorithm for boolean formulas of size $n^{2.49}$. We also give an alternative proof of the average-case lower bounds on such formulas.

Compression algorithms and circuit lower bounds. In Chapter 4, we explore the connections between compression algorithms and circuit lower bounds, where the compression algorithms take the truth table of a function and output a relatively succinct representation of the function. The compression algorithms rely on structural characterizations of "easy" functions, which are useful both for proving circuit lower bounds and for designing satisfiability algorithms. We also show that the existence of efficient compression algorithms would imply circuit lower bounds.

The work in Chapter 3 and 4 was done jointly with V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman; it appeared in the 29th IEEE Conference on Computational Complexity (CCC 2014) [CKK$^+$14].

Improved FormulaSAT algorithms. In Chapter 5, we get an improved analysis of the shrinkage property of boolean formulas, and this gives a non-trivial satisfiability algorithm for formulas of size $n^{2.63}$. This is a joint work with V. Kabanets and N. Saurabh; it will appear in the 39th International Symposium on. Mathematical Foundations of Computer Science (MFCS 2014) [CKS13].

Satisfiability algorithms and lower bounds for small linear-size boolean circuits. In Chapter 6, we revisit the gate-elimination method which was used to prove fixed linear-size circuit lower bounds, and generalize it to get concentrated circuit-size reduction under certain random restrictions. The approach is similar to the concentrated shrinkage of boolean formulas, and it leads to average-case lower bounds and #SAT algorithms for circuits over de Morgan basis of size almost $3n$, as well as for circuits over the full basis of size almost $2.5n$. This is an ongoing joint work with V. Kabanets.

Lower bounds against weakly-uniform circuits. In Chapter 7, we introduce the notion of weakly-uniform circuits, as a generalization of uniform circuits. Uniform circuits are

circuits that can be generated by algorithms, while weakly-uniform can be generated by algorithms taking certain amount of extra inputs (which are called advice). We generalize all previous known lower bounds against uniform circuits to weakly-uniform circuits. This is a joint work with V. Kabanets and J. Kinne. The conference version (with Kabanets) appeared in the 18th Annual International Computing and Combinatorics Conference (COCOON 2012) [CK12], and the journal version (with Kabanets and Kinne) appeared in Algorithmica, August 2013, Springer [CKK13a].

Chapter 2

Preliminaries

Solving a computational problem requires computational resources, such at time and space. Complexity classes categorize problems based on resource bounds in computation. In this chapter we will introduce terminologies on complexity classes, with emphasis on circuit complexity classes. We will then review circuit lower bounds, as an approach to separate complexity classes. Finally, we will survey the recent connections between algorithms and circuit lower bounds, which is the theme of this thesis.

2.1 Complexity Classes

2.1.1 Turing Machines, Time and Space Bounds

Let $\{0,1\}^* = \cup_{n\in\mathbb{N}}\{0,1\}^n$ be the collection of all binary strings. A *language* is a subset of $\{0,1\}^*$, or equivalently, a mapping from $\{0,1\}^*$ to $\{0,1\}$ such that $x \in L$ iff $L(x) = 1$. A decision problem for L is to decide whether or not a given string is in L.

A *Turing machine* (TM) is a tuple $M = (Q, \Sigma, \delta)$ where

- Q is a finite set of states, containing a start state q_0, an accepting state q_y, and a rejecting state q_n. The states q_y and q_n are distinct.
- Σ is the alphabet which is a collection of symbols, containing a special blank symbol $\perp$. The input does not contain the blank symbol $\perp$.
- δ is a transition function from $Q \times \Sigma$ to $Q \times \Sigma \times \{\leftarrow, \rightarrow, -\}$.

In its physical implementation, a Turing machine has a tape, a cursor, a state register, and a lookup table for transitions. The tape is divided into cells and initialized with blank symbols; we assume it extends infinitely at both ends. At the beginning, the tape contains the input string, the cursor locates at the first input symbol, and the machine starts with state q_0. At each step, given the current state q and the symbol a at the cursor, the machine follows the transition rule $\delta(q, a) = (q', b, d)$ by changing to state q', overwriting the cell at the cursor with b, and moving the cursor in the direction d.

Given a TM and an input string, a *configuration* describes the running context of the machine. Formally, a configuration is a tuple (q, u, w), where q is the current state, u is the string to the left the cursor, and w is the string to the right of the cursor.

We say a configuration c yields another configuration c' if the machine moves one step from c to c'. A TM M *accepts* an input string x if and only if there exists a sequence of configurations $c_0, c_1, \ldots, c_n$ such that c_0 is the start configuration, c_i yields c_{i+1} for $0 \leq i \leq n-1$, and c_n is a configuration with the accepting state. Given an input, a TM may accept it, reject it, or run forever. The language decided by a Turing machine M is $L(M) = \{w \in \Sigma^\star \mid M \text{ accepts } w\}$.

The running time of a TM M on input x, denoted by $t_M(x)$, is the number of steps M takes on input x before halting (M may not halt). A TM M is time bounded by $f\colon \mathbb{N} \to \mathbb{N}$ if M halts within $f(|x|)$ steps for any input x, that is, $\forall x[t_M(x) \leq f(|x|)]$. The running space of a TM M on input x, denoted by $s_M(x)$, is the number of tape cells that M reaches on input x before halting. A TM M is space bounded by $f\colon \mathbb{N} \to \mathbb{N}$ if M halts by reaching at most $f(|x|)$ cells for any input x, that is, $\forall x[s_M(x) \leq f(|x|)]$.

A *nondeterministic Turing Machine (NTM)* extends a TM with several possible transitions in each step of computation. The transition function δ is defined from $Q \times \Gamma$ to the power set of $Q \times \Gamma \times \{\leftarrow, \rightarrow, -\}$. We say a configuration c yields another configuration c' if there is one choice by the transition function such that the machine moves from c to c'. A NTM N *accepts* an input string x if and only if there exists a sequence of transitions that leads to an accepting configuration. The language decided by a NTM N is the set of strings that N accept.

A language L is *decidable* in time $t(n)$ (space $s(n)$) if there is a Turing machine M which runs in time $t(n)$ (space $s(n)$) and decides L. A language L is *verifiable* in time $t(n)$ (space $s(n)$) if there is a Turing machine M running in time $t(n)$ (space $s(n)$) such that, for any input $x \in \{0,1\}^n$, there is a string $y \in \{0,1\}^{t(n)}$ such that $A(x, y) = 1$ iff $x \in L$. The

string y is often called a *certificate*. Note that, language L is *verifiable* in time $t(n)$ (space $s(n)$) if and only if there is a nondeterministic Turing machine which decides L, where the certificate can be encoded as transition choices of the NTM.

Let DTIME$(t(n))$ be the class of languages decidable in time $O(t(n))$, and let NTIME$(t(n))$ be the class of languages verifiable in time $O(t(n))$. Let DSPACE$(s(n))$ be the class of languages decidable in space $O(s(n))$, and let NSPACE$(s(n))$ be the class of languages verifiable in space $O(s(n))$. We also have the following common complexity classes:

- P: languages decidable in polynomial time.
- NP: languages verifiable in polynomial time.
- PSPACE: languages decidable in polynomial space.
- EXP: languages decidable in exponential time.
- NEXP: languages verifiable in exponential time.
- EXPSPACE: languages decidable in exponential space.

By equivalent simulation of Turing machines, it is not hard to see the following inclusions.

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}$$

By the following time and space hierarchy theorems, we know some of inclusions are strict; however, we do not know which ones are strict.

We say a function $f\colon \mathsf{N} \to \mathsf{N}$ is *time (space) constructible* if, given any x with $|x| = n$, the value $f(n)$ is computable in time (space) $O(f(n))$.

Theorem 2.1 (Time Hierarchy Theorem [HS65, Zák83])**.** *Let f and g be time constructible functions.*

- *if $f(n) \log f(n) = o(g(n))$, then* DTIME$(f(n)) \subsetneq$ DTIME$(g(n))$*.*
- *if $f(n+1) = o(g(n))$, then* NTIME$(f(n)) \subsetneq$ NTIME$(g(n))$*.*

Theorem 2.2 (Space Hierarchy Theorem [HLS65, Imm88])**.** *Let f and g be space constructible functions and $f(n) = o(g(n))$. Then*

- DSPACE$(f(n)) \subsetneq$ DSPACE$(g(n))$*.*
- NSPACE$(f(n)) \subsetneq$ NSPACE$(g(n))$*.*

2.1.2 Polynomial Hierarchy

An *oracle TM* is a TM M with a special *oracle tape*, an *oracle language*, and three special states q_Q, q_Y, q_N. When M enters the state q_Q, it writes some input string w onto its oracle tape; and if w is in the oracle language, then M moves directly to q_Y, otherwise it moves to q_N. The output of M on input x with oracle language O is written as $M^O(x)$.

We let $\Delta_0^{\mathsf{P}} = \Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$.

- $\Delta_{i+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$.
- $\Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$.
- $\Pi_{i+1}^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_i^{\mathsf{P}}}$.

The *polynomial hierarchy* is defined as $\mathsf{PH} = \cup_{i \geq 0} \Sigma_i^{\mathsf{P}}$. In particular, we have the following.

- $\Sigma_1^{\mathsf{P}} = \mathsf{NP}$, $\Pi_1^{\mathsf{P}} = \mathsf{coNP}$.
- $\Sigma_i^{\mathsf{P}} = \mathsf{co}\Pi_i^{\mathsf{P}}$.
- $\mathsf{PH} = \mathsf{coPH}$.

It is known that $\mathsf{PH} \subseteq \mathsf{PSPACE}$, but it is not known whether the inclusion is strict. As a generalization of P, NP and coNP, the exact relationship between Σ_i^P, $\Sigma_{i+1}^{\mathsf{P}}$ and Π_i^{P} is still unknown. The following are some relative results on the "collapse" of the polynomial hierarchy.

- For $i \geq 1$, if $\Sigma_i^{\mathsf{P}} = \Pi_i^{\mathsf{P}}$ then $\mathsf{PH} = \Sigma_i^{\mathsf{P}}$, that is, PH collapse to the i-th level.
- If $\mathsf{NP} = \mathsf{coNP}$ then $\mathsf{PH} = \mathsf{NP}$.
- If $\mathsf{P} = \mathsf{NP}$ then $\mathsf{PH} = \mathsf{P}$.

2.1.3 Circuit Complexity

We have defined complexity classes for problems solvable by Turing machines with certain resource (time or space) bounds. Now we consider a non-uniform computational model, namely, boolean circuits, where we are allowed to design different algorithms for different input size.

Definition 2.3. An n-input, single-output *boolean circuit* is a directed acyclic graph (DAG) with

- n input vertices of in-degree zero;
- one output vertex of out-degree zero; and
- all non-input vertices, called *gates*, labeled with AND, OR or NOT, where the vertices labeled with AND and OR have fan-in 2 and the vertices labeled with NOT have fan-in 1.

In boolean circuits, the gates have unbounded out-degrees. Restricting the out-degree gives boolean formulas. That is, a *boolean formula* is a boolean circuit where all non-output gates have out-degree one.

A *circuit family* is a sequence of boolean circuits $\{C_n\}_{n\in\mathsf{N}}$, where each circuit C_n has n inputs and one single output. A boolean circuit on n inputs computes a boolean function from $\{0,1\}^n$ to $\{0,1\}$. A circuit family defines a language L such that, for any n and any $x \in \{0,1\}^n$, we have $x \in L$ iff $C_n(x) = 1$.

The complexity of a boolean circuit can be measured by its size and depth. The *size* of a boolean circuit C, denoted by $|C|$, is the total number of vertices in C. The *depth* of C is the length of its longest path from an input to the output.

Let $s\colon \mathbb{N} \to \mathbb{N}$ be a function. We say a circuit family $\{C_n\}_n$ has size $s(n)$ if we have $|C_n| \leq s(n)$ for every n. We denote by $\mathsf{SIZE}(s(n))$ the class of languages decidable by circuit families of size $O(s(n))$. In particular, we denote by $\mathsf{SIZE}(\mathsf{poly}) \equiv \cup_c(\mathsf{SIZE}(n^c))$ the collection of languages decidable by polynomial-size circuit families.

Circuit complexity classes can be equivalently characterized by TMs taking advice. An *advice-taking TM* has a read-only tape, called *advice tape*, which presents an advice string a_n for inputs of length n. Note that, one advice is used for all inputs of the same size.

Definition 2.4. A language L is in $\mathsf{DTIME}(t(n))/\alpha(n)$ if there exists a TM M and a sequence $\{a_n\}$ of strings with $a_n \in \{0,1\}^{\alpha(n)}$ such that

- M runs in time $t(n)$ on inputs of length $n + \alpha(n)$, and
- for any $x \in \{0,1\}^n$, $x \in L$ if and only if $M(x, a_n) = 1$.

In particular, we let $\mathsf{P/poly} = \cup_{c,d}\mathsf{DTIME}(n^c)/n^d$. The following is easy to show by hardwiring descriptions of circuits as advice.

Proposition 2.5.

$$\mathsf{P/poly} = \mathsf{SIZE(poly)}$$

$$\cup_{c,d}\mathsf{DTIME}(n^c)/n^d = \cup_c \mathsf{SIZE}(n^c).$$

A deterministic TM running in time $t(n)$ can be simulated by a circuit family of size polynomial in $t(n)$, and similarly, A non-deterministic TM running in time $t(n)$ can be simulated by a circuit family of size exponential in $t(n)$. This gives that $\mathsf{P} \subseteq \mathsf{P/poly}$, and $\mathsf{NP} \subseteq \mathsf{SIZE}(2^{\mathsf{poly}(n)})$. However, we do not know whether there is a better simulation with smaller circuit size. In fact, a simulation of NP with polynomial-size circuits implies the collapse of the polynomial hierarchy, and similarly, a simulation of EXP with polynomial-size circuits implies a more severe collapse of EXP to Σ_2^P.

Theorem 2.6 (Karp-Lipton [KL82])**.** *If* $\mathsf{NP} \subseteq \mathsf{P/poly}$*, then* $\mathsf{PH} = \Sigma_2^{\mathsf{P}} = \Pi_2^{\mathsf{P}}$*.*

Theorem 2.7 (Meyer [KL82])**.** *If* $\mathsf{EXP} \subseteq \mathsf{P/poly}$*, then* $\mathsf{EXP} = \Sigma_2^{\mathsf{P}}$*.*

Shannon [Sha49] showed that there are hard functions which requires large circuits.

Theorem 2.8 (Shannon [Sha49])**.** *For large enough* n*, there exists a boolean function which is not computable by any boolean circuit of size at most* $2^n/n$*.*

The proof is by a counting argument; that is, count the number of possible circuits of a bounded size. In fact, almost all boolean functions are hard to compute; the number of functions computable by circuits of size $2^n/n$ is bounded by $2^{2^n} \cdot 2^{-\Omega(2^n/n)}$, which is only a tiny portion of the 2^{2^n} functions. However, we do not know how to explicitly define such a function. In particular, we do not know whether there is such a hard function in NEXP.

Based on Shannon's result, Kannan [Kan82] showed that, there are hard languages which are not computable by circuits of any fixed polynomial size.

Theorem 2.9. *[Kan82] For any constant* c*, there exists a language in* Σ_2^{P} *which is not computable by boolean circuits of size* n^c*.*

Note that the theorem says, for any constant c, we have $\Sigma_2^{\mathsf{P}} \not\subseteq \mathsf{SIZE}(n^c)$. This does not imply any relationship between Σ_2^{P} and $\mathsf{P/poly} = \cup_c \mathsf{SIZE}(n^c)$. Analogously, for any c, we have $\mathsf{P} \not\subseteq \mathsf{TIME}(n^c)$, but $\mathsf{P} = \cup_c \mathsf{TIME}(n^c)$; for any c, we have $\mathsf{P/poly} \not\subseteq \mathsf{SIZE}(n^c)$, but $\mathsf{P/poly} = \cup_c \mathsf{SIZE}(n^c)$.

2.1.4 Bounded Depth Circuits

General boolean circuits are hard to analyze. Until now, we do not know whether there is a problem in NEXP which require circuits of super-polynomial size. In the following, we define circuits of bounded depth, where the depth restriction allows us to apply more analytical tools.

In order for a constant-depth circuit to access all the inputs, we allows the AND and OR gates to have unbounded fan-in. Also, we introduce modular and majority gates. A MOD$_m$ gate outputs 1 if and only if the summation of its inputs is divisible by m. A MAJ gate outputs 1 if and only if more than half of its inputs are 1. The following are well-studied constant-depth circuit classes.

- AC^0 is the class of circuit families of constant depth and polynomial size, with unbounded fan-in AND and OR gates.
- $\mathsf{AC}^0[m]$, for an integer m, extends AC^0 with unbounded fan-in MOD$_m$ gates.
- $\mathsf{ACC}^0 = \cup_m \mathsf{AC}^0[m]$.
- TC^0 extends AC^0 with unbounded fan-in MAJ (majority) gates.

It is known that

$$\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{P/poly}.$$

Proving lower bounds against bounded-depth circuits is a hard task for complexity theorists. It was known that PARITY is not in AC^0 [FSS84], and for distinct prime numbers p and q, MOD$_p$ is not in $\mathsf{AC}^0[q]$ [Raz87, Smo87]. There was little progress for decades, until Williams [Wil11] showed that a problem in NEXP is not in ACC^0. Proving a lower bound against TC^0 is the next goal for complexity theorists.

2.1.5 Boolean Formulas

A *(boolean) formula* over the basis B is a tree where each leaf is labeled by an input variable or a constant, each internal node is labeled by a function in B whose arity is the same as the in-degree of the node, and the root is designated as the output. A *De Morgan formula* is a formula over the basis $\{\neg, \wedge, \vee\}$. Constants in formulas can be eliminated by

the facts that $0 \wedge x = 0$, $1 \vee x = 1$, and $0 \vee x = 1 \wedge x = x$. Formulas also allow equivalent transformations by commutativity and De Morgan's laws:

$$x \wedge y = y \wedge x, \qquad x \vee y = y \vee x,$$
$$\neg(x \wedge y) = \neg x \vee \neg y, \qquad \neg(x \vee y) = \neg x \wedge \neg y.$$

Thus, one can transform a de Morgan formula such that $\neg$ appears only before a variable. Without loss of generality, we will assume that, a de Morgan formula is represented by a binary tree where each leaf is labeled by a literal (a variable or its negation), and each internal node is labeled by $\wedge$ or $\vee$.

The *size* of a formula is the number of leafs in its tree representation; the *depth* of a formula is the depth of the tree. The size is also the total number of appearances of all variables in the formula. We denote by $L(F)$ the size of a formula F. For a boolean function f and a basis B, its formula size $L(f)$ is the smallest size of all possible formulas over B computing f, and the formula depth $D(f)$ is the smallest depth of all formulas over B computing f.

We say a language L is computable by formulas of size $s(n)$ over a basis B if, for each input length n, there is a formula F_n over B such that, for every $x \in \{0,1\}^n$, we have $x \in L$ if and only if $F_n(x) = 1$. In other words, L is computable by a sequence of formulas $\{F_n\}$ such that $L(F_n) \leq s(n)$ for all n.

Shannon [Sha49] showed a non-explicit lower bound of $\Omega(2^n/\log n)$ using a counting argument; that is, there exists a function which is not computable by formulas of size $\delta \cdot 2^n/\log n$ for some $\delta > 0$ (over any finite basis). In fact, most functions are hard to compute by such formulas. However, we do not know how to explicitly find even one of them. Nevertheless, using Shannon's lower bound as a blackbox, Andreev [And87] constructed a function computable in P but not by formulas of size $n^{2-o(1)}$ over the full basis. For de Morgan formulas, the lower bound was improved to $n^{3-o(1)}$ by [Hås98].

2.2 Circuit Lower Bounds versus Circuit Satisfiability Algorithms

2.2.1 Lower Bounds from Satisfiability Algorithms

The Karp-Lipton and Meyer's theorems (Theorem 2.6 and 2.7) say that, simulating PH (EXP) with polynomial-size circuits imply the collapse of PH (EXP) to Σ_2^{P}. This can be

directly used to prove the following.

Theorem 2.10. *[KL82] If* $\mathsf{P} = \mathsf{NP}$*, then* $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$.

The proof is simple. Suppose $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$. Then by Theorem 2.7), $\mathsf{EXP} = \Sigma_2^{\mathsf{P}}$. Since $\mathsf{P} = \mathsf{NP}$ implies that $\Sigma_2^{\mathsf{P}} = \mathsf{P}$, we get a contradiction with the time hierarchy theorem (Theorem 2.1). This result can be strengthened in two ways. Let $\mathsf{E} = \mathsf{DTIME}(2^{O(n)})$.

- The condition $\mathsf{P} = \mathsf{NP}$ implies a stronger result $\mathsf{E} \not\subseteq \mathsf{SIZE}(2^n/n)$. Similar to Kannan's result (Theorem 2.9), E^{PH} contains a language which requires the maximal circuit size $O(2^n/n)$. Since $\mathsf{P} = \mathsf{NP}$ implies $\mathsf{PH} = \mathsf{P}$, we get $\mathsf{E}^{\mathsf{PH}} = \mathsf{E}$ and the result follows.
- A weaker condition $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{o(n)})$ still implies that $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$. Assuming $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$ implies $\mathsf{EXP} = \Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}}$. Since $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{o(n)})$, we get EXP is in $\mathsf{DTIME}(2^{o(n)})$ which still contradicts the time hierarchy theorem.

We can also get that $\mathsf{coNP} \subseteq \mathsf{NTIME}(2^{o(n)})$ implies $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$. This follows from the result by [IKW01] that $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ implies $\mathsf{NEXP} = \Pi_2^{\mathsf{P}}$.

The above results show that sub-exponential time SAT algorithms imply circuit lower bounds. However, such algorithms are still beyond our reach. Recently, Williams [Wil10] showed that a very weak improvement over brute-force search would imply circuit lower bounds for NEXP. Let $\mathcal{C}$ be a circuit class which is closed under composition and $\mathsf{AC}^0 \subseteq \mathcal{C} \subseteq \mathsf{P/poly}$ (for example, $\mathcal{C}$ could be $\mathsf{P/poly}, \mathsf{TC}^0, \mathsf{ACC}^0$).

Theorem 2.11. *[Wil10, Wil11] There is* $c > 0$ *such that, if* $\mathcal{C}$*-SAT is in* $O(2^n/n^c)$ *for circuits with* n *inputs and* n^k *size for any* k*, then* $\mathsf{NEXP} \not\subseteq \mathcal{C}$.

Similar as Theorem 2.10 and its strengthened results, the proof is via a contradiction with the (non-deterministic) time hierarchy theorem. That is, assuming both $\mathsf{NEXP} \subseteq \mathcal{C}$ and an efficient $\mathcal{C}$-SAT algorithm, every problem in nondeterministic $O(2^n)$ time is solvable in $o(2^n)$ time, which contradicts the non-deterministic time hierarchy theorem (Theorem 2.1).

The condition here only requires SAT algorithms of slightly better than brute-force search; this is much weaker than the previous conditions of polynomial or sub-exponential time SAT algorithms. Following this, Williams [Wil11] showed that ACC^0 circuits do have SAT algorithms running in the require time bound, and this gives the currelty best-known circuit lower bounds.

Theorem 2.12. *[Wil11]* $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$.

2.2.2 Structured Properties, Lower Bounds and Satisfiability Algorithms

Restricted circuit models have certain structured properties which can be exploited to either prove lower bounds or design better satisfiability algorithms. We survey here several circuit models, including k-CNF, AC^0, ACC^0 and boolean formulas, and their structured properties.

k-CNF. A CNF formula is an AND of ORs of literals, where a literal is a variable or its negation. A k-CNF is a CNF where each OR has at most k literals. The k-SAT problem is to decide the satisfiability of k-CNF formulas; for all $k \geq 3$, the problem is NP-complete [Coo71]. There has been a line of researches which get k-SAT algorithms running in time $2^{n-n/ck}$ for some fixed constant c [PPZ99, PPSZ98, Sch02]. (However, we note that the savings of the running time over 2^n diminishes as k increases.)

Paturi, Pudlak and Zane [PPZ99] gave a randomized algorithm for k-SAT running in time roughly $2^{n-n/k}$, and a deterministic algorithm in time $2^{n-n/2k}$. The algorithm is based a special property on the satisfying assignments to k-CNF, which is termed "Satisfiability Coding Lemma". It says that the satisfying assignments of a k-CNF can be encoded succinctly. In particular, it implies that a k-CNF has at most $2^{n-n/k}$ isolated solutions, where an isolated solution is a satisfying assignment such that, if we flip any one bit of the assignment, the formula is no longer satisfied. It was shown that a satisfiable k-CNF has either a nearly isolated solution or many satisfying assignments, and then one can either search in a space of small size (for an isolated solution), or randomly guess a solution.

The Satisfiability Coding Lemma was also used in [PPZ99] to get a lower bound for computing Parity with depth-3 circuits, that is, Parity requires depth-3 circuits of size $\theta(n^{1/4}2^{\sqrt{n}})$. Intuitively, Parity, which has a large number 2^{n-1} of isolated solutions, is hard to compute by small ORs of CNFs, which do not have many isolated solutions.

The approach by Paturi, Pudlak and Zane [PPZ99] shows an example of the connections between satisfiability algorithms and lower bounds. They identified a structured property of k-CNFs; on the one hand, this property helps to design a better satisfiability algorithm by reducing the search space, and on the other hand, this property implies the limitation of using k-CNF as a computational model, which gives a lower bound.

AC^0 Circuits. We now consider AC^0 circuits, which are boolean circuits with unbounded fan-in AND and OR gates but constant depth. This generalizes DNFs and CNFs from depth 2 to arbitrary constant depth. Lower bounds against AC^0 were obtained long time ago; it was shown by [FSS84, Ajt83] that Parity is not computable by polynomial-size AC^0 circuits.

Hastad [Hås86] improved this lower bound to exponential size $\exp(n^{\theta(\frac{1}{d-1})})$. The structured property of AC^0 exploited by Hastad [Hås86] is depth reduction under "random restrictions", known as Hastad's Switching Lemma. That is, for a depth-d AC^0 circuit, when we randomly assign values to a large fraction of the inputs, with high probability, the circuit (on the remaining inputs) can be transformed to be of depth $d-1$. By recursively applying this depth reduction, a small-size circuit can be transformed to be a constant even when there are still unassigned inputs. This means such circuits cannot compute Parity, since Parity is not a constant even when there is only one input left.

Recently, Impagliazzo, Matthews and Paturi [IMP12] extends Hastad's switching lemma and gives a randomized AC^0 satisfiability algorithm running in time $2^{n-n/\log(s/n)^{d-1}}$, where s is the size of the circuit. The algorithm is based on a partition of the input space into regions where the circuit becomes constant, and that the number of such regions is not too large. A deterministic AC^0 satisfiability algorithm was derived in [BIS12], which is based on an improvement of the lower bound technique of [Ajt83]. The properties exploited in [IMP12] and [BIS12] also give improved bounds on how AC^0 circuits can approximately compute Parity; finally, Hastad [Hås12] showed that AC^0 circuits of size s can compute Parity correctly on at most $1/2+\epsilon$ fraction of inputs where $\epsilon = \exp(-\Omega(n/\log^{d-1} s))$.

This line of work [Ajt83, Hås86, IMP12, BIS12, Hås12], which explored structured properties of AC^0, show another connection between circuit lower bounds and satisfiability algorithms.

ACC^0 **Circuits.** Since AC^0 circuits can not compute Parity, ACC^0 circuits extend AC^0 by allowing unbounded fan-in modulo gates. It was shown by [BT94, AG94] that an ACC^0 circuit of size s can be efficiently transformed to a depth-2 circuit with a symmetric gate at the top, and $\exp(\log^{O(1)}(s))$ AND gates in the middle. Here, a *symmetric* gate computes a function of its inputs such that the output is the same for any permutations of the inputs. Williams [Wil11] exploited this property to design a satisfiability algorithm running in time $2^{n-n^{\Omega(1)}}$, which allows him to apply Theorem 2.11 [Wil10] to get a new lower bound.

Boolean Formulas. For boolean formulas, we have the restriction that the non-input gates have fan-out 1. Boolean formulas have the structured property that the formula size shrinks non-trivially under random restrictions. This property was exploited to prove formula lower bounds. In particular, for boolean formulas over the de Morgan basis, Subbotovskaya [Sub61] showed that, when we randomly assign values to a fraction $1-p$ of the inputs, the formula size (on average) shrinks by a factor of $p^{1.5}$. Based on this shrinkage

property, Andreev [And87] constructed a function (computable in P) which is not computable by formulas of size $n^{2.5-o(1)}$. The shrinkage factor was improved to almost p^2 by Hastad [Hås98], and this implies a $n^{3-o(1)}$ formula lower bound for the same Andreev's function [And87].

Recently, Santhanam [San10] exploited the shrinkage property to design a satisfiability algorithm running in time $2^{n-\Omega(n)}$ for linear-size formulas. The algorithm is based on a concentrated analysis of the shrinkage property. That is, under a certain process which restricts a fraction $1-p$ of the inputs, the formula size shrinks in high probability, not just on average. This "concentrated" shrinkage allows [San10] to build a decision tree of size $2^{n-\Omega(n)}$ for linear-size formulas, and the algorithm follows from traversing the decision tree. This approach also allows Santhanam [San10] to give a correlation lower bound, that is, linear-size formulas over the de Morgan basis can compute Parity correctly on at most $1/2+2^{-\Omega(n)}$ of the inputs. Santhanam [San10]'s algorithm was extended to formulas over the full basis by Seto and Tamaki [ST12], to formulas of size $n^{2.49}$ by [CKK$^+$14] (which we include in this thesis), and finally to formulas of size $n^{3-o(1)}$ by [KRT13].

Chapter 3

Boolean Formulas: #SAT Algorithms and Lower Bounds

3.1 Introduction

Boolean formulas are a special case of boolean circuits where the underlying structures are trees instead of directed acyclic graphs. In other words, each internal gate in a boolean formula has out-degree at most one. This limitation in reusability makes formulas a less powerful computational model. However, it does allow more combinatorial analysis. In particular, we have better lower bounds, satisfiability algorithms, and pseudoranom generators for formulas than those for circuits.

A particularly interesting property for formulas is the size *shrinkage* under random restrictions, that is, the size of a formula shrinks non-trivially when some of its inputs are randomly assigned. We say the *shrinkage exponent* is Γ if the formula size shrinks by an expected factor of p^Γ under random restrictions which leave a fraction p of the inputs unassigned.

Subbotovskaya [Sub61] observed that, de Morgan formulas, i.e., formulas over the de Morgan basis $\{\neg, \vee, \wedge\}$, have shrinkage exponent $\Gamma \geq 1.5$. This immediately gives a lower bound: we need de Morgan formulas of size at least $\Omega(n^{1.5})$ to compute Parity. Khrapchenko [Khr71] proved a lower bound of $\Omega(n^2)$ using a different technique. Later, Andreev [And87] constructed a function (computable in polynomial time) which requires formulas of size at least $\Omega(n^{\Gamma+1-o(1)})$; this is at least $\Omega(n^{2.5-o(1)})$ for $\Gamma \geq 1.5$ by [Sub61].

The shrinkage exponent for de Morgan formulas was improved to $\Gamma \geq 1.55$ by [IN93] and to $\Gamma \geq 1.63$ by [PZ93]. They use similar techniques by introducing weight functions based on the underlying tree structures of formulas. Finally, Hastad [Hås98] proved that $\Gamma \geq 2-o(1)$; this is almost tight since we know $\Gamma \leq 2$, by the fact that Parity is computable by formulas of size n^2. This leads to an $\Omega(n^{3-o(1)})$ lower bound for Andreev's function [And87].

Recently, several new results on formulas were discovered: better-than brute-force satisfiability algorithms for small-size formulas [San10], pseudorandom generators [IMZ12], and average-case lower bounds for formulas [KR13, KRT13].

Santhanam [San10] observed that, the shrinkage result of [Sub61] can be strengthened to a high-probability version. That is, by deterministically choosing variables which appear frequently in the formula and randomly assigning 0 or 1 to such variables, the formula size shrinks with high probability (by a factor of $p^{1.5}$ when there are a fraction p of the variable left). This concentrated shrinkage property was used by [San10] to design a #SAT algorithm for linear-size de Morgan formulas which runs in time $2^{n-\Omega(n)}$, better than brute-force search. Meanwhile, this property gives a correlation lower bound for Parity; that is Parity can not be approximately computed by linear-size de Morgan formulas.

The average-case lower bound for de Morgan formulas was improved to $\Omega(n^{2.499})$ by [KR13] and to $\Omega(n^{2.499})$ by [KRT13], which is almost tight. Komargodski and Raz [KR13] designed a specific process of random restrictions such that the formula size shrinks (by a factor of $p^{1.5}$) with high probability, and they also used error-correcting codes to generalize the hardness of Andreev's function from the worst case to the average case. The restriction process defined in [KR13] still uses certain randomness for selecting variables to restrict, which is necessary for their average-case hardness. Using techniques of [IMZ12], the average-case lower bound was improved by [KRT13] to $\Omega(n^{2.999})$, matching with the worst-case hardness. Furthermore, this average-case hardness result implies a randomized #SAT algorithm running in time $2^{n-n^{\Omega(1)}}$ for de Morgan formulas of size $n^{2.999}$.

Impagliazzo, Meka and Zuckerman [IMZ12] constructed pseudorandom generators for formulas based on the shrinkage property. In particular, for de Morgan formulas of size s, their generator has seed length almost $s^{0.334}$. They proved that, under a family of pseudorandom restrictions, the de Morgan formula size shrinks by a factor of almost p^2 with high probability.

Along this line of research, we derive the following results on the concentrated shrinkage, #SAT-algorithms and average-case lower bounds for small formulas.

Shrinkage of formulas. This "shrinkage in expectation" result of Subbotovskaya [Sub61] is sufficient for proving worst-case de Morgan formula lower bounds [And87]. However, for designing SAT-algorithms and pseudorandom generators, as well as for proving strong average-case hardness results for small de Morgan formulas, it is important to have a "high-probability" version of such a shrinkage result, saying that "most" restrictions (of the appropriate kind) shrink the size of the original formula. Such a version of shrinkage for de Morgan formulas is implicit in [San10] (for linear-size formulas); Impagliazzo et al. [IMZ12] prove a version of shrinkage with respect to pseudo-random restrictions (for de Morgan formulas of size almost n^3); Komargodski and Raz [KR13] prove the shrinkage result for certain random restrictions (for de Morgan formulas of size about $n^{2.5}$).

We sharpen a structural characterization of small (de Morgan) formulas by proving a stronger version of the "shrinkage under random restrictions" result of [San10, KR13], with a cleaner and simpler argument.

Shrinkage Lemma: *Let F be a (de Morgan) formula or general branching program size s on n variables. Consider the following greedy randomized process:*

> *For $n-k$ steps (where $0 \leq k \leq n$), do the following: (1) choose the most frequent variable in the current formula; (2) assign it uniformly at random to 0 or 1; (3) simplify the resulting new formula.*

Then, with probability at least $1-2^{-k}$, this process produces a formula of size at most $2 \cdot s \cdot (k/n)^{\Gamma}$, where $\Gamma = 1.5$ for de Morgan formulas, and $\Gamma = 1$ for general formulas and branching programs.

Formula-#SAT. The fact that SAT is NP-complete [Coo71, Lev73], and so probably not solvable in polynomial time, does not deter researchers interested in "better-than-brute-force" SAT-algorithms. In particular, the case of CNF-SAT has been actively studied for a number of years (see [DH09] for a recent survey), while the study of Circuit-SAT algorithms for more general classes of circuits is more recent: see [CIP09, IMP12, BIS12] for AC^0-SAT, [San10, ST12] for Formula-SAT, and [Wil11] for ACC^0-SAT. Usually such algorithms exploit the same structural properties of the corresponding circuit class that are used in the circuit lower bounds for that class. In fact, the observation that circuit lower bound proofs and meta-algorithms are intimately related was first formulated in Zane's PhD thesis [Zan98] precisely in the context of depth-3 circuit lower bounds and improved CNF-SAT algorithms.

As a consequence of the Shrinkage Lemma above, we get a new "better-than-brute-force" deterministic algorithm for #SAT for (de Morgan) formulas and general branching programs of about quadratic size, as well as give a simplified analysis of the #SAT algorithms for linear-size (de Morgan) formulas from [San10, ST12].

#SAT algorithms: *Counting the number of satisfying assignments for n-variate de Morgan formulas of size $n^{2.49}$, formulas over the complete basis of size $n^{1.99}$, or branching programs of size $n^{1.99}$ can be done by a deterministic algorithm in time $2^{n-n^{\epsilon}}$, for some $\epsilon > 0$ (dependent on the size of the formula/branching program).*

Average-case formula lower bounds. Showing that explicit functions are average-case hard to compute by small circuits is an important problem in complexity theory, both for understanding "efficient computation", and for algorithmic applications (e.g., in cryptography and derandomization). Here, again, useful algorithmic ideas often contribute to proving lower bounds for the related model of computation. For example, strong average-case hardness results for *linear-size* (de Morgan) formulas are proved in [San10, ST12], using the same ideas that also gave SAT-algorithms for the corresponding formula classes.

We use our shrinkage lemma to give an alternative proof of a recent average-case lower bound against (de Morgan) formulas due to [KR13]: *There is a Boolean function $f : \{0,1\}^n \to \{0,1\}$ computable in* P *such that every de Morgan formula of size $n^{2.49}$ (any general formula of size $n^{1.99}$) computes $f(x)$ correctly on at most $1/2 + 2^{-n^{\sigma}}$ fraction of all n-bit inputs, for some constant $0 < \sigma < 1$.*

3.1.1 Restriction-Based Lower Bounds

Now we describe the "random restriction" method by Subbotovskaya [Sub61] for proving formula lower bounds. That is, if we randomly assign values to the inputs of a De Morgan formula, then the formula size shrinks non-trivially. More precisely, if a random restriction leaves p fraction of variables unrestricted, then the formula size shrinks in expectation by a factor of p^{Γ} for $\Gamma \geq 1.5$, where Γ is called the *shrinkage exponent*.

Let f be a boolean function on n variables and it is computed by a formula F which has the smallest possible size. Consider that we fix the variables of F one by one to be either 0 or 1. The tree of the formula can be simplified at each step via the following rules:

- If a constant (0 or 1) appears at a leaf, remove it and also possibly its sibling. In particular, replace $1 \vee \psi$ by 1, $0 \wedge \psi$ by 0, and $0 \vee \psi$ or $1 \wedge \psi$ by ψ.
- For any literal in a leaf, remove its variable from the sibling subtree. In particular, for $y \vee G$ where y is a literal and G is a formula, replace every appearance of y in G by 0 and $\overline{y}$ by 1; for $y \wedge G$, replace every appearance of y in G by 1, and $\overline{y}$ by 0.

The second rule guarantees that the sibling subtree of a leaf does not contain the variable in that leaf.

Suppose we choose a variable x randomly and assign it uniformly to be either 0 or 1. For each leaf containing x, we can remove this leaf, and also, with probability at least $1/2$, remove its sibling subtree. By the second rule above, the sibling subtree does not contain x. Thus on average we can remove at least 1.5 leaves for each appearance of x in F. Let F' be the new formula after applying the above simplification rules. Since on average each variable appears $L(F)/n$ times in F, the new formula size $L(F')$ satisfies that

$$\mathbf{E}[L(F')] \leq L(F) - \frac{L(F)}{n} \cdot \frac{3}{2} = L(F) \cdot \left(1 - \frac{3}{2n}\right) \leq L(F) \cdot \left(1 - \frac{1}{n}\right)^{1.5}.$$

The last inequality is by the fact that $1 - ax \leq (1-x)^a$ for $0 \leq x \leq 1$ and $a \geq 1$. Note that the expectation is taken over a random choice of variables and a random assignment to the variable.

Consider a step-by-step process where at each step we randomly choose a variable and randomly fix it. We simplify the formula using the simplification rules after each step. Suppose we run the process for $n-k$ steps (with k variables left unfixed). Let $F_0 = F$, and F_i be the simplified formula after step i. We have that, for each step i,

$$\mathbf{E}[L(F_i)] \leq \mathbf{E}[L(F_{i-1})] \cdot \left(1 - \frac{1}{n-i+1}\right)^{1.5},$$

and finally

$$\mathbf{E}[L(F_{n-k})] \leq L(F_0) \cdot \left(1 - \frac{1}{n}\right)^{1.5} \cdots \left(1 - \frac{1}{k+1}\right)^{1.5} = L(F_0) \cdot \left(\frac{k}{n}\right)^{1.5}.$$

Together with Markov's inequality, we essentially proved the following lemma.

Lemma 3.1. *Let f be a boolean function on n variables. After randomly fixing $n-k$ variables, let f' be the new function. Then we have*

$$\mathbf{E}[L(f')] \leq L(f) \cdot \left(\frac{k}{n}\right)^{1.5},$$

and

$$\mathbf{Pr}\left[L(f')] \leq 4 \cdot L(f) \cdot \left(\frac{k}{n}\right)^{1.5}\right] \geq \frac{3}{4}.$$

Using this shrinkage result, Andreev [And87] constructed an explicit function requiring almost $n^{2.5}$ formula size. Andreev's function uses Nechiporuk's "universal function" method. Andreev [And87] gave lower bounds based on the shrinkage exponent of the size of the computational model.

Theorem 3.2. *[And87] If a class of formulas has shrinkage exponent Γ, then there is an explicit boolean function $h\colon \{0,1\}^n \to \{0,1\}$ in* P *that is not computable by formulas of size at most $n^{\Gamma+1}/\mathrm{poly}(\log n)$.*

Andreev's function is constructed as follows. For simplicity, consider $2n$ bits of inputs: The first n bits defines a function on $\log n$ bits, denoted by H; the second n bits are divided into $\log n$ blocks, each with $n/\log n$ bits. Compute the the parity of the variables in each block, and then we apply H on the $\log n$ bits produced. Now if H is the truth table of a hardest function, then it requires formulas of size $O(n/\log\log n)$. Under random restriction, with certain probability, we have both that the formula size shrinks non-trivially and that each block in the second n bits has unrestricted variables left. Due to hardness of H, the original function must have a large size such that the size after the shrinkage is still big enough to accommodate H.

Now we prove the special case on De Morgan formulas.

Theorem 3.3. *[And87] There exists an explicit function which is not computable by formulas of size $n^{2.5-o(1)}$.*

Proof. We define a function f on $2n$ inputs, where the first n inputs will be fixed as a hard function, and the second n inputs serve as an index to the first n bits. Similarly, the second n inputs are divided into $\log n$ blocks, each of size $n/\log n$.

The function is computed as follows. For each block of the second n inputs, compute the parity of all bits inside the block; this generates $\log n$ bits, one from each block. Use these bits to locate one bit in the first n inputs and output it.

More specifically, suppose we have input bits $(x, y) = (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1})$. Let $m = n/\log n$ and let

$$r_i = y_{mi} \oplus \cdots \oplus y_{m(i+1)-1}, \qquad i = 0, \ldots, (\log n - 1).$$

The number $r = r_0 r_1 \dots r_{\log n - 1}$ is between 0 and $n-1$. The output of $f(x,y)$ is the bit x_r.

Let F be the smallest formula computing f. We fix a hard function h on $\log n$ bits which has formula size $L(h) \geq n/(2\log\log n)$, and fix the first n bits to be the truth table of h, denoted by x^h.

Consider a random restriction on the second n inputs which leaves $k = \log n \ln(4\log n)$ of the n inputs unfixed. For each block of m inputs, the probability that no bit is left unfixed is the following

$$\frac{\binom{n-m}{k}}{\binom{n}{k}} \leq \left(\frac{n-k}{n}\right)^m \leq e^{-\frac{k}{\log n}} = \frac{1}{4\log n}$$

By the union bound, the probability that some of the $\log n$ blocks has no bit unfixed is at most $\log n \cdot \frac{1}{4\log n} = 1/4$. Then with probability $3/4$, every block has some unfixed inputs.

Denote by F' the formula after the random restriction. With probability $3/4$, it computes the hard function h (as a sub-function), which mean that $L(F') \geq L(h)$. On the other hand, with probability $3/4$, we have $L(F') \leq 4(k/n)^{1.5} L(F)$. Therefore, we get

$$L(F) \geq \frac{1}{4}\left(\frac{n}{k}\right)^{1.5} L(h) \geq \frac{1}{4}\left(\frac{n}{k}\right)^{1.5} \frac{n}{2\log\log n} \geq n^{2.5-o(1)}.$$

□

3.2 Concentrated Shrinkage

The shrinkage under Subbotovskaya's random restriction happens on average. Santhanam [San10] observed that, by adaptively picking the most frequent variables, shrinkage happens with high probability; and furthermore, this concentrated shrinkage can be applied to design a satisfiability algorithm for small-size formulas, and also get average-case lower bounds.

We prove the concentrated shrinkage using the adaptive restrictions of [San10] (each time randomly restricting the most frequent variable in the formula). Following [KR13], our idea is to analyze how the size of a formula is changed after a single (most frequent) variable is randomly assigned. The new formula size is a random variable, which is expected to get smaller than the previous formula size. We would like to treat the sequence of these random variables as a supermartingale, and use the standard concentration results (Azuma's inequalities) to show that the final formula is very likely to have a small size.

One technical problem with this approach is that in one step the formula size may drop by an arbitrary amount, and we don't seem to get the boundedness condition (that a random variable changes by at most some fixed amount after each step) that is a condition for the standard version of Azuma's inequality. In [KR13], this technicality was circumvented by introducing some "dummy" variables into the formula to artificially keep the one-step change in the formula size bounded, and then apply the standard version of Azuma's inequality. However, it seems unnecessary to do that, since if the formula size drops by a lot in a single step, this should be even better for us!

Instead, we show a version of Azuma's inequality holds in the special case of random variables which take two values with equal probability and where the boundedness condition is *one-sided*: we just require that the next random value be *smaller* than the current value by at least some known amount, meanwhile allowing it to be arbitrarily small. This turns out to be precisely the setting in our case, and so we can bound the probability of producing a large formula by a direct application of Azuma's inequality. Apart from making the overall argument simpler, this also gives a quantitatively better bound. We give the details next.

3.2.1 A Variant of Azuma's Inequality

Lemma 3.4. *Let Y be a random variable taking two values with equal probability. If $\mathbf{E}[Y] \leq 0$ and there exists $c \geq 0$ such that $Y \leq c$, then for any $t \geq 0$, we have $\mathbf{E}[e^{tY}] \leq e^{t^2c^2/2}$.*

Proof. Suppose Y takes two values a and b where $a \leq b \leq c$, and $\mathbf{Pr}[Y = a] = \mathbf{Pr}[Y = b] = \frac{1}{2}$. Consider the following two cases. If $b \leq 0$, then

$$e^{tY} \leq e^{t \cdot 0} = 1 \leq e^{t^2c^2/2}.$$

If $b > 0$, since $\mathbf{E}[Y] = \frac{1}{2}(a + b) \leq 0$, we have $a \leq -b$ and

$$\mathbf{E}[e^{tY}] = \frac{1}{2}(e^{ta} + e^{tb}) \leq \frac{1}{2}(e^{-tb} + e^{tb}) \leq e^{t^2b^2/2} \leq e^{t^2c^2/2},$$

where we used the inequality $\frac{1}{2}(e^{-x} + e^{x}) \leq e^{x^2/2}$. □

Recall that a sequence of random variables $X_0, X_1, X_2, \ldots, X_n$ is a *supermartingale* with respect to a sequence of random variables $R_1, R_2, \ldots, R_n$ if

$$\mathbf{E}[X_i \mid R_{i-1}, \ldots, R_1] \leq X_{i-1}, \text{ for } 1 \leq i \leq n.$$

Lemma 3.5. *Let $\{X_i\}_{i=0}^n$ be a supermartingale with respect to $\{R_i\}_{i=1}^n$. Let $Y_i = X_i - X_{i-1}$. If, for every $1 \leq i \leq n$, the random variable Y_i (conditioned on $R_{i-1}, \ldots, R_1$) assumes two values with equal probability, and there exists a constant $c_i \geq 0$ such that $Y_i \leq c_i$, $\mathbf{E}[e^{tY_i} \mid R_{i-1}, \ldots, R_1] \leq e^{t^2 c_i^2/2}$, then, for any λ, we have*

$$\mathbf{Pr}[X_n - X_0 \geq \lambda] \leq \exp\left(-\frac{\lambda^2}{2\sum_{i=1}^n c_i^2}\right).$$

Proof. The following is an adaptation of the standard proof of Azuma's inequality to our case of "one-sided bounded" variables. Let $t \geq 0$ be arbitrary. Since $X_n - X_0 = \sum_{i=1}^n Y_i$, we have

$$\mathbf{Pr}[X_n - X_0 \geq \lambda] = \mathbf{Pr}\left[\sum_{i=1}^n Y_i \geq \lambda\right] = \mathbf{Pr}\left[e^{t\sum_{i=1}^n Y_i} \geq e^{\lambda t}\right] \leq e^{-\lambda t}\mathbf{E}\left[e^{t\sum_{i=1}^n Y_i}\right],$$

where the last inequality is by Markov's inequality. We note that the following bound does not depend on the conditioning part:

$$\mathbf{E}[e^{tY_i} \mid R_i, \ldots, R_1] \leq e^{t^2 c_i^2/2}.$$

We get

$$\mathbf{E}\left[e^{t\sum_{i=1}^n Y_i}\right] = \mathbf{E}\left[e^{\sum_{i=1}^{n-1} Y_i} e^{tY_n}\right] = \mathbf{E}\left[e^{t\sum_{i=1}^{n-1} Y_i} \cdot \mathbf{E}\left[e^{tY_n} \mid R_{n-1}, \ldots, R_1\right]\right] \leq \mathbf{E}\left[e^{t\sum_{i=1}^{n-1} Y_i}\right] \cdot e^{t^2 c_n^2/2},$$

where the last inequality is by Lemma 3.4. By induction, we get $\mathbf{E}\left[e^{t\sum_{i=1}^n Y_i}\right] \leq e^{t^2\sum_{i=1}^n c_i^2/2}$. Thus, $\mathbf{Pr}[X_n - X_0 \geq \lambda] \leq e^{-\lambda t + t^2\sum_{i=1}^n c_i^2/2}$. Choosing $t = \lambda/\sum_{i=1}^n c_i^2$ yields the required bound. □

3.2.2 Shrinkage Lemma

Recall our definition of a restriction. For a given de Morgan formula F on n variables, define $F_0 = F$. For $1 \leq i \leq n$, we define F_i to be the subformula obtained from F_{i-1} by uniformly at random assigning the most frequent variable of F_{i-1}.

We state the Shrinkage Lemma for the case of de Morgan formulas; the case of general formulas and branching programs is similar with the shrinkage exponent $\Gamma = 1$ used throughout instead of $\Gamma = 3/2$.

Lemma 3.6 (Shrinkage Lemma)**.** *Let F be any given de Morgan formula on n variables. For any $k \geq 4$, we have*

$$\mathbf{Pr}\left[L(F_{n-k}) \geq 2 \cdot L(F) \cdot \left(\frac{k}{n}\right)^{3/2}\right] < 2^{-k}.$$

For the proof, we will need the following auxiliary lemmas.

Lemma 3.7. *Let F be a de Morgan formula on n variables, and let $F' = F_1$ (obtained from F in one step of adaptive restriction defined above). Then $L(F') \leq L(F) \cdot \left(1 - \frac{1}{n}\right)$, and $\mathbf{E}[L(F')] \leq L(F) \cdot \left(1 - \frac{1}{n}\right)^{3/2}$.*

Proof. Let x be the most frequent variable in F. Then x appears at least $L(F)/n$ times (as a leaf label x or $\overline{x}$). Furthermore, since F is simplified, for each leaf labeled with x or $\overline{x}$, its sibling subtree does not contain x. By the simplification rules 1 and 2, after assigning x to be 0 or 1, we can remove at least one leaf for each appearance of x. That is, $L(F') \leq L(F) - L(F)/n = L(F) \cdot (1 - 1/n)$.

Moreover, for each appearance of x, we expect to remove its sibling with probability $1/2$. Since the sibling has size at least 1 and does not contain x, we have

$$\mathbf{E}[L(F')] \leq L(F) - \frac{L(F)}{n} - \frac{1}{2} \cdot \frac{L(F)}{n} = L(F) \cdot \left(1 - \frac{3}{2n}\right) \leq L(F) \cdot \left(1 - \frac{1}{n}\right)^{3/2},$$

where the last inequality is by $1 - ax \leq (1-x)^a$ true for $0 \leq x \leq 1$ and $a \geq 1$ (see below). □

Lemma 3.8. *For $0 \leq x \leq 1$ and $a \geq 1$, it holds that $(1 - ax) \leq (1 - x)^a$.*

Proof. For $1 \leq a < 2$, by Taylor' series,

$$\begin{aligned} & (1-x)^a \\ = \;& 1 - ax + \frac{a(a-1)}{2!}x^2\left(1 - \frac{a-2}{3}x\right) + \frac{a(a-1)(a-2)(a-3)}{4!}x^4\left(1 - \frac{a-5}{5}x\right) + \cdots \\ \geq \;& 1 - ax \end{aligned}$$

For $2 \leq a < 3$, we have $(1-x)^a \geq (1-x)\,(1-(a-1)x) \geq 1 - ax$. By induction, we can prove the inequality for all intervals $i \leq a < i+1$, for integers $i \geq 1$. □

Let R_i be the random value assigned to the restricted variable in step i. Set $L_i := L(F_i)$, and $l_i := \log L_i$. Define a sequence of random variables $\{Z_i\}$ as follows:

$$Z_i = l_i - l_{i-1} - \frac{3}{2}\log\left(1 - \frac{1}{n-i+1}\right).$$

Note that, given $R_1, \ldots, R_{i-1}$, the random variable Z_i assumes two values with equal probability.

Lemma 3.9. *Let $X_0 = 0$ and $X_i = \sum_{j=1}^{i} Z_j$. Then the sequence $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$, and, for each Z_i, we have*

$$Z_i \leq c_i := -\frac{1}{2}\log\left(1 - \frac{1}{n-i+1}\right).$$

$$\mathbf{E}[Z_i \mid R_i, \ldots, R_1] \leq 0,$$

$$\mathbf{E}[e^{tZ_i} \mid R_i, \ldots, R_1] \leq e^{t^2 c_i^2/2}.$$

Proof. Using Lemma 3.7, we get

$$l_i \leq l_{i-1} + \log\left(1 - \frac{1}{n-i+1}\right).$$

Then,

$$\begin{aligned} Z_i &= l_i - l_{i-1} - \frac{3}{2}\log\left(1 - \frac{1}{n-i+1}\right) \\ &\leq \log\left(1 - \frac{1}{n-i+1}\right) - \frac{3}{2}\log\left(1 - \frac{1}{n-i+1}\right) \\ &= -\frac{1}{2}\log\left(1 - \frac{1}{n-i+1}\right) \\ &\equiv c_i \end{aligned}$$

By Jensen's inequality, $\mathbf{E}[l_i \mid R_{i-1}, \ldots, R_1] \leq \log \mathbf{E}[L_i \mid R_{i-1}, \ldots, R_1]$, which, by Lemma 3.7, is at most

$$\log\left(L_{i-1} \cdot \left(1 - \frac{1}{n-i+1}\right)^{3/2}\right) = l_{i-1} + \frac{3}{2}\log\left(1 - \frac{1}{n-i+1}\right);$$

this implies $\mathbf{E}[Z_i \mid R_{i-1}, \ldots, R_1] \leq 0$, and so $\{X_i\}$ is indeed a supermartingale.

Conditioning on the first $i-1$ random bits, Z_i takes two values with equal probability, $\mathbf{E}[Z_i \mid R_{i-1}, \ldots, R_1] \leq 0$, and $Z_i \leq c_i$. By Lemma 3.4, we have $\mathbf{E}[e^{tZ_i} \mid R_1, \ldots, R_{i-1}] \leq e^{t^2 c_i^2/2}$.

□

Now we can complete the proof of the Shrinkage Lemma.

Proof of Lemma 3.6. Let λ be arbitrary, and let c_i's be as defined in Lemma 3.9. By Lemma 3.9 and Lemma 6.4, we get

$$\mathbf{Pr}\left[\sum_{j=1}^{i} Z_j \geq \lambda\right] \leq \exp\left(-\frac{\lambda^2}{2\sum_{j=1}^{i} c_j^2}\right).$$

For the left-hand side, we get by the definition of Z_j's that $\sum_{j=1}^{i} Z_j = l_i - l_0 - \frac{3}{2}\log\frac{n-i}{n}$. since

$$\begin{aligned}\sum_{j=1}^{i} Z_j &= l_i - l_{i-1} - \frac{3}{2}\log\left(1 - \frac{1}{n-(i-1)}\right) + \ldots + l_1 - l_0 - \frac{3}{2}\log\left(1-\frac{1}{n}\right)\\ &= l_i - l_0 - \frac{3}{2}\log\left(\frac{n-i}{n}\right),\end{aligned}$$

Hence,

$$\begin{aligned}\mathbf{Pr}\left[\sum_{j=1}^{i} Z_j \geq \lambda\right] &= \mathbf{Pr}\left[l_i - l_0 - \frac{3}{2}\log\left(\frac{n-i}{n}\right) \geq \lambda\right]\\ &= \mathbf{Pr}\left[\exp\left(l_i - l_0 - \frac{3}{2}\log\left(\frac{n-i}{n}\right)\right) \geq e^{\lambda}\right]\\ &= \mathbf{Pr}\left[L_i \geq e^{\lambda} L_0 \left(\frac{n-i}{n}\right)^{3/2}\right].\end{aligned}$$

For each $1 \leq j \leq i$, we have $c_j \leq \frac{1}{2}\cdot\frac{1}{n-j}$.

$$\begin{aligned}c_j &= -\frac{1}{2}\log\left(1 - \frac{1}{n-j+1}\right)\\ &= \frac{1}{2}\log\left(1 + \frac{1}{n-j}\right)\\ &\leq \frac{1}{2}\cdot\frac{1}{n-j},\end{aligned}$$

where the last inequality is by $\log(1+x) \leq x$. Thus, $\sum_{j=1}^{i} c_j^2$ is at most

$$\frac{1}{4}\sum_{j=1}^{i}\left(\frac{1}{n-j}\right)^2 \leq \frac{1}{4}\sum_{j=1}^{i}\left(\frac{1}{n-j-1} - \frac{1}{n-j}\right) = \frac{1}{4}\cdot\left(\frac{1}{n-i-1} - \frac{1}{n-1}\right) \leq \frac{1}{4}\cdot\frac{1}{n-i-1}.$$

Taking $i = n-k$, we get

$$\mathbf{Pr}\left[L_{n-k} \geq e^{\lambda} L_0 \left(\frac{k}{n}\right)^{3/2}\right] \leq \exp\left(-\frac{\lambda^2}{2\sum_{j=1}^{n-k} c_j^2}\right) \leq e^{-2\lambda^2(k-1)}.$$

Choosing $\lambda = \ln 2$ and $k \geq 4$, concludes the proof.

$$\mathbf{Pr}\left[L(F_{n-k}) \geq 2 \cdot L(F)\left(\frac{k}{n}\right)^{3/2}\right] < 2^{-k},$$

□

3.3 #SAT Algorithms for Formulas

3.3.1 $n^{2.49}$-Size Formulas

Here we show the existence of "better than brute-force" #SAT algorithms for formulas of about quadratic size.

Theorem 3.10. *There is a deterministic algorithm for counting the number of satisfying assignments in a given formula on n variables of size at most n^d which runs in time $t(n) \leq 2^{n-n^\delta}$, for some constant $0 < \delta < 1$ (dependent on d), where the constant d is such that*

- $d < 2.5$ *for de Morgan formulas, and*
- $d < 2$ *for formulas over the complete basis and for branching programs.*

Proof. We consider the case of de Morgan formulas only; the case of general formulas and branching programs is similar (using the shrinkage exponent $\Gamma = 1$ rather than $\Gamma = 1.5$). Suppose we have a formula F on n variables of size $n^{2.5-\epsilon}$ for a small constant $\epsilon > 0$. Let $k = n^\alpha$ and $\alpha < \frac{2}{3}\epsilon$. We build a restriction decision tree with 2^{n-k} branches as follows:

> Starting with F at the root, find the most frequent variable in the current formula, set the variable first to 0 then to 1, and simplify the resulting two subformulas. Make these subformulas the children of the current node. Continue until get a full binary tree of depth exactly $n - k$.

Note that constructing this decision tree takes time $2^{n-k}\mathsf{poly}(n)$. By the Shrinkage Lemma (Lemma 3.6), for all but at most 2^{-k} fraction of the leaves have the formula size $L(F_{n-k}) < 2 \cdot L(F)\left(\frac{k}{n}\right)^{3/2} = 2 \cdot n^{2.5-\epsilon} \cdot n^{1.5(\alpha-1)} = 2n^{1-\epsilon+1.5\alpha}$.

To solve #SAT for all "big" formulas (those that haven't shrunk), we use brute-force enumeration over all possible assignments to the k variables left. The running time is bounded by $2^{n-k} \cdot 2^{-k} \cdot 2^k \cdot \mathsf{poly}(n) \leq 2^{n-k} \cdot \mathsf{poly}(n)$.

For "small" formulas (those that shrunk to the size less than $2n^\gamma$ for some $\gamma = 1 - \epsilon + 1.5\alpha$), we use memoization. First, we enumerate all formulas of such size, and compute and store the number of satisfying assignments for each of them. Then, as we go over the leaves of the decision tree that correspond to small formulas, we simply look up the stored answers for these formulas.

There are at most $2^{O(n^\gamma \log n)}$ such formulas, and counting the satisfying assignments for each one (with k inputs) takes time $2^k \mathsf{poly}(n^\gamma) = 2^{n^\alpha} \cdot \mathsf{poly}(n)$. The total time of evaluating all formulas of size $2n^\gamma$ is bounded by $2^{O(n^\gamma \log n)} \cdot 2^{n^\alpha} \cdot \mathsf{poly}(n) \leq 2^{O(n^\gamma \log n)}$. Including pre-processing, computing #SAT for all small formulas takes time at most $2^{n-k} \cdot \mathsf{poly}(n) + 2^{O(n^\gamma \log n)} \leq 2^{n-n^\alpha} \cdot \mathsf{poly}(n)$.

Thus, the overall running time is bounded by 2^{n-n^δ} for some $\delta > 0$. □

3.3.2 Linear-Size Formulas

First, we give a simplified analysis of Santhanam's $2^{n-\delta n}$-time satisfiability algorithm [San10] for cn-size de Morgan formulas on n variables, getting an explicit bound on the savings δ along the way (in [San10], the savings δ was some unspecified inverse polynomial in c).

Theorem 3.11 ([San10])**.** *There is a deterministic algorithm for counting the number of satisfying assignments of a given cn-size de Morgan formula on n variables that runs in time $2^{n-\delta n}$, for $\delta \geq 1/(32 \cdot c^2)$.*

Proof. Let F be a de Morgan formula of linear size cn for some constant c. Let $p = \left(\frac{1}{4c}\right)^2$ and $k = pn$. We construct a decision tree of $n - k$ levels in exactly the same way as in the proof of Theorem 3.10. At each level, we choose the most frequent variable in the current formula, generate two branches by assigning the variable either 0 or 1, and then restrict the formula by the assignment. The final decision tree has 2^{n-k} branches. By the Shrinkage Lemma (Lemma 3.6), all but 2^{-k} fraction of leaves have the formula size $L(F_{n-k}) \leq 2 \cdot L(F) \left(\frac{k}{n}\right)^{3/2} = 2 \cdot cn \cdot p^{3/2} = 2cp^{1/2} \cdot pn = \frac{1}{2}pn = \frac{k}{2}$.

To compute #SAT for all "big" formulas, we use brute-force enumerations over all possible assignments to the k variables which are left. The running time in total is bounded by $2^{n-k} \cdot 2^{-k} \cdot 2^k \cdot \mathsf{poly}(n) = 2^{n-k} \cdot \mathsf{poly}(n)$.

For "small" formulas (with size less than $k/2$), there are at most $k/2$ variables left. To compute #SAT for all such formulas, the total running time is bounded by $2^{n-k} \cdot 2^{k/2} \cdot \mathsf{poly}(n) = 2^{n-k/2} \cdot \mathsf{poly}(n)$.

The overall running time of counting the number of satisfying assignments of a de Morgan formula of size cn is bounded by $2^{n-\delta n}\text{poly}(n)$ where $\delta = \frac{1}{32c^2}$. □

Remark 3.12. Santhanam's SAT algorithm relies on the fact that, under most restrictions, a given linear-size de Morgan formula will simplify to a formula that doesn't depend on all of the remaining variables. The same is not true for de Morgan formulas of size at least n^2, as such formulas can compute the parity function on n bits. It is an interesting question whether one can devise a non-trivial SAT algorithm for super-quadratic-size de Morgan formulas that uses, say, polynomial space.

We can also use the "supermartingale approach" to provide a different analysis of the #SAT algorithm for linear-size general formulas of [ST12]. At a high level, the argument of [ST12] is as follows. One runs a greedy branching process (picking variables to restrict, and restricting them to both 0 and 1) on a given general formula. Either at some point in this process, we get a subformula that is easy to check for satisfiability (using, e.g., linear algebra), or else the formula will keep shrinking (similarly to the case of de Morgan formulas). That is, assuming that we don't get a formula amenable to linear-algebraic methods, we can show that the formulas will behave similarly to de Morgan formulas and so keep shrinking with some shrinkage exponent slightly bigger than 1.

More precisely, Seto and Tamaki [ST12] show that if we don't get a simple enough formula to solve using linear algebra, then in each step of the branching process there will be a *constant number* of variables to restrict so that all the restrictions of these variables are guaranteed to make the formula "slightly" smaller (by a certain known value), and moreover, for at least half of such restrictions, the new formula gets "significantly" smaller. The latter is similar to what happens in the case of de Morgan formulas after one restricts *one* variable (albeit with much worse shrinkage parameters). The main difference is that for general formulas (of linear size), we need to restrict more than one but still at most some constant number of variables.

This suggests defining a supermartingale sequence for the sizes of the restricted formula after a certain constant number of variables are set, and applying Lemma 6.4 to that sequence. Indeed, this approach yields the running-time analysis of [ST12]'s SAT algorithm for cn-size general formulas on n variables, with the running time $2^{n-\delta n}$, for δ about c^{-c^3}.

Finally, we observe that the proof of Theorem 3.11 immediately yields an average-case

lower bound for linear-size de Morgan formulas [San10]. Indeed, by the proof of Theorem 3.11, every cn-size de Morgan formula F on n variables can be computed by a decision tree of height $n-k$, for $k = n/(16c^2)$, where all but 2^{-k} branches of the tree correspond to subformulas on at most $k/2$ of the remaining k variables. Any such subformula has zero correlation with the parity function. Hence, F can correctly compute parity with probability at most $1/2 + 2^{-k} = 1/2 + 2^{-n/(16c^2)}$.

Note that this average-case hardness is nontrivial for $c < \sqrt{n}$, i.e., for de Morgan formulas of size at most $n^{1.5}$. In the following section, we show how to get an average-case lower bound against de Morgan formulas of size about $n^{2.5}$.

3.4 Average-Case Hardness

Here we use our shrinkage result for adaptive restrictions to re-prove a recent result by Komargodski and Raz [KR13] on average-case hardness for de Morgan formulas. Our proof is more modular than the original argument of [KR13], and is arguably simpler. The main differences are: (i) we use restrictions that choose which variable to restrict in a completely deterministic way (rather than randomly), and (ii) we use an extractor for oblivious bit-fixing sources (instead of Andreev's extractor for block-structured sources).

3.4.1 Andreev's Original Argument

We sketch the original idea of Andreev first. Andreev [And87] defined a function $A\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ as follows: Given inputs $x, y \in \{0,1\}^n$, partition y into $\log n$ blocks $y_1, \ldots, y_{\log n}$ of size $n/\log n$ each. Let b_i be the parity of block y_i, and output the bit of x in the position $b_1 \ldots b_{\log n}$ (where we interpret the $\log n$-bit string $b_1 \ldots b_{\log n}$ as an integer between 0 and $n-1$). Note that the de Morgan formula complexity of $A(x,y)$ is at least that of $A(x_0, y)$ for any fixed string x_0. Andreev argued that if x_0 is a truth table of a function of maximal formula complexity, then the resulting function $A'(y) = A(x_0, y)$ will be hard for de Morgan formulas of certain size (dependent on the best available shrinkage exponent Γ).

The proof is by contradiction. Suppose we have a small de Morgan formula computing $A'(y)$. The argument relies on two observations. First, under a random restriction (with appropriate parameters), the restricted subformula of $A'(y)$ will have size considerably less

than n. Secondly, a random restriction is likely to leave at least one variable free (unrestricted) in each of the blocks. When both these events happen, we get a small-size de Morgan formula that can be used to compute the bits of x_0, which contradicts the assumed hardness of x_0.

Looking at Andreev's argument more closely, we observe that he uses the second string y to extract $\log n$ bits that are used as a position in the truth table x_0. He needs y to have the property that every $\log n$-bit string can be obtained from y even after y is hit by a random restriction, leaving few variables free. Intuitively, each unrestricted variable in y is a source of a truly random bit, and so the restricted string y is a weak source of randomness containing k truly random bits, where k is the number of unrestricted variables left in y. In fact, this is an oblivious bit-fixing source with k bits of min-entropy.

Andreev uses a very simple extractor for y (extracting one bit of randomness from each block in y), but this extractor works only for "sources of randomness" which have a "block structure", namely, every block contains at least one truly random bit. This dictates that the argument be constrained to use restrictions which in addition to leaving k unrestricted bits, also respect this "block structure" (at least with high probability). This is not an issue in Andreev's argument which uses random restrictions (that indeed respect the "block structure" with high probability). However, this creates difficulties if one wants to use other choices of restrictions as is the case in both [KR13] and the argument of this work.

3.4.2 Adapting to Arbitrary Restrictions, using Extractors

We will show that Andreev's argument can be adapted to work with *any* choice of restrictions (in particular, our adaptive restrictions that choose deterministically which variables to restrict). To this end, we shall use explicit extractors for oblivious bit-fixing sources; in fact, a disperser suffices in this context of worst-case hardness, but an extractor is needed for the case of average-case hardness that we consider later.

One difficulty we need to overcome when using an arbitrary extractor/disperser instead of Andreev's original extractor is an apparent need of *invertibility*: Given a position z into the truth table of x_0, and a restriction, we need to find extractor's pre-image y' of z that is consistent with the restriction. This task is very easy for Andreev's extractor, but quite non-trivial in general. We remark that [GS12] constructed dispersers for oblivious bit-fixing sources which are *invertible* in expected polynomial time. Naively, we seem to require an

inverting procedure that is computable by a small de Morgan formula, in order to argue that we get a small de Morgan formula for the assumed hard string x_0. However, we will show that for Andreev's argument, one can start with *any incompressible string* x_0, not just of high de Morgan formula complexity, but rather, say, of high *Kolmogorov complexity*. This makes the whole argument of deriving a contradiction to the assumed hardness of x_0 much simpler: we just need to argue that the existence of a small de Morgan formula for $A(x_0, y)$ implies the existence of a *short description in the Kolmogorov sense* for the string x_0. The reconstruction procedure for x_0 may take arbitrary amount of time, and so in particular, it is acceptable to use even brute-force inverting procedures for extractors/dispersers.

We provide the details on how to use dispersers in Andreev's worst-case hardness argument next. We define a modified version of Andreev's function using the following zero-error disperser.

Theorem 3.13 ([GS12])**.** *There exist $c > 1$ and $0 < \eta < 1$ such that, for all sufficiently large n, $k > (\log n)^c$, there is a $\mathrm{poly}(n)$-time computable zero-error disperser $D : \{0,1\}^n \to \{0,1\}^{k-o(k)}$ for oblivious (n, k)-bit-fixing sources.*

The modified function $B : \{0,1\}^{4n} \times \{0,1\}^n \to \{0,1\}$ is defined by $B(x, y) = x_{D(y)}$, where D is a disperser that extracts $\log(4n) = \log n + 2$ bits from oblivious bit-fixing sources containing $k = (\log n)^c$ random bits. That is, we use a more powerful disperser instead of Andreev's naive parity based disperser. In addition, we also increased the length of the first input x from n to $4n$. This is done for technical reasons related to the use of Kolmogorov complexity.

Next, fix a string x_0 of length $4n$ whose Kolmogorov complexity is $K(x_0) \geq 4n$, and consider the function $B'(y) = B(x_0, y)$. Suppose $B'(y)$ has a de Morgan formula F. The shrinkage result of Lemma 3.6 says that, after adaptively restricting $n - k$ variables via a random restriction ρ, the formula size will shrink with high probability. Denote by F' the formula after a restriction ρ, i.e., $F' = F|_\rho$. Then,

$$\mathbf{Pr}\left[L(F') \leq 2L(F)\left(\frac{k}{n}\right)^{3/2}\right] > 1 - \frac{1}{2^k}.$$

Fix a good restriction ρ and consider the formula F' obtained from F using the restriction ρ. We will use the descriptions of F' and ρ to reconstruct the string x_0, using the following procedure:

> Given a formula $F'(y')$, a restriction ρ, and n in binary, go over all values $0 \leq i \leq 4n-1$. For each i, find a pre-image $z = D^{-1}(i)$ consistent with the restriction ρ (by trying all possible values for the free variables y' and evaluating D on the input described by the restriction ρ plus the chosen values for y'), and output $F'(z')$, where z' is the part of z corresponding to the unrestricted variables y'.

For the correctness analysis, for each position $0 \leq i \leq 4n-1$, there will be a required preimage z to the disperser (since the disperser is zero-error). Since F correctly computes $B'(y)$, we get that $F'(z')$ equals the bit of x_0 in the position $D(z) = i$.

The input size that the above procedure for reconstructing x_0 takes is at most $L(F') \cdot \log L(F') + 2n + 2\log n + 2$ bits to describe the restricted formula F', the restriction ρ, and the input size n. Indeed, we can first describe n by repeating twice each bit of the $\log n$-bit string n, followed by the two-bit string 01, followed by $2n$-bit string describing the restriction ρ (saying for each position $0 \leq i \leq n-1$ of y whether it's 0, 1, or $*$), followed by the description of F'. We get

$$4n \leq K(x_0) \leq L(F') \cdot \log L(F') + 2n + 2\log n + c,$$

for some constant c (which takes into account the constant-size description of the Turing machine performing the reconstruction of x_0). Hence, $L(F') > n/\log n$. We conclude that $L(F) \geq n^{2.5}/\text{poly}\log n$, and hence also the function $B(x,y)$ requires de Morgan formulas of at least that size, up to a constant factor.

3.4.3 Average-Case Hardness for Formulas of Size $n^{2.49}$

Here we generalize the argument from the previous subsection to prove *average-case* hardness. We will use the following extractor by Rao [Rao09].

Theorem 3.14 ([Rao09])**.** *There exist constants $d < 1$ and $c \geq 1$ such that for every $k(n) > \log^c n$, there is a polynomial time computable extractor $E\colon \{0,1\}^n \to \{0,1\}^{k-o(k)}$ for (n,k)-bit-fixing sources, with error 2^{-k^d}.*

We also use the following binary code whose existence is a folklore result; for completeness, we sketch a possible construction of such a code.

Theorem 3.15. *Let $r = n^\gamma$, for any given $0 < \gamma < 1$. There exists a binary code C mapping $(4n)$-bit message to a codeword of length 2^r, such that C is (ρ, L)-list decodable*

for $\rho = 1/2 - O(2^{-r/4})$ and $L \leq O(2^{r/2})$. Furthermore, there is a polynomial-time algorithm for computing $C(x)$ in position z, for any given inputs $x \in \{0,1\}^{4n}$ and $z \in \{0,1\}^r$.

Proof sketch. For a parameter $\epsilon > 0$, let $S \subseteq \{0,1\}^{4n}$ be an explicit ϵ-biased sample space. Using a powering construction from [AGHP92], we get such a set of size $(4n/\epsilon)^2$, where for each $1 \leq i \leq |S|$, we can compute the ith string in S in time $\mathrm{poly}(n)$. For $x \in \{0,1\}^{4n}$ and position $1 \leq i \leq |S|$, we define the ith symbol of the codeword of x by $C(x)_i = \langle x, y_i \rangle \mod 2$, where y_i is the ith string in S. By construction, the code has relative minimum distance at least $1/2 - \epsilon$. Hence, by the Johnson bound, the code is $(1/2 - O(\sqrt{\epsilon}), O(1/\epsilon))$-list-decodable. We choose ϵ so that $|S| = 2^r$, which yields $\epsilon = O(2^{-r/2})$. □

Loosely speaking, as in [KR13], the code is used to perform "worst-case to average-case hardness amplification" in the spirit of [STV01]: When applied on a truth table x_0 of a function that is hard in the worst case, $C(x_0)$ is the truth table of a function that is hard on average. Here "hardness" refers to description size.

We extend the definition of the previous section and use the modified Andreev's function after applying the error-correcting code. Namely, let $f : \{0,1\}^{4n} \times \{0,1\}^n \to \{0,1\}$ be defined by $f(x,y) = C(x)_{E(y)}$, where C is the code from Theorem 3.15 and E is Rao's extractor (from Theorem 3.14) mapping n bits to $m = r = n^\gamma$ bits, for the min-entropy $k \geq 2m$. We will prove the following.

Theorem 3.16. *Let x_0 be any fixed $(4n)$-bit string of Kolmogorov complexity $K(x_0) \geq 3n$. Define $f'(y) = f(x_0, y)$. Then there exists a constant $0 < \sigma < 1$ such that, for any de Morgan formula F of size at most $n^{2.49}$ on n inputs, we have*

$$\mathbf{Pr}_{y \in \{0,1\}^n}[F(y) = f'(y)] < \frac{1}{2} + \frac{1}{2^{n^\sigma}}.$$

Proof. We will use an argument similar to that from the previous section, where we argued worst-case hardness. Towards a contradiction, suppose that there is a small de Morgan formula F computing $f'(y)$ well on average:

$$\mathbf{Pr}_{y \in \{0,1\}^n}[F(y) = f'(y)] \geq \frac{1}{2} + \frac{1}{2^{n^\sigma}}. \tag{3.1}$$

For $k = 2m = 2n^\gamma$, consider a restriction decision tree of depth $n - k$ for the formula F. We know by the Shrinkage Lemma (Lemma 3.6) that all but 2^{-k} fraction of leaves of the decision tree correspond to restricted subformulas of F of de Morgan formula size

$s < 2 \cdot L(F)(k/n)^{3/2}$. For a sufficiently small $\gamma > 0$, we can get that $s < n^{0.99}$, and hence, the description size of each such subformula is less than $n^{0.991}$.

Note that the restriction decision tree of depth $n - k$ partitions the universe $\{0,1\}^n$ into disjoint subsets of inputs of equal size 2^k each. Furthermore, the distribution of choosing a restriction by the specified process, and then uniformly selecting the unrestricted bits, induces a uniform n bit string. Hence, the probability on the left-hand side of Eq. (3.1) is equal to the average over all branches of this decision tree of the success probabilities of the restricted subformulas computing the corresponding restrictions of f'. Since there are at most 2^{-k} fraction of "bad" restrictions (which do not shrink the formula F), we conclude that the average over "good" restrictions ρ (those that shrink the formula F) of the success probabilities $\mathbf{Pr}_y[F|_\rho(y) = f'|_\rho(y)]$ is at most 2^{-k} smaller than the right hand-side of Eq. (3.1). By averaging, there exists a restriction ρ such that $F' = F|_\rho$ agrees with $f'|_\rho$ in at least $1/2 + 2^{-n^\sigma} - 2^{-k}$ fraction of the remaining 2^k inputs, and at the same time F' has the reduced size $s < n^{0.99}$.

Let y' denote the k unrestricted variables left in y. For any given k-bit string a, we denote by (ρ, a) the input to the function $f'(y)$ obtained using the restriction ρ and the values a for the unrestricted variables y'. We have

$$\mathbf{Pr}_{y' \in \{0,1\}^k}[F'(y') = C(x_0)_{E(\rho,y')}] \geq \frac{1}{2} + \frac{1}{2^{n^\sigma}} - \frac{1}{2^k}. \tag{3.2}$$

Note that the probability above is for a random experiment where we first choose a uniformly random $y' \in \{0,1\}^k$ which determines $z = E(\rho, y')$. Equivalently, we can first choose $z = E(\rho, y'')$ for a random $y'' \in \{0,1\}^k$, and then set y' to be a uniformly random k-bit string such that $E(\rho, y') = z$. Finally, consider a new experiment where we choose z uniformly at random from $\{0,1\}^r$, and then choose y' uniformly at random so that $E(\rho, y') = z$. Since E is an extractor with error at most 2^{-k^d} (by Theorem 3.14), the probability in Eq. (3.2) will reduce by at most 2^{-k^d}. Thus we get the following randomized algorithm for computing $C(x_0)$ at a given position z:

> Given n and the descriptions of F' and ρ, on input $z \in \{0,1\}^r$, pick a uniformly random $y' \in \{0,1\}^k$ such that $E(\rho, y') = z$, and output $F'(y')$. (Output an arbitrary value if there does not exist a y' such that $E(\rho, y') = z$).

By the discussion above, we have the described procedure computes $C(x_0)$ correctly with probability at least $\epsilon = 1/2 + 2^{-n^\sigma} - 2^{-k} - 2^{-k^d}$, where the probability is over both the

codeword position $z \in \{0,1\}^r$ and the internal randomness used to sample y'. By choosing σ sufficiently small as a function of γ and d, we can ensure that $\epsilon \geq 1/2 + 2^{-n^{\gamma d/2}} = 1/2 + 2^{-r^{d/2}}$.

Equivalently, we could implement the above procedure as follows: given z, consider all k-bit strings y' such that $E(\rho, y') = z$, calculate the fraction p_z of those strings y' from that set where $F'(y') = 1$, and output 1 with probability p_z, and 0 otherwise. This way, the internal randomness we need is the randomness to pick a uniformly random point on the unit interval $[0,1]$. This can be done up to an error 2^{-t}, using t uniformly random bits. By choosing $t = r$, we ensure that this modified algorithm succeeds with about the same probability, and that it uses t uniformly random bits for internal randomness that are independent of the string z. By averaging, there is a particular string $\alpha_0 \in \{0,1\}^t$ such that our algorithm correctly computes $C(x_0)$ on at least $1/2 + 2^{-r/4}$ fraction of positions $z \in \{0,1\}^r$, when using this α_0 as advice.

Thus we get a deterministic algorithm (with advice) that outputs some 2^r-bit string w that agrees with $C(x_0)$ in at least $1/2 + 2^{-r/4}$ fraction of positions. The amount of nonuniform advice needed by this algorithms is at most $n^{0.991} + 2n + r + O(\log n) \leq (2.1)n$ to describe the subformula F', restriction ρ, internal randomness α_0, and the input length n.

The list-decodability of the code C (Theorem 3.15) implies there are at most $O(2^{r/2})$ codewords that have such high agreement with w. We can describe the required codeword $C(x_0)$ by specifying its index of at most r bits in the collection of all such codewords (ordered lexicographically). This would add extra $r = n^\gamma$ bits of advice to our algorithm above. The overall amount of advice will be less than $(2.5)n$ bits.

Once we know $C(x_0)$, we can also recover the message x_0, using a uniform algorithm that does brute-force decoding. We conclude that $K(x_0) < 3n$, contradicting our choice of x_0. □

As a corollary, we get

Theorem 3.17. *There is a constant $0 < \sigma < 1$ such that, for any de Morgan formula F of size at most $n^{2.49}$ on $5n$ inputs, we have*

$$\mathbf{Pr}_{x \in \{0,1\}^{4n}, y \in \{0,1\}^n}[F(x,y) = f(x,y)] < \frac{1}{2} + \frac{1}{2^{n^\sigma}}.$$

Proof. The proof is by a simple averaging argument applied to Theorem 3.16. Suppose there is a de Morgan formula F that agrees with $f(x,y)$ on at least $1/2 + \epsilon$ fraction of pairs

(x, y), for $\epsilon = 2^{-n^\sigma}$. By averaging, there is a subset S containing at least $\epsilon/2$ fraction of strings x, such that for each x' from the subset we have $F(x', y) = f(x', y)$ on at least $1/2 + \epsilon/2$ fraction of y's.

On the other hand, the fraction of $4n$-bit strings that have Kolmogorov complexity less than $3n$ is at most $2^{3n}/2^{4n} = 2^{-n}$, which is much less than $\epsilon/2$. Hence, there is a $(4n)$-bit string x_0 with $K(x_0) \geq 3n$, such that $F(x_0, y)$ has non-trivial agreement with $f(x_0, y)$ over random y's. The latter contradicts Theorem 3.16. □

Since the function $f(x, y)$ is computable in P (using the fact that the code C and the extractor E are efficiently computable), we get an explicit function in P that has exponential average-case hardness with respect to de Morgan formulas of size $n^{2.49}$.

Remark 3.18. The average-case lower bound for general formulas and branching programs of size at most $n^{1.99}$ can be argued in exactly the same way, using the corresponding shrinkage result. In particular, we can prove the analogue of Theorem 3.16, by observing that a general formula (branching program) of size $n^{1.99}$ is also likes to shrink to size below $n^{0.99}$ (for the same parameter k), and then proceeding with the rest of the proof as before.

3.5 Linear-Size Formulas over the Full Basis

Seto and Tamaki [ST12] give a satisfiability algorithm for linear-size formulas over the full basis, generalizing Santhanam's algorithm [San10] for de Morgan formulas. Here we give a simplified analysis with the "supermartingale approach".

Theorem 3.19 ([ST12])**.** *There is a deterministic algorithm for counting the number of satisfying assignments of a cn-size Boolean formula over the complete basis that runs in time $2^{n-\delta n}$ for $\delta = 2^{-O(c^3 \log c)}$.*

The algorithm is based on a specific property of linear-size general formulas. Below we first state the property and the algorithm, and then analyze the running time of the algorithm.

Without loss of generality, we assume a Boolean formula over the complete basis is a tree in which each leaf is labeled by a literal (x or $\overline{x}$) and each internal node is labeled by a gate from $\{\wedge, \vee, \oplus\}$. Any Boolean formula over the complete basis can be efficiently transformed into this form by de Morgan's law and the fact that $\overline{x \oplus y} = \overline{x} \oplus y$.

Given a formula tree, we call a node *linear* if (1) it is a leaf, or (2) it is labeled by $\oplus$ and both of its child nodes are linear. We say a linear node is *maximal* if its parent node is not linear. For a node v in a formula F, we denote by F_v the subformula rooted at v. Note that for a linear node v, the subformula F_v computes the parity of all its leaves. We say two maximal linear nodes u and v are *mergable* if they are connected by a path in which every node is labeled by $\oplus$. We can merge u and v in the following way. Suppose we have $F_s = F_u \oplus F_{u'}$, and $F_t = F_v \oplus F_{v'}$, that is, s and t are the parent nodes of u and v respectively, and u' and v' are the siblings of u and v. Then we can replace F_u by $F_u \oplus F_v$ and F_t by $F_{v'}$.

We have the following simplification rules, in addition to the rules for de Morgan formulas: (1) If $0 \oplus \psi$ or $1 \oplus \psi$ appears, then replace it by ψ or $\overline{\psi}$, respectively. (2) If a variable x appears more than once (as x or $\overline{x}$) in a linear node, then eliminate redundancy by the commutativity of $\oplus$ and the facts that $x \oplus x = 0$ and $x \oplus \overline{x} = 1$. (3) Merge any mergable maximal linear nodes.

Based on these simplification rules, Seto and Tamaki [ST12] identify the following structural property of linear-size general formulas.

Lemma 3.20 ([ST12])**.** *Let F be a formula on n variables of size cn for some constant c. Then one of the following cases must be true:*

1. *The formula size is small: $c \leq 3/4$.*
2. *The total number of maximal linear nodes is less than $3n/4$.*
3. *There exists a variable appearing at least $c + \frac{1}{8c}$ times.*
4. *There exists a maximal linear node v with $L(F_v) \leq 8c$ such that the parent node of v is either $\wedge$ or $\vee$, and every variable in F_v appears at least c times in F.*

For completeness, the proof of this lemma is given in the appendix.

Lemma 3.20 was proved in [ST12] using the following two lemmas.

Lemma 3.21. *Let F be a simplified formula containing at least one node labeled by $\vee$ or $\wedge$. Let $\#MLin_{\vee\wedge}(F)$ be the number of maximal linear nodes with parents labeled by $\vee$ or $\wedge$. Let $\#MLin_{\oplus}(F)$ be the number of maximal linear nodes with parents labeled by $\oplus$. Then $\#MLin_{\vee\wedge}(F) \geq \#MLin_{\oplus}(F)$.*

Proof. It is easy to see that the result holds for $L(F) \leq 3$. Also, if there is exactly one internal node labeled by $\vee$ or $\wedge$, then it is easy to see that the result holds. The rest of the proof is by induction.

If there is a maximal linear node of size at least two, then it must have a subtree of two leaves $s \oplus t$. We can replace this subtree by a single node; this reduces the formula size by 1, but does not change $\#MLin_{\vee\wedge}(F)$ or $\#MLin_{\oplus}(F)$. By induction, the result holds.

If every maximal linear node has size exactly one, then there must be a subtree of two leaves $s \vee t$ or $s \wedge t$. Let u be the parent node of s and t, and v be the parent node of v. If v is labeled by $\vee$ or $\wedge$, we can replace u by a single node; this will reduce the formula size by 1 and $\#MLin_{\vee\wedge}(F)$ by 1, but $\#MLin_{\oplus}(F)$ does not change. By induction, the result holds.

If v is labeled by $\oplus$, let w be the sibling of u, then we can replace v by w; this will reduce the formula size by 2 and $\#MLin_{\vee\wedge}(F)$ by 2, and reduce $\#MLin_{\oplus}(F)$ by at most 1. By induction, the result also holds. □

Lemma 3.22. *Let F be a simplified formula of size cn, and each variable appears less than $c + \frac{1}{8c}$ times. (i) There are at most $n/8$ maximal linear nodes with more than $8c$ leaves. (ii) There are at most $n/8$ maximal linear nodes which has a variable appearing less than c times in F.*

Proof. (i) is by an averaging argument.

Let $freq(x)$ be the number of times x appearing in F. By assumption, no variable appears $c + 1/8c$ times. If a variables appears less than c times, it must also be less than $c+1/8c-1$, since the frequency is an integer. There are at most $n/8c$ such variables, since the other variables appear less than $c + 1/8c$ times. The total number of maximal linear nodes containing such variables is less than $c \cdot n/8c = n/8$. □

The satisfiability algorithm follows directly from this property. For case 1, a brute-force search is sufficient. For case 2, we again use a brute-force search, but this time to enumerate all possible assignments to maximal linear nodes, and, for each assignment, solve a system of linear equations using Gaussian elimination. In both cases the running time is $2^{3n/4}\mathsf{poly}(n)$. For cases 3 and 4, the algorithm is based on a step-by-step restriction. At each step, we are able to restrict a constant number of variables such that the shrinkage of the formula size is non-trivial.

The restriction defined in Seto and Tamaki [ST12] is analogous to the restriction for de Morgan formulas. For de Morgan formulas, we restrict the most frequent variable and this gives non-trivial shrinkage. For formulas over the complete basis, we are able to restrict a constant number of variables such that the formula shrinks non-trivially, where the constant depends on the linear size of the formula.

Now we define the restrictions for a formula in case 3 or 4 of the above analysis.

In particular, for case 3, we randomly restrict the first variable which appears at least $c + 1/8c$ times; that eliminates at least $c + 1/8c$ leaves.

For case 4, let u be the sibling of the maximal linear node v. Consider the following two sub-cases: (a) there exists a variable appearing in F_u but not in F_v; (b) all variables in F_u appear in F_v.

For case 4(a), we randomly restrict all variables in the subformula F_v. Suppose there are totally $b \leq 8c$ variables in F_v. Since each of them appears at least c times, we can eliminate at least bc leaves. Furthermore, since F_v takes value 0 or 1 with equal probability, and the parent node of v is labeled by either $\wedge$ or $\vee$, the sibling node of v can be eliminated with probability $1/2$. Since there is an extra variable in the sibling, we eliminate at least $bc + 1$ leaves with probability $1/2$.

For case 4(b), suppose x is one common variable in both F_v and F_u, and there are totally $b + 1 \leq 8c$ variables in F_v. We randomly restrict all variables in F_v except x. This eliminates at least $bc + 1$ leaves, since each variable appears at least c times, and at least one appearance of x in F_v and F_u can be eliminated.

To unify the cases 3, 4(a) and 4(b), in each case, we can deterministically find $1 \leq b \leq 8c$ variables such that by randomly restricting them, we eliminate at least bc leaves, and moreover, with probability $1/2$, eliminate at least $bc(1 + 1/8c^2)$ leaves. Denote by $l(F) := \log L(F)$ and let F' be the new formula after the restriction and simplification. Then we have $l(F') \leq l(F) + \log(1 - b/n)$; and with probability $1/2$, $l(F') \leq l(F) + \left(1 + 1/8c^2\right) \log(1 - b/n)$.

Now we consider a process of adaptive restrictions; this can be viewed as constructing a decision tree. At each step, we assume that only cases 3, 4(a) or 4(b) happens (otherwise, we directly run the brute-force search). As analyzed above, we deterministically find $1 \leq b \leq 8c$ variables and branch on assigning each variable to be 0 or 1. The process continues until at most k variables are free (k will be fixed later). We will argue that the formula size shrinks non-trivially on most of the branches.

We consider the decision tree virtually divided into layers of height $16c$, which means that at each layer, there are exactly $16c$ variables being restricted. For simplicity we assume $n-k$ is divisible by $16c$. Consider a node at the top of one layer; let G be the formula labeling the node, and suppose G is over n variables with size cn. Let G' be the new formula after adaptively restricting $16c$ variables (at the bottom of the layer). Then we have the following bounds on the size of G'.

Lemma 3.23. *It holds that*

$$l(G') \leq l(G) + \log\left(1 - \frac{16c}{n}\right).$$

Moreover, with probability at least $1/2$*,*

$$l(G') \leq l(G) + \log\left(1 - \frac{16c}{n}\right) + \frac{1}{8c^2}\log\left(1 - \frac{1}{n}\right).$$

Proof. Since each variable being restricted appears at least c times, the first inequality holds.

Consider any path in the decision tree starting from G. There must be one descendant node at distance $0 \leq h < 8c$ from G such that case 3, 4(a) or 4(b) happens and in consequence there are $1 \leq b \leq 8c$ variables restricted. Over all descendants of this particular node at the bottom of the layer, it holds with probability at least $1/2$ that

$$\begin{aligned} l(G') &\leq l(G) + \log\left(1 - \frac{h}{n}\right) + \left(1 + \frac{1}{8c^2}\right)\log\left(1 - \frac{b}{n-h}\right) + \log\left(1 - \frac{16c-h-b}{n-h-b}\right) \\ &\leq l(G) + \log\left(1 - \frac{16c}{n}\right) + \frac{1}{8c^2}\log\left(1 - \frac{1}{n}\right). \end{aligned}$$

Note that this inequality does not depend on the particular path in consideration. Thus it holds for all descendants of G at distance exactly $16c$. This ends the proof. □

Now we are ready to prove the shrinkage result for linear-size general formulas.

Lemma 3.24. *Denote by* F_{n-k} *the formula after restricting* $n-k$ *variables. For* $k > 160c$*,*

$$\mathbf{Pr}\left[L(F_{n-k}) \geq 2 \cdot L(F)\left(\frac{k}{n}\right)^{1+\frac{1}{256c^3}}\right] < 2^{-k}.$$

Proof. Consider the nodes in the decision tree at depth $16c \cdot i$, for $i = 0, 1, \ldots, (n-k)/16c$. We define a sequence of random variables

$$Z_i = l(F_{16ci}) - l(F_{16c(i-1)}) - \log\left(1 - \frac{16c}{n - 16c(i-1)}\right) - \frac{1}{16c^2}\log\left(1 - \frac{1}{n - 16c(i-1)}\right).$$

By Lemma 3.23, we have $Z_i \leq c_i := -\frac{1}{16c^2}\log\left(1-\frac{1}{n-16c(i-1)}\right)$. Let $R_1, R_2, \ldots, R_{16c(i-1)}$ be the random bits (the values of the assignments) used at each step. Conditioning on these random bits, it holds with probability $1/2$ that $Z_i \leq -c_i$. Therefore, conditioning on $R_1, \ldots, R_{16c(i-1)}$, Z_i is upper bounded by a variable taking $-c_i$ and c_i with equal probability. By Lemma 6.4, we have for any $\lambda \geq 0$,

$$\mathbf{Pr}\left[\sum_{j=1}^{i} Z_j \geq \lambda\right] \leq \exp\left(-\frac{\lambda^2}{2\sum_{j=1}^{i} c_j^2}\right).$$

Let $i = (n-k)/16c$. We first have that

$$\begin{aligned}
\sum_{j=1}^{i} Z_j &= l(F_{16ci}) - l(F_0) - \sum_{j=0}^{i-1}\log\left(1-\frac{16c}{n-16cj}\right) - \sum_{j=0}^{i-1}\frac{1}{16c^2}\log\left(1-\frac{1}{n-16cj}\right) \\
&\geq l(F_{n-k}) - l(F_0) - \log\left(\frac{k}{n}\right) - \frac{1}{256c^3}\log\left(\frac{k+16c-1}{n+16c-1}\right) \\
&\geq l(F_{n-k}) - l(F_0) - \left(1+\frac{1}{256c^3}\right)\log\left(\frac{k+16c-1}{n+16c-1}\right).
\end{aligned}$$

Here we use the inequality that

$$\begin{aligned}
\sum_{j=0}^{i-1}\log\left(1-\frac{1}{n-bj}\right) &= \frac{1}{b}\sum_{j=0}^{i-1}\log\left(1-\frac{1}{n-bj}\right)^b \\
&\leq \frac{1}{b}\sum_{j=0}^{i-1}\log\left(\left(1-\frac{1}{n-bj+b-1}\right)\cdots\left(1-\frac{1}{n-bj}\right)\right) \\
&= \frac{1}{b}\log\left(\frac{n-bi+b-1}{n+b-1}\right).
\end{aligned}$$

Hence,

$$\mathbf{Pr}\left[\sum_{j=1}^{i} Z_j \geq \lambda\right] \geq \mathbf{Pr}\left[L(F_{n-k}) \geq e^{\lambda} L(F_0)\left(\frac{k+16c-1}{n+16c-1}\right)^{1+\frac{1}{256c^3}}\right].$$

Then since $c_j \leq \frac{1}{16c^2}\cdot\frac{1}{n-16c(j-1)-1}$, we have that

$$\begin{aligned}
\sum_{j=1}^{i} c_j^2 &\leq \left(\frac{1}{16c^2}\right)^2 \sum_{j=1}^{i}\left(\frac{1}{n-16c(j-1)-1}\right)^2 \\
&\leq \left(\frac{1}{16c^2}\right)^2 \sum_{j=1}^{i}\left(\frac{1}{n-16cj-1} - \frac{1}{n-16c(j-1)-1}\right)\cdot\frac{1}{16c} \\
&\leq \frac{1}{16^3c^5}\cdot\frac{1}{n-16ci-1} = \frac{1}{16^3c^5}\cdot\frac{1}{k-1}.
\end{aligned}$$

Therefore,

$$\begin{aligned}\mathbf{Pr}\left[L(F_{n-k}) \geq e^{\lambda} L(F)\left(\frac{k+16c-1}{n+16c-1}\right)^{1+\frac{1}{256c^3}}\right] &\leq \exp\left(-\frac{\lambda^2}{2\cdot\frac{1}{16^3c^5}\cdot\frac{1}{k-1}}\right)\\ &= e^{-2048\lambda^2 c^5(k-1)}.\end{aligned}$$

In particular, for $\lambda = \ln(2/1.2)$ and $k > 160c$,

$$\mathbf{Pr}\left[L(F_{n-k}) \geq 2\cdot L(F)\left(\frac{k}{n}\right)^{1+\frac{1}{256c^3}}\right] < 2^{-k}.$$

□

Now we are ready to analyze the running time of the algorithm.

Proof of Theorem 3.19. Let F be a cn-size general formula on n variables. We build a decision tree based on adaptively restricting variables according to the cases in Lemma 3.20. Whenever the formula is in case 1 or 2, we run the brute-force search; otherwise we adaptively restrict a constant number of variables, and continue the process until there are at most k variables left.

Let $p = (4c)^{-256c^3}$ and $k = pn$. In the worst case, we build a decision tree of $n-k$ levels with 2^{n-k} branches. By Lemma 3.24, at most 2^{-k} fraction of the branches end with formula size

$$L(F_{n-k}) \geq 2\cdot L(F)\left(\frac{k}{n}\right)^{1+\frac{1}{256c^3}} = 2\cdot cn\cdot p^{1+\frac{1}{256c^3}} = 2cp^{\frac{1}{256c^3}}\cdot pn = \frac{1}{2}pn = \frac{k}{2}.$$

To compute #SAT for all such "big" formulas (of size at least $k/2$), we use brute-force enumerations over all possible assignments to the k free variables. The running time in total is bounded by $(2^{n-k}\cdot 2^{-k})\cdot 2^k\cdot \mathsf{poly}(n) = 2^{n-k}\cdot\mathsf{poly}(n)$.

For the other branches which end with "small" formulas (of size less than $k/2$), there are at most $k/2$ variables left. To compute #SAT for all such formulas, the total running time is bounded by $2^{n-k}\cdot 2^{k/2}\cdot\mathsf{poly}(n) = 2^{n-k/2}\cdot\mathsf{poly}(n)$.

The overall running time is bounded by $2^{n-\delta n}\mathsf{poly}(n)$ where $\delta = 2^{-O(c^3\log c)}$. □

Chapter 4

Compression Algorithms and Circuit Lower Bounds

We show that circuit lower bound proofs based on the method of random restrictions yield non-trivial *compression algorithms* for "easy" Boolean functions from the corresponding circuit classes. The compression problem is defined as follows: given the truth table of an n-variate Boolean function f computable by some *unknown small circuit* from a *known class* of circuits, find in deterministic time $\mathrm{poly}(2^n)$ a circuit C (no restriction on the type of C) computing f so that the size of C is less than the trivial circuit size $2^n/n$. We get non-trivial compression for functions computable by AC^0 circuits, (de Morgan) formulas, and (read-once) branching programs of the size for which the lower bounds for the corresponding circuit class are known.

These compression algorithms rely on the structural characterizations of "easy" functions, which are useful both for proving circuit lower bounds and for designing "meta-algorithms" (such as Circuit-SAT). For (de Morgan) formulas, such structural characterization is provided by the "shrinkage under random restrictions" results [Sub61, Hås98], strengthened to the "high-probability" version by [San10, IMZ12, KR13]. We give a new, simple proof of the "high-probability" version of the shrinkage result for (de Morgan) formulas, with improved parameters. We use this shrinkage result to get both compression and #SAT algorithms for (de Morgan) formulas of size about n^2. We also use this shrinkage result to get an alternative proof of the recent result by Komargodski and Raz [KR13] of the average-case lower bound against small (de Morgan) formulas.

Finally, we show that the existence of any non-trivial compression algorithm for a circuit class $\mathcal{C} \subseteq \mathsf{P/poly}$ would imply the circuit lower bound $\mathsf{NEXP} \not\subseteq \mathcal{C}$. This complements Williams's result [Wil10] that any non-trivial Circuit-SAT algorithm for a circuit class $\mathcal{C}$ would imply a superpolynomial lower bound against $\mathcal{C}$ for a language in NEXP.

4.1 Introduction

Circuit lower bounds (proved or assumed) have a number of algorithmic applications. The most notable examples are in cryptography, where a computationally hard problem is used to construct a secure cryptographic primitive [BM84, Yao82], and in derandomization of probabilistic polynomial-time algorithms, where a hard problem is used to construct a source of pseudorandom bits that can be used instead of truly random ones when simulating an efficient randomized algorithm [NW94]. In both cases, we in fact have an equivalence between the existence of appropriately hard computational problem and the existence of a corresponding algorithmic procedure (appropriate pseudorandom generator) [HILL99, NW94].

In both of the mentioned examples, a circuit lower bound is used in a "black-box" fashion: the knowledge that a lower bound holds is sufficient to derive algorithmic consequences (e.g., if some language in $\mathsf{DTIME}(2^{O(n)})$ requires circuit size $2^{\Omega(n)}$, then $\mathsf{BPP} = \mathsf{P}$ [IW97]). One would hope that looking inside the proofs (of the few circuit lower bounds that we actually have at present) may yield new algorithms (for the same computational model where we have the lower bounds).

This is indeed the case as witnessed by number of examples: a learning algorithm for AC^0-computable Boolean functions [LMN93], a Circuit-SAT algorithm for AC^0 circuits [IMP12, BIS12] (using Håstad's Switching Lemma, a main tool used in AC^0 lower bound proofs [Hås86]), a simple pseudorandom generator for AC^0 circuits [Bra10] (using [LMN93]), a Circuit-SAT algorithm for linear-size (de Morgan) formulas [San10, ST12], and a pseudorandom generator for small (de Morgan) formulas and branching programs [IMZ12] (using a generalization of the "shrinkage under random restrictions" result of [Sub61, Hås98]), to mention just a few.

Trying to understand the limitations of current circuit lower bound techniques, Razborov and Rudich [RR97] came up with the notion of a *natural property* that can be extracted from every lower bound proof known at the time. Loosely speaking, a natural property is

a deterministic polynomial-time algorithm that can distinguish the truth table of an easy Boolean function (computable by a small circuit from a given circuit class $\mathcal{C}$) from the truth table of a random Boolean function, when given the truth table of a function as input. They also argued that such an algorithm can be used to break strong pseudorandom generators computable in the circuit class $\mathcal{C}$; hence, if we assume sufficiently secure cryptography for a circuit class $\mathcal{C}$, then we must conclude that there is no natural property for the class $\mathcal{C}$. The latter is known as the "natural-proof barrier" to proving new circuit lower bounds.

Compression of Boolean functions. In this work, we focus on the "positive" part of the natural-property argument: known circuit lower bounds yield a natural property. One way to obtain such a natural property is to argue the existence of an efficient *compression algorithm* for easy functions from a given circuit class $\mathcal{C}$. Namely, given the truth table of n-variate Boolean function f from $\mathcal{C}$, we want to find some Boolean circuit (not necessarily of the type $\mathcal{C}$) computing f such that the size of the found circuit is less than $2^n/n$ (which is the trivial size achievable for any n-variate Boolean function)[1]. There are two natural parameters to minimize: the *size* of the found circuit and the *running time* of the compression algorithm. Since the algorithm is given the full truth table as input, we consider it efficient if it runs in time $2^{O(n)}$ (polynomial in its input size). Ideally, we would like to find a circuit as small as the promised size of the concise representation of a given function f. However, any non-trivial savings over the generic $2^n/n$ circuit size [Lup58] are interesting.[2]

Does every $\mathcal{C}$-circuit lower bound known today yield a compression algorithm for $\mathcal{C}$? The positive answer would strengthen the argument of [RR97] to show that every known lower bound proof yields a particular kind of natural property, *efficient compressibility*.

We hypothesize that the answer is 'Yes,' and make the first step in this direction by extracting a compression algorithm from the lower-bound proofs based on the method of *random restrictions*. These include the lower bounds for AC^0 circuits [FSS84, Yao85, Hås86], for de Morgan formulas [Sub61, And87, Hås98], for branching programs [Nec66], and for read-once branching programs (see, e.g., [ABCR99]).

---

[1] This is different than $\mathcal{C}$-circuit minimization considered by [AHM+08] where the task is to construct a small circuit of the type $\mathcal{C}$. Our setting is closer to that of computational learning theory (non-proper exact learning [Ang87]).

[2] The compression task as defined above can be viewed as *lossless* compression: we want the compressed image (circuit) to compute the given function exactly. One can also consider the notion of *lossy* compression where the task is to find a circuit that only approximates the given function. This is related to the concept of PAC learning [Val84]. The focus of the present work, however, will be on lossless compression algorithms.

Compression Theorem: *(1) Boolean n-variate functions computed by* AC^0 *circuits of size s and depth d are compressible in time* $\mathrm{poly}(2^n)$ *to circuits of size at most $2^{n-n/O(\log s)^{d-1}}$. (2) Boolean n-variate functions computed by de Morgan formulas of size at most $n^{2.49}$, by formulas over the complete basis of size at most $n^{1.99}$, or by branching programs of size at most $n^{1.99}$ are compressible in time* $\mathrm{poly}(2^n)$ *to circuits of size at most $2^{n-n^{\epsilon}}$, for some $\epsilon > 0$ (dependent on the size of the formula/branching program). (3) Boolean n-variate functions computed by read-once branching programs of size at most $2^{0.48 \cdot n}$ are compressible in time* $\mathrm{poly}(2^n)$ *to circuits of size at most $2^{0.99 \cdot n}$.*

Finding a succinct representation of a given object is an important natural problem studied in various settings under various names: e.g., data compression, circuit minimization, and computational learning. Designing efficient compression algorithms for "data" produced by small Boolean circuits of restricted type is an interesting task in its own right. In addition, such algorithmic focus helps us sharpen our understanding of the *structural properties* of easy Boolean functions, which may be exploited in both designing new *meta-algorithms*, algorithms that take Boolean functions as inputs (e.g., the full truth table as in the case of compression algorithms, or a small Boolean circuit computing the function, as in the case of Circuit-SAT algorithms), and proving stronger circuit lower bounds.

In this vein, we also have the following additional results.

4.1.1 Our Results

In addition to the Compression Theorem mentioned above, we have results on circuit lower bounds implied by compression algorithms. These are detailed next.

Circuit lower bounds from compression algorithms. There are a number of results showing that the existence of a meta-algorithm for a certain circuit class $\mathcal{C}$ implies super-polynomial lower bounds against that class for some function in (nondeterministic) exponential time [Kan82, HS82, NW94, IKW02, KI04, Agr05, FK06, Wil10]. In particular, the result by Williams [Wil10] essentially says that deciding the satisfiability of circuits from a class $\mathcal{C}$ in time slightly less than that of the trivial brute-force SAT-algorithm implies super-polynomial circuit lower bounds against $\mathcal{C}$ for a language in NEXP. Here we complement this result, by showing the following.

Compression implies circuit lower bounds: *Compressing Boolean functions from any subclass $\mathcal{C}$ of polynomial-size circuits to any circuit size less than $2^n/n$ implies superpolynomial lower bounds against the class $\mathcal{C}$ for a language in* NEXP*.*

Thus, both non-trivial SAT algorithms and non-trivial compression algorithms for a circuit class $\mathcal{C} \subseteq \mathsf{P/poly}$ imply superpolynomial lower bounds against that class. This suggests trying to get an alternative proof of Williams's lower bound $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$ [Wil11] via designing a compression algorithm for ACC^0 functions. Apart from getting an alternative proof, the hope is that such a compression algorithm would give us more insight into the structure of ACC^0 functions, which could lead to ACC^0 circuit lower bounds against a much more explicit Boolean function, say the one in NP or in P.

4.1.2 Our Proof Techniques

The circuit lower bounds proved by a method of random restrictions yield a nice structural characterization of the class of n-variate Boolean functions f computable by small circuits. Roughly, we get that the universe $\{0,1\}^n$ can be partitioned into "not too many" disjoint regions, such that the restriction of the original function f to "almost every" region is a "simple" function, where "simple" means of description size $O(n)$. This is reminiscent of the Set Cover problem: we want to cover all the 1s of the given function f using as few as possible subsets that correspond to the truth tables of "simple functions" of small description size. We show how to find such a collection of few simple functions, using a variant of the greedy heuristic for Set Cover.

For our compression algorithms, we use the "simplicity" of functions in the disjunction to argue that they have linear-size descriptions (as required in order to achieve $\mathrm{poly}(2^n)$ running time). For our #SAT algorithms, we use the "simplicity" of the functions to argue that there will be *few distinct* functions associated with the regions of the partition of $\{0,1\}^n$. Once we solve #SAT (using a brute-force algorithm) for all distinct subfunctions and store the results, we can solve #SAT for almost all regions by the table look-up, achieving a noticeable speed-up overall.

Our proof of the high-probability version of the shrinkage lemma for formulas follows the supermartingale approach of [KR13]: For a de Morgan formula F on n variables, we consider the sequence of random variables X_i, $1 \leq i \leq n$, where X_i corresponds to the size of the restricted and simplified subformula of F after i variables are set randomly. By

[Sub61], setting a single variable at random is expected to shrink the formula size (with the shrinkage exponent $3/2$). Thus, the sequence $\{X_i\}$ is a supermartingale. However, to apply standard concentration bounds (Azuma's inequality), one needs to show that the absolute value of $|X_i - X_{i-1}|$ is bounded. In our case, we have only one side of this bound, i.e., that $X_i - X_{i-1}$ is small. We show a variant of Azuma's inequality that holds in this case (for one-sided bounded random variables that take two possible values with equal probability), and apply this bound to complete the shrinkage analysis. This yields a simpler proof of the shrinkage result of [KR13] with the following differences: (1) our restrictions always choose deterministically which variable to restrict (as opposed to restrictions of [KR13] that define "heavy" and "light" variables, and either choose deterministically a heavy variable, if it exists, or randomly choose a light variable otherwise), (2) after setting $n-k$ variables, we get that all but at most 2^{-k} restricted formulas have shrunk in size (as opposed to $2^{-k^{1-o(1)}}$ in [KR13]). The fact that our restrictions are *deterministic* when choosing a variable to restrict leads to a *deterministic* #SAT algorithm for small (de Morgan) formulas. The fact that our error parameter is 2^{-k} leads to simplified analysis of Santhanam's #SAT algorithm for linear-size de Morgan formulas [San10].

Our proof of [KR13]'s average-case hardness result is more modular and simpler. In particular, we adapt Andreev's original lower bound argument [And87] to the case of not necessarily truly random restrictions (by using randomness extractors), and use the information-theoretic framework of Kolmogorov complexity to avoid unnecessary technicalities.

Finally, our proof of circuits lower bounds for NEXP from a compression algorithm for a circuit class $\mathcal{C} \subseteq P/\text{poly}$ is a generalization of the similar result from [IKW02], showing that the existence of a natural property (even without the "largeness" assumption) for P/poly implies $\text{NEXP} \not\subseteq \text{P}/\text{poly}$. Here we handle the case of any circuit class $\mathcal{C} \subseteq \text{P}/\text{poly}$. Since the existence on an efficient compression algorithm for a circuit class $\mathcal{C}$ implies a natural property for the same class, the required lower bound $\text{NEXP} \not\subseteq \mathcal{C}$ follows.

Independently, Williams also proves such a generalization of the result from [IKW02] (as part of his equivalence between proving $\mathcal{C}$-circuit lower bounds against NEXP and having polynomial-time computable properties useful against $\mathcal{C}$).

Other related work. Perhaps the earliest example of a compression algorithm for a general class of Boolean functions is due to Yablonski [Yab59], who observed that n-variate Boolean functions that "don't have too many distinct subfunctions" can be computed by a

circuit of size $\sigma \cdot 2^n/n$, for some $\sigma < 1$ (related to the number of distinct subfunctions). The complexity of circuit minimization was studied in [Mas79, KC00, AHM+08, Fel09]. In particular, [AHM+08, Fel09] show that finding an approximately *minimal*-size DNF for a given truth table of an n-variate Boolean function is NP-hard, for the approximation factor n^γ for some constant $0 < \gamma < 1$.

Concurrent independent work. As we were completing this work, we have found out from Ran Raz [private communication, March 2013] about his new paper with Komargodski and Tal [KRT13] that improves the average-case de Morgan formula lower bounds of [KR13] to handle formulas of size about n^3. In that paper, the authors prove a version of the high-probability shrinkage result for de Morgan formulas with Håstad's shrinkage exponent 2 (rather than Subbotovskaya's shrinkage exponent 1.5 used in [KR13]). Similarly to our work but independently, Komargodski et al. [KRT13] also adapt Andreev's method to the case of arbitrary (not necessarily completely random) restrictions by using appropriate randomness extractors.

4.1.3 Preliminaries

Circuits

Here we recall some basic definitions of circuit classes.

A *branching program* F on n input variables $x_1, \ldots, x_n$ is a directed acyclic graph with one source and two sinks (labeled 0 and 1), where each non-sink node is of out-degree 2 and is labeled by an input variable x_i, $1 \leq i \leq n$. The two outgoing edges of each non-terminal node are labeled by 0 and 1. The branching program computes by starting at the source node, and following the path in the graph using the edges corresponding to the values of the variables queried in the nodes. The program accepts if it reaches the sink labeled 1, and rejects otherwise. The size of a branching program F, denoted by $L(F)$, is the number of nodes in the underlying graph. A branching program is (syntactic) *read-once* if on every path no variable occurs more than once.

A *decision tree* is a branching program whose underlying graph is a tree; the size of a decision tree is the number of leaves.

A *restriction* ρ of the variables $x_1, \ldots, x_n$ is a an assignment of Boolean values to some subset of the variables; the assigned variables are called set, while the remaining variables

are called free. For a circuit (formula or branching program) F on input variables $x_1, \ldots, x_n$ and a restriction ρ, we define the restriction $F|_\rho$ as the circuit on the free variables of ρ, obtained from F after the set variables are "hard-wired" and the circuit is simplified.

A de Morgan formula can be simplified using the following *simplification rules*, which have been used in [Hås98, San10]. We denote by ψ an arbitrary subformula, and y a literal. The rules are: (1) If $0 \wedge \psi$ or $1 \vee \psi$ appears, then replace it by 0 or 1, respectively. (2) If $0 \vee \psi$ or $1 \wedge \psi$ appears, then replace it by ψ. (3) If $y \vee \psi$ appears, then replace all occurrences of y in ψ by 0 and $\overline{y}$ by 1; if $y \wedge \psi$ appears, then replace all occurrences of y in ψ by 1 and $\overline{y}$ by 0. We say a de Morgan formula is *simplified* if none of the above rules are applicable. Note that in a simplified formula, by the rule 3, if a leaf is labeled with x or $\overline{x}$, then its sibling subtree does not contain the variable x.

Given a (bounded fan-in) circuit of size s, we can describe it using $O(s \log s)$ bits (by specifying the gate type and at most two incoming gates for each of the s gates). The same bound is true also for formulas and branching programs of size s.

Extractors and codes

Let X be a distribution over $\{0,1\}^n$. The *min-entropy* of X is defined as $H_\infty(X) = \min_x \log(1/\mathbf{Pr}[X = x])$. We say two distributions X and Y over $\{0,1\}^n$ are ϵ-*close* if for any subset $A \subseteq \{0,1\}^n$, it holds that $|\mathbf{Pr}[X \in A] - \mathbf{Pr}[Y \in A]| \leq \epsilon$.

An *oblivious* (n,k)-*bit-fixing source* is a distribution X over $\{0,1\}^n$, where there is a subset $S \subseteq [n]$ of size k such that $X_{[n]\setminus S}$ is fixed, while X_S is uniformly distributed over $\{0,1\}^{|S|}$. A *seedless zero-error disperser* is a function $D\colon \{0,1\}^n \to \{0,1\}^m$ such that for any distribution X over $\{0,1\}^n$ with min-entropy at least k, the support of $D(X)$ is $\{0,1\}^m$. A *seedless* (k,ϵ)-*extractor* is a function $E\colon \{0,1\}^n \to \{0,1\}^m$ such that for any distribution X over $\{0,1\}^n$ with min-entropy at least k, $E(X)$ is ϵ-close to the uniform distribution over $\{0,1\}^m$.

A *binary* (n,k,d)-*code* is a function $C : \{0,1\}^k \to \{0,1\}^n$ (mapping k-bit messages to n-bit codewords) such that any two codewords are at least the Hamming distance d apart; the *relative minimum distance* of C is d/n. For $0 \leq \rho \leq 1$ and $L \geq 1$, we say a code C is (ρ, L)-*list-decodable* if for any $y \in \{0,1\}^n$, there are at most L codewords in C within the Hamming distance at most ρn from y. The Johnson bound (see, e.g., [AB09]) says that, for any $\delta \geq \sqrt{\epsilon}$, an $(n, k, (1/2 - \epsilon)n)$-code is $(1/2 - \delta, 1/(2\delta^2))$-list-decodable.

Kolmogorov complexity

The *Kolmogorov complexity* of a given n-bit string x, denoted by $K(x)$, is the length of a shortest string $\langle M \rangle w$, where $\langle M \rangle$ is a description of a Turing machine M, and w is a binary string such that M on input w produces x as an output. A simple counting argument shows that, for every n, there exists an n-bit string x with $K(x) \geq n$, and, more generally, for any $0 < \alpha < 1$, we have that $K(x) \geq \alpha n$ for all but at most $2^{-(1-\alpha)n}$ fraction of n-bit strings x.

4.2 Compression from Restriction-Based Circuit Lower Bounds

Here we prove the Compression Theorem stated in the Introduction.

4.2.1 Compression of DNFs, using the Greedy Set Cover Heuristic

As a warm-up, consider the case of DNFs. It is well-known that DNFs of almost minimum size can be computed from the truth table of $f : \{0,1\}^n \to \{0,1\}$ using a greedy Set Cover heuristic [Joh74, Lov75, Chv79]. We recall this heuristic next.

Let U be a universe, and let $S_1, \ldots, S_t \subseteq U$ be subsets. Suppose U can be covered by ℓ of the subsets. Then the following algorithm will find an approximately minimal set cover.

> Repeat the following, until all of U is covered: find a subset S_i that covers at least $1/\ell$ fraction of points in U which were not covered before, and add S_i to the set cover.

For the analysis, observe that since ℓ subsets cover U, they also cover every subset of U. Hence, in each iteration of the algorithm, there exists a subset that covers at least $1/\ell$ fraction of the not-yet-covered points. After each iteration, the size of the set of points that are not covered reduces by the factor $(1 - 1/\ell)$. Thus, after t iterations, the number of points not yet covered is at most $|U| \cdot (1 - 1/\ell)^t \leq |U| \cdot e^{-t/\ell}$, which is less than 1 for $t = O(\ell \cdot \ln |U|)$. Hence, this algorithm finds a set cover that is at most the factor $O(\ln |U|)$ larger than the minimal set cover.

It is easy to adapt the described algorithm to find approximately minimal DNFs. Let $f : \{0,1\}^n \to \{0,1\}$ be given by its truth table. Suppose that there exists a DNF computing f such that the DNF consists of ℓ terms (conjunctions). With each term a on n variables,

we associate the set $S_a = a^{-1}(1)$ of points of $\{0,1\}^n$ where it evaluates to 1. We enumerate over all possible terms a on n variables, and keep only those sets S_a where $S_a \subseteq f^{-1}(1)$ (i.e., S_a does not cover any zero of f); note that all ℓ terms of the minimal DNF for f will be kept. Next we run the greedy set cover algorithm on the universe $U = f^{-1}(1)$ and the collection of sets S_a chosen above. By the analysis above, we get $O(\ell \cdot \log |U|)$ terms such that their disjunction computes f. That is, we find a DNF for f of size at most $O(n)$ factor larger than that of the minimal DNF for f.

The running time of the described algorithm is polynomial in 2^n and the number of sets S_a. The latter is the number of all possible terms on n variables, which is at most 2^{2n} (we can use an n-bit string to describe the characteristic functions of a subset of n variables, and another n-bit string to describe the signs of the chosen variables). Thus, the overall running time is $\mathrm{poly}(2^n)$.

4.2.2 Compression of AC^0 Functions via DNFs

The known lower bounds for AC^0 circuits are based on the fact that almost all random restrictions simplify a small AC^0 circuit to a function that depends on fewer than the remaining unrestricted variables. Intuitively, this means that there is a partitioning of the Boolean cube $\{0,1\}^n$ into not too many disjoint regions such that the original AC^0 circuit is constant over each region. This intuition can be made precise using the Switching Lemma [Hås86, Raz93, Bea94, IMP12], yielding the following structural result saying that each small AC^0 circuit has an equivalent representation as a DNF with not too many terms.

Lemma 4.1 ([IMP12])**.** *Every depth d Boolean circuit C with s gates on n inputs has an equivalent DNF with at most $\mathrm{poly}(n) \cdot s \cdot 2^{n(1-\mu)}$ terms, where $\mu \geq 1/O(\log(s/n) + d \log d)^{d-1}$.*

Using this structural characterization and the greedy algorithm for Set Cover considered earlier, we immediately get the following.

Theorem 4.2. *There is a deterministic $\mathrm{poly}(2^n)$-time algorithm A satisfying the following. Let $f : \{0,1\}^n \to \{0,1\}$ be any Boolean function computable by an AC^0 circuit of depth d and size $s = s(n)$. Given the truth table of f as well as the parameters d and s, algorithm A produces a DNF for f with at most $\mathrm{poly}(n) \cdot s \cdot 2^{n(1-\mu)}$ terms, where $\mu \geq 1/O(\log s)^{d-1}$.*

Note the described algorithm achieves nontrivial compression for depth-d AC^0 circuits of size up to $2^{n^{1/(d-1)}}$, the size for which we know lower bounds against AC^0 for explicit

functions.

4.2.3 Formulas and Branching Programs

The known lower bounds for (de Morgan) formulas are also proved using the method of random restrictions. One of the earliest results here is by Subbotovskaya [Sub61] who argued that the size of a de Morgan formula shrinks in expectation when hit by a random restriction; this result was subsequently tightened by Håstad [Hås98]. However, these results are not strong enough to provide a kind of structure of easy functions that would be useful for compression. By analogy with the case of AC^0, we would like to say something like "for every small de Morgan formula, there is a partition of the Boolean cube into not too many regions such that the original formula is constant on each region". In particular, we need a "high probability" version of the classical shrinkage results of [Sub61, Hås98].

Recently, there have been several such shrinkage results proved for different purposes. Santhanam [San10] implicitly proved such a result for linear-size de Morgan formulas and used it to obtain a deterministic SAT algorithm for such formulas that runs in time better than that of the "brute-force" algorithm. Impagliazzo et al. [IMZ12] proved a version of shrinkage result with respect to certain *pseudorandom* restrictions, in order to construct a non-trivial pseudorandom generator for small de Morgan formulas. Komargodski and Raz [KR13] proved a shrinkage result for certain random restrictions (different from the ones in [San10]), and used it to get a strong average-case lower bound against small de Morgan formulas.

We will give an improved and simplified proof of the shrinkage result due to [San10, KR13]. We use the same notion of random restrictions as in [San10], which will allow us later to get a "better than brute force" *deterministic* SAT algorithms for *super-quadratic*-size de Morgan formulas. We get a smaller error probability than that of [KR13], which allows us to analyze Santhanam's SAT algorithm for linear-size de Morgan formulas as an easy corollary. Finally, we get a clean and simple proof which avoids some of the *ad hoc* technicalities from [KR13].

Structure of functions computable by small formulas

First, we state our version of the shrinkage result. Let F be a de Morgan formula on n variables. As in [San10], we consider *adaptive* restrictions that proceed in i rounds, for

$0 \leq i \leq n$, and in each round set uniformly at random the most frequent variable in the current formula, and simplify the resulting new formula (using the standard simplification rules). Note that these restrictions are not completely random: the next variable to be restricted is chosen completely deterministically (as the most frequent one), but the value assigned to this variable is then chosen uniformly at random to be either 0 or 1.

For a given de Morgan formula F, define $F_0 = F$. For $1 \leq i \leq n$, we define F_i to be the random formula obtained from F_{i-1} by uniformly at random assigning the most frequent variable of F_{i-1}, and simplifying the result. Note that F_i is a formula on $n - i$ remaining (unrestricted) variables.

Lemma 4.3 (Shrinkage Lemma)**.** *Let F be any given (de Morgan) formula or a branching program on n variables. For any $k \geq 4$, we have*

$$\mathbf{Pr}\left[L(F_{n-k}) \geq 2 \cdot L(F) \cdot \left(\frac{k}{n}\right)^{\Gamma}\right] < 2^{-k},$$

where $\Gamma = 3/2$ for the case of de Morgan formulas, and $\Gamma = 1$ for the case of formulas over the complete basis and for the case of branching programs.

We gave the proof of Shrinkage Lemma in Section 3.2. Now we apply this lemma to obtain the following structural characterization of small formulas and branching programs, which will be useful for compression.

Corollary 4.4. *Let $F(x_1, \ldots, x_n)$ be any formula (branching program) of size $O(n^d)$, where the constant d is such that $d < 2.5$ for de Morgan formulas, and $d < 2$ for formulas over the complete basis and for branching programs. There exist constants $0 < \delta, \gamma < 1$ (dependent on d) such that for $k = \lceil n^{\delta} \rceil$ the following holds. The Boolean function computed by F is computable by a decision tree of depth $n - k$ whose leaves are labeled by the restrictions of F (determined by the path leading to the leaf) such that all but 2^{-k} fraction of the leaf labels are formulas (branching programs) on k variables of size less than n^{γ} .*

Proof. We consider the case of de Morgan formulas only; the case of formulas over the complete basis or branching programs can be argued analogously. Let $d = 2.5 - \nu$, for some constant $\nu > 0$. Set $\delta := \nu/3$, and $\gamma := 1 - \nu/2$. By Lemma 4.3 applied to F, we get that for all but 2^{-k} fraction of the branches of the restriction decision tree of depth $n - k$, the restricted formula has size less than $O(n^d/n^{1.5(1-\delta)}) = O(n^{1-\nu/2})$. □

Generalized greedy Set-Cover heuristic

The Shrinkage Lemma allows us to decompose the Boolean cube into not too many regions so that, over almost all regions, the original formula simplifies to a formula of sublinear size. This falls short of our original hope to get a constant function over most regions. In fact, the latter cannot be achieved since a de Morgan formula of size $O(n^2)$ computes the parity of n bits, and the parity function doesn't simplify to a constant unless all of its variables are fixed.

Fortunately, we can still use a version of the greedy Set Cover heuristic to compress de Morgan formulas of size about $n^{2.5}$. The reason is that a similar algorithm works also for a function $f : \{0,1\}^n \to \{0,1\}$ computed by a circuit of the form $\vee_{i=1}^{\ell+1} C_i$, for $\ell \leq 2^n$, where all but one circuit are small, while the remaining circuit accepts few inputs. More precisely, we have the following.

Theorem 4.5. *There is a deterministic* $\mathrm{poly}(2^n)$*-time algorithm* A *satisfying the following. Let* $f : \{0,1\}^n \to \{0,1\}$ *be any function computable by a circuit* $\vee_{i=1}^{\ell+1} C_i$*, for* $1 \leq \ell \leq 2^n$*, where the circuits* $C_1, \ldots, C_\ell$ *have both circuit size and description size at most* cn *for a constant* $c > 0$*, while the last circuit* $C_{\ell+1}$ *evaluates to* 1 *on at most fraction* α *of points in* $\{0,1\}^n$*, for some* $0 \leq \alpha < 1$*.*

Given the truth table of f *and the parameters* ℓ*,* c*, and* α*, algorithm* A *finds a circuit for* f *of the form* $\vee_{i=1}^{m} D_i$*, where* $m = O(n \cdot \ell)$*, the circuits* $D_1, \ldots, D_{m-1}$ *are of size* $O(n)$ *each, and the circuit* D_m *is a DNF with* $O(\alpha 2^n)$ *terms. Hence the overall size of the found circuit is* $O(\ell n^2 + \alpha n 2^n)$*.*

Proof. Let $U = f^{-1}(1)$, and let $\beta = |U|/2^n$. If $\beta \leq 2\alpha$, then our algorithm A outputs the circuit which is a DNF with $\beta 2^n$ terms, where each term evaluates to 1 on a single point in U, and is 0 everywhere else. Note that the size of this circuit is $O(\alpha n 2^n)$, as required.

If $\beta > 2\alpha$, then algorithm A does the following.

> Enumerate[3] all linear-size circuits C of description size at most cn, keeping only those C where $C^{-1}(1) \subseteq f^{-1}(1)$. Call the kept circuits *legal*. Let $\mathcal{S} = \emptyset$.
>
> Repeat the following until the number of not-yet-covered points of U becomes at most $2\alpha 2^n$: find a legal circuit C such that the set $C^{-1}(1)$ covers at least

[3] Here we assume the correspondence between circuits and their descriptions is efficiently computable and is known.

$1/(2\ell)$ fraction of not-yet-covered points in U, and add C to the set $\mathcal{S}$.

Once the number of non-covered points in U becomes at most $2\alpha 2^n$, construct a DNF D that evaluates to 1 on each non-covered point, and is 0 everywhere else. Output the disjunction of D and the circuits in $\mathcal{S}$.

For the analysis, let $W = C_{\ell+1}^{-1}(1)$, and let let $V = U \setminus W$. We claim that at each iteration of the algorithm before the last iteration, the set of not-yet-covered points in V is at least as big as the set of not-yet-covered points in W. Indeed, otherwise the total number of not-yet-covered points at that iteration is at most $2 \cdot |W| \leq 2\alpha 2^n$, making this the last iteration of the algorithm.

Next observe that at each iteration before the last one, the set of not-yet-covered points in V is non-empty, and is covered by ℓ legal circuits. Hence, there is a legal circuit that covers at least $1/\ell$ fraction of non-covered points in V, which, by the earlier remark, constitutes at least $1/(2\ell)$ fraction of all non-covered points of U. Thus our algorithm will always find a required legal circuit C. It follows that after each iteration, the size of not-yet-covered points in U decreases by the factor $(1 - 1/(2\ell))$, and hence the total number of iterations is $t = O(\ell \cdot \log |U|) = O(\ell \cdot n)$.

Thus, after at most t iterations, at most $2\alpha 2^n$ points of U are still not covered. We denote the t found circuits $D_1, \ldots, D_t$, and let D_{t+1} be the DNF with at most $2\alpha 2^n$ terms which evaluates to 1 on the non-covered points of U, and is 0 everywhere else. Note that the circuit size of D_{t+1} is $O(\alpha n 2^n)$, while all D_i's, for $1 \leq i \leq t$, are of circuit size $O(n)$ by construction. Also note that the overall running time of the described algorithm is $\mathsf{poly}(2^n, t) = \mathsf{poly}(2^n)$. The theorem follows. □

Using this generalized algorithm, we get the following.

Theorem 4.6. *There is an efficient compression algorithm that, given the truth table of a formula (branching program) F on n variables of size $L(F) \leq n^d$, the algorithm produces an equivalent Boolean circuit of size at most 2^{n-n^ϵ}, for some constant $0 < \epsilon < 1$ (dependent on d), where the constant d is such that*

- $d < 2.5$ *for de Morgan formulas, and*
- $d < 2$ *for formulas over the complete basis and for branching programs.*

Proof. Let F be a de Morgan formula, a complete-basis formula, or a branching program of the size stated in the theorem. By Corollary 4.4, this F can be computed by a decision tree of depth $m := n - n^\delta$ such that all but at most $\alpha := 2^{-n^\delta}$ fraction of the leaves correspond to restricted subformulas of F of size n^γ on $k := n^\delta$ variables, for some constants $0 < \delta, \gamma < 1$ dependent on d.

Each leaf of the decision tree corresponds to a restriction of some subset of m input variables. Let us associate with each leaf i, $1 \leq i \leq 2^m$, of the decision tree, the conjunction c_i of m literals that defines the corresponding restriction. Also let F_i, for $1 \leq i \leq 2^m$, denote the restriction of the original F corresponding to the restriction given by c_i. We get that $F \equiv \vee_{i=1}^{2^m}(c_i \wedge F_i)$.

We know that all but $b := \alpha \cdot 2^m$ of formulas F_i are sublinear-size n^γ. Let us assume, without loss of generality, that all the first $\ell := 2^m - b$ formulas F_i are small. Define the circuits $C_i := (c_i \wedge F_i)$, for $1 \leq i \leq \ell$, and $C_{\ell+1} := \vee_{i=\ell+1}^{2^m}(c_i \wedge F_i)$.

Observe that the circuit $C_{\ell+1}$ can evaluate to 1 on at most $b \cdot 2^k = \alpha \cdot 2^n$ inputs from $\{0,1\}^n$ (since the decision tree of depth m partitions the set $\{0,1\}^n$ into 2^m disjoint subsets of size 2^k each, and $C_{\ell+1}$ corresponds to b such subsets). Each circuit C_i, for $1 \leq i \leq \ell$, is of size at most $O(m + n^\gamma) \leq O(n)$. We also claim that each such circuit can be described by a string of $O(n)$ bits. Indeed, we can specify the conjunction c_i using $2n$ bits (n bits to describe the subset of variables in the conjunction, and another n bits to specify the signs of the variables), and we can specify the formula (branching program) F_i of size n^γ by at most $O(n^\gamma \log n) \leq O(n)$ bits in the standard way.

Thus we get that $F \equiv \vee_{i=1}^{\ell+1} C_i$ satisfies the assumption of Theorem 4.5. Running the greedy algorithm of Theorem 4.5, we get a circuit for F of total size at most $O(\ell n^2 + \alpha n 2^n) \leq \mathsf{poly}(n) \cdot 2^{n-n^\delta}$. □

4.2.4 Read-Once Branching Programs

Read-once branching programs are quite well-understood, with strongly exponential lower bounds known. A property that makes a function f hard for read-once branching programs is that of being m-mixed: for every set S of variables such that $|S| = m$ every two distinct assignments a and b to variables in S give rise to different functions $f_a \not\equiv f_b$. Any read-once branching program computing an m-mixed Boolean function must have at least $2^m - 1$ nodes [SZ96].

On the other hand, a function that has a small read-once branching program cannot be m-mixed for large m. Intuitively, such a function can be represented by a decision tree of depth m, whose leaves are labeled by subfunctions g (in the remaining $n - m$ variables) so that many of the leaves share the same subfunction. If a program has size s, then the number of distinct such subfunctions is at most s. Thus, f can be computed as an OR of at most s subformulas, where each subformula encodes the conjunction of a particular subfunction g and the DNF describing all branches leading to this subfunction g. The fact that f can be represented as an OR of few simple formulas allows us to use the greedy SetCover heuristic to compress such f. We provide the details next.

It is convenient for us to use the following canonical form of a read-once branching program. We call a program *full* if, for every node v of the program, all paths leading from the start node to v query the same set of variables (not necessarily in the same order).

Lemma 4.7. *Every read-once branching program F of size s on n inputs has an equivalent full read-once branching program F' of size $s' \leq 3n \cdot s$.*

Proof. Given F, construct F' inductively as follows. Consider nodes of F in the topological order from the start node. The start node obviously satisfies the fullness property. For every node v of F with distinct predecessor nodes $u_1, \ldots, u_t$, for $t \geq 2$, let X_i denote the set of variables queried by the paths from start to u_i; note that, by the inductive hypothesis, all paths leading to u_i query the same set X_i of variables. Let $X = \cup_{i=1}^{t} X_i$. For every $i \in \{1, \ldots, t\}$, let $\Delta_i = X \setminus X_i$ be the set of "missing" variables. If $\Delta_i \neq \emptyset$, replace the edge (u_i, v) by a multi-path $u_i, w_1, w_2, \ldots, w_r, v$, for $r = |\Delta_i|$, where w_j's are new nodes labeled by the "missing" variables from Δ_i (in any fixed order), with the edge (u_i, w_1) labeled as the edge (u_i, v), and each w_j has two edges to its successor node on the path, labeled by 0 and by 1, respectively.

Since our original program is read-once, no variable from the set X for a node v can occur after v. Thus, adding the queries to the "missing" variables for every predecessor of v preserves the property of being read-once, and preserves the functionality of the branching program. It also makes the node v and all of its predecessors satisfy the fullness property. Hence, after considering all nodes v, we obtain a required full read-once branching program F' equivalent to F. The size of F' is at most $s + 2sn$ since we add at most n dummy nodes for each of at most $2s$ edges of F. □

Theorem 4.8. *There is a deterministic* $\mathrm{poly}(2^n)$*-time algorithm* A *satisfying the following. Let* $f : \{0,1\}^n \to \{0,1\}$ *be any Boolean function computable by a read-once branching program of size* s*. Given the truth table of* f*, algorithm* A *produces a formula for* f *of size at most* $O(sn^3 \cdot 2^{n/2})$*.*

Proof. By Lemma 4.7, f is computable by a full read-once branching program F of size $s' = 3sn$. For $0 \leq k \leq n$ to be chosen later, consider the set B of all nodes at distance $n-k$ from the start node. Clearly, there are at most s' such nodes. For every such node v, let X_v be the set of $n-k$ variables queried on every path from the start to v. Let Y_v be the remaining k variables. Associate with v the function h_v in the variables X_v computed by the branching subprogram with v as the new accepting terminal node (and the same start node), and the function g_v in the variables Y_v computed by the branching subprogram with v as the new start node (and the same terminal nodes). We may assume that the functions g_v are distinct for distinct nodes in B; otherwise, we merge all nodes with the same g_v (on the same subset of k variables) into a single node. We have

$$f \equiv \vee_{v \in B}(h_v \wedge g_v). \tag{4.1}$$

Consider any $v \in B$. Let ρ be a restriction of the variables X_v corresponding to some path from the start to v. We have $g_v = f|_\rho$, and h_v is the disjunction of all restrictions ρ' of the variables X_v such that $f|_{\rho'} = g_v = f|_\rho$. Thus, to describe any disjunct in the representation of f given by Eq. (4.1), it suffices to specify a restriction of some subset of $n-k$ variables of f; this can be described using $O(n)$ bits.

We now run the greedy Set-Cover heuristic to find at most $O(s'n)$ functions, each describable by a restriction of some $n-k$ variables as explained above, whose disjunction equals f. For each restriction ρ specifying one of these functions, the corresponding function can be computed as an AND of a DNF of size 2^k (for the function $f|_\rho$ on k variables) and a DNF of size 2^{n-k} (for all restrictions ρ' on $n-k$ variables that yield $f|_{\rho'} = f|_\rho$). The overall circuit size of each of these $O(s'n)$ functions is then $O(n(2^k + 2^{n-k}))$, and the overall size of the circuit computing f is $O(s'n^2(2^k + 2^{n-k}))$, which is at most $O(sn^3 \cdot 2^{n/2})$, if we set $k = n/2$. The running time of the compression algorithm is $\mathrm{poly}(2^n)$ since we only need to enumerate all $O(n)$-size descriptions. □

4.3 Circuit Lower Bounds from Compression

Since in-compressibility of Boolean functions is a special case of a natural property in the sense of [RR97], the existence of compression algorithms for a circuit class $\mathcal{C}$ implies that there is no strong PRG in $\mathcal{C}$. Here we argue that such compression algorithms would also yield circuit lower bounds against $\mathcal{C}$ for a language in NEXP.

4.3.1 Arbitrary Subclass of Polynomial-Size Circuits

It was shown in [IKW02] that the existence of a natural property for $\mathrm{P/poly}$ would imply that $\mathrm{NEXP} \not\subseteq \mathrm{P/poly}$. In particular, the same conclusion follows if we assume the existence of a compression algorithm for $\mathrm{P/poly}$-computable Boolean functions. Here we generalize this result by proving that the same is true if we replace $\mathrm{P/poly}$ with any subclass $\mathcal{C} \subseteq \mathrm{P/poly}$.

Theorem 4.9. *Let $\mathcal{C} \subseteq \mathrm{P/poly}$ be any circuit class. Suppose that for every $c \in \mathbb{N}$ there is a deterministic polynomial-time algorithm that compresses a given truth table of an n-variate Boolean function $f \in \mathcal{C}[n^c]$ to a circuit of size less than $2^n/n$. Then $\mathrm{NEXP} \not\subseteq \mathcal{C}$.*

Proof. Suppose, for the sake of contradiction, that $\mathrm{NEXP} \subseteq \mathcal{C} \subseteq \mathrm{P/poly}$. The following is a refinement of a result in [IKW02] who showed the result for the case $\mathcal{C} = \mathrm{P/poly}$. We show how to strengthen it to any subclass $\mathcal{C} \subseteq \mathrm{P/poly}$.

Claim 4.10. *If $\mathrm{NEXP} \subseteq \mathcal{C}$, then for every $L \in \mathrm{NEXP}$ there is a $c \in \mathbb{N}$ such that, for all sufficiently large n, every n-bit string $x \in L$ has a witness computable by a $\mathcal{C}$-circuit of size n^c.*

Proof. By [IKW02], the assumption $\mathrm{NEXP} \subseteq \mathrm{P/poly}$ implies that, for every language $L \in \mathrm{NEXP}$, there exists a constant $c_L \in \mathbb{N}$ such that every sufficiently large input $x \in L$ has a NEXP-witness that is the truth table of some Boolean function of circuit complexity n^{c_L}. For every $L \in \mathrm{NTIME}(2^{n^e})$, define a new language $L' \in \mathrm{EXP}$ as follows: on inputs x, y, where $|x| = n$ and $|y| = n^e$, search through the circuits of size n^{c_L} until find an NEXP-witness for $x \in L$. If no such witness is found, then output 0. Otherwise, output the yth bit of the found witness (which is the truth table of a n^{c_L}-size circuit). We get that for every $x \in L$, a string y is such that $(x, y) \in L'$ iff the yth bit of the lexicographically first witness for x (as found by the algorithm enumerating all n^{c_L} size circuits) is 1. Since $\mathrm{EXP} \subseteq \mathcal{C}$, we get that $L' \in \mathcal{C}$.

So, every $x \in L$ has a witness that is the truth table of Boolean function computable by a polynomial-size $\mathcal{C}$-circuit. □

Consider now a universal language L for NE, with $L \in \mathsf{NTIME}(2^{n^2})$. For $\mathsf{NTIME}(2^{cn})$ for every $c \in \mathbb{N}$, the witness size for inputs of size n is bounded by $2^{cn} \leq 2^{n^2}$ for large enough n. We think of witnesses for NE languages (on inputs of size n) as the truth tables of m-variate Boolean functions for $m = n^2$: such a string of length 2^m is a witness iff its prefix of appropriate length is a witness. By Claim 4.10 above, we get that there is a constant $c_0 \in \mathbb{N}$ such that yes-instances x, $|x| = n$, of every language in NE have witnesses that are truth tables of $m = n^2$-variate Boolean functions computable in $\mathcal{C}[m^{c_0}]$.

Suppose we have a deterministic $\mathrm{poly}(2^n)$-time compression algorithm for n-variate Boolean functions in $\mathcal{C}[n^{2c_0}]$. Consider the following NE algorithm:

> On input x of size n, nondeterministically guess a binary string of length 2^n. Run the compression algorithm on the guessed string. Accept iff the compression algorithm didn't produce a circuit of size less than $2^n/n$ for this string.

Observe that the described algorithm accepts every input x since there are incompressible strings of every length 2^n. Its running time is $\mathrm{poly}(2^n)$ dependent on the running time of the assumed compression algorithm. Note that every witness for an input x is a string that our compression algorithm fails to compress, which means that the witness is the truth table of an n-variate Boolean function that requires $\mathcal{C}$-circuits of size greater than n^{2c_0}. If we think of this 2^n-bit witness as the prefix of a 2^{n^2}-bit truth table of an $m = n^2$-variate Boolean function, we conclude that the latter m-variate Boolean function requires $\mathcal{C}$ circuits of size greater than m^{c_0}. But this contradicts the fact we established earlier that every NE language must have $\mathcal{C}[m^{c_0}]$ computable witnesses. □

It is easy to get an analogue of Theorem 4.9 also for deterministic *lossy* compression algorithms.

Remark 4.11. If we could show that ACC^0-computable functions are compressible, we would get an alternative proof of Williams's lower bound $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$ [Wil11]. Interestingly, while such a compression algorithm would yield a natural property for ACC^0, the overall lower bound proof would still use non-natural arguments and non-relativizing arguments that come from the use of [IKW02] in the proof of Claim 4.10.

4.3.2 Other Function Classes That Are Hard to Compress

Large AC^0 circuits. Compressing functions computable by "large" AC^0 circuits (of size 2^{n^ϵ} with $\epsilon \gg 1/d$, where d is the depth of the circuit) is difficult since every function computable by a polynomial-size NC^1 circuit has an equivalent AC^0 circuit of size 2^{n^ϵ} (and some depth d dependent on ϵ). The existence of a compression algorithm for such large AC^0 circuits would imply a *natural property* in the sense of [RR97] useful against NC^1. The latter implies that no strong enough PRG can be computed by NC^1 circuits [RR97, AHM+08]. Also, using Theorem 4.9, we get that such compression would imply that $\mathsf{NEXP} \not\subset \mathsf{NC}^1$.

Theorem 4.12. *For every $\epsilon > 0$ there is a $d \in \mathbb{N}$ such that the following holds. If there is a deterministic polynomial-time algorithm that compresses a given truth table of an n-variate Boolean function $f \in \mathsf{AC}^0_d[2^{n^\epsilon}]$ to a circuit of size less than $2^n/n$, then $\mathsf{NEXP} \not\subset \mathsf{NC}^1$.*

Monotone functions. Every monotone Boolean function on n variables can be computed by a (monotone) circuit of size $O(2^n/n^{1.5})$ [Pip77, Red79]. We argue that compressing polynomial-size monotone functions is as hard as compressing arbitrary functions in $\mathsf{P/poly}$.

Theorem 4.13. *If there is an efficient algorithm that compresses a given truth table of an m-variate monotone Boolean function of monotone circuit size $\mathrm{poly}(m)$ to a (not necessarily monotone) circuit of size at most $2^m/m^{1.51}$, then there is an efficient algorithm for compressing arbitrary n-variate $\mathsf{P/poly}$-computable Boolean functions to circuits of size less than $2^n/n$.*

Proof sketch. The idea is to use the well-known connection between non-monotone functions and monotone slice functions [Ber82]. We use an optimal embedding of an arbitrary n-variate Boolean function f into the middle slice of a monotone slice function g on m variables for $m = n+(\log n)/2+\Theta(1)$ due to [KKM12]. Given a truth table of f, we can efficiently construct the truth table of this monotone function g. The mapping between n-bit inputs of f and the corresponding m-bit inputs of g (of Hamming weight $m/2$) is computable and invertible in time $\mathrm{poly}(m) = \mathrm{poly}(n)$. Hence, a circuit for g of size at most $2^m/m^{1.51}$ yields a circuit for f of size at most $O((2^n/n^{1.01}) + \mathrm{poly}(n))$, which is less than $2^n/n$ for large enough n. Appealing to Theorem 4.9 concludes the proof. □

Thus, a compression algorithm for monotone functions of polynomial monotone-circuit complexity would yield a natural property for the class $\mathsf{P/poly}$, as well as a proof that $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

4.4 Open Questions

We have shown efficient compressibility of functions computable by small circuits from several classes $\mathcal{C}$ where known lower bounds are proved using the method of random restrictions. Can we extend this to other circuit classes with known lower bounds, e.g., constant-depth circuits with prime-modular gates for which the polynomial-approximation method was used [Raz87, Smo87]? Can we compress functions computable by ACC^0 circuits? More generally, can we argue that all known circuit lower bound proofs yield compression algorithms for the corresponding circuit classes?

The compressed circuit sizes for our compression algorithms are barely less than exponential. Is it possible to achieve better compression for the circuit classes considered?

We have used the ideas of our compression algorithm for small formulas to get also a #SAT-algorithm for small formulas. Is there a general connection between compression and SAT algorithms?

Using the independent work by Komargodski et al. [KRT13] on the "high-probability version of shrinkage" for de Morgan formulas, we can get compression and #SAT algorithms for de Morgan formulas of size almost n^3. However, unlike our #SAT-algorithm (for $n^{2.5}$-size de Morgan formulas), the #SAT-algorithm resulting from [KRT13] is only *randomized* (due to the notion of random restrictions used in [KRT13]). It is an interesting open question to get a *deterministic* such algorithm for n^3-size de Morgan formulas. (A similar problem is also open for AC^0-SAT algorithms, where there is a quantitative gap between the AC^0 circuit size that can be handled by the randomized algorithm of [IMP12] and the deterministic algorithm of [BIS12].) Very recently, building on the results in the present paper, [CKS13] make a step in that direction: they give a deterministic #SAT-algorithm for de Morgan formulas of size $n^{2.63}$, with the running time $2^{n-n^{\Omega(1)}}$.

Finally, the focus of the present work has been on lossless compression. For small AC^0 circuits and small AC^0 circuits with few threshold gates, one can get nontrivial *lossy* compression using the Fourier transform [LMN93, GS10]. What about lossy compression for other circuit classes?

For example, for polynomial-size AC^0 circuits with parity-gates, we know by the results of Razborov and Smolensky [Raz87, Smo87] that every such function can be approximated by a $(\mathrm{poly}\log n)$-degree polynomial over $GF(2)$ to within error $1/n$. This polynomial is a binary Reed-Muller codeword of order $\mathrm{poly}\log n$ that disagrees with our received word (the given truth table of a function) in at most $1/n$ fraction of positions. The problem of lossy compression leads to the following natural question on decoding: Given a received word x of size 2^n such that there is a Reed-Muller codeword (of order $\mathrm{poly}\log n$) within the Hamming ball of relative radius $1/n$ around x, find in time $\mathrm{poly}(2^n)$ *some* codeword that is at most $1/n$ away from x. Note that this is different from the usual list-decoding question: here the number of codewords within this Hamming ball can be huge, and so we don't ask to find all of them, but rather any single one. (The only result in this direction that we are aware of is [TW11] for the case of binary Reed-Muller codes of order 2.)

Chapter 5

Improved #SAT Algorithm for De Morgan Formulas

We give a deterministic #SAT algorithm for de Morgan formulas of size up to $n^{2.63}$, which runs in time $2^{n-n^{\Omega(1)}}$. This improves upon the deterministic #SAT algorithm of [CKK$^+$13b], which has similar running time but works only for formulas of size less than $n^{2.5}$.

Our new algorithm is based on the shrinkage of de Morgan formulas under random restrictions, shown by Paterson and Zwick [PZ93]. We prove a *concentrated* and *constructive* version of their shrinkage result. Namely, we give a deterministic polynomial-time algorithm that selects variables in a given de Morgan formula so that, *with high probability* over the random assignments to the chosen variables, the original formula shrinks in size, when simplified using a *deterministic polynomial-time* formula-simplification algorithm.

5.1 Introduction

Subbotovskaya [Sub61] introduced the method of *random restrictions* to prove that PARITY requires de Morgan formulas of size $\Omega(n^{1.5})$, where a de Morgan formula is a boolean formula over the basis $\{\vee, \wedge, \neg\}$. She showed that a random restriction of all but a fraction p of the input variables yields a new formula whose size is expected to reduce by at least the factor $p^{1.5}$. That is, the *shrinkage exponent* Γ for de Morgan formulas is at least 1.5, where the shrinkage exponent is defined as the least upper bound on γ such that the expected formula size shrinks by the factor p^{γ} under a random restriction leaving p fraction

of variables free.

Impagliazzo and Nisan [IN93] argued that Subbotovskaya's bound $\Gamma \geq 1.5$ is not optimal, by showing that $\Gamma \geq 1.556$. Paterson and Zwick [PZ93] improved upon [IN93], getting $\Gamma \geq (5-\sqrt{3})/2 \approx 1.63$. Finally, Håstad [Hås98] proved the tight bound $\Gamma = 2$; combined with Andreev's construction [And87], this yields a function in P requiring de Morgan formulas of size $\Omega(n^{3-o(1)})$.

While the original motivation for the shrinkage results of [Sub61, IN93, PZ93, Hås98] was to prove formula lower bounds, the same results turn out to be useful also for designing nontrivial SAT algorithms for small de Morgan formulas. Santhanam [San10] strengthened Subbotovskaya's *expected* shrinkage result to *concentrated* shrinkage, i.e., shrinkage with high probability, and used this to get a deterministic #SAT algorithm (counting the number of satisfying assignments) for linear-size de Morgan formulas, with the running time $2^{n-\Omega(n)}$. Santhanam's algorithm deterministically selects a most frequent variable in the current formula, and recurses on the two subformulas obtained by restricting the chosen variable to 0 and 1; after $n - \Omega(n)$ recursive calls, almost all obtained formulas depend on fewer than the actual number of free variables remaining, which leads to nontrivial savings over the brute-force SAT algorithm for the original formula. A similar algorithm works also for formulas of size less than $n^{2.5}$, with the running time $2^{n-n^{\Omega(1)}}$ [CKK$^+$13b].

Motivated by average-case formula lower bounds, Komargodksi et al. [KRT13] (building upon [IMZ12]) showed a concentrated-shrinkage version of Håstad's optimal result for the shrinkage exponent $\Gamma = 2$. Combined with the aforementioned algorithm of Chen et al. [CKK$^+$13b], this yields a nontrivial *randomized* zero-error #SAT algorithm for de Morgan formulas of size $n^{3-o(1)}$, running in time $2^{n-n^{\Omega(1)}}$.

Is there a *deterministic* #SAT algorithm with similar running time that works for formulas of size close to n^3? We make a step in that direction, by giving such an algorithm for formulas up to size $n^{2.63}$.

5.1.1 Our Main Results and Techniques

Our main result is a *deterministic* #SAT algorithm for de Morgan formulas of size up to $n^{2.63}$, running in time $2^{n-n^{\Omega(1)}}$.

Theorem 5.1 (Main)**.** *There is a deterministic algorithm for counting the number of satisfying assignments in a given de Morgan formula on n variables of size at most $n^{2.63}$ which*

runs in time at most $2^{n-n^{\delta}}$*, for some constant* $0 < \delta < 1$*.*

As in [San10, CKK$^+$13b], we use a deterministic algorithm to choose a next variable to restrict, and then recurse on the two resulting restrictions of this variable to 0 and 1. Instead of Subbotovskaya-inspired selection procedure (choosing the most frequent variable), we use the weight function introduced by Paterson and Zwick [PZ93], which measures the potential savings for each one-variable restriction, and selects a variable with the biggest savings. Since [PZ93] gives the shrinkage exponent $\Gamma \approx 1.63$, rather than Subbotovskaya's 1.5, this could potentially lead to an improved #SAT algorithm for larger de Morgan formulas.

However, computing the savings, as defined by [PZ93], is NP-hard, as it requires computing the size of a smallest logical formula equivalent to a given one-variable restriction. In fact, the shrinkage result of [PZ93] is *nonconstructive* in the following sense: the expected shrinkage in size is proved for the minimal logical formula computing the restricted boolean function, rather than for the formula obtained from the original formula using efficiently computable simplification rules. In contrast, the shrinkage results of [Sub61, Hås98] are constructive: the restricted formula is expected to shrink in size when simplified using a certain explicit set of logical rules, so that the new, simplified formula is computable in polynomial time from the original restricted formula.

While the constructiveness of shrinkage is unimportant for proving formula lower bounds, it is crucial for designing shrinkage-based #SAT algorithms for de Morgan formulas, such as those in [San10, CKK$^+$13b, KRT13]. Our main technical contribution is a proof of the *constructive* version of the result in [PZ93]: we give deterministic polynomial-time algorithms for formula simplification and extend the analysis of [PZ93] to show expected shrinkage of formulas with respect to this efficiently computable simplification procedure. The same simplification procedure allows us to choose, in deterministic polynomial-time, which variable should be restricted next. The merit of deterministic variable selection and concentrated and constructive shrinkage, for a shrinkage exponent Γ, is that they yield a deterministic satisfiability algorithm for de Morgan formulas up to size $n^{\Gamma+1-o(1)}$, using an approach of [CKK$^+$13b].

Namely, once we have this constructive shrinkage result, based on restricting one variable at a time, we apply the martingale-based analysis of [KR13, CKK$^+$13b] to derive a *concentrated* version of constructive shrinkage, showing that almost all random settings of the selected variables yield restricted formulas of reduced size, where the restricted formulas are simplified by our efficient procedure. The shrinkage exponent $\Gamma = (5-\sqrt{3})/2 \approx 1.63$

is the same as in [PZ93]. Using [CKK+13b], we then get a deterministic #SAT algorithm, running in time $2^{n-n^{\Omega(1)}}$, that works for de Morgan formulas of size up to $n^{\Gamma+1-o(1)} \approx n^{2.63}$.

Related work

The deep interplay between lower bounds and satisfiability algorithms has been witnessed in several circuit models. For example, Paturi, Pudlak and Zane [PPZ99] give a randomized algorithm for k-SAT running in time $O(n^2 s 2^{n-n/k})$, where n is the number of variables and s is the formula size; they also show that PARITY requires depth-3 circuits of size $\Omega(n^{1/4}2^{\sqrt{n}})$. More generally, Williams [Wil10] shows that a "better-than-trivial" algorithm for Circuit Satisfiability, for a class $\mathcal{C}$ of circuits, implies a super-polynomial lower bounds against the circuit class $\mathcal{C}$ for some language in NEXP; using this approach, Williams [Wil11] obtains a super-polynomial lower bound against ACC^0 circuits[1] by designing a nontrivial SAT algorithm for ACC^0 circuits.

Following [San10], Seto and Tamaki [ST12] get a nontrivial #SAT algorithm for general linear-size formulas (over an arbitrary basis). Impagliazzo et al. [IMP12] use a generalization of Håstad's Switching Lemma [Hås86], an analogue of shrinkage for AC^0 circuits[2], to give a nontrivial randomized zero-error #SAT algorithm for depth-d AC^0 circuits on n inputs of size up to $2^{n^{1/(d-1)}}$. Beame et al. [BIS12] give a nontrivial deterministic #SAT algorithm for AC^0 circuits, however, only for circuits of much smaller size than that of [IMP12].

Recently, the method of (pseudo) random restrictions has also been used to get pseudorandom generators (yielding additive-approximation #SAT algorithms) for small de Morgan formulas [IMZ12] and AC^0 circuits [TX13].

Remainder of this chapter. Section 5.2 contains our efficient formula-simplification procedures. We use these procedures in Section 5.3 to prove a constructive and concentrated shrinkage result for de Morgan formulas. This is then used in Section 5.4 to describe and analyze our #SAT algorithm from Theorem 5.1. Section 5.5 contains some open questions.

[1] constant-depth, unbounded fanin circuits, using AND, OR, NOT, and (MOD m) gates, for any integer m

[2] constant-depth, unbounded fanin circuits, using AND, OR, and NOT gates

Preliminaries

Recall that a *(de Morgan) formula* is a binary tree where each leaf is labeled by a literal (a variable x or its negation $\overline{x}$) or a constant (0 or 1), and each internal node is labeled by $\wedge$ or $\vee$. Let F be a formula with no constant leaves. We define the *size* of F, denoted by $L(F)$, as number of leaves in F. Following [PZ93], we define a *twig* to be a subtree with exactly two leaves. Let $T(F)$ be the number of twigs in F. We define the *weight* of F as $w(F) = L(F) + \alpha \cdot T(F)$, where $\alpha = \sqrt{3} - 1 \approx 0.732$. For convenience, if F is a constant, we define $L(F) = w(F) = 0$. We say F is *trivial* if it is a constant or a literal. Note that we define the size and weight only for formulas which are either constants or with no constant leaves; this is without loss of generality since constants can always be eliminated using a simplification procedure below.

It is easy to see that $L(F) + \alpha \leq w(F) \leq L(F)(1 + \alpha/2)$, since the number of twigs in a formula is at least one and at most half of the number of leaves.

We denote by $F|_{x=1}$ the formula obtained from F by substituting each appearance of x by 1 and $\overline{x}$ by 0; $F|_{x=0}$ is similar. We say a formula $\vee$*-depends (*$\wedge$*-depends)* on a literal y if there is a path from the root to a leaf labeled by y such that every internal node on the path (including the root) is labeled by $\vee$ (by $\wedge$).

5.2 Formula Simplification Procedures

5.2.1 Basic Simplification

We define a procedure **Simplify** to eliminate constants, redundant literals and redundant twigs in a formula. The procedure includes the standard constant simplification rules and a natural extension of the one-variable simplification rules from [Hås98].

$\mathbf{Simplify}(F)$:

If F is trivial, done. Otherwise, apply the following transformations whenever applicable. We denote by y a literal and G a subformula.

1. **Constant elimination.**
   (a) If a subformula is of the form $0 \wedge G$, replace it by 0.
   (b) If a subformula is of the form $1 \vee G$, replace it by 1.

(c) If a subformula is of the form $1 \wedge G$ or $0 \vee G$, replace it by G.

2. **One-variable simplification.**

(a) If a subformula is of the form $y \vee G$ and y or $\overline{y}$ appears in G, replace the subformula by $y \vee G|_{y=0}$.

(b) If a subformula is of the form $y \wedge G$ and y or $\overline{y}$ appears in G, replace the subformula by $y \wedge G|_{y=1}$.

(c) If a subformula G is of the form $G_1 \vee G_2$ for non-trivial G_1 and G_2, and G $\vee$-depends on a literal y, then replace G by $y \vee G|_{y=0}$.

(d) If a subformula G is of the form $G_1 \wedge G_2$ for non-trivial G_1 and G_2, and G $\wedge$-depends on a literal y, then replace G by $y \wedge G|_{y=1}$.

We call a formula *simplified* if it is invariant under **Simplify**. Note that a simplified formula may not be a smallest logically equivalent formula; e.g., $(x \wedge y) \vee (\overline{x} \wedge y)$ is already simplified but it is logically equivalent to y.

The rules 1(a)–(c) and 2(a)–(b) are from [Hås98, San10]. Rules 2(c)–(d) are a natural generalization of the one-variable rule of [Hås98], which allow us to eliminate more redundant literals and reduce the formula weight. For example, the formula $(x \vee y) \vee (x \wedge y)$ simplifies to $x \vee y$ under our rules but not the rules in [Hås98, San10]. For another example, the formula $(x \vee y) \vee (z \wedge w)$ with weight $4 + 2\alpha$ simplifies to $x \vee (y \vee (z \wedge w))$ with weight $4 + \alpha$.

The next lemma shows **Simplify** is efficient.

Lemma 5.2. Simplify *runs in polynomial time.*

To prove Lemma 6.1, we first show that **Simplify** reduces the formula size but it does not increase the number of twigs.

Lemma 5.3. *Let F be a formula with no constant leaves. Suppose we substitute k leaves of F by constants, and run* **Simplify** *which produces a new formula F'. Then $L(F') \leq L(F) - k$ and $T(F') \leq T(F)$.*

Proof. If F is a literal, this is obvious. Suppose that F is not trivial.

We first consider constant-elimination rules. Each replacement removes at least one leaf, so the formula size reduces by at least k. For rules 1(a)–(b), at most one new twig may be formed, but at least one old twig is removed. For rule 1(c), if G is not a literal, the

twigs will not change; if G is a literal, one old twig is removed and at most one new twig is formed.

Now consider one-variable simplification rules. For rules 2(a)–(b), new constants are introduced, which will be eliminated later; the number of twigs does not increase by constant elimination. For rules 2(c)–(d), the formula size does not increase; if $G|_{y=0}$ is a literal, then a new twig is formed but at least one old twig will be removed; otherwise, the twigs will not change. □

Proof. (Lemma 6.1) **Simplify** checks if any simplification rule is applicable, and terminates when none of the rules are applicable. At each round, it takes polynomial time to check whether a rule applies and, if so, to transform the formula by the rule. Next we bound the number of rounds.

For rules 1(a)–(c), at least one leaf is eliminated. For rules 2(a)–(b), at least one constant leaf is introduced and will then be eliminated. So, the number of rounds where one of 1(a)–(c) or 2(a)–(b) is active is at most $2 \cdot L(F)$. Next we bound the number of rounds where one of 2(c)–(d) is active.

Call a nontrivial subformula G without constant leaves *stable* if none of the rules 2(a)–(d) are applicable to G; that is, G is either $y \vee H$ or $y \wedge H$ where y or $\overline{y}$ does not appear in H, or G does not $\vee$-depend or $\wedge$-depend on any literal. For a non-trivial formula F without constant leaves, we define $q(F) = \sum_G L(G)$ where G ranges over all subformulas of F which are unstable.

Consider rules 2(c)–(d). When we replace an unstable subformula G by $y \vee G|_{y=0}$ or $y \wedge G|_{y=1}$ and eliminate the constants, the quantity $q(F)$ reduces by at least one, since either all of G is eliminated or the new unstable formula is smaller than G. Since $q(F) \leq poly(L(F))$, the number of rounds where one of 2(c)–(d) is active is at most $poly(L(F))$. □

5.2.2 Simplification under All One-Variable Restrictions

Here we consider how a formula simplifies when one of its variables is restricted. Let F be a formula. We define a recursive procedure **RestrictSimplify** which produces a collection of formulas for F under all one-variable restrictions. We denote the output of the procedure by $\{F_y\}$, where y ranges over all literals. Note that each F_y is logically equivalent to $F|_{y=1}$.

The idea behind the transformations in **RestrictSimplify** is the following. When a formula simplifies to a literal under some one-variable restriction, then the formula must be

logically equivalent to some special form. For example, if we know that $F|_{x=1}$ simplifies to a literal y, then F itself must be logically equivalent to $(x \wedge y) \vee (\overline{x} \wedge G)$ for some G. This logically equivalent form may help to simplify F under other one-variable restrictions.

$\mathbf{RestrictSimplify}(F)$:

If F is a constant c, then let $F_y := c$ for all y. If F is a literal, then let $F_y := F|_{y=1}$ for all y.

If F is $G \vee H$ or $G \wedge H$, recursively call **RestrictSimplify** to compute $\{G_y\}$ and $\{H_y\}$, and initialize each $F_y := \mathbf{Simplify}(G_y \vee H_y)$ or $F_y := \mathbf{Simplify}(G_y \wedge H_y)$, respectively. Then apply the following transformations whenever possible. We suppose there are two literals x and y over distinct variables such that $F_x = y$.

1. If $F_{\overline{x}} = y$, then let $F_w := y|_{w=1}$ for every literal w.
2. If $F_{\overline{x}} = z$ for some literal $z \notin \{x, \overline{x}, y\}$, then let $F_w := \mathbf{Simplify}((x \wedge y) \vee (\overline{x} \wedge z)|_{w=1})$ for every literal w.
3. (a) If neither x nor $\overline{x}$ appears in F_y, then let $F_y := 1$; (b) otherwise, let $F_y := \mathbf{Simplify}(x \vee (F_y|_{x=0}))$.
4. (a) If neither x nor $\overline{x}$ appears in $F_{\overline{y}}$, then let $F_{\overline{y}} := 0$; (b) otherwise, let $F_{\overline{y}} := \mathbf{Simplify}(\overline{x} \wedge (F_{\overline{y}}|_{x=0}))$.
5. For $z \notin \{x, \overline{x}, y, \overline{y}\}$, if neither x nor $\overline{x}$ appears in F_z, then let $F_z := y$.

Correctness of RestrictSimplify. The above transformations are based on logical implications. In case 1, $F_x = F_{\overline{x}} = y$ implies that $F \equiv y$. In case 2, $F_x = y$ and $F_{\overline{x}} = z$ implies that $F \equiv (x \wedge y) \vee (\overline{x} \wedge z)$. Note that in this case z might be $\overline{y}$. In case 3, we have $F_y|_{x=1} \equiv F_x|_{y=1} = 1$; if neither x nor $\overline{x}$ appears in F_y then $F_y = F_y|_{x=1} \equiv 1$, otherwise $F_y \equiv x \vee (F_y|_{x=0})$. Case 4 is dual to case 3. In case 5, if neither x nor $\overline{x}$ appears in F_z then $F_z = F_z|_{x=1} \equiv F_x|_{z=1} = y$.

Remark 5.4. It is possible to introduce more simplifications rules in **RestrictSimplify**, e.g., when F_x is a constant for some literal x, or when, in case 5, x or $\overline{x}$ appears in F_z[3]. However, such simplifications are not needed for our proof of constructive shrinkage.

[3]then we could let $F_z := (x \wedge y) \vee (\overline{x} \wedge \mathbf{Simplify}(F_z|_{x=0}))$

It is easy to show that **RestrictSimplify** is effiicent. We prove a proof of the following lemma in the Appendix.

Lemma 5.5. RestrictSimplify *runs in polynomial time.*

Proof. The base case is obvious. For induction, suppose $F = G \vee H$, where F is on n variables. The procedure makes two recursive calls on G and H, and then simplifies the collection $\{F_y\}$. The transformations on the collection $\{F_y\}$, except case 3(b) and 4(b), reduce the formulas to the smallest possible size. In case 3(b) (similarly 4(b)), F_y becomes constant 1, or literal x, or non-trivial; this will not trigger another transformation. Thus the transformations on the collection $\{F_y\}$ run in time $poly(n, L(F))$.

We conclude that the time spent at each node of the formula F is $poly(n, L(F))$, and so the overall time is $L(F) \cdot poly(n, L(F)) = poly(n, L(F))$, as required. □

We will need the following basic property of **RestrictSimplify**.

Claim 5.6. *For $F = G \vee H$ or $F = G \wedge H$, we have $w(F_y) \leq w(G_y) + w(H_y)$, for all literals y except those where G_y and H_y are literals over distinct variables.*

Proof. Let $F = G \vee H$; the other case is identical. We initialize $F_y := \mathbf{Simplify}(G_y \vee H_y)$, and so the required inequality holds initially. All transformations, except 3(b) and 4(b), produce the smallest logically equivalent formula; rules 3(b) and 4(b) do not increase the weight of the formula. □

The *solo structure* of a formula F is the relation on literals defined by $x \Rightarrow y$ if $F_x = y$, where the collection of formulas $\{F_x\}$ is produced by the procedure **RestrictSimplify**. The following lemma gives all possible solo structures; it resembles the characterization of solo structures for boolean functions from [PZ93]. We provide the proof in the Appendix.

Lemma 5.7. *The solo structure of a non-trivial formula F must be in one of the following forms:*

(i) the empty relation,

(ii) there exists y such that for all literals $x \notin \{y, \overline{y}\}$ we have $x \Rightarrow y$ in the relation,

(iii) $\{x_1 \Rightarrow y,\ \ldots,\ x_k \Rightarrow y\}$ for some $k \geq 1$ and x_i's are over distinct variables,

(iv) $\{x \Rightarrow y,\ y \Rightarrow x,\ \overline{x} \Rightarrow \overline{y},\ \overline{y} \Rightarrow \overline{x}\}$ for some literals x, y,

(v) $\{x \Rightarrow y,\ \overline{x} \Rightarrow z\}$ *for some literals* x, y, z*,*

(vi) $\{x \Rightarrow y,\ y \Rightarrow x\}$ *for some literals* x, y*,*

(vii) $\{x \Rightarrow y,\ \overline{y} \Rightarrow \overline{x}\}$ *for some literals* x, y*.*

Proof. If none of F_x is a literal, then this is case (i). Otherwise, suppose that $F_x = y$ for some literals x, y. If x is the only literal such that F_x is a literal, then this is case (iii) with $k = 1$. Next we assume there is another literal x' such that $F_{x'} = y'$ for some literal y'. We consider different possibilities of x' and the implications by the transformations in **RestrictSimplify**.

If $x' = \overline{x}$, consider different cases of y'. If $y' = y$, then by the transformation 1 in **RestrictSimplify** we have $F_w = y$ for all $w \notin \{y, \overline{y}\}$ and this gives case (ii). If $y' \notin \{x, \overline{x}, y\}$, then by the transformation 2 in **RestrictSimplify** we have $F_w := \mathbf{Simplify}((x \wedge y) \vee (\overline{x} \wedge y')|_{w=1})$; this gives either case (iv) if $y' = \overline{y}$ or case (v) if $y' \notin \{x, \overline{x}, y, \overline{y}\}$.

If $x' = y$, then we have both $F_x = y$ and $F_y = y'$. By the transformation 3(a)-(b) in **RestrictSimplify**, the only possibility for y' is that $y' = x$. This gives either case (iv) if $F_{\overline{x}} = \overline{y}$ or case (vi) otherwise.

If $x' = \overline{y}$, then we have both $F_x = y$ and $F_{\overline{y}} = y'$. By the transformation 4(a)-(b) in **RestrictSimplify**, the only possibility for y' is that $y' = \overline{x}$. This gives either case (iv) if $F_{\overline{x}} = \overline{y}$ or case (vii) otherwise.

If $x' \notin \{x, \overline{x}, y, \overline{y}\}$, then by the transformation 5 in **RestrictSimplify**, the only possibility for y' is $y' = y$. Note that y' cannot be x or $\overline{x}$ since $y'|_{x=1} = F_{x'}|_{x=1} \equiv F_x|_{x'=1} = y|_{x'=1} = y$. This gives case (iii) with $k \geq 2$. □

5.3 Constructive and Concentrated Shrinkage

Here we prove a constructive and concentrated version of the shrinkage result from [PZ93]. For each literal y of a given formula F, we define the savings (reduction in weight of F) when we replace F by the new formula F_y, as computed by the procedure **RestrictSimplify**. We first prove that the lower bound on the average savings (over all variables of F) shown by [PZ93] continues to hold with respect to our efficiently computable one-variable restrictions F_y.

5.3.1 Average Savings under One-Variable Restrictions

Assume a formula F is simplified; otherwise, let $F := \mathbf{Simplify}(F)$. For a formula F and a literal y, we define $\sigma_y(F) = w(F) - w(F_y)$, where F_y is produced by **RestrictSimplify**. Let $\sigma(F) = \sum_x(\sigma_x(F) + \sigma_{\overline{x}}(F))$, where the summation ranges over all variables of F. The quantity $\sigma(F)$ measures the total *savings* under all one-variable restrictions.

Theorem 5.8. *For any formula F, it holds that $\sigma(F)/w(F) \geq 2\gamma$, where $\gamma = (5-\sqrt{3})/2 \approx 1.63$.*

The proof is by induction, as in [PZ93]. The difficulty here is that we need to apply the "syntactic simplifications" defined by the procedure **RestrictSimplify**, instead of using the smallest logically equivalent formulas as in [PZ93].

For the base case, the following lemma can be proved by enumerating all possible formulas of size at most 4 (the proof is given in the Appendix).

Lemma 5.9. *For any simplified F of size at most 4, we have $\sigma(F)/w(F) \geq 2\gamma$.*

Proof. Table 5.1 lists all simplified formulas (or their duals) of size at most 4, together with the savings. The cases labeled by * were not consider in [PZ93] since they are not the smallest logically equivalent formulas.

Table 5.1: Savings for all formulas with $2 \leq L(F) \leq 4$

	Formula	Weight $w(F)$	Savings $\sigma(F)$	$\sigma(F)/w(F)$
	$x \vee y$	$2+\alpha$	$6+4\alpha$	$=2\gamma$
	$x \vee (y \wedge z)$	$3+\alpha$	$10+3\alpha$	$=2\gamma$
	$x \vee (y \vee z)$	$3+\alpha$	$12+3\alpha$	$>2\gamma$
	$x \vee (y \wedge (z \wedge w))$	$4+\alpha$	$17+4\alpha$	$>2\gamma$
	$x \vee (y \wedge (z \vee w))$	$4+\alpha$	$15+2\alpha$	$>2\gamma$
	$x \vee (y \vee (z \wedge w))$	$4+\alpha$	$16+2\alpha$	$>2\gamma$
	$x \vee (y \vee (z \vee w))$	$4+\alpha$	$20+4\alpha$	$>2\gamma$
	$(x \wedge y) \vee (z \wedge w)$	$4+2\alpha$	$12+8\alpha$	$=2\gamma$
*	$(x \wedge y) \vee (x \wedge y)$	$4+2\alpha$	$14+8\alpha$	$>2\gamma$
*	$(x \wedge y) \vee (\overline{x} \wedge y)$	$4+2\alpha$	$14+8\alpha$	$>2\gamma$
	$(x \wedge y) \vee (\overline{x} \wedge \overline{y})$	$4+2\alpha$	$12+8\alpha$	$=2\gamma$
*	$(x \wedge y) \vee (x \wedge z)$	$4+2\alpha$	$16+9\alpha$	$>2\gamma$
	$(x \wedge y) \vee (\overline{x} \wedge z)$	$4+2\alpha$	$14+8\alpha$	$>2\gamma$

□

For formulas of size larger than 4, we consider whether one child of the root is trivial. Without loss of generality, we assume the root is labeled by $\vee$; the other case is dual. The following lemma considers if one child of the root is trivial. The proof is similar to [PZ93] and is given in the Appendix.

Lemma 5.10. *If F is a simplified formula of the form $x \vee G$ for some literal x and subformula G, and $L(F) \geq 5$, then $\sigma(F)/w(F) \geq 2\gamma$.*

Proof. The proof is similar to [PZ93]. Without loss of generality, assume x is a variable. Since F is simplified, we get that x does not appear in G. Let k be the number of literals y such that G_y is a literal. We will show that the k twigs produced by restricting these literals can be compensated. For $k \leq 4$, by the induction hypothesis on G and the fact that $w(F) = 1 + w(G)$, we have

$$\begin{aligned} \sigma(F) &= \sigma_{\overline{x}}(F) + \sigma_x(F) + \sigma(G) - k\alpha \\ &\geq 1 + w(F) + 2\gamma \cdot (w(F) - 1) - 4\alpha \\ &= 2\gamma \cdot w(F) + w(F) - (4\alpha + 2\gamma - 1) \geq 2\gamma \cdot w(F) \end{aligned}$$

since $w(F) \geq L(F) + \alpha > 5.7 > (4\alpha + 2\gamma - 1) \approx 5.2$.

If $k \geq 5$, then

$$\begin{aligned} \sigma(F) &\geq 1 + w(F) + 5(w(F) - (2 + \alpha)) \\ &\geq 6w(F) - (9 + 5\alpha) \geq 2\gamma \cdot w(F) \end{aligned}$$

since $w(F) > 5.7 > (9 + 5\alpha)/(6 - 2\gamma) \approx 4.7$. □

Now we consider formulas where both children of the root are non-trivial.

Lemma 5.11. *Suppose F is of the form $G \vee H$ with $L(F) \geq 5$ and G, H are non-trivial. Then $\sigma(F)/w(F) \geq 2\gamma$.*

Intuitively, we need to take care of the cases where both G and H simplify to literals on distinct variables (thereby forming a new twig); otherwise the result holds by the induction hypothesis. Suppose $G_x \vee H_x$ is a twig for some literal x. Then $\sigma_x(F) = \sigma_x(G) + \sigma_x(H) - \alpha$, i.e., we get the savings from restricting x in G and H, but then need to pay the penalty α for the twig created. We will argue that there are "extra savings" from restricting other literals in the formula F that can be used to compensate for the penalty α at x.

Proof. We first prove that, for a literal x, if G_x and H_x are not literals over distinct variables, then $\sigma_x(F) \geq \sigma_x(G) + \sigma_x(H)$. Since $w(F) = w(G) + w(H)$, the claim follows from $w(F_x) \leq w(G_x) + w(H_x)$, which holds by Claim 5.6.

Let k be the number of different literals x such that $G_x \vee H_x$ is a twig (i.e., G_x and H_x are literals over distinct variables). Thus there are k twigs created as we consider all possible one-variable restrictions. We will argue that, for different cases of k, the weight $k\alpha$ of these new twigs can be compensated from savings in other restrictions.

Case $k = 0$**:** We have $\sigma_y(F) \geq \sigma_y(G) + \sigma_y(H)$ for all literals y, and thus $\sigma(F) \geq \sigma(G) + \sigma(H)$. The result follows directly by the induction hypothesis on G and H.

Case $1 \leq k \leq 2$**:** Let x be such that $G_x = y$ and $H_x = z$. Without loss of generality, assume x, y, z are distinct variables. Consider F under the restrictions $y = 1$ and $z = 1$. We will argue that the extra savings from applying **Simplify** on $G_y \vee H_y$ and $G_z \vee H_z$ are at least $2 > k\alpha$.

Since $G_x = y$, transformation 3(a)–(b) in **RestrictSimplify** guarantee that either G_y is constant 1 or it $\vee$-depends on x. Similarly either H_z is constant 1 or it $\vee$-depends on x. Since $H_y|_{x=1} \equiv H_x|_{y=1} = z$, we get that H_y is not a constant (it depends on z), and if it is a literal it must be z. Similarly G_z is not a constant (it depends on y), and if it is a literal it must be y.

We first consider the case that either G_y or H_z is constant 1. If $G_y = H_z = 1$, then there are at least 2 savings from simplifying $G_y \vee H_y$ and $G_z \vee H_z$ by eliminating constants. If $G_y = 1$ and H_y is not a literal, then there are at least 2 savings from simplifying $G_y \vee H_y$. If $G_y = 1$, $H_y = z$ and $H_z \neq 1$, we first have one saving from simplifying $G_y \vee H_y$; then since $H_y = z$ and $H_z \neq 1$, by the transformation 3(b) in **RestrictSimplify** H_z $\vee$-depends on y, and since G_z depends on y, we get another saving from simplifying $G_z \vee H_z$. The cases where $H_z = 1$ are similar.

Next we consider that both G_y and H_z $\vee$-depends on x. In the following we analyze different possibilities for H_y and G_z.

- If x appears in both H_y and G_z, then there are at least 2 savings from simplifying $G_y \vee H_y$ and $G_z \vee H_z$ by eliminating x.
- If x appears in H_y but not G_z, then by the transformation 5 in **RestrictSimplify** we have $G_z = y$, and thus G_y $\vee$-depends on both x and z. Then since H_y depends on

both x and z, we have two savings from simplifying $G_y \vee H_y$ by eliminating both x and z from H_y.

- If x appears in G_z but not H_y, this is similar to the previous case.
- If x appears in neither H_y nor G_z, then by the transformation 5 in **RestrictSimplify** we have $G_z = y$ and $H_y = z$. Thus G_y $\vee$-depends on both x and z, and H_z $\vee$-depends on both x and y. Therefore we have at least 2 savings, one from simplifying $G_y \vee H_y$ by eliminating z, and another from simplifying $G_z \vee H_z$ by eliminating y.

Case $k \geq 3$: By Lemma 5.7, the solo structure of G and H must be one of cases (ii), (iii), or (iv).

First assume that either G or H is in case (ii) of Lemma 5.7. Without loss of generality, suppose G is in case (ii); then G is logically equivalent to a literal y but itself is non-trivial, which implies that $w(G) \geq 4 + \alpha$. (The smallest non-trivial, simplified formula equivalent to a literal has size at least 4). We have that $w(G_z) = 1$ for at least k literals $z \notin \{y, \overline{y}\}$, and $w(G_y) = w(G_{\overline{y}}) = 0$. Then by the fact that $w(F) = w(G) + w(H)$ and the induction hypothesis on H, we have

$$\begin{aligned} \sigma(F) &\geq k(w(G) - 1) + 2w(G) + \sigma(H) - k\alpha \\ &\geq 2\gamma \cdot w(F) + (2 + k - 2\gamma)w(G) - k(1 + \alpha) \geq 2\gamma \cdot w(F). \end{aligned}$$

If both G and H are in case (iv), then, under each restriction, they reduce to literals on the same variable. Since in case (iii) all x_i's are over distinct variables, it is not possible that one of G and H is in case (iv) while the other is in case (iii). Thus, we now only need to analyze if both G and H are in case (iii).

Without loss of generality, suppose that $x_1, \ldots, x_k, y, z$ are distinct variables such that $G_{x_i} = y$ and $H_{x_i} = z$ for $i = 1, \ldots, k$. By the transformation 3 in **RestrictSimplify**, either $G_y = 1$ or G_y $\vee$-depends on $x_1, \ldots, x_k$; and H_z is similar.

If every x_i appears in H_y, then there are k savings from simplifying $G_y \vee H_y$ by eliminating x_i's. Similarly, if every x_i appears in G_z, there are also k savings from simplifying $G_z \vee H_z$.

If some x_i does not appear in H_y and some x_i does not appear in G_z. By the transformation 5 in **RestrictSimplify**, we have $H_y = z$ and $G_z = y$. Therefore,

$$\begin{aligned}
\sigma_{x_i}(F) &= w(F) - (2+\alpha), \;\; i = 1, \ldots, k \\
\sigma_y(F) &\geq 1 + (w(H) - 1) = w(H) \\
\sigma_z(F) &\geq 1 + (w(G) - 1) = w(G) \\
\sum_v \sigma_{\overline{v}}(F) &\geq L(F) \geq w(F)/(1+\alpha/2), \;\; v \text{ ranges over all variables of } F
\end{aligned}$$

Summing the above cases together yields $\sigma(F) \geq 2\gamma \cdot w(F)$. □

Proof of Theorem 5.8. The proof is by combining the base case in Lemma 5.9 and the two inductive cases in Lemma 5.10 and Lemma 5.11. □

5.3.2 Concentrated Shrinkage

Theorem 5.8 characterizes the average shrinkage of the weight of a formula when a randomly chosen literal is restricted. Given a formula F on n variables, if we randomly pick one variable and randomly assign it 0 or 1, the weight of the restricted formula (produced by **RestrictSimplify**) reduces by at least $\gamma \cdot w(F)/n$ on average.

The procedure **RestrictSimplify** also allows us to deterministically pick the variable with the best savings in polynomial time. That is, given a formula F, we run **RestrictSimplify** to produce a collection of formulas $\{F_y\}$, and then pick a variable x such that $\sigma_x(F) + \sigma_{\overline{x}}(F)$ is maximized. We show that randomly restricting such a variable significantly reduces the expected weight of the simplified formula.

Lemma 5.12. *Let F be a formula on n variables. Let x be the variable such that $\sigma_x(F) + \sigma_{\overline{x}}(F)$ is maximized. Let F' be F_x or $F_{\overline{x}}$ with equal probability. Then we have $w(F') \leq w(F) - 1$ and*

$$\mathbf{E}[w(F')] \leq \left(1 - \frac{1}{n}\right)^{\gamma} \cdot w(F).$$

Proof. Restricting one variable eliminates at least one leaf; therefore $w(F') \leq w(F) - 1$. By Theorem 5.8, $n(\sigma_x(F) + \sigma_{\overline{x}}(F)) \geq \sigma(F) \geq 2\gamma \cdot w(F)$. Then we have

$$\mathbf{E}[w(F')] = w(F) - \frac{1}{2}(\sigma_x(F) + \sigma_{\overline{x}}(F)) \leq \left(1 - \frac{\gamma}{n}\right) \cdot w(F) \leq \left(1 - \frac{1}{n}\right)^{\gamma} \cdot w(F).$$

□

Next we use the martingale-based analysis from [KR13, CKK$^+$13b] to derive a "high-probability shrinkage" result from Lemma 5.12. Let $F_0 = F$ be a formula on n variables. For $1 \leq i \leq n$, let F_i be the (random) formula obtained from F_{i-1} by assigning the variable with the best savings with a random value $R_i \in \{0, 1\}$. The following Lemma shows the weight of a given de Morgan formula reduces with high probability under the restriction process. The proof, which is similar to [CKK$^+$13b], is provided in the Appendix.

Lemma 5.13 (Concentrated weight shrinkage)**.** *Let F be any given de Morgan formula on n variables. For any $k > 10$, we have*

$$\mathbf{Pr}\left[w(F_{n-k}) \geq 2 \cdot w(F) \cdot \left(\frac{k}{n}\right)^{\gamma}\right] < 2^{-k/10}.$$

Recall that a sequence of random variables $X_0, X_1, X_2, \ldots, X_n$ is a *supermartingale* with respect to a sequence of random variables $R_1, R_2, \ldots, R_n$ if $\mathbf{E}[X_i \mid R_{i-1}, \ldots, R_1] \leq X_{i-1},$ for $1 \leq i \leq n$. We use the following version of Azuma's inequality.

Lemma 5.14 ([CKK$^+$13b])**.** *Let $\{X_i\}_{i=0}^n$ be a supermartingale with respect to $\{R_i\}_{i=1}^n$. Let $Y_i = X_i - X_{i-1}$. If, for every $1 \leq i \leq n$, the random variable Y_i (conditioned on $R_{i-1}, \ldots, R_1$) assumes two values with equal probability, and there exists a constant $c_i \geq 0$ such that $Y_i \leq c_i$, then, for any λ, we have*

$$\mathbf{Pr}[X_n - X_0 \geq \lambda] \leq \exp\left(-\frac{\lambda^2}{2\sum_{i=1}^n c_i^2}\right).$$

Let $F_0 = F$ be a formula on n variables. For $1 \leq i \leq n$, let F_i be the (random) formula obtained from F_{i-1} by assigning the variable with the best savings with a random value $R_i \in \{0, 1\}$. We define random variables $W_i := w(F_i)$, $w_i := \log W_i$ and

$$Z_i := w_i - w_{i-1} - \gamma \log\left(1 - \frac{1}{n-i+1}\right).$$

We have the following.

Lemma 5.15. *Let $X_0 = 0$ and $X_i = \sum_{j=1}^i Z_j$. Then the sequence $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$, and, for each Z_i, we have $Z_i \leq c_i := -\gamma \log\left(1 - \frac{1}{n-i+1}\right)$.*

Proof. Since $w_i \leq w_{i-1}$, we have $Z_i \leq c_i$. By Jensen's inequality and Lemma 5.12, we get

$$\begin{aligned}
\mathbf{E}[w_i \mid R_{i-1}, \ldots, R_1] &\leq \log \mathbf{E}[W_i \mid R_{i-1}, \ldots, R_1] \\
&\leq \log\left(W_{i-1}\left(1 - \frac{1}{n-i+1}\right)^{\gamma}\right) \\
&= w_{i-1} + \gamma \log\left(1 - \frac{1}{n-i+1}\right).
\end{aligned}$$

This implies $\mathbf{E}[Z_i \mid R_{i-1}, \ldots, R_1] \leq 0$ and so $\{X_i\}$ is a supermartingale. □

Now we can prove that the weight of a given de Morgan formula reduces with high probability under the restriction process defined above.

Proof of Lemma 5.13. Let λ be arbitrary, and let c_i's be as defined in Lemma 5.15. By Lemmas 5.15 and 6.4, we get

$$\mathbf{Pr}\left[\sum_{j=1}^{i} Z_j \geq \lambda\right] \leq \exp\left(-\frac{\lambda^2}{2\sum_{j=1}^{i} c_j^2}\right).$$

For the left-hand side, we get by the definition of Z_j's that $\sum_{j=1}^{i} Z_j = w_i - w_0 - \gamma \log \frac{n-i}{n}$. Hence,

$$\mathbf{Pr}\left[\sum_{j=1}^{i} Z_j \geq \lambda\right] = \mathbf{Pr}\left[w_i - w_0 - \gamma \log\left(\frac{n-i}{n}\right) \geq \lambda\right] = \mathbf{Pr}\left[W_i \geq e^{\lambda} W_0 \left(\frac{n-i}{n}\right)^{\gamma}\right].$$

For each $1 \leq j \leq i$, we have $c_j \leq \frac{\gamma}{n-j}$, using the inequality $\log(1+x) \leq x$. Thus, $\sum_{j=1}^{i} c_j^2$ is at most

$$\gamma^2 \sum_{j=1}^{i} \left(\frac{1}{n-j}\right)^2 \leq \gamma^2 \sum_{j=1}^{i} \left(\frac{1}{n-j-1} - \frac{1}{n-j}\right) = \gamma^2 \cdot \left(\frac{1}{n-i-1} - \frac{1}{n-1}\right) \leq \gamma^2 \cdot \frac{1}{n-i-1}.$$

Taking $i = n - k$, we get

$$\mathbf{Pr}\left[W_{n-k} \geq e^{\lambda} W_0 \left(\frac{k}{n}\right)^{\gamma}\right] \leq \exp\left(-\frac{\lambda^2}{2\sum_{j=1}^{n-k} c_j^2}\right) \leq e^{-\lambda^2(k-1)/2\gamma^2}.$$

Choosing $\lambda = \ln 2$ concludes the proof. □

Finally, by $w(F)/(1+\alpha/2) \leq L(F) \leq w(F)$ for all F, we get from Lemma 5.13 the desired concentrated constructive shrinkage with respect to the restriction process defined above.

Corollary 5.16 (Concentrated constructive shrinkage)**.** *Let F be an arbitrary de Morgan formula. There exist constants $c, d > 1$ such that, for any $k > 10$,*

$$\mathbf{Pr}\left[L(F_{n-k}) \geq c \cdot L(F) \cdot \left(\frac{k}{n}\right)^{\gamma}\right] < 2^{-k/d}.$$

5.4 #SAT Algorithm for $n^{2.63}$-Size Formulas

Here we prove our main result.

Theorem 5.17. *There is a deterministic algorithm for counting the number of satisfying assignments in a given formula on n variables of size at most $n^{2.63}$ which runs in time $t(n) \leq 2^{n-n^{\delta}}$, for some constant $0 < \delta < 1$.*

Proof. Suppose we have a formula F on n variables of size $n^{1+\gamma-\epsilon}$ for a small constant $\epsilon > 0$. Let $k = n^{\alpha}$ such that $\alpha < \epsilon/\gamma$. We build a restriction decision tree with 2^{n-k} branches as follows:

> Starting with F at the root, run **RestrictSimplify** to produce a collection $\{F_y\}$, pick the variable x which will make the largest reduction in the weight of the current formula. Make the two formulas F_x and $F_{\overline{x}}$ the children of the current node. Continue recursively on F_x and $F_{\overline{x}}$ until get a full binary tree of depth exactly $n - k$.

Note that constructing this decision tree takes time $2^{n-k}\text{poly}(n)$, since the procedure **RestrictSimplify** runs in polynomial time. By Corollary 5.16, all but at most $2^{-k/d}$ fraction of the leaves have the formula size $L(F_{n-k}) < c \cdot L(F) \left(\frac{k}{n}\right)^{\gamma} = cn^{1-\epsilon+\gamma\alpha}$.

To solve #SAT for all "big" formulas (those that haven't shrunk), we use brute-force enumeration over all possible assignments to the k free variables left. The running time is at most $2^{n-k} \cdot 2^{-k/d} \cdot 2^{k} \cdot \text{poly}(n) \leq 2^{n-k/d} \cdot \text{poly}(n)$.

For "small" formulas (those that shrunk to the size less than $cn^{1-\epsilon+\gamma\alpha}$), we use memoization. First, we enumerate all formulas of such size, and compute and store the number of satisfying assignments for each of them. Then, as we go over the leaves of the decision tree that correspond to small formulas, we simply look up the stored answers for these formulas.

There are at most $2^{O(n^{1-\epsilon+\gamma\alpha})}\text{poly}(n)$ such formulas, and counting the satisfying assignments for each one (with k inputs) takes time $2^{k}\text{poly}(n)$. Including pre-processing, computing #SAT for all small formulas takes time at most $2^{n-k} \cdot \text{poly}(n) + 2^{O(n^{1-\epsilon+\gamma\alpha})} \cdot 2^{k} \cdot \text{poly}(n) \leq 2^{n-k} \cdot \text{poly}(n)$.

The overall running time of our #SAT algorithm is bounded by $2^{n-n^{\delta}}$ for some $\delta > 0$. □

5.5 Open Questions

The main open question is whether there is a nontrivial deterministic #SAT algorithm for de Morgan formulas of size up to $n^{3-o(1)}$. Is it possible to derandomize the randomized zero-error algorithm of [KRT13] that is based on Håstad's shrinkage result [Hås98]?

Is it possible to improve the analysis of the shrinkage result of [PZ93] (by considering more general patterns than just twigs), getting a better shrinkage exponent? If so, this could lead to a deterministic #SAT algorithm for larger de Morgan formulas.

Chapter 6

Correlation Bounds and #SAT for Small Circuits

We revisit the gate-elimination method, generalize it to prove correlation bounds of boolean circuits with Parity, and also derive deterministic #SAT algorithms for small linear-size circuits. In particular, we prove that, for boolean circuits of size $3n - n^{0.51}$, the correlation with Parity is at most $2^{-n^{\Omega(1)}}$, and there is a #SAT algorithm running in time $2^{n-n^{\Omega(1)}}$; for boolean circuits of size $2.99n$, the correlation with Parity is at most $2^{-\Omega(n)}$, and there is a #SAT algorithm running in time $2^{n-\Omega(n)}$. Similar correlation bounds and algorithms also exist for circuits over the full binary basis B_2 of size almost $2.5n$.

6.1 Introduction

Proving circuit lower bounds is one of the most important tasks in computational complexity. However, little progress was made on lower bounds for general circuits. The best known explicit lower bounds (for functions even in NP) is $3n - o(n)$ for circuits over the full binary basis B_2, and $5n - o(n)$ for circuits over $U_2 = B_2 \setminus \{\oplus, \equiv\}$.

The connection between circuit lower bounds and efficient algorithms has been explicitly exploited in many recent breakthroughs. For certain restricted circuit classes, such as boolean formulas and AC^0 circuits, several previous work derive non-trivial satisfiability algorithms from known lower bounds proofs; oftentimes, such results also imply average-case lower bounds (correlation bounds).

Santhanam [San10] gave a #SAT algorithm for linear-size formulas over U_2 running in time $2^{n-\Omega(n)}$; the algorithm is based on a generalization of the "shrinkage under random restrictions" result which was used to prove lower bounds [Sub61, Hås98]. Moreover, Santhanam [San10] showed the correlation of linear-size formulas with Parity is at most $2^{-\Omega(n)}$. This result was extended to linear-size formulas over B_2 [ST12] and $n^{2.49}$-size formulas over U_2 [KR12, CKK$^+$14]. Finally, Komargodski, Raz and Tal [KRT13] improved the analysis to give a function in P which has correlation at most $2^{-n^{\Omega(1)}}$ with $n^{2.99}$-size formulas over U_2; the analysis also implies a randomized $2^{n-n^{\Omega(1)}}$-time #SAT algorithm for $n^{2.99}$-size formulas.

We note that the above correlation bounds of formulas with Parity [San10, KRT13] are optimal, up to constant factors in the exponent. This can be seen from the following construction of approximate formulas for Parity. We divide the n inputs into l groups each of size n/l, use formulas of size $(n/l)^2$ to compute Parity for each group, and then take the disjunction of the outputs from all groups. This formula outputs 1 on most inputs, and moreover, whenever it outputs 0, it agrees with Parity. Thus the correlation of this formula with Parity is at least 2^{-l}. The formula size is at most $(n/l)^2 \cdot l + l < 2n^2/l$.

6.1.1 Our results

In this work, we consider correlation bounds and #SAT algorithms for small linear-size boolean circuits. We prove that, for U_2-circuits of size $3n - n^{\epsilon}$ for $\epsilon > 0.5$, the correlation with Parity is at most $2^{-n^{\Omega(1)}}$, and there is a #SAT algorithm running in time $2^{n-n^{\Omega(1)}}$; for U_2-circuits of size $3n - \epsilon n$ for $\epsilon > 0$, the correlation is at most $2^{-\Omega(n)}$, and there is a #SAT algorithm running in time $2^{n-\Omega(n)}$. We note that, the correlation bounds here are optimal, up to constant factors in the exponents. Using a similar construction as the one for formulas, we can get a circuit of size $3n - l$ which has correlation at least $2^{-\Omega(l)}$ with Parity.

For B_2-circuits, we also give a similar #SAT algorithms for circuit size almost $2.5n$, and show the average-case hardness of affine extractors against such circuits.

Table 6.1: Lower bounds and upper bounds for computing Parity

	Worst-Case Lower Bounds	Average-Case Upper / Lower Bounds	
AC^0	$s = \exp(n^{\theta(\frac{1}{d-1})})$ [Yao85, Hås86]	$\epsilon = 2^{-\Omega(n/(\log s)^{d-1})}$ [Hås12]	
De Morgan formulas	$s = n^{2-\theta(1)}$ [Sub61]	$\epsilon = 2^{-\Omega(n^2/s)}$	$\epsilon = 2^{-\Omega(n/\sqrt{s})}$ [BBC$^+$01, Rei11] $\epsilon = 2^{-\Omega(n/c^2)}$ for $s = cn$ [San10]
U_2-circuits	$s = 3n - \theta(1)$ [Sch74]	$\epsilon = 2^{-\Omega(3n-s)}$	$\epsilon = 2^{-\Omega((3n-s)^2/n)}$ (This work)

6.1.2 Preliminaries

Circuits

For two nodes u and v, we will write $u \to v$ if there is a directed edge from u to v. We call a circuit over the basis B as *B-circuit*. We consider two binary bases: the full basis B_2, which contains all boolean functions on two variables, and the basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$. Specifically, the basis B_2 contains the following 16 functions $f(x, y)$:

- six degenerating functions: $0,\ 1,\ x,\ \neg x,\ y,\ \neg y$;
- eight $\wedge$-type functions: $x \wedge y,\ x \vee y$, and their variations by negating one or both inputs;
- two $\oplus$-type functions: $x \oplus y,\ x \equiv y$.

The *size* of a circuit C, denoted by $s(C)$, is the number of gates in C. The circuit size of a function $f\colon \{0,1\}^n \to \{0,1\}$ is the minimal size of a boolean circuit computing f. For convenience, we define $\mu(C) = s(C) + N(C)$, where $N(C)$ is the number of inputs that C depends on. We let $\mu(C) = 0$ if C is constant, and $\mu(C) = 1$ if C is a literal.

A *restriction* ρ is a mapping from the inputs to $\{0, 1, *\}$. For a circuit C, the restricted circuit $C|_\rho$ is obtained by assigning $x_i = b$ for all x_i such that $\rho(x_i) = b \in \{0, 1\}$.

It is convenient to work with circuits with no redundant nodes or wires. We will call a non-constant circuit (over U_2 or B_2) *simplified* if it does not contain

1. a node labeled by a constant,
2. a gate labeled by a degenerating function, or
3. a non-output gate with out-degree 0, or

4. an input x and two gates u, v with three wires $x \to u$, $x \to v$, $u \to v$.

Lemma 6.1. *For any circuit C, there is a polynomial-time algorithm transforming C into an equivalent simplified circuit C' such that $s(C') \leq s(C)$ and $\mu(C') \leq \mu(C)$.*

Proof. Cases (1)-(3) are trivial. For case (4), suppose w is the other node with $w \to u$. If C is over B_2, then v computes a binary function of x and w; if C is over U_2, then v computes an $\wedge$-type function of x and w (because any $\oplus$-type function requires at least 3 gates). In either case, we can connect w directly to v, remove the wire $u \to v$, and change the label at v. By checking through each input and gate, the transformation can be done in polynomial time. □

Correlation

Definition 6.2. Let f and g be two boolean functions on n inputs. The correlation of f and g is defined as

$$\text{Corr}(f, g) = |\mathbf{Pr}[f(x) = g(x)] - \mathbf{Pr}[f(x) \neq g(x)]| = |2\mathbf{Pr}[f(x) = g(x)] - 1|\,,$$

where x is chosen uniformly from $\{0,1\}^n$.

Note that, f has correlation c with g if and only if f computes g or its negation correctly a fraction $(1 + c)/2$ of the inputs. Correlation bound is also referred to as average-case lower bound in the literature.

Decision Tree

A *decision tree* is a tree where (1) each internal node is labeled by a variable x, and has two outgoing edges labeled by $x = 0$ and $x = 1$, and (2) each leaf is labeled by a constant 0 or 1. A decision tree computes a boolean function by tracking the paths from the root to leaves. The *size* of a decision tree is the number of leaves in the tree.

A *parity decision tree* extends a decision tree such that each internal node is labeled by the parity function of a subset of variables. We insist that, for each path from the root to a leaf, the parities appearing in the internal nodes are linearly independent.

Concentration bounds

Theorem 6.3 (Chernoff bounds)**.** *[AB09] Let $\{X_i\}_{i=1}^n$ be mutually independent random variables over $\{0,1\}$, and let $\mu = \sum_{i=1}^n \mathbf{E}[X_i]$. Then, for every $c > 0$,*

$$\mathbf{Pr}\left[\left|\sum_{i=1}^{n} X_i - \mu\right| \geq c\mu\right] \leq 2 \cdot e^{-\min\{c^2/4,\, c/2\}\mu}.$$

A sequence of random variables $X_0, X_1, \ldots, X_n$ is called a *supermartingale* with respect to a sequence of random variables $R_1, \ldots, R_n$ if $\mathbf{E}[X_i \mid R_{i-1}, \ldots, R_1] \leq X_{i-1}$, for $1 \leq i \leq n$. The following is a variant of Azuma's inequality which holds for supermartingales with one-side bounded differences.

Lemma 6.4. *[CKK+14] Let $\{X_i\}_{i=0}^n$ be a supermartingale with respect to $\{R_i\}_{i=1}^n$. Let $Y_i = X_i - X_{i-1}$. If, for every $1 \leq i \leq n$, the random variable Y_i (conditioned on $R_{i-1}, \ldots, R_1$) assumes two values with equal probability, and there exists a constant $c_i \geq 0$ such that $Y_i \leq c_i$, then, for any $\lambda \geq 0$, we have*

$$\mathbf{Pr}[X_n - X_0 \geq \lambda] \leq \exp\left(-\frac{\lambda^2}{2\sum_{i=1}^n c_i^2}\right).$$

6.2 U_2-circuits

For U_2-circuits, the best known lower bound is $5n - o(n)$ by Iwama et al. [ILMR02], improving a $3n - c$ lower bound for Parity by Schnorr [Sch74], and a $4n - c$ lower bound for certain symmetric functions by Zwick [Zwi91]. All these lower bounds were proved using the gate-elimination method.

We generalize the gate-elimination method by defining a random process of restrictions under which the circuit size shrinks with high probability. This allows us to get a #SAT algorithm for U_2-circuits of size almost $3n$, and also prove a correlation bound with Parity.

6.2.1 Concentrated shrinkage under restrictions

We call an $\wedge$-type function of two variables as a *twig*. We now define a random process of restrictions, where, at each step, we uniformly assign a random value (0 or 1) to either a variable or a twig, and simplify the circuit by eliminating unnecessary gates. The choice of variables or twigs at each step is determined based on the following cases:

- If the circuit is a literal, choose the variable in the literal.
- If there is an input x with out-degree at least two, choose x.
- Otherwise, there must be a gate u which is fed by two variables each with out-degree 1; we choose u (which computes a twig).

Let C be a simplified U_2-circuit on inputs $x_1, \ldots, x_n$. Let C' be the simplified circuit obtained after one step of restriction. Then we have the following lemma on the shrinkage of $\mu(C)$.

Lemma 6.5. *Suppose $\mu(C) \geq 4$. Let $\sigma = \mu(C) - \mu(C')$. Then we have $\sigma \geq 3$, and $\mathbf{E}[\sigma] \geq 4$.*

Proof. We consider the following cases:

1. Suppose there is an input x_i feeding into two gates u and v. By Lemma 6.1, there are no edges between u and v. We randomly assign 0 or 1 to x_i, and consider the following sub-cases on the successors of u and v.

   (a) If u and v feed into two different successors, we have the following possibilities on gate elimination. (1) Under one assignment to x_i, both u and v become constants (eliminating x_i, u, v and the two successors), and under the other assignment to x_i, neither of u, v become constants (eliminating x_i, u, v); then we have $\mathbf{Pr}[\sigma \geq 5] \geq 1/2$, and $\sigma \geq 3$. (2) Under both assignments to x_i, exactly one of u, v becomes a constant (eliminating x_i, u, v and one successor); then $\sigma \geq 4$.

   (b) If u and v feed into one single common successor w, we have similar situations as above. (1) Under one assignment to x_i, both u and v become constants (eliminating x_i, u, v, w and a successor of w), and under the other assignment to x_i, neither of u, v become constants (eliminating x_i, u, v). (2) Under both assignments, exactly one of u, v becomes a constant (eliminating x_i, u, v, w).

2. When all inputs have out-degree 1, there must be a gate u fed by two inputs, say x_i and x_j. We randomly assign 0 or 1 to u; this will eliminate x_i, x_j, u and at least one successor of u. Then we have $\sigma \geq 4$.

In all cases we have $\sigma \geq 3$, and $\mathbf{E}[\sigma] \geq 4$. □

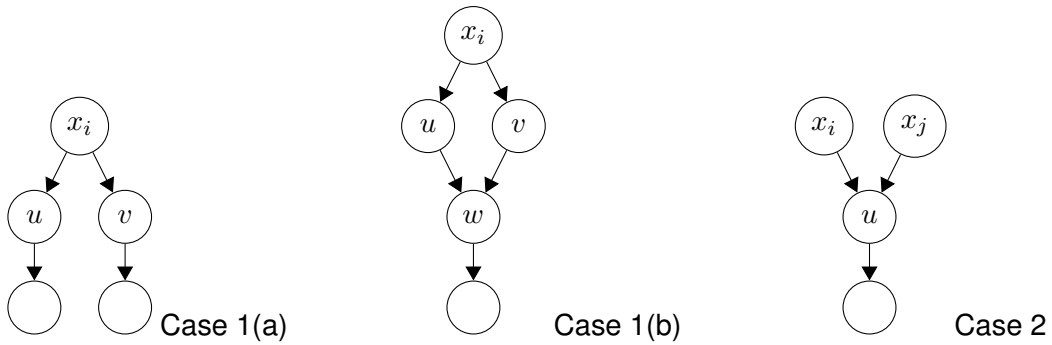

Figure 6.1: Cases in Lemma 6.5

We next consider the shrinkage of $\mu(C)$ under multiple steps of restrictions. Let $C_0 = C$, and, for $i = 1, \ldots, d$, let C_i be the circuit obtained at step i. For convenience, we let $\mu_i := \mu(C_i)$. Denote by R_i the random value assigned to the variable or twig at each step. We define a sequence of random variables $\{Z_i\}$ as follows:

$$Z_i = \begin{cases} \mu_i - (\mu_{i-1} - 4), & \mu_{i-1} \geq 4, \\ 0, & \mu_{i-1} < 4. \end{cases}$$

Note that $0 < \mu_{i-1} < 4$ holds only when C_{i-1} itself is a literal or a twig, which means C_i is a constant and $\mu_i = 0$.

Lemma 6.6. *Let $X_0 = 0$ and $X_i = \sum_{j=1}^{i} Z_i$. Then we have $Z_i \leq 1$, and $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$.*

Proof. By Lemma 6.5, conditioning on $R_1, \ldots, R_{i-1}$, when $\mu_{i-1} \geq 4$, we have $\mu_i \leq \mu_{i-1} - 3$ and $\mathbf{E}[\mu_i] \leq \mu_{i-1} - 4$. Therefore, we get $Z_i \leq 1$, $\mathbf{E}[Z_i] \leq 0$, and $\mathbf{E}[X_i] \leq X_{i-1}$. Thus $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$. □

Lemma 6.7. *For $\lambda \geq 0$,*

$$\mathbf{Pr}\left[\mu_d \geq \max\{\mu_0 - 4d + \lambda, 1\}\right] \leq \exp(-\lambda^2/2d).$$

Proof. Conditioning on $R_1, \ldots, R_{i-1}$, the variable Z_i assumes two values with equal probability. By Lemma 6.6, we have $\{X_i\}$ is a supermartingale with respect to $\{R_i\}$, and

$Z_i \leq c_i \equiv 1$. Applying the bound in Lemma 6.4, we have

$$\mathbf{Pr}\left[\sum_{i=1}^{d} Z_i \geq \lambda\right] \leq \exp\left(-\frac{\lambda^2}{2d}\right).$$

When $\mu_d > 0$, we have $\sum_{i=1}^{d} Z_i = \mu_d - \mu_0 + 4d$. Let E_1 be the event that $\mu_d > 0$, and let E_2 be the event that $\sum_{i=1}^{d} Z_i \geq \lambda$. Then the final probability is $\mathbf{Pr}[E_1 \wedge E_2] \leq \mathbf{Pr}[E_2] \leq \exp(-\lambda^2/2d)$.

□

6.2.2 #SAT algorithms

We now give a #SAT algorithm for circuits of size almost $3n$ based on the concentrated shrinkage in Section 6.2.1.

Theorem 6.8. *For U_2-circuits of size $s < 3n$, there is a deterministic #SAT algorithm running in time $2^{n-\Omega((3n-s)^2/n)}$.*

Proof. Let C be a circuit on n inputs $x_1, \ldots, x_n$ with size $s < 3n$. Let $\mu_0 := \mu(C) \leq s + n$.

We use the following procedure to construct a generalized decision tree, where each internal node might be labeled by a variable or a twig. We start with the root node and C.

- If C is a constant, label the current node by this constant and return.
- Use the cases in Section 6.2.1 to find either an input variable or a twig; denote it by u. Label the current node by u.
- Build two outgoing edges with labeled by $u = 0$ and $u = 1$. For each child node, simplify the circuit, and recurse.

We say an assignment to $x_1, \ldots, x_n$ is *consistent* with a path if it satisfies the assignments to individual variables and twigs along the path. Since an assignment could be consistent with only one path, the paths give a disjoint partitioning of the boolean cube $\{0,1\}^n$. To count the number of satisfying assignments for C, one can count the assignments consistent with each path, and sum over the paths with leaves labeled by 1. Edges along each path essentially defines a read-once 2CNF, for which counting can be done in polynomial time. We next only need to bound the total number of paths.

We wish to bound the probability that a random path has length bigger than $n-k$, for k to be chosen later. Let $\lambda = 4(n-k)-\mu_0+1$. Then by Lemma 6.7, at depth $n-k$, the restricted circuit becomes a constant with probability at least $1-\exp(-\lambda^2/2(n-k)) \geq 1-2^{-c\lambda^2/n}$ for some constant c. The number of paths with length bigger than $n-k$ is at most

$$2^{n-k} \cdot 2^{-c\lambda^2/n} \cdot 2^k \leq 2^{n-c\lambda^2/n}.$$

Therefore, the size of the decision tree is at most $2^{n-k}+2^{n-c\lambda^2/n}$. Choosing $k=(3n-s)/8$, both the size of the decision tree and the running time of the counting algorithm are bounded by $2^{n-\Omega((3n-s)^2/n)}$.

□

Corollary 6.9. *(1) For U_2-circuits of size $3n-\epsilon n$ with $\epsilon > 0$, there is a deterministic #SAT algorithm running in time $2^{n-\Omega(n)}$. (2) For U_2-circuits of size $3n-n^\epsilon$ with $\epsilon > 0.5$, there is a deterministic #SAT algorithm running in time $2^{n-n^{\Omega(1)}}$.*

6.2.3 Correlation with Parity

Schnorr [Sch74] proved a $3n-c$ lower bound for computing Parity using the following property: A simplified U_2-circuit computing Parity can not have any input with out-degree exactly 1. Indeed, if such an input x exists, one could substitute all other variables by constants such that the gate fed by x becomes a constant, and this makes the function independent of x.

We next generalize this lower bound to the average-case, and show that a U_2-circuit of size less than $3n$ does not correlate well with Parity. We will convert the decision tree constructed in the proof of Theorem 6.8 into a regular decision tree without twigs, and argue that the decision tree size will not blow up too much.

Lemma 6.10. *Any function computed by a U_2-circuit of size $s < 3n$ has a decision tree of size $2^{n-\Omega((3n-s)^2/n)}$.*

Proof. Let T be the (generalized) decision tree as constructed in the proof of Theorem 6.8. We next expand each node labeled by a twig into two nodes labeled by variables in the twig. For example, suppose we have a node labeled by $x \vee y$, we replace it by a node labeled by x and another node labeled by y under $x=0$; attach the original subtree under $x \vee y = 0$ to the new edge $y=0$, and attach the original subtree under $x \vee y = 1$ to both

$x = 1$ and $y = 1$. We denote the new decision tree by T', and we wish to bound the size of T'.

For a twig, such as $x \vee y$, we say an assignment is *good* if it allows three different configurations of the two variable involved, e.g., $x \vee y = 1$; otherwise, we say the assignment is *bad*, e.g., $x \vee y = 0$. We note that, for a path in T which has l twigs with good assignments, it will be replace by 2^l paths in T'.

We first consider paths in T of length bigger than $n - k$. As shown in Theorem 6.8, at depth $n - k$ of T, there are at most $2^{n-k} \cdot 2^{-c\lambda^2/n}$ nodes which are not leaves. Let v be such a node, and let l be the number of twigs on the path from the root to v. Then all paths in T passing through v will be replaced by at most $2^l \cdot 2^{k-l} = 2^k$ paths in T'. Therefore, all paths in T of length bigger than $n - k$ will be replaced by at most $2^{n-c\lambda^2/n}$ paths.

For a path in T of length at most $n - k$, let l be the number of twigs with good assignments along the path. If $l \leq k/2$, then this path is replaced by at most $2^{k/2}$ paths in T'. For all paths in T with length at most $n - k$ and $l \leq k/2$, they will be replaced by at most $2^{n-k} \cdot 2^{k/2} = 2^{n-k/2}$ paths.

Consider a path of length at most $n - k$ which has $l > k/2$ twigs with good assignments. After expanding the twigs, it is replaced by 2^l paths. When expanding a twig with a bad assignment, the path length increases by 1; when expanding a twig with a good assignment, the path becomes two paths with length increased by 0 and 1, respectively. Thus, by Chernoff bounds (choosing $\mu = l/2$ and $c = 1/2$ in Theorem 6.3), over the 2^l new paths, at most a fraction $2 \cdot e^{-l/32} < 2^{-k/d}$ (for some constant d) will have length bigger than $n - l + 3l/4 = n - l/4 < n - k/8$. Therefore, for all paths in T with length at most $n - k$ and having $l > k/2$ twigs with good assignments, they will be replaced by at most $2^{n-k/8}$ paths of length less than $n - k/8$, and at most $2^n \cdot 2^{-k/d} = 2^{n-k/d}$ longer paths.

Choosing $k = (3n - s)/8$ gives the result.

□

The following is a simple relationship between the size of a decision tree and its correlation with Parity. It was used in [San10, IMP12] to derive correlation bounds for de Morgan formulas and AC^0 circuits.

Lemma 6.11. *A decision tree of size 2^{n-k} has correlation at most 2^{-k} with Parity.*

Proof. Consider a path from the root to a leaf in the decision tree. If the path has length smaller than n, then, over the inputs that are consistent with this path, the correlation of

the decision tree with Parity is zero. The total number of paths of length n is at most 2^{n-k}, thus the decision tree can compute Parity correctly on at most a fraction $(1+2^{-k})/2$ of the inputs. □

Theorem 6.12. *Let C be a U_2-circuit of size $s < 3n$. Then its correlation with Parity is at most $2^{-\Omega((3n-s)^2/n)}$. In particular, for $s = 3n - \epsilon n$ with $\epsilon > 0$, the correlation is at most $2^{-\Omega(n)}$; for $s = 3n - n^{\epsilon}$ with $\epsilon > 0.5$, the correlation is at most $2^{-n^{\Omega(1)}}$.*

Remark. Iwama et al. [ILMR02] proves a $5n - o(n)$ lower bound for functions which are called strongly two-dependent, that is, functions such that fixing any two inputs results in four different sub-functions. The proof of this worst-case lower bound relies on a detailed case analysis on the circuit structure. It is not clear on how to generalize it to the average case; a major difficulty is that an approximate circuit may no longer have the "strongly two-dependent" property.

6.2.4 Applications

Lemma 6.10 shows that, a circuit of size almost $3n$ has a decision tree of non-trivial size. As applications of this property, one can get compression algorithms following [CKK+14] and Fourier concentration result as in [IK14].

Corollary 6.13. *There is an algorithm running in time $2^{O(n)}$ such that, given the truth table of an (unknown) n-input boolean circuit of size $s < 3n$, the algorithm produces an equivalent DNF of size $2^{n-\Omega((3n-s)^2/n)} \cdot \mathrm{poly}(n)$.*

Corollary 6.14. *Let f be a function computable by a boolean circuit of size $s < 3n$. Then,*

$$\sum_{A \subseteq [n]\colon |A| \geq n-\Omega((3n-s)^2/n)} \widehat{f}(A) \leq 2^{-\Omega((3n-s)^2/n)}.$$

6.3 B_2-circuits

In this section, we derive #SAT algorithms and correlation lower bounds for B_2-circuits of size almost $2.5n$.

6.3.1 Concentrated shrinkage and #SAT algorithms

Given a simplified circuit C, we will construct a generalized parity decision tree for C, where each internal node of the tree is labeled by either a twig or the parity of a subset of variables (this includes one variable as a special case). Starting with the root node of the tree and the circuit C, we use the following case analysis to identify labels and build branches recursively.

If the circuit is a constant, we label the current node by the constant, and the node becomes a leaf. If the circuit is a literal or a gate fed by two variables, we choose the variable in the literal or the circuit itself as the label, and build two branches of the decision tree with the constant circuits. Otherwise, consider a topological order on the gates of the circuit, and let u be the first gate which is either $\oplus$-type of out-degree at least 2 or $\wedge$-type.

1. If u is a $\oplus$-type gate of out-degree at least 2, then it computes a function $\oplus_{i\in I}x_i$ (or its negation) for some subset $I \in [n]$. We choose $\oplus_{i\in I}x_i$ as the label, and build two branches; for the circuit in the branch with assignment $\oplus_{i\in I}x_i = b$, we can replace u by a constant, and substitute an arbitrary variable x_j for $j \in I$ by a sub-circuit $\oplus_{i\in I\setminus\{j\}}x_i \oplus b$. In both branches, we can eliminate one input variable x_j, and at least 3 gates (u and its two successors).

2. If u is an $\wedge$-type gate fed by some $\oplus$-type gate v, suppose the other gate/input feeding into u is w.

   - If w has out-degree 1, then we choose the parity function computed at v as the label, and build two branches similar to the previous case. In one branch, we can eliminate some input x_j and two gates v and u; in the other branch, we can eliminate additionally w and a successor of u.
   - If w has out-degree at least 2, then it must be a variable. We choose w as the label, and build two branches. In one branch, we can eliminate w and its two successors (including u); in the other branch, we can eliminate two more gates (v and a successor of u).

3. If u is an $\wedge$-type gate fed by two inputs x_i and x_j where x_i has out-degree at least 2, then we choose x_i as the label and build two branches. In one branch, we can eliminate x_i and its two successors; in the other branch, we can eliminate one more gate (a successor of u).

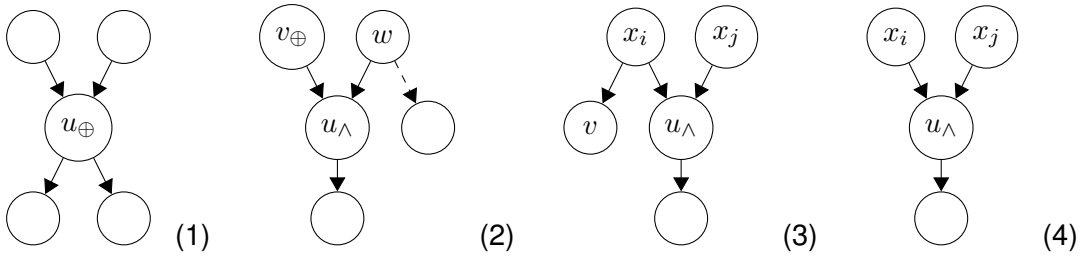

Figure 6.2: Cases for eliminating gates in B_2-circuits

4. If u is an $\wedge$-type gate fed by two inputs each of out-degree 1, then choose the twig computed at u as the label. In both branches, we can eliminate x_i, x_j, u and a successor of u.

Consider a random path from the root of the decision tree to its leaves. Let $C_0 = C$, and let C_i be the restricted circuit obtained at depth i. Let $\mu_i := \mu(C_i)$. The following lemma follows directly from the above case analysis.

Lemma 6.15. *If $\mu_i > 4$, then $\mu_i - \mu_{i+1} \geq 3$, and $\mathbf{E}[\mu_i - \mu_{i+1}] \geq 3.5$. If $\mu_i \leq 4$, then $\mu_{i+1} = 0$.*

Then we have the following concentrated shrinkage; the proof is essentially the same as Lemma 6.7.

Lemma 6.16. *For $\lambda \geq 0$,*

$$\mathbf{Pr}\left[\mu_d \geq \max\{\mu_0 - 3.5d + \lambda, 1\}\right] \leq \exp(-\lambda^2/2d).$$

Theorem 6.17. *For B_2-circuits of size $s < 2.5n$, there is a deterministic #SAT algorithm running in time $2^{n-\Omega((2.5n-s)^2/n)}$. In particular, for $s = 2.5n - \epsilon n$ with $\epsilon > 0$, the algorithm runs in time $2^{n-\Omega(n)}$; for $s = 2.5n - n^\epsilon$ with $\epsilon > 0.5$, the algorithm runs in time $2^{n-n^{\Omega(1)}}$.*

The proof, which we omit here, is similar to the proof of Theorem 6.8. The algorithms runs by constructing the generalized parity decision tree as stated above, and the count the number of satisfying assignments for each path of the tree. The size of the tree can be bounded by applying Lemma 6.16.

6.3.2 Correlation bounds

Demenkov and Kulikov [DK11] proved that affine dispersers for sources of dimension d requires B_2-circuits of size $3n - \Omega(d)$. In the following, we extend this to show that affine extractors have small correlations with B_2-circuits of size almost $2.5n$.

Definition 6.18. Let F_2 be the finite field with elements $\{0, 1\}$. A function $\mathsf{AE}\colon F_2^n \to F_2$ is a (k, ϵ)-*affine extractor* if for any uniform distribution X over some k-dimensional affine subspace of F_2^n,

$$|\mathbf{Pr}[\mathsf{AE}(X) = 1] - 1/2| \leq \epsilon.$$

We will need the following known constructions of affine extractors.

Theorem 6.19. *[Bou07, Yeh11, Li11] (1) For any $\delta > 0$ there exists a polynomial-time computable (k, ϵ)-affine extractor $\mathsf{AE}_1\colon \{0,1\}^n \to \{0,1\}$ with $k = \delta n$ and $\epsilon = 2^{-\Omega(n)}$. (2) There exists a constant $c > 0$ and a polynomial-time computable (k, ϵ)-affine extractor $\mathsf{AE}_2\colon \{0,1\}^n \to \{0,1\}$ with $k = cn/\sqrt{\log\log n}$ and $\epsilon = 2^{-n^{\Omega(1)}}$.*

We will prove correlation bounds using the following representation of small B_2-circuits by parity decision trees. Although the correlation bounds can be proved more directly, the representation with parity decision trees might be of independent interests.

Lemma 6.20. *Any function computed by a B_2-circuit of size $s < 2.5n$ is computable by a parity decision tree of size $2^{n-\Omega((2.5n-s)^2/n)}$.*

The proof is this lemma is almost the same as the proof of Lemma 6.10. That is, one can construct a generalized parity decision tree which might have twigs, and then expand the twigs and argue that the tree size will not blow up too much. We note the property that, when we restrict a twig, the two variables of the twig will be completely eliminated from the circuit; when we restrict a parity function, one of the variables is substituted, and thus all restrictions are linearly independent.

Lemma 6.21. *(1) For any $\delta > 0$, a parity decision tree of size 2^{n-k} for $k = \delta n$ has correlation at most $2^{-\Omega(n)}$ with AE_1. (2) There is a constant $c > 0$ such that a parity decision tree of size 2^{n-k} for $k = cn/\sqrt{\log\log n}$ has correlation at most $2^{-n^{\Omega(1)}}$ with AE_2.*

Proof. Consider a parity decision tree of size 2^{n-k} for $k = \delta n$. Note that all paths in the tree gives a disjoint partitioning of the boolean cube $\{0, 1\}^n$.

For a path of length at most $n - k/2$, the inputs that are consistent with the path form an affine subspace of dimension at least $k/2$. Over all such short paths, the parity decision tree computes AE_1 correctly on at most $2^n \cdot (1/2 + 2^{-\Omega(n)})$ of the inputs. For long paths of length larger than $n - k/2$, there are at most 2^{n-k} of them, and the number of inputs that are consistent with such paths is at most $2^{n-k} \cdot 2^{k/2} = 2^{n-k/2}$. Therefore, the parity decision tree computes AE_1 correctly on at most a fraction $1/2 + 2^{-\Omega(n)} + 2^{-k/2} = 1/2 + 2^{-\Omega(n)}$ of the inputs.

The proof for the second case is similar. □

The next theorem follows directly by combining Lemma 6.20 and Lemma 6.21.

Theorem 6.22. *(1) For any $\delta > 0$ and any B_2-circuit of size $2.5n - \delta n$, its correlation with AE_1 is at most $2^{-\Omega(n)}$. (2) There exists a constant $c > 0$ such that, for any B_2-circuit of size $2.5n - cn/\sqrt[4]{\log \log n}$, its correlation with AE_2 is at most $2^{-n^{\Omega(1)}}$.*

Chapter 7

Lower Bounds Against Weakly-Uniform Circuits

An ongoing line of research has shown super-polynomial lower bounds for uniform and slightly-non-uniform small-depth threshold and arithmetic circuits [All99, KP09, JS11]. We give a unified framework that captures and improves each of the previous results. Our main results are that PERMANENT does not have threshold circuits of the following kinds.

1. Depth $O(1)$, $n^{o(1)}$ bits of non-uniformity, and size $n^{O(1)}$.
2. Depth $O(1)$, $\mathrm{polylog}(n)$ bits of non-uniformity, and size $s(n)$ such that for all constants c the c-fold composition of s, $s^{(c)}(n)$, is less than 2^n.
3. Depth $o(\log \log n)$, $\mathrm{polylog}(n)$ bits of non-uniformity, and size $n^{O(1)}$.

(1) strengthens a result of Jansen and Santhanam [JS11], who obtained similar parameters but for *arithmetic* circuits of constant depth rather than Boolean threshold circuits. (2) and (3) strengthen results of Allender [All99] and Koiran and Perifel [KP09], respectively, who obtained results with similar parameters but for completely uniform circuits. Our main technical contribution is to simplify and unify earlier proofs in this area, and adapt the proofs to handle some amount of non-uniformity. We also develop a notion of circuits with a small amount of non-uniformity that naturally interpolates between fully uniform and fully non-uniform circuits. We use this notion, which we term *weak uniformity*, rather than the earlier and essentially equivalent notion of *succinctness* used by Jansen and Santhanam because

the notion of weak uniformity more fully and easily interpolates between full uniformity and non-uniformity of circuits.

7.1 Introduction

Understanding the power and limitation of efficient algorithms is the major goal of complexity theory, with the "P vs. NP" problem being the most famous open question in the area. While proving that no NP-complete problem has a uniform polynomial-time algorithm would suffice for separating P and NP, a considerable amount of effort was put into the more ambitious goal of trying to show that no NP-complete problem can be decided by even a *nonuniform* family of polynomial-size Boolean circuits. More generally, an important goal in complexity theory has been to prove strong (exponential or super-polynomial) circuit lower bounds for "natural" computational problems that may come from complexity classes larger than NP, e.g., the class NEXP of languages decidable in nondeterministic exponential time. By the counting argument of Shannon [Sha49], a randomly chosen n-variate Boolean function requires circuits of exponential size. However, the best currently known circuit lower bounds for *explicit* problems are only linear for NP problems [LR01, IM02], and polynomial for problems in the polynomial-time hierarchy PH [Kan82] and counting hierarchy CH [Tod91]. Super-polynomial lower bounds are known only for classes such as MAEXP [BFT98, MVW99].

To make progress, researchers introduced various restrictions on the circuit classes. In particular, for Boolean circuits of *constant* depth, with NOT and unbounded fan-in AND and OR gates (AC^0 circuits), exponential lower bounds are known for the PARITY function [FSS84, Yao85, Hås86]. For constant-depth circuits that additionally have (unbounded fan-in) MOD_p gates, one also needs exponential size to compute the MOD_q function, for any distinct primes p and q [Raz87, Smo87]. With little progress for decades, Williams [Wil11] has recently shown that a problem in NEXP is not computable by polynomial-size ACC^0 circuits, which are constant-depth circuits with NOT gates and unbounded fan-in AND, OR and MOD_m gates, for any integer $m > 1$. However, no lower bounds are known for the class TC^0 of constant-depth *threshold circuits* with unbounded fan-in majority gates[1], a class of circuits that includes ACC^0 circuits as a sub-class (see, e.g., [BIS90]).

[1] A plausible explanation of this "barrier" is given by the "natural proofs" framework of [RR97], who argue it is hard to prove lower bounds against the circuit classes that are powerful enough to implement cryptography.

To make more progress, another restriction has been added: *uniformity* of circuits. Roughly speaking, a circuit family is called uniform if there is an efficient algorithm that can construct any circuit from the family. There are two natural variations of this idea. One can ask for an algorithm that outputs the entire circuit in time polynomial in the circuit size; this notion of uniformity is known as P-uniformity. In the more restricted notion, one asks for an algorithm that describes the local structure of the circuit: given two gate names, such an algorithm determines if one gate is the input to the other gate, as well as determines the types of the gates, in time linear (or polynomial) in the input size (which is logarithmic or polylogarithmic time in the size of the circuit described by the algorithm); such an algorithm is said to decide the *direct-connection language* of the given circuit. This restricted notion is called DLOGTIME- (or POLYLOGTIME-) uniformity and is commonly viewed as the viewed as "the right" notion of uniformity for circuit classes below P [Ruz81, BIS90, AG94]. We will use the notion of POLYLOGTIME-uniformity by default, and, for brevity, will omit the word POLYLOGTIME.

It is easy to show (by diagonalization) that, for any fixed exponential function $s(n) = 2^{n^c}$ for a constant $c \geq 1$, there is a language in EXP (deterministic exponential time) that is not computable by a uniform (even P-uniform) family of Boolean $s(n)$-size circuits.[2] Similarly, as observed in [All99], a PSPACE-complete language requires exponential-size uniform TC^0 circuits – due to the space hierarchy theorem and the fact that uniform TC^0 circuits can be decided by a logarithmic space Turing machine. For the smaller complexity class $\#\mathsf{P} \subseteq \mathsf{PSPACE}$, Allender and Gore [AG94] showed PERMANENT (which is complete for $\#\mathsf{P}$ [Val79]) is not computable by uniform ACC^0 circuits of sub-exponential size. Later, Allender [All99] proved that PERMANENT cannot be computed by uniform TC^0 circuits of size $s(n)$ for any function s such that, for all k, $s^{(k)}(n) = o(2^n)$ (where $s^{(k)}$ means the function s composed with itself k times). Finally, Koiran and Perifel [KP09] extended this result to show that PERMANENT is not computed by polynomial-size uniform threshold circuits of depth $o(\log \log n)$.

Recently, Jansen and Santhanam [JS11] have proposed a natural relaxation of uniformity, termed *succinctness*, which allows one to interpolate between non-uniformity and uniformity. According to [JS11], a family of $s(n)$-size circuits $\{C_n\}$ is succinct if the direct-connection language of C_n is decided by some circuit of size $s(n)^{o(1)}$. In other words, while

[2] Unlike the nonuniform setting, where every n-variate Boolean function is computable by a circuit of size about $2^n/n$ [Lup58], *uniform* circuit lower bounds can be $> 2^n$.

there may not be an efficient algorithm for describing the local structure of a given $s(n)$-size circuit C_n, the local structure of C_n can be described by a *non-uniform* circuit of size $s(n)^{o(1)}$. Note that if we allow the non-uniform circuit to be of size $s(n)$, then the family of circuits $\{C_n\}$ would be completely non-uniform. So, intuitively, the restriction to the size $s(n)^{o(1)}$ makes the notion of succinctness close to that of non-uniformity.

The main result of [JS11] is that PERMANENT does not have succinct polynomial-size *arithmetic* circuits of constant depth, where arithmetic circuits have unbounded fan-in addition and multiplication gates and operate over integers. While relaxing the notion of uniformity, [JS11] were only able to prove a lower bound for the *weaker* circuit class, as polynomial-size constant-depth arithmetic circuits can be simulated by polynomial-size TC^0 circuits. A natural next step was to prove a super-polynomial lower bound for PERMANENT against succinct TC^0 circuits. This is achieved in the present work.

7.1.1 Weakly-Uniform Circuit Families

Before stating our main results, we introduce the notion of *weakly-uniform* circuits which we use to phrase our results. Recall that the direct-connection language for a circuit describes the local structure of the circuit. The language answers questions such as – is a particular gate an AND gate, or is the output of a particular gate connected to the input of another particular gate? For the definition of weakly-uniform circuits an important point is that for circuits $\{C_n\}$ of size $s(n)$, queries to the direct-connection language L_{dc} for $\{C_n\}$ are of length $O(\log s(n))$ – because any gate in the circuit can be indexed by this many bits. That is, let m denote the input length of queries to L_{dc}; then queries concerning C_n are of length $m = O(\log s(n))$. We define weakly-uniform circuits as follows.

Definition 7.1. Let $\{C_n\}$ be a circuit family of size $s(n)$, and let m be the input length of its direct-connection language L_{dc}. Then for a function $\alpha : \mathbb{N} \to \mathbb{N}$, $\{C_n\}$ is α-*weakly-uniform* if L_{dc} can be decided using $\alpha(m)$ bits of advice in $\mathrm{poly}(m + \alpha(m))$ time, where the advice string only depends on the input size m.

More precise definitions of the direct connection language and weakly-uniform circuits are given in Section 7.1.5. Readers unfamiliar with definitions of uniform circuits are encouraged to consult that section before continuing. We stress that here we have parameterized the advice as a function of the input length for L_{dc}, which is a logarithm of the size of the circuit. We view this as the correct definition because the advice is associated

with the machine deciding L_{dc}, as opposed to being associated with the circuits C_n directly. With $m = O(\log s(n))$, as a function of the input length n, a circuit of size $s(n)$ that is $2^{o(m)}$-weakly-uniform uses $s^{o(1)}(n)$ bits of advice for its direct-connection language, and a circuit that is $m^{O(1)}$-weakly-uniform uses $\mathrm{polylog}(s(n))$ bits of advice for its direct-connection language. The notion of α-weakly-uniform is essentially equivalent to the notion of α-succinct introduced in [JS11]. One notable difference is that our definition fully interpolates between non-uniform and uniform circuits; setting $\alpha(m) = 2^m$ in our definition corresponds to fully non-uniform circuits, while setting $\alpha(m) = 0$ corresponds to fully uniform circuits. On the other hand, there is no setting of parameters for succinct circuits which corresponds to fully uniform circuits. See Section 7.1.5 for details.

7.1.2 Our Main Results

We improve upon [JS11] by showing that PERMANENT does not have succinct polynomial-size TC^0 circuits, in our terminology – weakly-uniform, polynomial-size TC^0 circuits. In addition to strengthening the main result from [JS11], we also give a simpler proof. Our argument is quite general and allows us to extend to the weakly-uniform setting all previously known uniform circuit lower bounds of [AG94, All99, KP09]. Our main results are as below. First, we strengthen the lower bound of [JS11].

Theorem 7.2. PERMANENT *is not computable by* $2^{o(m)}$*-weakly-uniform* poly*-size* TC^0 *circuits, where* $m = O(\log n)$*.*

We state the weak-uniformity in terms of the input length m to the direct-connection language due to the reasons stated above. Note that in Theorem 7.2, the amount of advice, the weak-uniformity, is $n^{o(1)}$. Let us call a function $s(n)$ *sub-subexponential* if, for any constant $k > 0$, we have that the k-wise composition $s^{(k)}(n) \leq 2^{n^{o(1)}}$. We use subsubexp to denote the class of all sub-subexponential functions $s(n)$. We extend a result of Allender [All99] to the "weakly-uniform" setting.

Theorem 7.3. PERMANENT *is not computable by* $m^{O(1)}$*-weakly-uniform* subsubexp*-size* TC^0 *circuits, where* $m = O(\log s(n))$*.*

Note that the amount of advice in Theorem 7.3 is $\mathrm{polylog}(s(n))$. We extend the result of [KP09] to the weakly-uniform setting as well.

Theorem 7.4. PERMANENT *is not computable by* $m^{O(1)}$*-weakly-uniform* poly*-size threshold circuits of depth* $o(\log\log n)$*, where* $m = O(\log n)$*.*

Note that the amount of advice in Theorem 7.4 is $\text{polylog}(n)$.

Other Results

The proofs of Theorems 7.2, 7.3, and 7.4 all use the same strategy, but require different settings of parameters. We also state a single parameterized result that implies a tradeoff between the amount of non-uniformity, circuit size, and depth. The precise statement is given in Section 7.4 and implies Theorems 7.2, 7.3, and 7.4. Finally, we obtain lower bounds for weakly-uniform ACC^0, AC^0, and general circuits. These results are stated and proved in Section 7.5.

7.1.3 Our Techniques

The proofs of our main results are *indirect diagonalization* arguments. Such arguments begin by assuming the intended lower bound does not hold, and then using this assumption and known separation results to obtain a contradiction. The choice of which known separation to use points the direction for much of the rest of the argument. We give two different proofs of our main results, resulting from using two different known separations as the basis of an indirect diagonalization argument. Both proofs are similar, but each implies corollaries that cannot be achieved by the other. We give an outline of how each of the two different proofs comes to a contradiction if we assume PERMANENT has $2^{o(m)}$-weakly-uniform polynomial-size constant-depth threshold circuits (Theorem 7.2). For the setting of poly-size circuits, m, the input length for the direct connection language of the circuit, is $O(\log n)$ – meaning the non-uniformity is $n^{o(1)}$. As this fact is key to the proofs, we refer to $n^{o(1)}$-weak-uniformity throughout the proof outlines. Before outlining the two different proofs we discuss the key ingredients that are common to both proofs.

Collapse of the counting hierarchy if PERMANENT is easy The counting hierarchy is an analogue of the polynomial hierarchy, where counting is used in place of non-determinism. There are a number of equivalent formulations of the counting hierarchy. One is based on PP machines – nondeterministic machines where an input x is defined to be accepted by

the machine if a majority of computation paths accept. The d^{th} level of the counting hierarchy, CH_d, is defined as $\mathsf{PP}^{\mathsf{CH}_{d-1}}$ where $\mathsf{CH}_0 = \mathsf{P}$. One of the main properties we use of PERMANENT is that PERMANENT is hard for PP, and thus if PERMANENT is easy then the counting hierarchy collapses. For example, suppose Theorem 7.2 does not hold, namely that PERMANENT has constant-depth poly-size threshold circuits that use $n^{o(1)}$ bits of advice. Consider a language in $\mathsf{CH}_2 = \mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}} = \mathsf{PP}^{\mathsf{PP}}$. Using the paddability of PERMANENT and the hardness of PERMANENT for PP, all of the PP oracle queries can be replaced by queries to a single $n^{o(1)}$-weakly-uniform poly-size constant-depth threshold circuit. The CH_2 computation can be replaced, then, by a PP machine using $n^{o(1)}$ bits of advice. Using the assumed easiness of PERMANENT again converts this PP computation with advice into a poly-size threshold circuit that uses $n^{o(1)}$ additional bits of advice. The argument can be carried out inductively. A language in CH_d is a PP computation with an oracle to CH_{d-1}. If PERMANENT is easy, the CH_{d-1} oracle queries can be solved using a poly-size constant-depth threshold circuit that uses $n^{o(1)}$ bits of advice. Plugging the circuit into the definition of CH_d results in a PP computation that uses advice, which in turn is converted into a poly-size constant-depth threshold circuit that uses $n^{o(1)}$ additional bits of advice. The resulting constant-depth threshold circuit requires $N^{o(1)}$ bits of advice for each level in the induction, where N is the length of queries to PERMANENT . Furthermore, N grows by potentially a polynomial factor at each level. As the advice needed is $N^{o(1)}$ and there are a constant number of levels to collapse, the total amount of advice is still $n^{o(1)}$.

Equivalent notions of CH Our proofs use two alternate definitions of the counting hierarchy. First, CH is equal to the set of languages decided by uniform constant-depth threshold circuits of size $2^{\mathsf{poly}(n)}$. Second, CH is equal to the set of languages decided by *threshold Turing machines* which run in polynomial time and make a constant number of alternations. Threshold Turing machines are based on nondeterministic Turing machines, but where each configuration is labeled, depending on the state of the finite control, an existential, universal, or threshold configuration. These configurations are defined to be accepting if, respectively, at least one computation path from the configuration accepts, all paths accept, or a majority of computation paths accept. The machine makes an alternation when there is a switch in the configuration's type between existential, universal, or threshold. In the equivalence between these two definitions of CH, the existential, universal, and threshold configurations of poly-time threshold Turing machines are essentially

equivalent to the unbounded fan-in AND, OR, and majority gates of an exponential-size uniform threshold circuit. The equivalence between uniform threshold circuits and threshold Turing machines extends to the setting of weak uniformity. In particular, for the setting of Theorem 7.2 considered here, a $n^{o(1)}$-weakly-uniform constant-depth poly-size threshold circuit can be simulated by a linear-time threshold machine that uses a constant number of alternations and $n^{o(1)}$ bits of advice [3].

First proof of Theorem 7.2 Our first proof takes as its starting point a time hierarchy theorem for threshold Turing machines. If PERMANENT has constant-depth $n^{o(1)}$-weakly-uniform threshold circuits, our discussion above implies that every language in CH also has such circuits. On the other hand, we show that the assumed easiness of PERMANENT together with the time hierarchy for threshold Turing machines imply a language in CH that is hard for these circuits – a contradiction. Consider a language L_{P} that is complete for P. By the known uniform reductions from P to PERMANENT and the assumed easiness of PERMANENT , L_{P} has constant-depth $n^{o(1)}$-weakly-uniform threshold circuits of depth d, for some constant d. By the completeness of L_{P} for P, any language decided by poly-size $n^{o(1)}$-weakly-uniform circuits also has $n^{o(1)}$-weakly-uniform threshold circuits of depth d, and therefore also can be decided by a linear-time threshold Turing machine that makes at most d alternations and uses $n^{o(1)}$ bits of advice. The time hierarchy theorem for such threshold Turing machines states that there is a language L_{hard} computable by a poly-time threshold Turing machine that makes d alternations and differs from all such languages, thus differing from all languages computable by poly-size $n^{o(1)}$-weakly-uniform circuits. L_{hard} differs from such machines but at the same time resides in CH, which can (by the assumed easiness of PERMANENT and the resulting collapse of CH) be computed by poly-size $n^{o(1)}$-weakly-uniform constant-depth threshold circuits – a contradiction.

Second proof of Theorem 7.2 The first proof uses the assumed easiness of PERMANENT to construct a language in CH that is hard for constant-depth $n^{o(1)}$-weakly-uniform circuits of any polynomial size. For the second proof, we start with an *unconditional* lower bound that holds for a *fixed* polynomial. Namely, for any constant k there exists a language L_{hard} in P^{PP} that cannot be computed by circuits of size n^k. As with the first proof, we also

[3] Fully uniform constant-depth poly-size threshold circuits can be simulated by threshold Turing machines running in polylog time, but the relaxed $n^{o(1)}$-weak-uniformity translates to threshold Turing machines with longer running time.

use the easiness of PERMANENT to work in the other direction – to show that if PERMANENT has $n^{o(1)}$-weakly-uniform constant-depth threshold circuits then L_{hard} can be computed by circuits of size less than n^k, as follows. The easiness of PERMANENT implies that any language in P^{PP}, in particular L_{hard}, can be computed by $n^{o(1)}$-weakly-uniform constant-depth threshold circuits C_{hard} of polynomial size. The circuits C_{hard} can in turn be collapsed in much the same manner we discussed collapsing CH above. The first level of threshold gates closest to the inputs can be viewed as PP questions of size $\mathrm{poly}(\log(n) + n^{o(1)})$; using the assumed easiness of PERMANENT a circuit C_1 of size $n^{o(1)}$ can be used in place of the threshold gates on the first level. A similar argument shows that the second level of threshold gates reduce to PP questions of size $\mathrm{poly}(|C_1|)$, which can be replaced by a circuit of size $\mathrm{poly}(\mathrm{poly}(|C_1|))$ using the assumed easiness of PERMANENT . This process is repeated for each level of threshold gates in C_{hard}. If C_{hard} has depth d, we obtain a circuit of size $p^{(d)}(\log(n) + n^{o(1)}) + O(n)$ for some polynomial p after iterating for each level of threshold gates in C_{hard}. The conclusion is a contradiction – we have constructed a circuit of size $O(n)$ for computing C_{hard} although it should require size n^k.

Notes on the proofs One of the key ingredients for both proofs is that if PERMANENT is easy then the counting hierarchy collapses, even in the presence of $n^{o(1)}$ bits of advice. Equivalently, weakly-uniform circuits for PERMANENT imply the collapse of weakly-uniform threshold circuits. The same basic argument as those given above is used for each of Theorems 7.2, 7.3, and 7.4. In fact, for our second proof we prove a single parameterized statement that implies the theorems as corollaries. We have phrased our first proof in terms of threshold Turing machines with advice, and our second proof in terms of weakly-uniform threshold circuits. Due to the equivalence between the two models, both proofs could be given in terms of either model. The Turing machine model is natural for the first proof due to the reliance on a hierarchy theorem for Turing machines for L_{hard}. The circuit model is natural for the second proof due to its use of a circuit lower bound for L_{hard}.

7.1.4 Relation to the previous work

A similar indirect-diagonalization strategy was used (explicitly or implicitly) in all previous papers showing uniform or weakly-uniform circuit lower bounds for PERMANENT [AG94, All99, KP09, JS11]. Our proofs are most closely related to those of [All99, KP09]. The main difference is that we work in the weakly-uniform setting, which means that we need to

handle a certain amount of non-uniform advice. To that end, we have adapted the method of indirect diagonalization, making it modular (as outlined above) and sufficiently general to work also in the setting with advice. Due to this generality of our proof argument, we are able to extend the aforementioned lower bounds from the uniform setting to the weakly uniform setting. The approach adopted by [JS11] goes via the well-known connection between derandomization and circuit lower bounds (cf. [HS82, KI04, Agr05]). Since the authors of [JS11] work with the algebraic problem of Polynomial Identity Testing (given an arithmetic circuit computing some polynomial over integers, decide if the polynomial is identically zero), their final lower bounds are also in the algebraic setting: for weakly-uniform arithmetic constant-depth circuits. By making the diagonalization arguments in [JS11] more explicit (along the lines of [All99, KP09]), we are able to get the lower bound for weakly-uniform Boolean (TC^0) circuits, thereby both strengthening the results and simplifying the proofs from [JS11].

The remainder of the chapter. We give the necessary background in Section 7.1.5. Section 7.2 provides the tools needed for our proofs. These tools are then used in Sections 7.3 and 7.4 to give the two proofs of our main results (Theorems 7.2–7.4 above). We give other weakly-uniform circuit lower bounds in Section 7.5. We give concluding remarks in Section 7.6.

7.1.5 Preliminaries

Circuits

Recall that a *Boolean circuit* C_n on n inputs $x_1, \ldots, x_n$ is a directed acyclic graph with one single output gate (the node of out-degree 0), n nodes of in-degree 0 (input gates labeled $x_1, \ldots, x_n$), and internal nodes of in-degree 2 (for AND and OR gates) or 1 (for NOT gates). The *size* of the circuit C_n is defined to be the number of gates, and is denoted by $|C_n|$. For a function $s : \mathbb{N} \to \mathbb{N}$ and a circuit family $\{C_n\}_{n\geq 0}$, we say that the circuit family is in $\mathsf{SIZE}(s)$, if for all sufficiently large n we have $|C_n| \leq s(n)$. The *depth* of a circuit C_n is defined as the length of a longest path from some input gate to the output gate. We will be talking about constant-depth circuits, in which case we allow all gates (other than the NOT gates) to have unbounded fan-in. In addition to AND and OR, we may have other types of gates: MAJ (which is 1 iff more than half of its inputs are 1), or MOD_m gate for some

integer $m > 0$ (which is 1 iff the integer sum of the inputs is divisible by m). AC^0 circuits are constant-depth Boolean circuits with NOT gates, unbounded fan-in AND and OR gates, and constant gates 0 and 1. For a positive integer m, $\mathsf{AC}^0[m]$ is the set of languages decided by AC^0 circuits that are extended with unbounded fan-in MOD_m gates; ACC^0 is the union of AC^0 over all m. Finally, TC^0 circuits are AC^0 circuits extended with unbounded fan-in MAJ (or threshold) gates, where MAJ(x) = 1 if and only if more than half of the bits in x are 1. For a function $s : \mathbb{N} \to \mathbb{N}$ and a circuit type $\mathcal{C} \in \{\mathsf{AC}^0, \mathsf{ACC}^0, \mathsf{TC}^0\}$, we denote by $\mathcal{C}(s)$ the class of families of $s(n)$-size n-input circuits of type $\mathcal{C}$. When $s(n)$ is a polynomial in n, we may drop it and simply write $\mathcal{C}$ to denote the class of polynomial-size $\mathcal{C}$-circuits. Finally, we drop the superscript 0 in $\mathsf{AC}^0, \mathsf{ACC}^0$, and TC^0, when we want to talk about the corresponding type of circuits where the depth $d(n)$ may be a function of the input size n.

Weakly-uniform circuit families

Following [Ruz81, AG94], we define the *direct connection language* of a circuit family $\{C_n\}$ as $L_{dc} = \{(n, g, h) \colon g = h$ and g is a gate in C_n, or $g \neq h$ and h is an input to $g\}$, where n is in binary representation, and each of g and h is a binary string encoding both the type and the name of the gate. The *name* of a gate is just its indexing label. The *type* of a gate could be constant 0 or 1, Boolean logic gate NOT, AND, or OR, majority gate MAJ, modulo gate MOD_m for some integer m, or input $x_1, x_2, \ldots, x_n$. For a circuit family of size $s(n)$, we need $c_0 \log s(n)$ bits to encode (n, g, h), where c_0 is a small constant at most 4. A circuit family $\{C_n\}$ is *uniform* [BIS90, AG94] if its direct connection language is decidable in time polynomial in its input length $|(n, g, h)|$; this was referred to as POLYLOGTIME-uniformity in [AG94]. We say a function $f(n)$ is *constructible* if there is a deterministic TM that computes $f(n)$ in binary in time $O(f(n))$, when given n in binary as the input[4]. Following [JS11], for a constructible function $\alpha : \mathbb{N} \to \mathbb{N}$, we say that a circuit family $\{C_n\}$ of size $s(n)$ is *α-succinct* if its direct connection language L_{dc} is in $\mathsf{SIZE}(\alpha)$; i.e., L_{dc} has (non-uniform) Boolean circuits of size $\alpha(m)$, where $m = c_0 \log s(n)$ is the input size for L_{dc}. Trivially, for $\alpha(m) \geq 2^m$, every circuit family is α-succinct. The notion becomes nontrivial when $\alpha(m) \ll 2^m/m$. We will use $\alpha(m) = 2^{o(m)}$ (slightly succinct) and $\alpha(m) = m^{O(1)}$

[4]We note that $f(n)$ is constructible in our sense if and only if $2^{f(n)}$ is constructible according to Allender's definition in [All99].

(highly succinct). We stress that here we have parameterized the succinctness as a function of the logarithm of the size of the circuit. As a function of the input length n, a circuit of size $s(n)$ is slightly succinct if the direct connection language is decided by a circuit of size $s^{o(1)}(n)$, and is highly succinct if the direct connection language is decided by a circuit of size $\mathrm{poly}\log(s(n))$. We recall the definition of Turing machines with advice from [KL82]. Given functions $t\colon \mathbb{N} \to \mathbb{N}$ and $\alpha\colon \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{DTIME}(t)/\alpha$, if there is a deterministic Turing machine M and a sequence of advice strings $\{a_n\}$ of length $\alpha(n)$ such that, for any $x \in \{0,1\}^n$, machine M on inputs (x, a_n) decides whether $x \in L$ in time $t(n + \alpha(n))$. If the function $t(n + m)$ is upper-bounded by a polynomial in $n + m$, we say that $L \in \mathsf{P}/\alpha$.

Definition 7.5 (Restatement of Definition 7.1)**.** A circuit family $\{C_n\}$ of size $s(n)$ is α-*weakly-uniform* if its direct connection language is decided in P/α; recall that the input size for the direct-connection language describing C_n is $m = c_0 \log s(n)$, and so the size of the advice string needed in this case is $\alpha(c_0 \log s(n))$.

The two notions are closely related.

Lemma 7.6. *In the notation above, $\alpha(m)$-succinctness implies $\alpha(m)\log\alpha(m)$-weak uniformity, and conversely, $\alpha(m)$-weak uniformity implies $(\alpha(m)+m)^{O(1)}$-succinctness.*

sketch. A Boolean circuit of size s can be represented by a binary string of size $O(s \log s)$; and a Turing machine running in time t can be simulated by a circuit family of size $O(t \log t)$. □

The notion of weak uniformity (succinctness) interpolates between full uniformity on one end and full non-uniformity on the other end. For example, 0-weak uniformity is the same as uniformity. On the other hand, α-weak uniformity for $\alpha(m) \geq 2^m$ is the same as non-uniformity. For that reason, we will assume that the function α in "α-weakly-uniform" is such that $0 \leq \alpha(m) \leq 2^m$.

Definition 7.7. We say a circuit family $\{C_n\}$ is subexp-*weakly-uniform* if it is α-weakly-uniform for $\alpha(m) \in 2^{o(m)}$; similarly, we say $\{C_n\}$ is poly-*weakly-uniform* if it is α-weakly-uniform for $\alpha(m) \in m^{O(1)}$.

Alternating Turing machines

Both the counting hierarchy and uniform threshold circuits can equivalently be defined using threshold Turing machines, which are generalizations of alternating Turing machines. As we use this view in some of our proofs, we recall the definitions – and state the equivalence in the next subsection. Following [CKS81, AG94], an *alternating Turing machine (ATM)* is a nondeterministic Turing machine with two kinds of states: universal states and existential states. In the usual definition of an ATM, each configuration has either zero or two successor configurations; configurations with no successors, which are called *leaves*, are halting configurations; a configuration in universal (existential) state is accepting iff all (at least one) of its successors are accepting. We also consider the generalized ATMs where each configuration has a a set of successors, obtained by replacing a subtree of "bounded branching" configurations by a single configuration. We assume an ATM has random access to the input. We say an ATM has k *alternations* if it switches from an existential to a universal sate or vice versa for no more than $k-1$ times. A *threshold Turing machine* is an alternating Turing machine with not only existential and universal states but also majority (MAJ) states. A configuration in a *majority state* is accepting if and only if more than half of its successors are accepting. We say a threshold TM has k *alternations* if it switches from one kind of state (existential, universal, or majority) to another kind of state for no more than $k-1$ times. We point out that computations such as "majority of majorities" can be simulated by switching between threshold, existential, and threshold configurations. Alternatively, the threshold Turing machine could be augmented with the ability to "declare" a threshold computation as the "root" of a new threshold tree – thus counting a "majority of majorities" computation as consisting of a single alternation.

We denote by $\mathsf{Th}_{d(n)}\mathsf{TIME}(t(n))$ the class of languages accepted by threshold Turing machines having at most $d(n)$ alternations and running in time $O(t(n))$. Note that the class $\mathsf{Th}_{d(n)}\mathsf{TIME}(t(n))$ is closed under complement, since the negation of majority is the majority of negated inputs[5].

Recall that a language A is in PP ($\mathsf{C}_{=}\mathsf{P}$) if there is a nondeterministic polynomial-time Turing machine M such that $x \in A$ iff the number of accepting paths of M on input x is greater than (equal to) the number of rejecting paths. The *counting hierarchy* [Wag86,

[5]This is true for MAJ with an odd number of inputs, which is easily achieved by replacing $\mathsf{MAJ}(x_1, x_2, \ldots, x_k)$ with $\mathsf{MAJ}(x_1, x_1, x_2, x_2, \ldots, x_k, x_k, 0)$. This presupposes our definition for MAJ – that MAJ is 1 if strictly more than half of the inputs is 1.

Tor91] is defined as $\mathsf{CH} = \cup_{d\geq 0}\mathsf{CH}_d$, where $\mathsf{CH}_0 = \mathsf{P}$ and $\mathsf{CH}_{d+1} = \mathsf{PP}^{\mathsf{CH}_d}$. This definition is unchanged if we replace PP with $\mathsf{C}_=\mathsf{P}$. The counting hierarchy can be equivalently defined via threshold Turing machines: $\mathsf{CH}_d = \mathsf{Th}_d\mathsf{TIME}(n^{O(1)})$.

Alternating Turing machines can be also equipped with modulo states MOD_m for some fixed m; a MOD_m configuration is accepting if and only if the number of its accepting successors is 0 modulo m. We say an ATM with MOD_m states has k *alternations* if it switches from one kind of state (existential, universal, or MOD_m) to another kind of state for no more than $k-1$ times. We denote by $\mathsf{Mod}_{d(n)}\mathsf{TIME}(t(n))$ the class of languages decided by ATMs with MOD_m states for some fixed $m > 0$ dependent on the language, making at most $d(n)$ alternations and running in time $O(t(n))$. Following [GKR$^+$95, All99], we denote by ModPH the class $\cup_{d\geq 0}\mathsf{Mod}_d\mathsf{TIME}(n^{O(1)})$. It is well-known that threshold states can be used to simulate MOD_m states, and thus also $\mathsf{ModPH} \subseteq \mathsf{CH}$.

In general, on different inputs, an ATM may follow computation paths with different sequences of alternations; however, by introducing dummy states, it is always possible to transform the machine into an equivalent machine such that all computation paths on inputs of the same size will follow the same sequence of alternations, whereas the number of alternations and the running time will change only by a constant factor; see [AG94] for details.

Weak uniformity vs. alternating Turing machines with advice

It is well-known that uniform $\mathsf{AC}^0(2^{\mathsf{poly}(n)})$ equals the polynomial-time hierarchy PH [FSS84]. Similarly, the correspondence exists between uniform $\mathsf{ACC}^0(2^{\mathsf{poly}(n)})$ and ModPH [GKR$^+$95, AG94], as well as between uniform $\mathsf{TC}^0(2^{\mathsf{poly}(n)})$ and the counting hierarchy CH [BIS90, All99]; see Table 7.1 below for the summary. More precisely, for constructible $t(n)$ such that $t(n) = \Omega(\log n)$, we have $\cup_{d\geq 0}\mathsf{Mod}_d\mathsf{TIME}(\mathsf{poly}(t(n)))$ is precisely the class of languages decided by uniform $\mathsf{ACC}^0(2^{\mathsf{poly}(t(n))})$, and $\cup_{d\geq 0}\mathsf{Th}_d\mathsf{TIME}(\mathsf{poly}(t(n)))$ is precisely the class of languages decided by uniform $\mathsf{TC}^0(2^{\mathsf{poly}(t(n))})$. The following lemma gives the correspondence between weakly-uniform threshold circuits and threshold TMs with advice.

Lemma 7.8. *Let L be any language decided by a family of α-weakly-uniform $d(n)$-depth threshold circuits of size $s(n)$. Then L is decidable by a threshold Turing machine with $d'(n) = 3d(n)+2$ alternations, taking advice of length $\alpha(m)$ for $m = c_0 \log s(n)$, and running in time $t(n) = d'(n) \cdot \mathsf{poly}(m + \alpha(m))$.*

Table 7.1: Correspondence between hierarchies and uniform circuit classes.

Alternation	Hierarchy	Circuits	Reference
$\exists, \forall$	PH	uniform AC^0	[FSS84]
$\exists, \forall, \mathsf{MOD}_2, \mathsf{MOD}_3, \ldots$	ModPH	uniform ACC^0	[GKR$^+$95, AG94]
$\exists, \forall, \mathsf{MAJ}$	CH	uniform TC^0	[BIS90, All99]

Proof. The proof follows directly from [AG94] where ACC^0 circuits are considered. Let $\{C_n\}$ be the circuit family deciding L. Its direct connection language L_{dc} is accepted by some Turing machine U, on input size $m = c_0 \log s(n)$, taking advice a_m of size $\alpha(m)$ and running in time $\mathsf{poly}(m + \alpha(m))$. We will construct a threshold Turing machine M which takes advice and decides L. For any input x of length n, machine M takes advice $b_n \equiv a_m$, and does the following:

- ($\exists$) guess gate g of C_n, and check that U accepts (n, g, g), i.e., g is a gate in C_n;
- ($\forall$) guess gate h and check that U rejects (n, h, g), i.e., g is the output;
- Call Eval(g), which is a recursive procedure defined below.

The procedure Eval(g) is as follows:

- ($\exists$) If g is an OR gate, then guess its input h; if U rejects (n, g, h) then reject, otherwise call Eval(h).
- ($\forall$) If g is an AND gate, then guess its input h; if U rejects (n, g, h) then accept, otherwise call Eval(h).
- (MAJ) If g is a MAJ gate, then guess its input h and a bit $b \in \{0, 1\}$; if U rejects (n, g, h), then accept when $b = 1$ and reject when $b = 0$, otherwise call Eval(h).
- If g is a constant gate, then accept iff it is 1.
- If g is an input, then accept iff the corresponding input bit is 1.

It is easy to verify that M with advice b_n accepts x iff $C_n(x) = 1$. The number of alternations that M takes on any computation path is at most $d(n) + 2$. However, each path may follow a different sequence of states. To resolve this, we replace each state on each path

by a sequence of three states $(\exists, \forall, \text{MAJ})$, where two of them are dummy. This gives a machine with each computation path following the same alternations, and the total number of alternations is at most $3d(n) + 2$. The access to inputs is only at the last step of each computation path (corresponding to the bottom level of the circuit). At each alternation, the machine simulates U and runs in time $\text{poly}(m + \alpha(m))$. Therefore, the total running time is bounded by $d'(n) \cdot \text{poly}(m + \alpha(m))$.

□

Similar to Lemma 7.8, we have the following correspondence between weakly-uniform ACC circuits and alternating Turing machines with modulo states.

Lemma 7.9. *Let L be any language decided by a family of α-weakly-uniform $d(n)$-depth* ACC *circuits of size $s(n)$ with MOD$_r$ gates, for some integer $r > 0$. Then L is decidable by an alternating Turing machine with MOD$_r$ states and $d'(n) = O(d(n))$ alternations, taking advice of length $\alpha(m)$ where $m = c_0 \log s(n)$, and running in time $d'(n) \cdot \text{poly}(m + \alpha(m))$.*

PERMANENT

The PERMANENT problem is to compute the *permanent* of an $n \times n$ matrix X:

$$Perm_n(X) = \sum_{\sigma} \prod_{i=1}^{n} x_{i,\sigma(i)},$$

where the summation is over all permutations σ of $1, \ldots, n$. Valiant [Val79] showed that PERMANENT for matrices over the set of integers $\mathbb{Z}$ is #P-complete. The main property of PERMANENT needed for our results is PP-hardness. It was shown in [Zan91], which builds on [Val79], that any language in PP reduces to the 0-1-PERMANENT with a quasi-linear size uniform AC^0 reduction, where quasi-linear means $n \cdot \text{polylog}(n)$.

7.2 Indirect Diagonalization

Here we establish the components needed for our indirect diagonalization, as outlined in Section 7.1.3. First, in Section 7.2.1, we give the ingredients needed for our first proof. One result is a diagonalization argument against alternating Turing machines with advice, getting a language in the counting hierarchy CH that is "hard" against weakly-uniform TC^0 circuits of certain size. Another result formalizes the intuition that if a P-complete language

has weakly-uniform small-depth threshold circuits of a certain size, then any language L decided by weakly-uniform circuits must have weakly-uniform small-depth threshold circuits, with parameters related to the assumed circuits for the P-complete language and for L. Section 7.2.2 contains the tools needed for the second proof of our main results. In particular we state and prove the circuit lower bound that is used in the second proof: that E^{PP} contains a language that requires non-uniform circuits of size $2^{\Theta(n)}$. Finally, in Section 7.2.3, we state and prove the key lemma that is used in both proofs. Namely, using the assumption that PERMANENT has small weakly-uniform TC^0 circuits, we show that CH collapses, and our assumed hard languages are in fact decidable by weakly-uniform s'-size Boolean circuits, which is a contradiction. Our actual argument is more general: we consider threshold circuits of not necessarily constant depth $d(n)$, and non-constant levels of the counting hierarchy.

7.2.1 Ingredients for the First Proof

Diagonalization against ATMs with advice

Lemma 7.10. *For any constructible functions $\alpha, d, t, T : \mathbb{N} \to \mathbb{N}$ such that $\alpha(n) \in o(n)$ and $t(n) \log t(n) = o(T(n))$, there is a language $D \in \mathsf{Th}_{d(n)}\mathsf{TIME}(T(n))$ which is not decided by threshold Turing machines with $d(n)$ alternations running in time $t(n)$ and taking advice of length $\alpha(n)$.*

Proof. The proof is by diagonalization. Define the language D consisting of those inputs x of length n that have the form $x = (M, y)$ (using some pairing function) such that the threshold TM M with advice y, where $|y| = \alpha(n)$, rejects input (M, y) in time $t(n)$ using at most $d(n)$ alternations. Language D is decided in $\mathsf{Th}_{d(n)}\mathsf{TIME}(T(n))$ by simulating M and flipping the result[6]. For contradiction, suppose that D is decided by some threshold Turing machine M_0 with $d(n)$ alternations taking advice $\{a_n\}$ of size $\alpha(n)$. Consider the input (M_0, a_n) with $|M_0| = n - \alpha(n)$; we assume that each TM has infinitely many equivalent descriptions (by padding), and so for large enough n, there must exist such a description of

[6] $\mathsf{Th}_{d(n)}\mathsf{TIME}(T(n))$ is closed under complement since the negation of MAJ is MAJ of negated inputs when MAJ has an odd number of inputs; the latter is easy to achieve by replacing $MAJ(x_1, \ldots, x_k)$ with $MAJ(x_1, x_1, \ldots, x_k, x_k, 0)$. Allender [All99] uses a lazy diagonalization argument [Zák83] for nondeterministic TMs. However, that argument seems incapable of handling the amount of advice we need. Fortunately, the basic diagonalization argument we use here is sufficient for our purposes.

size $n - \alpha(n)$. By the definition of D, we have (M_0, a_n) is in D iff M_0 with advice a_n rejects it; but this contradicts the assumption that M_0 with advice $\{a_n\}$ decides D.

□

The following diagonalization result, combing with Lemma 7.9, says that the hierarchy ModPH contains languages that are "hard" against weakly-uniform ACC circuits of certain size.

Lemma 7.11. *For any constructible functions $\alpha, d, t, T : \mathbb{N} \to \mathbb{N}$ such that $\alpha(n) \in o(n)$ and $t(n) \log t(n) = o(T(n))$, and for any integer $m > 1$, there is a language $D \in \mathsf{Mod}_{d(n)+1}\mathsf{TIME}(T(n))$ which is not decided by alternating Turing machines with MOD$_m$ states and $d(n)$ alternation running in time $t(n)$ and taking advice of length $\alpha(n)$.*

sketch. The proof is similar to the proof of Lemma 7.10, except that when flipping the result, the negation can be simulated by a MOD$_m$ state, using the identity $\neg x = \mathsf{MOD}_m(x)$.

□

If P is unusually easy

Let L_0 be a P-complete language under uniform projections (functions computable by uniform Boolean circuits with NOT gates only). For example, the standard P-complete set $\{(M, x, 1^t) : M$ accepts x in time $t\}$ works.

Lemma 7.12. *Suppose L_0 is decided by a family of α-weakly-uniform $d(n)$-depth threshold circuits of size $s(n)$. Then, for any constructible function $t(n) \geq n$ and $0 \leq \beta(m) \leq 2^m$, every language L in β-weakly-uniform $\mathsf{SIZE}(t(n))$ is decided by $\mu(n)$-weakly-uniform $d(\mathsf{poly}(t(n)))$-depth threshold circuits of size $s'(n) = s(\mathsf{poly}(t(n)))$ on n inputs, where $\mu(n) = \alpha(c_0 \log s'(n)) + \beta(c_0 \log t(n))$.*

Proof. Let U be an advice-taking algorithm deciding the direct-connection language for the $t(n)$-size circuits for L. For any string y of length $\beta(m)$ for $m = c_0 \log t(n)$, we can run U with the advice y to construct some circuit C^y of size $t(n)$ on n inputs. We can construct the circuit C^y in time at most $\mathsf{poly}(t(n))$, and then evaluate it in time $\mathsf{poly}(t(n))$ on any given input of size n. Consider the language $L' = \{(x, y, 1^{t(n)}) \mid |x| = n,\ |y| = \beta(m),\ C^y(x) = 1\}$. By the above, we have $L' \in \mathsf{P}$. Hence, by assumption, L' is decided by an α-weakly-uniform $d(l)$-depth threshold circuits of size $s(l)$, where $l = |(x, y, 1^{t(n)})| \leq \mathsf{poly}(t(n))$. To get a circuit

for L, we simply use as y the advice of size $\beta(m)$ needed for the direct-connection language of the $t(n)$-size circuits for L. Overall, we need $\alpha(c_0 \log s(l)) + \beta(m)$ amount of advice to decide L by weakly-uniform $d(\text{poly}(t(n)))$-depth threshold circuits of size $s(\text{poly}(t(n)))$.

□

7.2.2 Ingredients for Second Proof

The second proof uses the following to obtain a hard language in the indirect diagonalization. In the theorem statement *maximum circuit complexity* is the largest $h(n)$ such that there exists a language that does not have circuits of size $h(n) - 1$. For completeness, we provide a proof.

Theorem 7.13 ([Kan82])**.** *Let $c > 0$ be a constant such that there are at most $2^{(h(n))^c}$ circuits of size $h(n)$ at input length n. Let $h(n)$ be a time-constructible function such that for all n, $n \leq h(n)$, $(h(n))^c < 2^n$, and $h(n)$ is less than the maximum circuit complexity. There is a language L_{hard} in $\text{TIME}^{\text{PP}}(\text{poly}(h(n)))$ that does not have circuits of size $h(n)$.*

Proof. Let $x_1, ..., x_{(h(n))^c+1}$ be the $(h(n))^c + 1$ lexicographically smallest inputs of length n. The PP language we use as oracle is

$$A = \{(1^n, j, b_1, ..., b_{(h(n))^c+1}) |\ C(x_j) = b_j \text{ for at most 1/2}$$

$$\text{of the circuits } C \text{ of size } h(n) \text{ that satisfy } C(x_i) = b_i \text{ for all } 1 \leq i < j.\}$$

A can be decided in PP by a machine as follows. The machine guesses a circuit of size $h(n)$; if the circuit does not agree with one of the b_i between 1 and $j-1$ then the PP machine splits into two nondeterministic paths with one accepting and one rejecting; otherwise the PP machine accepts iff $C(x_j) \neq b_j$. Then there are at least half accepting paths iff at least half of the circuits in question disagree with b_j on x_j. As we can evaluate a circuit of size $h(n)$ in $\text{poly}(h(n))$ time, the running time for A is $\text{poly}(h(n))$, which is polynomial in the input length, so $A \in$ PP.

L_{hard} is defined as follows. $L_{hard}(x_1) = A(1^n, 1, 0, 0, ..., 0)$, and already L_{hard} differs from at least half of the circuits of size $h(n)$. $L_{hard}(x_2) = A(1^n, L_{hard}(x_1), 1, 0, ..., 0)$. So now L_{hard} differs from at least 3/4 of the circuits of size $h(n)$. And so on. As there are at most $2^{(h(n))^c}$ circuits of size $h(n)$, we will have differed from all in at most $(h(n))^c + 1$ steps.

For inputs not in the set $\{x_1, ..., x_{h(n)+1}\}$ we can define L_{hard} arbitrarily (e.g., set it to 0). Notice that L_{hard} can be decided in $\mathsf{poly}(h(n))$ time with access to the PP oracle A.

□

Since separations for high resources imply separations for low resources, it will be optimal to set $h(n)$ large. Because there exist languages that require circuits of size $2^{n/c}$ for all constants $0 < c < 1$ [Sha49] we have the following corollary, which we use in the second proof of our main results.

Corollary 7.14. *For every $0 < c < 1$, there is a language L_{hard} in $\mathsf{DTIME}^{\mathsf{PP}}(2^{O(n)})$ that does not have circuits of size $2^{n/c}$.*

7.2.3 Key Lemma – Collapse of CH if PERMANENT is Easy

Since PERMANENT is hard for the first level of the counting hierarchy CH, assuming that PERMANENT is "easy" implies the collapse of CH (see, e.g., [All99]). It was observed in [KP09] that it is also possible to collapse super-constant levels of CH, under the same assumption. Below we argue the collapse of super-constant levels of CH by assuming that PERMANENT has "small" weakly-uniform circuits. We use the notation $f \circ g$ to denote the composition of the functions f and g, and the notation $f^{(i)}$ is used to denote the composition of f with itself for i times; we use the convention that $f^{(0)}$ is the identity function.

Lemma 7.15. *Suppose that* PERMANENT *is in γ-weakly-uniform $\mathsf{SIZE}(s(n))$, for some $\gamma(m) \leq 2^{o(m)}$. For every $d(n) \leq n^{o(1)}$, every language A in $\mathsf{Th}_{d(n)}\mathsf{TIME}(\mathsf{poly})$ is also in $(2d(n) \cdot \gamma)$-weakly-uniform $\mathsf{SIZE}((s \circ q)^{(d(n)+1)}(n))$, for some polynomial q dependent on A.*

Proof. The idea of the proof is to convert the computation of A into a threshold circuit and then replace majority gates by the circuit for the permanent, beginning with the level of gates closest to the input and working inductively higher – as described in the outline of the second proof of Theorem 7.2 in section 7.1.3. The language A is computable by a uniform threshold circuit family $\{C_n\}$ of depth $d(n)$ and size $\mathsf{poly}(n)$, as follows. Let M be a polynomial-time TM deciding the direct-connection language of $\{C_n\}$. More precisely, we identify the gates of the circuit with the configurations of the given threshold TM for A; the output gate is the initial configuration; leaf (input) gates are halting configurations; deciding if one gate is an input to the other gate is deciding if one configuration follows from the other according to our threshold TM, and so can be done in polynomial time (dependent

on A); finally, given a halting configuration, we can decide if it is accepting or rejecting also in polynomial time (dependent on A). Consider an arbitrary n. Let $d = d(n)$. For a gate g of C, we denote by C_g the sub-circuit of C that determines the value of the gate g. We say that g is at depth i, for $1 \leq i \leq d$, if the circuit C_g is of depth i. Note that each gate at depth $i \geq 1$ can be simulated by a majority gate because threshold gates can simulate AND and OR gates, and NOT gates can be pushed to the inputs. For every $0 \leq i \leq d$, let B_i be a circuit that, given $x \in \{0,1\}^n$ and a gate g at depth i, outputs the value $C_g(x)$.

Claim 7.16. *There are polynomials q and q' dependent on A such that, for each $0 \leq i \leq d$, there are $2i\gamma$-weakly-uniform circuits B_i of size $(s \circ q)^{(i)} \circ q'$.*

Proof. We argue by induction on i. For $i = 0$, to compute $B_0(x, g)$, we need to decide if the halting configuration g of our threshold TM for A on input x is accepting or not; by definition, this can be done by the TM M in deterministic polynomial time. Hence, B_0 can be decided by a completely uniform circuit of size at most $q'(n)$ for some polynomial q' dependent on the running time of M. Assume we have the claim for i. Let s' be the size of the γ'-weakly-uniform circuit B_i, where $s' \leq (s \circ q)^{(i)} \circ q'$ and $\gamma' \leq 2i\gamma$. Consider the following TM N:

> "On input $z = (x, g, U, y, 1^{s'/2})$, where $|x| = n$, g is a gate of C, $|U| = \gamma(c_0 \log s')$, $|y| = \gamma'(c_0 \log s')$, interpret U as a Turing machine that takes advice y to decide the direct-connection language of some circuit D of size s' on inputs of length $|(x, g)|$. Construct the circuit D using U and y, where to evaluate U on a given input we simulate U for at most s' steps. Enter the MAJ state. Nondeterministically guess a gate h of C and a bit $b \in \{0,1\}$. If h is not an input gate for g, then accept if $b = 1$ and reject if $b = 0$; otherwise, accept if $D(x, h) = 1$ and reject if $D(x, h) = 0$."

We will be interested in the case where U is a polynomial-time TM. For any such U, the running time on any input is bounded by $\mathrm{poly}(c_0 \log s' + \gamma'(c_0 \log s'))$, which is less than s' by our assumptions that $\gamma(m) \leq 2^{o(m)}$ and $d \leq (s')^{o(1)}$. Thus, to evaluate U on a particular input, it suffices to simulate U for at most s' steps, which is independent of what the actual polynomial time bound of U is. It follows that we can construct the circuit D (given U and y) in time $p(s')$, where p is a polynomial that does not depend on U. Also, to decide if h is an input gate to g, we use the polynomial-time TM M. We conclude that N is a

PP machine which runs in some polynomial time (dependent on A). Since PERMANENT is PP-hard [Val79, Zan91], we have a uniform reduction mapping z (an input to N) to an instance of PERMANENT of size $q(|z|)$, for some polynomial q (dependent on A). By our assumption on the easiness of PERMANENT, we get that the language of N is decided by γ-weakly-uniform circuits C_N of size at most $s'' = s(q(s'))$. If we plug in for U and y the actual TM description and the advice needed to decide the direct-connection language of B_i, we get from C_N the circuit B_{i+1}. Note that the direct-connection language of this circuit B_{i+1} is decided in polynomial time (using the algorithm for direct-connection language of C_N) given the advice needed for C_N plus the advice needed to describe U and y. The total advice size is at most $\gamma(c_0 \log s'') + \gamma(c_0 \log s'') + \gamma'(c_0 \log s') \leq 2(i+1)\gamma(c_0 \log s'')$.

□

Finally, we take the circuit B_d and use it to evaluate $A(x)$ by computing the value $B_d(x, g)$ where g is the output gate of C, which can be efficiently constructed (since this is just the initial configuration of our threshold TM for A on input x). By fixing g to be the output gate of C, we get the circuit for A which is $2d\gamma$-weakly-uniform of size at most $(s \circ q)^{(d)}(r(n))$, where the polynomial r depends on the language A. Upper-bounding r by $(s \circ q)$ yields the result.

□

7.3 First Proof of Main Results

Here we use the technical tools from the previous section in order to give the first proof of our main results, as outlined in Section 7.1.3. The proofs for each of Theorems 7.2, 7.3, and 7.4 follow the same steps, with the key difference being how parameters are set. It is possible to leave all of the parameters free to obtain a single statement that would encapsulate each of the main theorems. As doing so would make the proofs seem overly complicated, we instead prove each of the results separately. For the second proof of our main results in the next section, we take the other approach – proving a single parameterized statement that implies each of the main theorems. Recall that we let L_0 be a P-complete language under uniform projection, e.g., $L_0 = \{(M, x, 1^t) : M \text{ accepts } x \text{ in time } t\}$. Also recall that we use "subexp-weakly-uniform" and "poly-weakly-uniform" as shorthands for $2^{o(m)}$-weakly-uniform and $m^{O(1)}$-weakly-uniform, respectively.

7.3.1 Proof of Theorem 7.2

First, assuming L_0 is easy, we construct a hard language in CH.

Lemma 7.17. *Suppose L_0 is in* subexp-*weakly-uniform* TC^0 *of depth d. Then, for a constant d' dependent on d, there is a language $L_{diag} \in \mathsf{CH}_{d'}$ which is not in* subexp-*weakly-uniform* $\mathsf{SIZE}(\mathrm{poly})$.

Proof. Let $\alpha(m) \in 2^{o(m)}$ be such that L_0 is in α-weakly-uniform TC^0 of depth d. Consider an arbitrary language L in β-weakly-uniform $\mathsf{SIZE}(\mathrm{poly})$, for an arbitrary $\beta(m) \in 2^{o(m)}$. By Lemma 7.12, L has $\mu(n)$-weakly uniform threshold circuits of depth d and polynomial size, where $\mu(n) = \alpha(O(\log n)) + \beta(O(\log n)) \leq n^{o(1)}$. By Lemma 7.8, we have that L is decided by a threshold Turing machine with $d' = O(d)$ alternations, taking advice of length $\mu(n) \leq n^{o(1)} \leq n/\log^2 n$, and running in time $d' \cdot \mathrm{poly}(O(\log n) + n^{o(1)}) \leq n^{o(1)} \leq n/\log^2 n$. Note that $n/\log^2 n$ can be replaced by any $f(n)$ such that $f(n)\log n \leq o(n)$. We conclude that *every* language in subexp-weakly-uniform $\mathsf{SIZE}(\mathrm{poly})$ is also decided by some threshold TM in time $n/\log^2 n$, using d' alternations and advice of size $n/\log^2 n$. Using Lemma 7.10, define L_{diag} to be the language in $\mathsf{Th}_{d'}\mathsf{TIME}(n)$ which is not decidable by any threshold Turing machine in time $n/\log^2 n$, using d' alternations and advice of size $n/\log^2 n$. It follows that L_{diag} is different from every language in subexp-weakly-uniform $\mathsf{SIZE}(\mathrm{poly})$.

□

Next, assuming PERMANENT is easy, we have that every language in CH is easy. The proof is immediate by Lemma 7.15.

Lemma 7.18. *If* PERMANENT *is in* subexp-*weakly-uniform* $\mathsf{SIZE}(\mathrm{poly})$*, then every language in* CH *is in* subexp-*weakly-uniform* $\mathsf{SIZE}(\mathrm{poly})$.

We now show that L_0 and PERMANENT cannot both be easy. The proof is immediate by Lemmas 7.17 and 7.18.

Theorem 7.19. *At least one of the following must be false:*

1. *L_0 is in* subexp-*weakly-uniform* TC^0*;*
2. PERMANENT *is in* subexp-*weakly-uniform* $\mathsf{SIZE}(\mathrm{poly})$.

To unify the two items in Theorem 7.19, we use the next lemma and its corollary.

Lemma 7.20 ([Val79, AG94])**.** *For every language $L \in \mathsf{P}$, there are uniform AC^0-computable functions M (mapping a binary string to a polynomial-size Boolean matrix) and f such that, for every x, we have $x \in L$ if and only if $f(\textsc{Permanent}(M(x))) = 1$.*

This lemma immediately yields the following.

Corollary 7.21. *If* PERMANENT *has α-weakly-uniform $d(n)$-depth threshold circuits of size $s(n)$, then L_0 has α-weakly-uniform $(d(n^{O(1)})+O(1))$-depth threshold circuits of size $s(n^{O(1)})$.*

Now we prove Theorem 7.2, which we re-state below.

Theorem 7.22. PERMANENT *is not in* subexp*-weakly-uniform* TC^0*.*

Proof. Otherwise by Corollary 7.21, both claims in Theorem 7.19 would hold, which is impossible. □

7.3.2 Proof of Theorem 7.3

Recall that a function $r(n)$ is sub-subexponential if, for every constant $k > 0$, $r^{(k)}(n) \leq 2^{n^{o(1)}}$. Also recall that subsubexp denotes the class of all sub-subexponential functions $r(n)$. Below, we will use the simple fact that, for every constant $k > 0$, the composition of k sub-subexponential functions is also sub-subexponential.

Lemma 7.23. *Suppose that the* P*-complete language L_0 is in* poly*-weakly-uniform* $\mathsf{TC}^0(\text{subsubexp})$ *of depth d. Then, for a constant $d' = O(d)$, there is a language $L_{diag} \in \mathsf{CH}_{d'}$ which is not in* poly*-weakly-uniform* SIZE(subsubexp)*.*

Proof. The proof is similar to that of Lemma 7.17. Let $\alpha(m) \in \text{poly}(m)$ and $s(n) \in \text{subsubexp}$ be such that L_0 is in α-weakly-uniform d-depth $\mathsf{TC}^0(s(n))$. Consider an arbitrary language L in β-weakly-uniform $\mathsf{SIZE}(t(n))$, for arbitrary $\beta(m) \in \text{poly}(m)$ and $t(n) \in \text{subsubexp}$. By Lemma 7.12, L is in $\mu(n)$-weakly uniform d-depth $\mathsf{TC}^0(s'(n))$, where $s'(n) = s(\text{poly}(t(n)))$ and $\mu(n) = \alpha(c_0 \log s'(n)) + \beta(c_0 \log t(n)) \leq n^{o(1)}$ (since s' and t are sub-subexponential). By Lemma 7.8, we have that L is decided by a threshold Turing machine with $d' = O(d)$ alternations, taking advice of length $\mu(n) \leq n^{o(1)} \leq n/\log^2 n$, and running in time $d' \cdot \text{poly}(c_0 \log s'(n) + \alpha(c_0 \log s'(n))) \leq n^{o(1)} \leq n/\log^2 n$. We conclude that *every* language in poly-weakly-uniform $\mathsf{SIZE}(\text{subsubexp})$ is also decided by some threshold Turing machine in time $n/\log^2 n$, using d' alternations and advice of size $n/\log^2 n$. Using Lemma 7.10, define

L_{diag} to be the language in $\mathsf{Th}_{d'}\mathsf{TIME}(n)$ which is not decidable by any threshold Turing machine in time $n/\log^2 n$, using d' alternations and advice of size $n/\log^2 n$. It follows that L_{diag} is different from every language in poly-weakly-uniform SIZE(subsubexp).

□

Now we are ready to prove Theorem 7.3, which we re-state below.

Theorem 7.24 (Theorem 7.3 restated)**.** PERMANENT *is not in the class* poly*-weakly-uniform* TC^0(subsubexp).

Proof. Suppose that, for some $\alpha(m) \in \mathsf{poly}(m)$ and $s(n) \in \mathsf{subsubexp}$, PERMANENT is in α-weakly-uniform $\mathsf{TC}^0(s(n))$; this also implies that PERMANENT is in α-weakly-uniform $\mathsf{SIZE}(\mathsf{poly}(s(n))$. By Corollary 7.21, L_0 is in α-weakly-uniform $\mathsf{TC}^0(\mathsf{poly}(s(n)))$, and so, by Lemma 7.23, there is a language $L_{diag} \in \mathsf{CH}$ which is not in poly-weakly-uniform SIZE(subsubexp). But, by Lemma 7.15, every language L in CH is in poly-weakly-uniform SIZE(subsubexp). A contradiction.

□

7.3.3 Proof of Theorem 7.4

Lemma 7.25. *Suppose* L_0 *is in* poly*-weakly-uniform* poly*-size threshold circuits of depth* $o(\log\log n)$*. Then there is a language* $L_{diag} \in \mathsf{Th}_{\log\log n}\mathsf{TIME}(n)$ *which is not in* poly*-weakly-uniform* $\mathsf{SIZE}(n^{\mathsf{poly}(\log n)})$.

Proof. Let $\alpha(m) \in \mathsf{poly}(m)$, $s(n) \in \mathsf{poly}(n)$, and $d(n) \in o(\log\log n)$ be such that L_0 is computable by α-weakly-uniform $d(n)$-depth threshold circuits of size $s(n)$. Consider an arbitrary language L in β-weakly-uniform $\mathsf{SIZE}(t(n))$, for arbitrary $\beta(m) \in \mathsf{poly}(m)$ and $t(n) \in n^{\mathsf{poly}(\log n)}$. By Lemma 7.12, L is in $\mu(n)$-weakly uniform $d'(n)$-depth threshold circuits of size $s'(n)$, where $d'(n) = d(\mathsf{poly}(t(n))) \leq o(\log\log n)$, $s'(n) = s(\mathsf{poly}(t(n))) \leq n^{\mathsf{poly}(\log n)}$, and $\mu(n) = \alpha(c_0 \log s'(n)) + \beta(c_0 \log t(n)) \leq \mathsf{poly}(\log n)$.

By Lemma 7.8, we have that L is decided by a threshold Turing machine with at most $O(d'(n)) < \log\log n$ alternations, taking advice of length $\mu(n) \leq n^{o(1)} \leq n/\log^2 n$, and running in time $O(d'(n)) \cdot \mathsf{poly}(c_0 \log s'(n) + \alpha(c_0 \log s'(n))) \leq n^{o(1)} \leq n/\log^2 n$. We conclude that *every* language in poly-weakly-uniform $\mathsf{SIZE}(n^{\mathsf{poly}(\log n)})$ is also decided by some threshold TM in time $n/\log^2 n$, using $\log\log n$ alternations and advice of size $n/\log^2 n$.

Using Lemma 7.10, define L_{diag} to be the language in $\mathsf{Th}_{\log\log n}\mathsf{TIME}(n)$ which is not decidable by any threshold TM in time $n/\log^2 n$, using $\log\log n$ alternations and advice of size $n/\log^2 n$. It follows that L_{diag} is the required language.

□

Now we prove Theorem 7.4, restated below.

Theorem 7.26 (Theorem 7.4 restated)**.** PERMANENT *is not computable by* poly*-weakly-uniform* poly*-size threshold circuits of depth* $o(\log\log n)$*.*

Proof. Assume otherwise. Then PERMANENT is also in poly-weakly-uniform $\mathsf{SIZE}(\mathsf{poly})$, and so, by Lemma 7.15, every language in $\mathsf{Th}_{\log\log n}\mathsf{TIME}(n)$ is in poly-weakly-uniform $\mathsf{SIZE}(n^{\mathsf{poly}(\log n)})$. On the other hand, by Corollary 7.21, L_0 is computable by poly-weakly-uniform threshold circuits of poly-size and depth $o(\log\log n)$, and so, by Lemma 7.25, there is a language $L_{diag} \in \mathsf{Th}_{\log\log n}\mathsf{TIME}(n)$ such that L_{diag} is not in poly-weakly-uniform $\mathsf{SIZE}(n^{\mathsf{poly}(\log n)})$. A contradiction. □

7.4 Second Proof of Main Results

In this section we give a second proof of our main results. Both proofs use the same key ingredient – the collapse of the counting hierarchy under the assumed easiness of PERMANENT (Lemma 7.15). The proofs differ in how this collapse is used to derive a contradiction to a known lower bound.

7.4.1 Parameterized Statement and Proof

Our second proof yields the following parameterized result. This result is proved using the strategy outlined in Section 7.1.3, but letting the circuit size, depth, and amount of advice be parameters. Let L_0 be the P-complete language used earlier.

Theorem 7.27. *Let* $s(n)$ *be time-constructible, and let* $m = O(\log s(n))$ *be the input length for a uniformity Turing machine for a circuit of size* $s(n)$*. Let* $s(n) \geq n$*,* $\alpha(m)$*, and* $d(n)$ *be non-decreasing functions such that* $\alpha(m)$ *and* $d(n) \leq s(n)$ *for all* n*. Assume also that* $\alpha(m) \leq 2^{o(m)}$ *and* $d(n) \leq (\log s(n))^{o(1)}$*. Let* $N = \mathsf{poly}(s(O(2^n)))$*,* $M = O(\log s(N))$ *and*

$$s' = (s \circ q)^{O(d(N))}(\log(s(N)) + \alpha(M))$$

where each big-O constant is an absolute constant independent of the other parameters. If $s' < 2^{n/c}$ then either

- PERMANENT *does not have $\alpha(m)$-weakly-uniform* SIZE$(s(n))$ *circuits,*
- *Or L_0 does not have $\alpha(m)$-weakly-uniform threshold circuits of size $s(n)$ and depth $d(n)$.*

Since L_0 reduces to PERMANENT , a corollary is that unconditionally PERMANENT does not have weakly-uniform threshold circuits with the given parameters. Each of Theorems 7.2, 7.3, and 7.4 can be obtained by setting the parameters in Theorem 7.27 appropriately. To prove Theorem 7.27, we combine the hard language L_{hard} resulting from Corollary 7.14 (which is in E^{PP} and requires circuits of size $2^{\Theta(n)}$) with the following two claims.

Claim 1. Let $s(n)$ be time-constructible, and let $m = O(\log s(n))$ be the input length for a uniformity Turing machine for a circuit of size $s(n)$. Let $s(n) \geq n$, $\alpha(m)$, and $d(n)$ be non-decreasing functions such that $\alpha(m)$ and $d(n) \leq s(n)$ for all n.

Suppose PERMANENT is in $\alpha(m)$-weakly-uniform SIZE$(s(n))$, and L_0 has $\alpha(m)$-weakly-uniform threshold circuits of size $s(n)$ and depth $d(n)$. Then L_{hard} has $O(\alpha(M))$-weakly uniform threshold circuits of depth $O(d(N))$ and size $O(s(N))$, for $N = \mathsf{poly}(s(O(2^n)))$ and $M = O(\log s(N))$.

Claim 1 is proved by plugging in the assumed computations for PERMANENT and L_0 into the E^{PP} computation of L_{hard}.

Proof. Consider the E^{PP} computation of L_{hard} of Corollary 7.14, which asks at most 2^n queries of its PP oracle on any given input. From the proof of Theorem 7.13, the PP oracle A from the definition of L_{hard} is computable in polynomial PPTIME, and the instances of A needed to solve L_{hard} are of size $O(2^n)$. These can be reduced to instances of PERMANENT that are also of some length $n_A = O(2^n)$ [7]. Given the assumed easiness of PERMANENT , the oracle queries can be decided by a weakly-uniform circuit C_A of size $\mathsf{poly}(s(O(2^n)))$ with advice $\alpha(O(\log s(O(2^n))))$. Deciding membership in L_{hard} amounts to querying the oracle A on at most 2^n inputs. This gives an oracle circuit that makes

[7] We can assume all queries are the same size because there are paddable PP-complete languages, including language versions of PERMANENT . A language is paddable if queries of smaller length can efficiently, e.g. by a uniform AC^0 reduction, be made longer to match the longest query.

exponentially many adaptive queries to A. In this circuit we replace each oracle gate with the circuit C_A, obtaining a single circuit deciding L_{hard} that is of size $\mathsf{poly}(2^n \cdot s(O(2^n)))$ that uses $\alpha(O(\log s(O(2^n))))$ bits of advice. This circuit can be viewed as a circuit value problem of size $\mathsf{poly}(2^n \cdot s(O(2^n)))$. By the P-completeness of L_0, this computation can be reduced to an instance of L_0 of size $N = \mathsf{poly}(2^n \cdot s(O(2^n)))$. Let $M = O(\log s(N))$.

By using a uniform AC^0 reduction to L_0 and using the assumed weakly-uniform threshold circuits for L_0, L_{hard} can be computed by a weakly-uniform threshold circuit of depth $O(d(N))$ and size $O(s(N))$ that uses $\alpha(O(\log s(O(2^n))))$ advice for the creation of the circuit C_A and $\alpha(O(\log s(N)))$ advice from the application of the easiness assumption for L_0. The total advice is $O(\alpha(O(\log s(N))))$. N can be simplified to $N = \mathsf{poly}(s(O(2^n)))$ since $s(n) \geq n$.

□

Claim 2. Let $s(n)$ be time-constructible, and let $m = O(\log s(n))$ be the input length for a uniformity Turing machine for a circuit of size $s(n)$. Let $s(n) \geq n$, $\alpha(m)$, and $d(n)$ be non-decreasing functions such that $\alpha(m)$ and $d(n) \leq s(n)$ for all n. Assume also that $\alpha(m) \leq 2^{o(m)}$ and $d(n) \leq (\log s(n))^{o(1)}$. Suppose PERMANENT is in $\alpha(m)$-weakly-uniform $\mathsf{SIZE}(s(n))$, and L_0 has $\alpha(m)$-weakly-uniform threshold circuits of size $s(n)$ and depth $d(n)$.

Then L_{hard} is contained in

$$\mathsf{SIZE}(s \circ q)^{O(d(N))}(\log s(N) + \alpha(M))$$

for some polynomial q, for $N = \mathsf{poly}(s(O(2^n)))$ and $M = O(\log s(N))$.

To prove Claim 2, we use the threshold circuit from Claim 1 and use the assumed easiness of PERMANENT to "collapse" the threshold circuit. For the latter we apply Lemma 7.15 – the same key step in both proofs of the main result.

Proof. Under the assumptions of the claim, we have a threshold circuit for L_{hard} due to Claim 1. We would like to apply Lemma 7.15. To do so, we need the computation for L_{hard} to be contained in $\mathsf{Th}_{d'(n)}\mathsf{TIME}(\mathsf{poly}(n'))$ for some d' and n' such that $d'(n') \leq n'^{o(1)}$. Due to the equivalence of weakly-uniform threshold circuits and threshold Turing machines with advice, we have that L_{hard} is in $\mathsf{Th}_{O(d(N))}\mathsf{TIME}(d(N) \cdot \mathsf{poly}(\log s(N) + \alpha(M)))$ using $O(\alpha(M))$ advice, with N and M from Claim 1. We set

$$n' = d(N) + \log(s(N)) + \alpha(M).$$

Then the running time for the threshold computation of L_{hard} is poly(n') with depth $O(d(N))$. Assuming $d(N) \leq (\log s(N))^{o(1)}$, we have that the depth is $n'^{o(1)}$. We have also assumed that the amount of advice $\alpha(m)$ in the weakly-uniform circuit for PERMANENT is $\leq 2^{o(m)}$, which is required to apply Lemma 7.15. The only remaining issue before applying Lemma 7.15 is that the lemma does not allow for the initial threshold computation for L_{hard} to use advice. An examination of the proof of Lemma 7.15 shows that a linear amount of advice does not change the parameters – the advice is passed through the argument and is added onto the amount of advice needed by the final circuit. In the current application, the threshold computation for L_{hard} uses $O(\alpha(M))$ advice, which is indeed $O(n')$. By our assumption that $d(N) \leq (\log s(N))^{o(1)}$ we have that $n' = O(\log s(N) + \alpha(M)$. Plugging into Lemma 7.15 we obtain a circuit for L_{hard} that is of size $(s \circ q)^{(O(d(N))}(n') = (s \circ q)^{O(d(N))}(\log s(N) + \alpha(M))$ for some polynomial q. □

(Proof of Theorem 7.27.) If the size of the circuit for L_{hard} in Claim 2 is less than the hardness proved for L_{hard} in Corollary 7.14, which is $2^{n/c}$, we conclude that one of the assumptions in the claim must be false.

7.4.2 Corollary to the Second Proof

In this section we observe that Theorem 7.27 can be improved by examining the proof more carefully. We state the corollary and then argue why it holds.

Corollary 7.28. *Let $s(n)$ be time-constructible, and let $m = O(\log s(n))$ be the input length for a uniformity Turing machine for a circuit of size $s(n)$. Let $s(n) \geq n$, $\alpha(m)$, and $d(n)$ be non-decreasing functions such that $\alpha(m)$ and $d(n) \leq s(n)$ for all n. Assume also that $\alpha(m) \leq 2^{o(m)}$ and $d(n) \leq (\log s(n))^{o(1)}$. Let $N =$ poly$(s(s(O(2^n))))$, $M = O(\log s(N))$ and*

$$s' = (s \circ q)^{O(d(N))}(\log s(N) + \alpha(M))$$

where each big-O constant is an absolute constant independent of the other parameters. If $s' < 2^{n/c}$ then either

- PERMANENT *does not have non-uniform circuits of size $s(n)$,*
- *Or* SAT *does not have $\alpha(m)$-weakly-uniform threshold circuits of size $s(n)$ and depth $d(n)$.*

The easiness of PERMANENT is used in the proof of Theorem 7.27 for two key purposes.

(i) Corollary 7.14 and Claim 1 show that if PERMANENT has weakly-uniform circuits and L_0 has small-depth weakly-uniform threshold circuits, L_{hard} has large weakly-uniform small-depth threshold circuits.

(ii) Claim 2 shows that if PERMANENT has small circuits, the circuit from (i) can be iteratively made smaller by appealing to Lemma 7.15.

For step (i), we can replace the combination of PERMANENT and L_0 by any language that, if assumed to have small-depth threshold circuits, implies a small-depth threshold circuit for a language with high circuit complexity. For example, we can use an NP-complete language and the following fact.

Theorem 7.29 ([Kan82, MVW99])**.** *For any $0 < c < 1$, there is a language L_{hard2} in* $\mathsf{TIME}^{\Sigma_2^p}(2^{O(n)})$ *that does not have circuits of size $2^{n/c}$.*

Using an NP-complete language such as SAT, Claim 1 becomes instead the following.

Claim 3. Let $s(n)$ be time-constructible, and let $m = O(\log s(n))$ be the input length for a uniformity Turing machine for a circuit of size $s(n)$. Let $s(n) \geq n$, $\alpha(m)$, and $d(n)$ be non-decreasing functions such that $\alpha(m)$ and $d(n) \leq s(n)$ for all n. Suppose SAT has $\alpha(m)$-weakly-uniform threshold circuits of size $s(n)$ and depth $d(n)$. Then L_{hard2} has $O(\alpha(M))$-weakly uniform threshold circuits of depth $O(d(N))$ and size $O(s(N))$, for $N = \mathsf{poly}(s(s(O(2^n))))$ and $M = O(\log s(N))$.

The change in the value of N is due to working in the third level of the exponential alternating hierarchy, whereas in Claim 1 the hard language was in the second level of the exponential counting hierarchy. For step (ii), the proof only requires that PERMANENT has small general circuits – the small-depth and uniformity are not used in the argument. Combining these two observations, we have a result stating that if both (1) SAT has small weakly-uniform small-depth threshold circuits, and (2) PERMANENT has small general circuits, then L_{hard2} has small circuits. Specifically, we have the following claim in place of Claim 2.

Claim 4. Let $s(n)$ be time-constructible, and let $m = O(\log s(n))$ be the input length for a uniformity Turing machine for a circuit of size $s(n)$. Let $s(n) \geq n$, $\alpha(m)$, and $d(n)$ be

non-decreasing functions such that $\alpha(m)$ and $d(n) \leq s(n)$ for all n. Assume also that $\alpha(m) \leq 2^{o(m)}$ and $d(n) \leq (\log s(n))^{o(1)}$. Suppose PERMANENT is in non-uniform SIZE$(s(n))$, and SAT has $\alpha(m)$-weakly-uniform threshold circuits of size $s(n)$ and depth $d(n)$. Then L_{hard2} is contained in

$$\text{SIZE}(s \circ q)^{O(d(N))}(\log s(N) + \alpha(M))$$

for some polynomial q, for $N = \text{poly}(s(s((O(2^n)))))$ and $M = O(\log s(N))$.

If the resulting circuit is of size less than $2^{n/c}$, then the assumed circuits for either SAT or PERMANENT must not exist.

7.5 Other Lower Bounds

Here we use diagonalization against advice classes to prove exponential lower bounds for weakly-uniform circuits, of both constant and unbounded depth.

7.5.1 Lower Bounds for ACC^0 and AC^0

The following result generalizes the result in [AG94] on uniform ACC^0 circuits.

Theorem 7.30. PERMANENT *is not in* poly*-weakly-uniform* $\text{ACC}^0(2^{n^{o(1)}})$.

Proof. It is shown in [BT94, AG94] that every language L in uniform $\text{ACC}^0(2^{n^{o(1)}})$ is also decidable by uniform depth-two circuits of related size $s'(n) \in 2^{n^{o(1)}}$ where *(i)* the bottom level consists of AND gates of fan-in $(\log s'(n))^{O(1)}$, and *(ii)* the top level is a symmetric gate (whose value depends only on the number of inputs that evaluate to one). Using this fact as well as the #P-hardness of PERMANENT [Val79], Allender and Gore [AG94] argue that L is in $\text{DTIME}(n^9)^{\text{PERMANENT}[1]}$ (with a single oracle query to PERMANENT). This result can be easily generalized to the case when L has weakly-uniform circuits. That is, for $\alpha(m) = m^{O(1)}$ where $m = O(\log s(n))$, any language in α-weakly-uniform $\text{ACC}^0(2^{n^{o(1)}})$ is also in $\text{DTIME}(n^9)^{\text{PERMANENT}[1]}/\gamma(n)$ for some $\gamma(n) = n^{o(1)}$.

For the sake of contradiction, suppose that PERMANENT is in α-weakly-uniform $\text{ACC}^0(2^{n^{o(1)}})$. Consider a language $L \in \text{DTIME}(n^{10})^{\text{PERMANENT}[1]}$ which is not in $\text{DTIME}(n^9)^{\text{PERMANENT}[1]}/n^{o(1)}$; the existence of such an L is easy to argue by diagonalization (similarly to the proof of

Lemma 7.10). Let M be the corresponding oracle machine deciding L. Consider the following languages:

$$L' = \{(x, y) \colon M \text{ uses } y \text{ as the answer of the oracle query and accepts } x\},$$

$$L'' = \{(x, i) \colon \text{the } i\text{th bit of the oracle query made by } M \text{ on input } x \text{ is 1}\}.$$

Clearly, both L' and L'' are in P. Since P is reducible to PERMANENT via uniform AC^0 reduction, we get that both L' and L'' are in α-weakly-uniform $\mathsf{ACC}^0(2^{n^{o(1)}})$. To construct circuits for L, on any input x, we use the circuit for L'' to construct the oracle query, use the circuit for PERMANENT to answer the query, and then use the circuit for L' to decide whether $x \in L$. Since L', L'' and PERMANENT all have α-weakly-uniform $\mathsf{ACC}^0(2^{n^{o(1)}})$ circuits, the resulting circuit is also in α-weakly-uniform $\mathsf{ACC}^0(2^{n^{o(1)}})$. This implies that L is in $\mathsf{DTIME}(n^9)^{\textsc{Permanent}[1]}/n^{o(1)}$. A contradiction.

□

We note that one can also show a lower bound for NP against weakly-uniform AC^0 circuits.

Theorem 7.31. NP *is not in* poly*-weakly-uniform* $\mathsf{AC}^0(\mathsf{subsubexp})$.

sketch. The proof is analogous to that of Theorem 7.3, by replacing PERMANENT with SAT, CH with PH, and threshold circuits with Boolean circuits.

□

Note, however, that this lower bound is weaker than the well-known result that PARITY requires *exponential-size non-uniform* AC^0 circuits [Hås86].

7.5.2 Lower Bounds for General Circuits

We use the following diagonalization result.

Lemma 7.32 ([HM95, Pol06])**.** *For any constants* c *and* d*,* EXP $\not\subseteq$ $\mathsf{DTIME}(2^{n^d})/n^c$*, and* PSPACE $\not\subseteq$ $\mathsf{DSPACE}(n^d)/n^c$*.*

The proof of Lemma 7.32 follows a very similar pattern as the proof that E^{PP} has a language that requires circuits of size $2^{\Theta(n)}$, which was proved in Section 7.2.

Corollary 7.33. EXP *is not in* poly*-weakly-uniform* $\mathsf{SIZE}(2^{n^{o(1)}})$.

Proof. Let L be an arbitrary language in poly-weakly-uniform SIZE$(2^{n^{o(1)}})$. For any input length n, given advice of length poly$(\log 2^{n^{o(1)}}) \leq n^{o(1)}$, we can construct a circuit for L of size $2^{n^{o(1)}}$ in time at most $2^{n^{o(1)}}$, and evaluate it on any given input of size n in time at most $2^{n^{o(1)}}$. Thus, $L \in$ DTIME$(2^{n^{o(1)}})/n^{o(1)}$. Using Lemma 7.32, construct $L_{diag} \in$ EXP which is not in DTIME$(2^n)/n$. By the above, this L_{diag} is not in poly-weakly-uniform SIZE$(2^{n^{o(1)}})$.

□

Recall that a Boolean circuit is called a *formula* if the underlying DAG is a tree (i.e., the fan-out of each gate is at most 1). We denote by FSIZE$(s(n))$ the class of families of Boolean formulas of size $s(n)$. We use a modified definition of the the direct-connection language for bounded fan-in formulas with AND, OR, and NOT gates: we assume that, for any given gate in the formula, we can determine in polynomial time who its parent gate is, and who its left and right input gates are. Lynch [Lyn77] gave a log-space algorithm for the Boolean formula evaluation problem, which can be adapted to work also in the case of input formulas given by the direct connection language (instead of the usual infix notation).

Lemma 7.34 (implicit in [Lyn77])**.** *Let $\{F_n\}$ be a uniform family of Boolean formulas of size $s(n)$. There is a* poly$(\log s(n))$*-space algorithm that, on input x of length n, computes $F_n(x)$.*

sketch. The input formula can be viewed as a tree, where each node has at most two children, and the evaluation algorithm will traverse the tree following specific rules. We assume that the formula is well-formed, which can be verified in poly$(\log s(n))$-space. The traversal starts from the left-most leaf, which can be identified in space poly$(\log s(n))$. Then, we traverse the tree such that, for each node A, *(i)* when we arrive at A from its left child, we either go to its parent (if the value of the left child fixes the value of A), or go to its right child and continue traversing the tree; *(ii)* when we arrive at A from its right child, we go directly to A's parent (the value of A is now determined by the value of the right child, as we know the left child has already been visited). The final node in this traversal is the root, which has no parent. The traversal is in poly$(\log s(n))$-space since we only need to remember the current node of the tree (and the direct-connection language is decided in time, and hence also in space, at most poly$(\log s(n))$) .

□

We have the following.

Theorem 7.35. PSPACE *is not in* poly*-weakly-uniform* FSIZE$(2^{n^{o(1)}})$.

Proof. Let L be an arbitrary language decided by a family $\{F_n\}$ of poly-weakly-uniform Boolean formulas of size $2^{n^{o(1)}}$; its direct connection language is decided in deterministic time $n^{o(1)}$ with advice of size $n^{o(1)}$. Using Lemma 7.34 (generalized in the straightforward way to handle weakly-uniform formulas), we get that L can be decided in $\mathsf{DSPACE}(n^{o(1)})/n^{o(1)}$. Appealing to Lemma 7.32 completes the proof.

□

7.6 Conclusion

We have shown how to use indirect diagonalization to prove lower bounds against weakly-uniform circuit classes. In particular, we have proved that PERMANENT cannot be computed by polynomial-size TC^0 circuits that are only slightly uniform (whose direct-connection language can be efficiently computed using sub-linear amount of advice). We have also extended to the weakly-uniform setting other circuit lower bounds that were previously known for the uniform case.

One obvious open problem is to improve the TC^0 circuit lower bound for PERMANENT to be exponential, which is not known even for the uniform case. Another problem is to get super-polynomial uniform TC^0 lower bounds for a language from a complexity class below $\#\mathsf{P}$ (e.g., PH). Strongly exponential lower bounds even against uniform AC^0 would be very interesting. One natural problem is to prove a better lower bound against *uniform* AC^0 (say for PERMANENT) than the known non-uniform AC^0 lower bound for PARITY.

Another natural question is if our techniques allow the $n^{o(1)}$ amount of non-uniformity in our results to be pushed any higher. It seems progress in this direction will need new ideas and/or a new framework. The framework used in this and previous work all encounter a roughly inverse relationship between the size of circuits in the lower bound and the amount of non-uniformity that can be handled. In Theorem 7.27 hardness holds if the inequality stated in the theorem holds. The inequality requires that the amount of advice be an inverse of $s^{(d(n))}$. This arises in the proof due to the nature in which the assumed easiness of PERMANENT is used repeatedly in Lemma 7.15, and a similar issue arises in earlier work in this area [All99, KP09, JS11]. Furthermore, the proofs of our main results relativize, but it is known that proving results with larger non-uniformity, say $\geq n$ bits, requires non-relativizing techniques. Thus to make progress we ought to look at utilizing techniques such as the interactive proofs for the PERMANENT, random self-reducibility, and combinatorial

properties of threshold circuits.

Chapter 8

Conclusions and Future Work

In this thesis we explore connections between meta-algorithms (algorithms analyzing computing objects such as circuits) and circuit lower bounds, deriving several non-trivial satisfiability algorithms and average-case lower bounds for small boolean formulas and circuits. Furthermore, we introduce the notion of compressibility of truth tables, show that non-trivial compression algorithms imply circuit lower bounds, and also derive compression algorithms for several restricted circuit classes.

It was derived in [KRT13] the best known average-case lower bounds for de Morgan formulas of size $O(n^{2.99})$ and formulas over the full basis of size $O(n^{1.99})$, almost matching with the worst-case lower bounds. Combining with our techniques in Chapter 3 and 4, the results in [KRT13] give non-trivial compression algorithms and randomized #SAT algorithms for formulas of sizes for which the lower bounds hold. The best known deterministic #SAT algorithms for de Morgan formulas are for size $O(n^{2.63})$ as shown in Chapter 5. An immediate open question is to get a deterministic #SAT algorithms for de Morgan formulas of size $O(n^{2.99})$. More generally, we will need new techniques to get lower bounds or non-trivial algorithms for de Morgan formulas of size $\Omega(n^3)$.

In Chapter 6, we get average-case lower bounds and non-trivial #SAT and compression algorithms for small-sized boolean circuits (almost $3n$-size over the de Morgan basis and almost $2.5n$-size over the full basis). There is still a gap to match with the best known worst-case lower bounds (size $5n$ for the de Morgan basis and $3n$ for the full basis). Another open question is to get pseudo-random generators for circuits of such small size. A long-term goal along this direction is to derive superlinear-size circuit lower bounds.

Bibliography

[AB09] S. Arora and B. Barak. *Complexity theory: a modern approach*. Cambridge University Press, New York, 2009. 56, 94

[ABCR99] A.E. Andreev, J. L. Baskakov, A. E. F. Clementi, and J. D. P. Rolim. Small pseudo-random sets yield hard functions: New tight explict lower bounds for branching programs. In *ICALP*, pages 179–189, 1999. 51

[AG94] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM Journal on Computing*, 23(5):1026–1049, 1994. 18, 107, 109, 113, 115, 117, 118, 119, 128, 135

[AGHP92] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost $k-$wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992. (preliminary version in FOCS'90). 39

[Agr05] M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proceedings of the Twenty-Fifth Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 92–105, 2005. 52, 114

[AHM$^+$08] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M.E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM Journal on Computing*, 38(1):63–84, 2008. 51, 55, 68

[Ajt83] M. Ajtai. σ_1^1-formulae on finite structures. *APAL*, 24:1–48, 1983. 17, 18

[All99] E. Allender. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, 1999. 105, 107, 109, 113, 114, 115, 118, 119, 121, 124, 138

[And87] A.E. Andreev. On a method of obtaining more than quadratic effective lower bounds for the complexity of π-schemes. *Vestnik Moskovskogo Universiteta. Matematika*, 42(1):70–73, 1987. English translation in *Moscow University Mathematics Bulletin*. 15, 19, 20, 21, 22, 25, 35, 51, 54, 72

[Ang87] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987. 51

[BBC+01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001. 92

[Bea94] P. Beame. A switching lemma primer. Technical report, Department of Computer Science and Engineering, University of Washington, 1994. 58

[Ber82] S.J. Berkowitz. On some relationships between monotone and non- circuit complexity. Technical report, University of Toronto, 1982. 68

[BFT98] H. Buhrman, L. Fortnow, and L. Thierauf. Nonrelativizing separations. In *Proceedings of the Thirteenth Annual IEEE Conference on Computational Complexity*, pages 8–12, 1998. 106

[BIS90] D.A.M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990. 106, 107, 115, 118, 119

[BIS12] Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating ac^0 by small height decision trees and a deterministic algorithm for $\#ac^0$ sat. In *Proceedings of the 2012 IEEE Conference on Computational Complexity (CCC)*, CCC '12, 2012. 18, 22, 50, 69, 74

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984. 50

[Bou07] J. Bourgain. On the construction of affine-source extractors. *Geometric and Functional Analysis*, 17(1):33–57, 2007. 103

[Bra10] M. Braverman. Polylogarithmic independence fools AC^0 circuits. *Journal of the Association for Computing Machinery*, 57:28:1–28:10, 2010. 50

[BT94] R. Beigel and J. Tarui. On acc. *Computational Complexity*, 4:350–366, 1994. 18, 135

[Chv79] V. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979. 57

[CIP09] C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009*, pages 75–85, 2009. 22

[CK12] Ruiwen Chen and Valentine Kabanets. Lower bounds against weakly uniform circuits. In *COCOON*, pages 408–419, 2012. 7

[CKK13a] R. Chen, V. Kabanets, and J. Kinne. Lower bounds against weakly-uniform threshold circuits. *Algorithmica*, August 2013. 7

[CKK+13b] R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Electronic Colloquium on Computational Complexity*, 20(57), 2013. 71, 72, 73, 74, 86

[CKK+14] R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman. Mining circuit lower bound proofs for meta-algorithms. In *Proceedings of the 29th Annual IEEE Conference on Computational Complexity*, CCC '14, 2014. 6, 19, 91, 94, 100

[CKS81] A.K. Chandra, D. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981. 117

[CKS13] R. Chen, V. Kabanets, and N. Saurabh. An improved deterministic #sat algorithm for small de morgan formulas. *Electronic Colloquium on Computational Complexity*, 20(150), 2013. 6, 69

[Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971. 17, 22

[DH09] E. Dantsin and E.A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. 2009. 22

[DK11] Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a 3n - o(n) lower bound on the circuit complexity of affine dispersers. In *MFCS*, pages 256–265, 2011. 103

[Fel09] V. Feldman. Hardness of approximate two-level logic minimization and PAC learning with membership queries. *Journal of Computer and System Sciences*, 75(1):13–26, 2009. 55

[FK06] L. Fortnow and A. Klivans. Linear advice for randomized logarithmic space. In *Proceedings of the Twenty-Third Annual Symposium on Theoretical Aspects of Computer Science*, pages 469–476, 2006. 52

[FSS84] M. Furst, J.B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, April 1984. 14, 17, 51, 106, 118, 119

[GKR+95] F. Green, J. Köbler, K.W. Regan, T. Schwentick, and J. Toran. The power of the middle bit of a $\#\mathrm{P}$ function. *Journal of Computer and System Sciences*, 50:456–467, 1995. 118, 119

[GS10] P. Gopalan and R. A. Servedio. Learning and lower bounds for AC^0 with threshold gates. In *Proceedings of the 13th international conference on Approximation, and 14th International conference on Randomization, and combinatorial*

optimization: algorithms and techniques, APPROX/RANDOM'10, pages 588–601, Berlin, Heidelberg, 2010. Springer-Verlag. 69

[GS12] A. Gabizon and R. Shaltiel. Increasing the output length of zero-error dispersers. *Random Struct. Alg.*, 40:74–104, 2012. 36, 37

[Hås86] J. Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 6–20, 1986. 18, 50, 51, 58, 74, 92, 106, 136

[Hås98] J. Håstad. The shrinkage exponent of de Morgan formulae is 2. *SIAM Journal on Computing*, 27:48–64, 1998. 15, 19, 21, 49, 50, 51, 56, 59, 72, 73, 75, 76, 89, 91

[Hås12] Johan Håstad. On the correlation of parity and small-depth circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:137, 2012. 18, 92

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:1364–1396, 1999. 50

[HLS65] J. Hartmanis, P.L. Lewis II, and R.E. Stearns. Hierarchies of memory-limited computations. In *Proceedings of the Sixth Annual IEEE Symposium on Switching Circuit Theory and Logic Design*, pages 179–190, 1965. 10

[HM95] S. Homer and S. Mocas. Nonuniform lower bounds for exponential time classes. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science*, MFCS '95, pages 159–168, London, UK, 1995. Springer-Verlag. 136

[HS65] J. Hartmanis and R.E. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965. 10

[HS82] J. Heintz and C.-P. Schnorr. Testing polynomials which are easy to compute. *L'Enseignement Mathématique*, 30:237–254, 1982. 52, 114

[IK14] Russell Impagliazzo and Valentine Kabanets. Fourier concentration from shrinkage. In *Proceedings of the 29th Annual IEEE Conference on Computational Complexity*, CCC '14, 2014. 100

[IKW01] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity*, pages 1–11, 2001. 16

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002. 52, 54, 66, 67

[ILMR02] Kazuo Iwama, Oded Lachish, Hiroki Morizumi, and Ran Raz. An explicit lower bound of 5n - o(n) for boolean circuits. In *Proc. of MFCF*, pages 353–364. Springer-Verlag, 2002. 94, 100

[IM02] K. Iwama and H. Morizumi. An explicit lower bound of $5n - o(n)$ for boolean circuits. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, MFCS '02, pages 353–364, London, UK, UK, 2002. Springer-Verlag. 106

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. 10

[IMP12] R. Impagliazzo, W. Matthews, and R. Paturi. A satisfiability algorithm for AC^0. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 961–972, 2012. 18, 22, 50, 58, 69, 74, 99

[IMZ12] Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from shrinkage. In *FOCS*, pages 111–119, 2012. 21, 22, 49, 50, 59, 72, 74

[IN93] R. Impagliazzo and N. Nisan. The effect of random restrictions on formula size. *Random Structures and Algorithms*, 4(2):121–134, 1993. 21, 72

[IW97] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997. 50

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974. 57

[JS11] M.J. Jansen and R. Santhanam. Permanent does not have succinct polynomial size arithmetic circuits of constant depth. In *Proceedings of the Thirty-Eighth International Colloquium on Automata, Languages, and Programming, Part I*, pages 724–735, 2011. 105, 107, 108, 109, 113, 114, 115, 138

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55:40–56, 1982. 13, 52, 106, 123, 134

[KC00] V. Kabanets and J.-Y. Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 73–79, 2000. 55

[Khr71] V.M. Khrapchenko. A method of determining lower bounds for the complexity of π-schemes. *Matematicheskie Zametki*, 10(1):83–92, 1971. English translation in *Mathematical Notes of the Academy of Sciences of the USSR*. 20

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1–2):1–46, 2004. 52, 114

[Kin12] Jeff Kinne. On tc0 lower bounds for the permanent. In *COCOON*, pages 420–432, 2012.

[KKM12] G. Karakostas, J. Kinne, and D. van Melkebeek. On derandomization and average-case complexity of monotone functions. *Theoretical Computer Science*, 434:35–44, 2012. 68

[KL82] R.M. Karp and R.J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(3-4):191–209, 1982. 13, 16, 116

[KP09] P. Koiran and S. Perifel. A superpolynomial lower bound on the size of uniform non-constant-depth threshold circuits for the permanent. In *Proceedings of the Twenty-Fourth Annual IEEE Conference on Computational Complexity*, pages 35–40, 2009. 105, 107, 109, 113, 114, 124, 138

[KR12] I. Komargodski and R. Raz. Average-case lower bounds for formula size. *Electronic Colloquium on Computational Complexity (ECCC)*, 19, 2012. (to appear in STOC'13). 91

[KR13] I. Komargodski and R. Raz. Average-case lower bounds for formula size. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, pages 171–180, 2013. 21, 22, 23, 26, 27, 35, 36, 39, 49, 53, 54, 55, 59, 73, 86

[KRT13] Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for demorgan formula size. In *FOCS*, pages 588–597, 2013. 19, 21, 55, 69, 72, 73, 89, 91, 140

[Lev73] L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. 22

[Li11] X. Li. A new approach to affine extractors and dispersers. In *IEEE Conference on Computational Complexity*, pages 137–147, 2011. 103

[LMN93] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the Association for Computing Machinery*, 40(3):607–620, 1993. 50, 69

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975. 57

[LR01] O. Lachish and R. Raz. Explicit lower bound of $4.5n - o(n)$ for boolena circuits. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 399–408, New York, NY, USA, 2001. ACM. 106

[Lup58] O.B. Lupanov. On the synthesis of switching circuits. *Doklady Akademii Nauk SSSR*, 119(1):23–26, 1958. English translation in *Soviet Mathematics Doklady*. 51, 107

[Lyn77] N.A. Lynch. Log space recognition and translation of parenthesis languages. *Journal of the Association for Computing Machinery*, 24:583–590, 1977. 137

[Mas79] W.J. Masek. Some NP-complete set covering problems. Manuscript, 1979. 55

[MVW99] P. B. Miltersen, N.V. Vinodchadran, and O. Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In T. Asano, H. Imai, D.T. Lee, S. Nakano, and T. Tokuyama, editors, *Proceedings of the Fifth Annual International Conference on Computing and Combinatorics*, volume 1627 of *Lecture Notes in Computer Science*, pages 210–220. Springer Verlag, 1999. (COCOON'99). 106, 134

[Nec66] E.I. Nechiporuk. On a Boolean function. *Doklady Akademii Nauk SSSR*, 169(4):765–766, 1966. English translation in *Soviet Mathematics Doklady*. 51

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994. 50, 52

[Pip77] N. Pippenger. The complexity of monotone boolean functions. *Theory of Computing Systems*, 11:289–316, 1977. 68

[Pol06] C. Pollett. Languages to diagonalize against advice classes. *Computational Complexity*, 14:341–361, 2006. 136

[PPSZ98] R. Paturi, P. Pudlák, M.E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the Thirty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 628–637, 1998. 17

[PPZ99] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 1999. 17, 74

[PZ93] M. Paterson and U. Zwick. Shrinkage of de Morgan formulae under restriction. *Random Structures and Algorithms*, 4(2):135–150, 1993. 21, 71, 72, 73, 74, 75, 79, 80, 81, 82, 89

[Rao09] A. Rao. Extractors for low-weight affine sources. In *Proceedings of the Twenty-Fourth Annual IEEE Conference on Computational Complexity*, pages 95–101, 2009. 38

[Raz87] A.A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes*, 41:333–338, 1987. 14, 69, 70, 106

[Raz93] A.A. Razborov. Bounded arithmetic and lower bounds in boolean complexity. In *Feasible Mathematics II*, pages 344–386. Birkhauser, 1993. 58

[Red79] N.P. Red'kin. On the realization of monotone boolean functions by contact circuits. *Problemy Kibernetiki*, 35:87–110, 1979. (in Russian). 68

[Rei11] B. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 560–569, 2011. 92

[RR97] A.A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55:24–35, 1997. 50, 51, 66, 68, 106

[Ruz81] W.L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. 107, 115

[San10] R. Santhanam. Fighting perebor: New and improved algorithms for formula and qbf satisfiability. In *Proceedings of the Fifty-First Annual IEEE Symposium on Foundations of Computer Science*, pages 183–192, 2010. 19, 21, 22, 23, 26, 33, 35, 42, 49, 50, 54, 56, 59, 72, 73, 74, 76, 91, 92, 99

[Sch74] Claus-Peter Schnorr. Zwei lineare untere schranken für die komplexität boolescher funktionen. *Computing*, 13(2):155–171, 1974. 92, 94, 98

[Sch02] U. Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32:615–623, 2002. 17

[Sha49] C.E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949. 13, 15, 106, 124

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 77–82, 1987. 14, 69, 70, 106

[ST12] Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. In *IEEE Conference on Computational Complexity*, pages 107–116, 2012. 19, 22, 23, 34, 42, 43, 45, 50, 74, 91

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. 39

[Sub61] B.A. Subbotovskaya. Realizations of linear functions by formulas using and, or, not. *Soviet Math. Doklady*, 2:110–112, 1961. 18, 20, 21, 22, 23, 49, 50, 51, 54, 59, 71, 72, 73, 91, 92

[SZ96] P. Savický and S. Zák. A large lower bound for 1-branching programs. *Electronic Colloquium on Computational Complexity*, TR96-036, 1996. 63

[Tod91] Seinosuke Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. 106

[Tor91] J. Torán. Complexity classes defined by counting quantifiers. *Journal of the Association for Computing Machinery*, 38:752–773, 1991. 118

[TW11] M. Tulsiani and J. Wolf. Quadratic Goldreich-Levin theorems. In *Proceedings of the Fifty-Second Annual IEEE Symposium on Foundations of Computer Science*, pages 619–628, 2011. 70

[TX13] L. Trevisan and T. Xue. A derandomized switching lemma and an improved derandomization of AC^0. In *Proceedings of the Twenty-Eighth Annual IEEE Conference on Computational Complexity*, pages 242–247, 2013. 74

[Val79] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979. 107, 120, 126, 128, 135

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. 51

[Wag86] K.W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Inf.*, pages 325–356, 1986. 118

[Wil10] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the Forty-Second Annual ACM Symposium on Theory of Computing*, 2010. 16, 18, 50, 52, 74

[Wil11] R. Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of the Twenty-Sixth Annual IEEE Conference on Computational Complexity*, pages 115–125, 2011. 14, 16, 18, 22, 53, 67, 74, 106

[Yab59] S.V. Yablonski. On the impossibility of eliminating perebor in solving some problems of circuit theory. *Doklady Akademii Nauk SSSR*, 124(1):44–47, 1959. English translation in *Soviet Mathematics Doklady*. 54

[Yao82] A.C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982. 50

[Yao85] A.C. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the Twenty-Sixth Annual IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985. 51, 92, 106

[Yeh11] A. Yehudayoff. Affine extractors over prime fields. *Combinatorica*, 31(2):245–256, 2011. 103

[Zák83] S. Zák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26:327–333, 1983. 10, 121

[Zan91] Viktória Zankó. #p-completeness via many-one reductions. *Int. J. Found. Comput. Sci.*, 2(1):77–82, 1991. 120, 126

[Zan98] F. Zane. *Circuits, CNFs, and Satisfiability*. PhD thesis, UCSD, 1998. 22

[Zwi91] Uri Zwick. A 4n lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic boolean functions. *SIAM J. Comput.*, 20(3):499–505, 1991. 94