# Mining Circuit Lower Bound Proofs for Meta-Algorithms*

Ruiwen Chen† Valentine Kabanets‡ Antonina Kolokolova § Ronen Shaltiel¶
David Zuckerman‖

April 3, 2013

**Abstract**

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 $\mathsf{poly}(2^n)$ 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 $\mathsf{AC}^0$ 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, KR12]. 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 [KR12] 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 $\mathsf{NEXP}$.

*This paper subsumes the earlier technical report [KK13] by the second and third authors.

†School of Computing Science, Simon Fraser University, Burnaby, BC, Canada; ruiwenc@sfu.ca

‡School of Computing Science, Simon Fraser University, Burnaby, BC, Canada; kabanets@cs.sfu.ca

§Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada; kol@cs.mun.ca

¶Department of Computer Science, University of Haifa, Haifa, Israel; ronen@cs.haifa.ac.il

‖Department of Computer Science, University of Texas, Austin, TX, USA; diz@cs.utexas.edu

# 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 $\mathsf{AC}^0$-computable Boolean functions [LMN93], a Circuit-SAT algorithm for $\mathsf{AC}^0$ circuits [IMP12, BIS12] (using Håstad's Switching Lemma, a main tool used in $\mathsf{AC}^0$ lower bound proofs [Hås86]), a simple pseudorandom generator for $\mathsf{AC}^0$ 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 paper, 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

[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]).

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 $\mathsf{AC}^0$ 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]).

**Compression Theorem:** *(1) Boolean $n$-variate functions computed by $\mathsf{AC}^0$ circuits of size $s$ and depth $d$ are compressible in time $\mathsf{poly}(2^n)$ 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 $\mathsf{poly}(2^n)$ 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 $\mathsf{poly}(2^n)$ 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.

### 1.1 Our results

In addition to the Compression Theorem mentioned above, we have results on shrinkage of (de Morgan) formulas, #SAT-algorithms and average-case lower bounds for small (de Morgan) formulas, and circuit lower bounds implied by compression algorithms. These are detailed next.

**Shrinkage of formulas.** The classical result of Subbotovskaya [Sub61] shows that if one randomly chooses $n - k$ variables of a given $n$-variate de Morgan formula, and sets each to 0 or 1 uniformly at random, then the expected size of the resulting formula is about $(k/n)^\Gamma \cdot |F|$, where $\Gamma$ (called the *shrinkage exponent*) is $3/2$; this $\Gamma$ was subsequently improved to the optimal value 2 by Håstad [Hås98].

[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 paper, however, will be on lossless compression algorithms.

This "shrinkage in expectation" result 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 [KR12] 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, KR12], 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 $0 \leqslant k \leqslant n$), 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 $\mathsf{NP}$-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 $\mathsf{AC}^0$-SAT, [San10, ST12] for Formula-SAT, and [Wil11] for $\mathsf{ACC}^0$-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 [KR12]: *There is a Boolean function* $f : \{0,1\}^n \to \{0,1\}$ *computable in* $\mathsf{P}$ *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* $0 < \sigma < 1$.

**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 superpolynomial 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 superpolynomial circuit lower bounds against $\mathcal{C}$ for a language in $\mathsf{NEXP}$. 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* $\mathsf{NEXP}$.

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 $\mathsf{ACC}^0$ functions. Apart from getting an alternative proof, the hope is that such a compression algorithm would give us more insight into the structure of $\mathsf{ACC}^0$ functions, which could lead to $\mathsf{ACC}^0$ circuit lower bounds against a much more explicit Boolean function, say the one in $\mathsf{NP}$ or in $\mathsf{P}$.

### 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 descritpion 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 $\mathsf{poly}(2^n)$ 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 [KR12]: For a de Morgan formula $F$ on $n$ variables, we consider the sequence of random variables $X_i$, $1 \leqslant i \leqslant n$, 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 [KR12] with the following differences: (1) our restrictions always choose deterministically which variable to restrict (as opposed to restrictions of [KR12] 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 [KR12]). 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 [KR12]'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.

**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 $\mathsf{NP}$-hard, for the approximation factor $n^\gamma$ for some constant $0 < \gamma < 1$.

**Concurrent independent work.** As we were completing this manuscript, 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 [KR12] 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 [KR12]). Similarly to our paper but independently of our work, Komargodski et al. [KRT13] also adapt Andreev's method to the case of arbitrary (not necessarily completely random) restrictions by using appropriate randomness extractors.

**The remainder of the paper.** We give basic definitions in Section 2. We prove our compression theorem in Section 3, and the shrinkage result in Section 4. We give our #SAT algorithms in Section 5. Average-case formula lower bounds are proved in Section 6. We prove that compression implies circuit lower bounds in Section 7. We conclude with open questions in Section 8. The appendix contains some technical details omitted from the main body of the paper.

# 2 Preliminaries

## 2.1 Circuits

Here we recall some basic definitions of circuit classes considered in our paper; for more background on circuit complexity, consult any of the following [Weg87, BS90, Juk12].

A *literal* is either a variable, or the negation of a variable; the sign of the variable is said to be positive in the first case, and negative otherwise. A *DNF* is a disjunction of terms, where each *term* is a conjunction of literals. The following is a basic fact: For any subset $S \subseteq \{0,1\}^n$ of size $t$, there is a DNF $D(x_1, \ldots, x_n)$ on $t$ terms that evaluates to 1 on each $a \in S$, and is 0 outside of $S$.

A *Boolean circuit* on $n$ inputs is a directed acyclic graph with a single node of out-degree 0 (the output gate), and $n$ in-degree 0 nodes (input gates), where each input gate is labeled by one of the variables $x_1, \ldots, x_n$, and each non-input gate by a logical function on at most 2 inputs (e.g., AND, OR, and NOT). The *size* of the circuit is the total number of gates; the *depth* is the length of a longest path in the circuit from an input gate to the output gate. The class $\mathsf{AC}^0$ is a class of constant-depth circuits with NOT, AND and OR gates, where AND and OR gates have unbounded fan-in. For a circuit class $\mathcal{C}$ and a size function $s(n)$, we denote by $\mathcal{C}[s(n)]$ the class of $s(n)$-size $n$-input circuits of the type $\mathcal{C}$. When no $s(n)$ is explicitly mentioned, it is assumed to be some $\mathsf{poly}(n)$.

A *Boolean formula* $F$ on $n$ input variables $x_1, \ldots, x_n$ is a tree whose root node is the output gate, and whose leaves are labeled by literals over the variables $x_1, \ldots, x_n$; all non-input gates are labeled by logical functions over 2 inputs. The size of the formula $F$, denoted by $L(F)$, is the total number of leaves. A *de Morgan formula* is a formula where the only logical functions used are AND and OR.

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 $x_i$, $1 \leqslant i \leqslant n$. 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* $\rho$ 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 $\rho$, we define the restriction $F|_\rho$ as the circuit on the free variables of $\rho$, 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 $\psi$ 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 $\psi$. (3) If $y \vee \psi$ appears, then replace all occurrences of $y$ in $\psi$ by 0 and $\overline{y}$ by 1; if $y \wedge \psi$ appears, then replace all occurrences of $y$ in $\psi$ 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 $s$.

### 2.2 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 $\epsilon$-*close* if for any subset $A \subseteq \{0,1\}^n$, it holds that $|\mathbf{Pr}[X \in A] - \mathbf{Pr}[Y \in A]| \leqslant \epsilon$.

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* $(k,\epsilon)$-*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 $\epsilon$-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 $0 \leqslant \rho \leqslant 1$ and $L \geqslant 1$, we say a code $C$ is $(\rho, L)$-*list-decodable* if for any $y \in \{0,1\}^n$, there are at most $L$ codewords in $C$ within the Hamming distance at most $\rho n$ from $y$. The Johnson bound (see, e.g., [AB09]) says that, for any $\delta \geqslant \sqrt{\epsilon}$, an $(n,k,(1/2-\epsilon)n)$-code is $(1/2-\delta, 1/(2\delta^2))$-list-decodable.

### 2.3 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 $K(x) \geqslant n$, and, more generally, for any $0 < \alpha < 1$, we have that $K(x) \geqslant \alpha n$ for all but at most $2^{-(1-\alpha)n}$ fraction of $n$-bit strings $x$.

## 3 Compression from restriction-based circuit lower bounds

Here we prove the Compression Theorem stated in the Introduction.

### 3.1 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 $\ell$ 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 $\ell$ 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 $|U| \cdot (1 - 1/\ell)^t \leqslant |U| \cdot e^{-t/\ell}$, 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 $\ell$ 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 $\ell$ 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 $\mathsf{poly}(2^n)$.

### 3.2 Compression of $\mathsf{AC}^0$ functions via DNFs

The known lower bounds for $\mathsf{AC}^0$ circuits are based on the fact that almost all random restrictions simplify a small $\mathsf{AC}^0$ 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 $\mathsf{AC}^0$ 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 $\mathsf{AC}^0$ circuit has an equivalent representation as a DNF with not too many terms.

**Lemma 3.1** ([IMP12])**.** *Every depth $d$ Boolean circuit $C$ with $s$ gates on $n$ inputs has an equivalent DNF with at most $\mathsf{poly}(n) \cdot s \cdot 2^{n(1-\mu)}$ terms, where $\mu \geqslant 1/O(\log(s/n) + d \log d)^{d-1}$.*

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

**Theorem 3.2.** *There is a deterministic $\mathsf{poly}(2^n)$-time algorithm $A$ satisfying the following. Let $f : \{0,1\}^n \to \{0,1\}$ be any Boolean function computable by an $\mathsf{AC}^0$ 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 $\mathsf{poly}(n) \cdot s \cdot 2^{n(1-\mu)}$ terms, where $\mu \geqslant 1/O(\log s)^{d-1}$.*

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

### 3.3 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 $\mathsf{AC}^0$, 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 [KR12] 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, KR12]. 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 [KR12], 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 [KR12].

### 3.3.1 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 $0 \leqslant i \leqslant n$, 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 $1 \leqslant i \leqslant n$, 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 3.3** (Shrinkage Lemma)**.** *Let $F$ be any given (de Morgan) formula or a branching program on $n$ variables. For any $k \geqslant 4$, we have*

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

*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 postpone the proof of Shrinkage Lemma till Section 4. Now we apply this lemma to obtain the following structural characterization of small formulas and branching programs, which will be useful for compression.

**Corollary 3.4.** *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 $n^\gamma$ .*

*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 3.3 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 $O(n^d/n^{1.5(1-\delta)}) = O(n^{1-\nu/2})$. □

### 3.3.2 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 $\ell \leqslant 2^n$, where all but one circuit are small, while the remaining circuit accepts few inputs. More precisely, we have the following.

**Theorem 3.5.** *There is a deterministic* $\mathsf{poly}(2^n)$*-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 $1 \leqslant \ell \leqslant 2^n$, 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 $\alpha$ of points in $\{0,1\}^n$, for some $0 \leqslant \alpha < 1$.*

*Given the truth table of $f$ and the parameters $\ell$, $c$, and $\alpha$, 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 $\beta \leqslant 2\alpha$, 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 $\mathcal{S} = \varnothing$.
>
> 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 $1/(2\ell)$ 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 $\mathcal{S}$.

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

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 $2 \cdot |W| \leqslant 2\alpha 2^n$, 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 $\ell$ 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 $1 \leqslant i \leqslant t$, 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 3.6.** *There is an efficient compression algorithm that, given the truth table of a formula (branching program) $F$ on $n$ variables of size $L(F) \leqslant n^d$, the algorithm produces an equivalent Boolean circuit of size at most $2^{n-n^\epsilon}$, 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 3.4, 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 $n^\gamma$ 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 $i$, $1 \leqslant i \leqslant 2^m$, of the decision tree, the conjunction $c_i$ of $m$ literals that defines the corresponding restriction. Also let $F_i$, for $1 \leqslant i \leqslant 2^m$, 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 $n^\gamma$. 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 $1 \leqslant i \leqslant \ell$, 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 $1 \leqslant i \leqslant \ell$, is of size at most $O(m + n^\gamma) \leqslant O(n)$. 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 $n^\gamma$ by at most $O(n^\gamma \log n) \leqslant O(n)$ bits in the standard way.

Thus we get that $F \equiv \vee_{i=1}^{\ell+1} C_i$ satisfies the assumption of Theorem 3.5. Running the greedy algorithm of Theorem 3.5, we get a circuit for $F$ of total size at most $O(\ell n^2 + \alpha n 2^n) \leqslant \mathsf{poly}(n) \cdot 2^{n-n^\delta}$. □

### 3.4 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 3.7.** *Every read-once branching program $F$ of size $s$ on $n$ inputs has an equivalent full read-once branching program $F'$ of size $s' \leqslant 3n \cdot s$.*

*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 $t \geqslant 2$, 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 $\Delta_i \neq \varnothing$, 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 $\Delta_i$ (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 3.8.** *There is a deterministic $\mathsf{poly}(2^n)$-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 3.7, $f$ is computable by a full read-once branching program $F$ of size $s' = 3sn$. For $0 \leqslant k \leqslant n$ 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

$$f \equiv \vee_{v \in B}(h_v \wedge g_v). \tag{1}$$

Consider any $v \in B$. Let $\rho$ 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 $\rho'$ 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. (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 $\rho$ 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 $\rho'$ 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 $\mathsf{poly}(2^n)$ since we only need to enumerate all $O(n)$-size descriptions. □

# 4 Shrinkage of de Morgan Formulas

Here we prove the Shrinkage Lemma. We use the adaptive restrictions of [San10] (each time randomly restricting the most frequent variable in the formula). Following [KR12], 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 [KR12], 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.

## 4.1 A variant of Azuma's Inequality

**Lemma 4.1.** *Let $Y$ be a random variable taking two values with equal probability. If $\mathbf{E}[Y] \leqslant 0$ and there exists $c \geqslant 0$ such that $Y \leqslant c$, then for any $t \geqslant 0$, we have $\mathbf{E}[e^{tY}] \leqslant e^{t^2c^2/2}$.*

*Proof.* Suppose $Y$ takes two values $a$ and $b$ where $a \leqslant b \leqslant c$, and $\mathbf{Pr}[Y = a] = \mathbf{Pr}[Y = b] = \frac{1}{2}$. Consider the following two cases. If $b \leqslant 0$, then $e^{tY} \leqslant e^{t\cdot 0} = 1 \leqslant e^{t^2c^2/2}$. If $b > 0$, since $\mathbf{E}[Y] = \frac{1}{2}(a + b) \leqslant 0$, we have $a \leqslant -b$ and $\mathbf{E}[e^{tY}] = \frac{1}{2}(e^{ta} + e^{tb}) \leqslant \frac{1}{2}(e^{-tb} + e^{tb}) \leqslant e^{t^2b^2/2} \leqslant e^{t^2c^2/2}$, where we used the inequality $\frac{1}{2}(e^{-x} + e^{x}) \leqslant e^{x^2/2}$. □

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

**Lemma 4.2.** *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 $1 \leqslant i \leqslant n$, the random variable $Y_i$ (conditioned on $R_{i-1}, \ldots, R_1$) assumes two values with equal probability, and there exists a constant $c_i \geqslant 0$ such that $Y_i \leqslant c_i$, then, for any $\lambda$, we have*

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

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

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

where the last inequality is by Markov's inequality. We get

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

where the last inequality is by Lemma 4.1. By induction, we get $\mathbf{E}\left[e^{t\sum_{i=1}^n Y_i}\right] \leqslant e^{t^2\sum_{i=1}^n c_i^2/2}$. Thus, $\mathbf{Pr}[X_n - X_0 \geqslant \lambda] \leqslant e^{-\lambda t + t^2\sum_{i=1}^n c_i^2/2}$. Choosing $t = \lambda/\sum_{i=1}^n c_i^2$ yields the required bound. □

## 4.2 Shrinkage lemma

Recall our definition of a restriction. For a given de Morgan formula $F$ on $n$ variables, define $F_0 = F$. For $1 \leqslant i \leqslant n$, 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 re-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 4.3** (Shrinkage Lemma)**.** *Let $F$ be any given de Morgan formula on $n$ variables. For any $k \geqslant 4$, we have*

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

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

**Lemma 4.4.** *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 $L(F') \leqslant L(F) \cdot \left(1 - \frac{1}{n}\right)$, and $\mathbf{E}[L(F')] \leqslant L(F) \cdot \left(1 - \frac{1}{n}\right)^{3/2}$.*

*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, $L(F') \leqslant L(F) - L(F)/n = L(F) \cdot (1 - 1/n)$.

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

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

where the last inequality is by $1 - ax \leqslant (1-x)^a$ true for $0 \leqslant x \leqslant 1$ and $a \geqslant 1$ (see the Appendix). □

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

*Proof.* Using Lemma 4.4, we get $l_i \leqslant l_{i-1} + \log \left(1 - \frac{1}{n-i+1}\right)$; this implies $Z_i \leqslant c_i$. By Jensen's inequality, $\mathbf{E}[l_i \mid R_{i-1}, \ldots, R_1] \leqslant \log \mathbf{E}[L_i \mid R_{i-1}, \ldots, R_1]$, which, by Lemma 4.4, 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 $\mathbf{E}[Z_i \mid R_{i-1}, \ldots, R_1] \leqslant 0$, and so $\{X_i\}$ is indeed a supermartingale. □

Now we can complete the proof of the Shrinkage Lemma.

*Proof of Lemma 4.3.* Let $\lambda$ be arbitrary, and let $c_i$'s be as defined in Lemma 4.5. By Lemma 4.5 and Lemma 4.2, we get

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

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

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

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

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

Taking $i = n-k$, we get

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

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

# 5 #SAT algorithms for formulas

## 5.1 $n^{2.49}$-size de Morgan formulas and $n^{1.99}$-size general formulas

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

**Theorem 5.1.** *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 $t(n) \leqslant 2^{n-n^{\delta}}$, 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 4.3), 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 $2^{n-k} \cdot 2^{-k} \cdot 2^{k} \cdot \mathsf{poly}(n) \leqslant 2^{n-k} \cdot \mathsf{poly}(n)$.

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 $2^k \mathsf{poly}(n^\gamma) = 2^{n^\alpha} \cdot \mathsf{poly}(n)$. Including pre-processing, computing #SAT for all small formulas takes time at most $2^{n-k} \cdot \mathsf{poly}(n) + 2^{O(n^\gamma \log n)} \leqslant 2^{n-n^\alpha} \cdot \mathsf{poly}(n)$.

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

## 5.2 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 $\delta$ along the way (in [San10], the savings $\delta$ was some unspecified inverse polynomial in $c$).

**Theorem 5.2** ([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 $\delta \geqslant 1/(32 \cdot c^2)$.*

*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 5.1. By the Shrinkage Lemma (Lemma 4.3), all but $2^{-k}$ fraction of leaves have the formula size $L(F_{n-k}) \leqslant 2 \cdot L(F) \left(\frac{k}{n}\right)^{3/2} = 2 \cdot cn \cdot p^{3/2} = 2cp^{1/2} \cdot pn = \frac{1}{2}pn = \frac{k}{2}$.

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 $2^{n-\delta n}\mathsf{poly}(n)$ where $\delta = \frac{1}{32c^2}$. □

**Remark 5.3.** 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 4.2 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 $\delta$ about $c^{-c^3}$. We provide the details in the Appendix.

Finally, we observe that the proof of Theorem 5.2 immediately yields an average-case lower bound for linear-size de Morgan formulas [San10]. Indeed, by the proof of Theorem 5.2, 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 $n^{2.5}$.

# 6 Average-case hardness for de Morgan formulas of size $n^{2.49}$

Here we use our shrinkage result for adaptive restrictions to re-prove a recent result by Komargodski and Raz [KR12] on average-case hardness for de Morgan formulas. Our proof is more modular than the original argument of [KR12], 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).

## 6.1 Andreev's original argument

We sketch the original idea of Andreev first. Andreev [And87] defined a function $A : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ 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 $\Gamma$).

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 [KR12] and the argument of this paper.

### 6.2 Adapting Andreev's argument 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 6.1** ([GS12])**.** *There exist $c > 1$ and $0 < \eta < 1$ such that, for all sufficiently large $n$, $k > (\log n)^c$, there is a $\mathsf{poly}(n)$-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 $K(x_0) \geqslant 4n$, and consider the function $B'(y) = B(x_0, y)$. Suppose $B'(y)$ has a de Morgan formula $F$. The shrinkage result of Lemma 4.3 says that, after adaptively restricting $n-k$ variables via a random restriction $\rho$, the formula size will shrink with high probability. Denote by $F'$ the formula after a restriction $\rho$, i.e., $F' = F|_\rho$. Then,

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

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

> Given a formula $F'(y')$, a restriction $\rho$, and $n$ in binary, go over all values $0 \leqslant i \leqslant 4n-1$. For each $i$, find a pre-image $z = D^{-1}(i)$ consistent with the restriction $\rho$ (by trying all possible values for the free variables $y'$ and evaluating $D$ on the input described by the restriction $\rho$ 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 $0 \leqslant i \leqslant 4n-1$, 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 $\rho$, 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 $\rho$ (saying for each position $0 \leqslant i \leqslant n-1$ of $y$ whether it's 0, 1, or $*$), followed by the description of $F'$. We get

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

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 $L(F) \geqslant n^{2.5}/\mathsf{poly}\log n$, and hence also the function $B(x, y)$ requires de Morgan formulas of at least that size, up to a constant factor.

### 6.3 Average-case hardness

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

**Theorem 6.2** ([Rao09])**.** *There exist constants $d < 1$ and $c \geqslant 1$ 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 6.3.** *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 $(\rho, L)$-list decodable for $\rho = 1/2 - O(2^{-r/4})$ and $L \leqslant O(2^{r/2})$. 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 $\epsilon$-biased sample space. Using a powering construction from [AGHP92], we get such a set of size $(4n/\epsilon)^2$, where for each $1 \leqslant i \leqslant |S|$, we can compute the $i$th string in $S$ in time $\mathsf{poly}(n)$. For $x \in \{0,1\}^{4n}$ and position $1 \leqslant i \leqslant |S|$, we define the $i$th symbol of the codeword of $x$ by $C(x)_i = \langle x, y_i \rangle \mod 2$, where $y_i$ is the $i$th 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 $\epsilon$ so that $|S| = 2^r$, which yields $\epsilon = O(2^{-r/2})$. □

Loosely speaking, as in [KR12], 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 6.3 and $E$ is Rao's extractor (from Theorem 6.2) mapping $n$ bits to $m = r = n^\gamma$ bits, for the min-entropy $k \geqslant 2m$. We will prove the following.

**Theorem 6.4.** *Let $x_0$ be any fixed $(4n)$-bit string of Kolmogorov complexity $K(x_0) \geqslant 3n$. 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:

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

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 4.3) 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. (2) 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 $\rho$ (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. (2). By averaging, there exists a restriction $\rho$ 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 $(\rho, a)$ the input to the function $f'(y)$ obtained using the restriction $\rho$ and the values $a$ for the unrestricted variables $y'$. We have

$$\mathbf{Pr}_{y' \in \{0,1\}^k}[F'(y') = C(x_0)_{E(\rho,y')}] \geqslant \frac{1}{2} + \frac{1}{2^{n^\sigma}} - \frac{1}{2^k}. \tag{3}$$

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 6.2), the probability in Eq. (3) 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 $\rho$, 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 $\sigma$ sufficiently small as a function of $\gamma$ and $d$, we can ensure that $\epsilon \geqslant 1/2 + 2^{-n^{\gamma d/2}} = 1/2 + 2^{-r^{d/2}}$.

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 $\alpha_0$ 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 $n^{0.991} + 2n + r + O(\log n) \leqslant (2.1)n$ to describe the subformula $F'$, restriction $\rho$, internal randomness $\alpha_0$, and the input length $n$.

The list-decodability of the code $C$ (Theorem 6.3) 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 6.5.** *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 6.4. 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 $K(x_0) \geqslant 3n$, such that $F(x_0,y)$ has non-trivial agreement with $f(x_0,y)$ over random $y$'s. The latter contradicts Theorem 6.4. □

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

**Remark 6.6.** 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 6.4, by observing that a general formula (branching program) of size $n^{1.99}$ is also likey to shrink to size below $n^{0.99}$ (for the same parameter $k$), and then proceeding with the rest of the proof as before.

# 7 Circuit lower bounds from compression

Since incompressibility 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 $\mathsf{NEXP}$.

## 7.1 Arbitrary subclass of polynomial-size circuits

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

**Theorem 7.1.** *Let $\mathcal{C} \subseteq \mathsf{P/poly}$ 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 $\mathsf{NEXP} \not\subseteq \mathcal{C}$.*

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

**Claim 7.2.** *If $\mathsf{NEXP} \subseteq \mathcal{C}$, then for every $L \in \mathsf{NEXP}$ 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 $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ implies that, for every language $L \in \mathsf{NEXP}$, there exists a constant $c_L \in \mathbb{N}$ such that every sufficiently large input $x \in L$ has a $\mathsf{NEXP}$-witness that is the truth table of some Boolean function of circuit complexity $n^{c_L}$. For every $L \in \mathsf{NTIME}(2^{n^e})$, define a new language $L' \in \mathsf{EXP}$ 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 $\mathsf{NEXP}$-witness for $x \in L$. If no such witness is found, then output 0. Otherwise, output the $y$th 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 $y$th bit of the lex first witness for $x$ (as found by the algorithm enumerating all $n^{c_L}$ size circuits) is 1. Since $\mathsf{EXP} \subseteq \mathcal{C}$, 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 $\mathsf{NE}$, 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 $2^{cn} \leqslant 2^{n^2}$ for large enough $n$. We think of witnesses for $\mathsf{NE}$ 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 7.2 above, we get that there is a constant $c_0 \in \mathbb{N}$ such that yes-instances $x$, $|x| = n$, of every language in $\mathsf{NE}$ 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 $\mathsf{poly}(2^n)$-time compression algorithm for $n$-variate Boolean functions in $\mathcal{C}[n^{2c_0}]$. Consider the following $\mathsf{NE}$ 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 $\mathsf{poly}(2^n)$ 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 $\mathsf{NE}$ language must have $\mathcal{C}[m^{c_0}]$ computable witnesses. □

It is easy to get an analogue of Theorem 7.1 also for deterministic *lossy* compression algorithms.

**Remark 7.3.** If we could show that $\mathsf{ACC}^0$-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 $\mathsf{ACC}^0$, 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 7.2.

## 7.2 Other function classes that are hard to compress

**Large $\mathsf{AC}^0$ circuits.** Compressing functions computable by "large" $\mathsf{AC}^0$ circuits (of size $2^{n^\epsilon}$ with $\epsilon \gg 1/d$, where $d$ is the depth of the circuit) is difficult since every function computable by a polynomial-size $\mathsf{NC}^1$ circuit has an equivalent $\mathsf{AC}^0$ circuit of size $2^{n^\epsilon}$ (and some depth $d$ dependent on $\epsilon$). The existence of a compression algorithm for such large $\mathsf{AC}^0$ circuits would imply a *natural*

*property* in the sense of [RR97] useful against $\mathsf{NC}^1$. The latter implies that no strong enough PRG can be computed by $\mathsf{NC}^1$ circuits [RR97, AHM+08]. Also, using Theorem 7.1, we get that such compression would imply that $\mathsf{NEXP} \not\subset \mathsf{NC}^1$.

**Theorem 7.4.** *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 7.5.** *If there is an efficient algorithm that compresses a given truth table of an $m$-variate monotone Boolean function of monotone circuit size $\mathsf{poly}(m)$ 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 $\mathsf{poly}(m) = \mathsf{poly}(n)$. Hence, a circuit for $g$ of size at most $2^m/m^{1.51}$ yields a circuit for $f$ of size at most $O((2^n/n^{1.01}) + \mathsf{poly}(n))$, which is less than $2^n/n$ for large enough $n$. Appealing to Theorem 7.1 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 $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$.

# 8 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 $\mathsf{ACC}^0$ 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 recent 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 $\mathsf{AC}^0$-SAT algorithms, where there is a quantitative gap between the $\mathsf{AC}^0$ circuit size that can be handled by the randomized algorithm of [IMP12] and the deterministic algorithm of [BIS12].)

Finally, the focus of the present paper has been on lossless compression. For small $\mathsf{AC}^0$ circuits and small $\mathsf{AC}^0$ 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 $\mathsf{AC}^0$ circuits with parity-gates, we know by the results of Razborov and Smolensky [Raz87, Smo87] that every such function can be approximated by a $(\mathsf{poly}\log n)$-degree polynomial over $GF(2)$ to within error $1/n$. This polynomial is a binary Reed-Muller codeword of order $\mathsf{poly}\log n$ 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 $\mathsf{poly}\log n$) within the Hamming ball of relative radius $1/n$ around $x$, find in time $\mathsf{poly}(2^n)$ *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.)

**Acknowledgements** We thank Ran Raz for answering our questions on [KR12] and for telling us about [KRT13], and Dieter van Melkebeek for answering our questions on [KKM12]. We also thank Avi Wigderson for helpful discussions.

## References

[AB09] S. Arora and B. Barak. *Complexity theory: a modern approach.* Cambridge University Press, New York, 2009.

[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.

[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.

[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.

[AHM+08] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M.E. Saks. Minimizing disjunctive normal form formulas and $\mathrm{AC}^0$ circuits given a truth table. *SIAM Journal on Computing*, 38(1):63–84, 2008.

[And87] A.E. Andreev. On a method of obtaining more than quadratic effective lower bounds for the complexity of $\pi$-schemes. *Vestnik Moskovskogo Universiteta. Matematika*, 42(1):70–73, 1987. English translation in *Moscow University Mathematics Bulletin.*

[Ang87] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[Bea94] P. Beame. A switching lemma primer. Technical report, Department of Computer Science and Engineering, University of Washington, 1994.

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

[BIS12] P. Beame, R. Impagliazzo, and S. Srinivasan. Approximating $\mathsf{AC}^0$ by small height decision trees and a deterministic algorithm for $\#\mathsf{AC}^0\mathrm{SAT}$. In *Proceedings of the Twenty-Seventh Annual IEEE Conference on Computational Complexity*, pages 117–125, 2012.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[Bra10] M. Braverman. Polylogarithmic independence fools $AC^0$ circuits. *Journal of the Association for Computing Machinery*, 57:28:1–28:10, 2010.

[BS90] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of theoretical computer science (vol. A)*, pages 757–804. MIT Press, Cambridge, MA, USA, 1990.

[Chv79] V. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[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.

[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.

[DH09] E. Dantsin and E.A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. 2009.

[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.

[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.

[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.

[GS10] P. Gopalan and R. A. Servedio. Learning and lower bounds for $\mathsf{AC}^0$ 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, 2010.

[GS12] A. Gabizon and R. Shaltiel. Invertible zero-error dispersers and defective memory with stuck-at errors. In *APPROX-RANDOM*, pages 553–564, 2012.

[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.

[Hås98] J. Håstad. The shrinkage exponent of de morgan formulae is 2. *SIAM Journal on Computing*, 27:48–64, 1998.

[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.

[HS82] J. Heintz and C.-P. Schnorr. Testing polynomials which are easy to compute. *L'Enseignement Mathématique*, 30:237–254, 1982.

[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.

[IMP12] R. Impagliazzo, W. Matthews, and R. Paturi. A satisfiability algorithm for $\mathrm{AC}^0$. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 961–972, 2012.

[IMZ12] R. Impagliazzo, R. Meka, and D. Zuckerman. Pseudorandomness from shrinkage. In *Proceedings of the Fifty-Third Annual IEEE Symposium on Foundations of Computer Science*, 2012.

[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.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[Juk12] S. Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer, 2012.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55:40–56, 1982.

[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.

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

[KK13] V. Kabanets and A. Kolokolova. Compression of boolean functions. *Electronic Colloquium on Computational Complexity*, 20(24), 2013.

[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.

[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).

[KRT13] I. Komargodski, R. Raz, and A. Tal. Improved average-case lower bounds for DeMorgan formula size. manuscript, March 2013.

[Lev73] L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973.

[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.

[Lov75] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[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*.

[Mas79] W.J. Masek. Some NP-complete set covering problems. Manuscript, 1979.

[Nec66] E.I. Nechiporuk. On a Boolean function. *Doklady Akademii Nauk SSSR*, 169(4):765–766, 1966. English translation in *Soviet Mathematics Doklady*.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[Pip77] N. Pippenger. The complexity of monotone boolean functions. *Theory of Computing Systems*, 11:289–316, 1977.

[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.

[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.

[Raz93] A.A. Razborov. Bounded arithmetic and lower bounds in boolean complexity. In *Feasible Mathematics II*, pages 344–386. Birkhauser, 1993.

[Red79] N.P. Red'kin. On the realization of monotone boolean functions by contact circuits. *Problemy Kibernetiki*, 35:87–110, 1979. (in Russian).

[RR97] A.A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55:24–35, 1997.

[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.

[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.

[ST12] K. Seto and S. Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. In *Proceedings of the Twenty-Seventh Annual IEEE Conference on Computational Complexity*, pages 107–116, 2012.

[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.

[Sub61] B.A. Subbotovskaya. Realizations of linear function by formulas using $\vee$, &, $^-$. *Doklady Akademii Nauk SSSR*, 136(3):553–555, 1961. English translation in *Soviet Mathematics Doklady*.

[SZ96] P. Savický and S. Zák. A large lower bound for 1-branching programs. *Electronic Colloquium on Computational Complexity*, TR96-036, 1996.

[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.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Weg87] I. Wegener. *The Complexity of Boolean Functions*. J. Wiley, New York, 1987.

[Wil10] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the Forty-Second Annual ACM Symposium on Theory of Computing*, pages 231–240, 2010.

[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.

[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*.

[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.

[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.

[Zan98] F. Zane. *Circuits, CNFs, and Satisfiability*. PhD thesis, UCSD, 1998.

# A Proof of $(1-ax) \leqslant (1-x)^a$

**Lemma A.1.** *For $0 \leqslant x \leqslant 1$ and $a \geqslant 1$, it holds that $(1-ax) \leqslant (1-x)^a$.*

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

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

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

# B #SAT algorithm for linear-size general formulas

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

**Theorem B.1** ([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 $\psi$ 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 B.2** ([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: $c \leqslant 3/4$.*
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 $L(F_v) \leqslant 8c$ such that the parent node of $v$ is either $\wedge$ or $\vee$, and every variable in $F_v$ appears at least $c$ times in $F$.*

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.

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 subcases: (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 $b \leqslant 8c$ 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 $b + 1 \leqslant 8c$ 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 $1 \leqslant b \leqslant 8c$ 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 $l(F') \leqslant l(F) + \log(1 - b/n)$; and with probability $1/2$, $l(F') \leqslant l(F) + \left(1 + 1/8c^2\right)\log(1 - b/n)$.

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 $1 \leqslant b \leqslant 8c$ 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 B.3.** *It holds that*

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

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

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

*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 $0 \leqslant h < 8c$ from $G$ such that case 3, 4(a) or 4(b) happens and in consequence there are

$1 \leqslant b \leqslant 8c$ variables restricted. Over all descendants of this particular node at the bottom of the layer, it holds with probability at least $1/2$ that

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

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 B.4.** *Denote by $F_{n-k}$ the formula after restricting $n-k$ variables. For $k > 160c$,*

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

*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 B.3, we have $Z_i \leqslant c_i := -\frac{1}{16c^2}\log\left(1 - \frac{1}{n-16c(i-1)}\right)$. 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 $Z_i \leqslant -c_i$. 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 4.2, we have for any $\lambda \geqslant 0$,

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

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

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

Here we use the inequality that

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

Hence,

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

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

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

Therefore,

$$\mathbf{Pr}\left[L(F_{n-k}) \geqslant e^{\lambda} L(F)\left(\frac{k+16c-1}{n+16c-1}\right)^{1+\frac{1}{256c^3}}\right] \;\leqslant\; \exp\left(-\frac{\lambda^2}{2\cdot\frac{1}{16^3c^5}\cdot\frac{1}{k-1}}\right) = e^{-2048\lambda^2c^5(k-1)}.$$

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

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

□

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

*Proof of Theorem B.1.* 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 B.2. 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 B.4, at most $2^{-k}$ fraction of the branches end with formula size

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

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 $\delta = 2^{-O(c^3\log c)}$. □