Mining Circuit Lower Bound Proofs for Meta-algorithms

Mining Circuit Lower Bound Proofs for Meta-Algorithms

Ruiwen Chen*, Valentine Kabanets*, Antonina Kolokolova†, Ronen Shaltiel‡ and David Zuckerman§
*School of Computing Science, Simon Fraser University, Burnaby, BC, Canada; {ruiwenc, kabanets}@cs.sfu.ca
†Dept. of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada; kol@cs.mun.ca
‡Dept. of Computer Science, University of Haifa, Haifa, Israel; ronen@cs.haifa.ac.il
§Dept. of Computer Science, University of Texas, Austin, TX, USA; diz@cs.utexas.edu

***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 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 [52], [21], strengthened to the "high-probability" version by [48], [26], [33]. 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 [33] 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 [55] that any non-trivial Circuit-SAT algorithm for a circuit class $\mathcal{C}$ would imply a superpolynomial lower bound against $\mathcal{C}$ for a language in NEXP[1].

***Keywords*-average-case circuit lower bounds; Circuit-SAT algorithms; compression; meta-algorithms; natural property; random restrictions; shrinkage of de Morgan formulas**

I. 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 [8], [59], 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 [41]. 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) [22], [41].

In both 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 $\mathsf{BPP} = \mathsf{P}$ [27]). One would hope that looking inside the proofs (of the few circuit lower bounds that we actually have at present) may yield new algorithms (for the same computational model where we have the lower bounds).

This is indeed the case as witnessed by number of examples: a learning algorithm for AC^0-computable Boolean functions [36], a Circuit-SAT algorithm for AC^0 circuits [25], [6] (using Håstad's Switching Lemma, a main tool used in AC^0 lower bound proofs [20]), a simple pseudorandom generator for AC^0 circuits [9] (using [36]), a Circuit-SAT algorithm for linear-size (de Morgan) formulas [48], [50], and a pseudorandom generator for small (de Morgan) formulas and branching programs [26] (using a generalization of the "shrinkage under random restrictions" result of [52], [21]), to mention just a few.

Trying to understand the limitations of current circuit lower bound techniques, Razborov and Rudich [46] 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

[1]Independently, Williams [57] also proves such an implication.

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)[2]. There are two natural parameters to minimize: the *size* of the found circuit and the *running time* of the compression algorithm. Since the algorithm is given the full truth table as input, we consider it efficient if it runs in time $2^{O(n)}$ (polynomial in its input size). Ideally, we would like to find a circuit as small as the promised size of the concise representation of a given function f. However, any non-trivial savings over the generic $2^n/n$ circuit size [38] are interesting.[3]

Does every $\mathcal{C}$-circuit lower bound known today yield a compression algorithm for $\mathcal{C}$? The positive answer would strengthen the argument of [46] to show that every known lower bound proof yields a particular kind of natural property, *efficient compressibility*.

We hypothesize that the answer is 'Yes,' and make the first step in this direction by extracting a compression algorithm from the lower-bound proofs based on the method of *random restrictions*. These include the lower bounds for AC^0 circuits [18], [60], [20], for de Morgan formulas [52], [3], [21], for branching programs [40], and for read-once branching programs (see, e.g., [4]).

Compression Theorem: *(1) Boolean n-variate functions computed by 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.

A. 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 [52] 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 Γ (called the *shrinkage exponent*) is $3/2$; this Γ was subsequently improved to the optimal value 2 by Håstad [21].

This "shrinkage in expectation" result is sufficient for proving worst-case de Morgan formula lower bounds [3]. 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 [48] (for linear-size formulas); Impagliazzo et al. [26] prove a version of shrinkage with respect to pseudo-random restrictions (for de Morgan formulas of size almost n^3); Komargodski and Raz [33] 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 [48], [33], with a cleaner and simpler argument.

Shrinkage Lemma: *Let F be a (de Morgan) formula or general branching program of 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.

[2]This is different than $\mathcal{C}$-circuit minimization considered by [2] where the task is to construct a small circuit of the type $\mathcal{C}$.

[3]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.

Formula-#SAT: The fact that SAT is NP-complete [14], [35], 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 [15] for a recent survey), while the study of Circuit-SAT algorithms for more general classes of circuits is more recent: see [10], [25], [6] for AC^0-SAT, [48], [50] for Formula-SAT, and [56] for ACC^0-SAT. Usually such algorithms exploit the same structural properties of the corresponding circuit class that are used in the circuit lower bounds for that class. In fact, the observation that circuit lower bound proofs and meta-algorithms are intimately related was first formulated in Zane's PhD thesis [61] 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 [48], [50].

#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 [48], [50], 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 [33]: *There is a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ computable in P such that every de Morgan formula of size $n^{2.49}$ (any general formula of size $n^{1.99}$) computes $f(x)$ correctly on at most $1/2 + 2^{-n^{\sigma}}$ fraction of all n-bit inputs, for some constant $0 < \sigma < 1$.*

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 [32], [23], [41], [24], [31], [1], [17], [55]. In particular, the result by Williams [55] 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 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 super-polynomial lower bounds against the class $\mathcal{C}$ for a language in NEXP.*

Thus, both non-trivial SAT algorithms and non-trivial compression algorithms for a circuit class $\mathcal{C} \subseteq \mathsf{P/poly}$ imply superpolynomial lower bounds against that class. This suggests trying to get an alternative proof of Williams's lower bound $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$ [56] via designing a compression algorithm for ACC^0 functions. Apart from getting an alternative proof, the hope is that such a compression algorithm would give us more insight into the structure of ACC^0 functions, which could lead to ACC^0 circuit lower bounds against a much more explicit Boolean function, say the one in NP or in P.

B. Our proof techniques

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

For our compression algorithms, we use the "simplicity" of functions in the disjunction to argue that they have linear-size descriptions (as required in order to achieve $\mathsf{poly}(2^n)$ 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 [33]: 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 [52], 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 [33] with the following differences: (1) our restrictions always choose deterministically which variable to restrict (as opposed to restrictions of [33] 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 [33]). 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 [48].

Our proof of [33]'s average-case hardness result is more modular and simpler. In particular, we adapt Andreev's original lower bound argument [3] to the case of not necessarily truly random restrictions (by using randomness extractors), and use the information-theoretic framework of Kolmogorov complexity to avoid unnecessary technicalities.

Finally, our proof of circuits lower bounds for NEXP from a compression algorithm for a circuit class $\mathcal{C} \subseteq P/\mathsf{poly}$ is a generalization of the similar result from [24], showing that the existence of a natural property (even without the "largeness" assumption) for P/poly implies $\mathsf{NEXP} \not\subseteq \mathsf{P}/\mathsf{poly}$. Here we handle the case of any circuit class $\mathcal{C} \subseteq \mathsf{P}/\mathsf{poly}$. Since the existence on an efficient compression algorithm for a circuit class $\mathcal{C}$ implies a natural property for the same class, the required lower bound $\mathsf{NEXP} \not\subseteq \mathcal{C}$ follows.

Independently, Williams [57] also proves such a generalization of the result from [24] (as part of his equivalence between proving $\mathcal{C}$-circuit lower bounds against NEXP and having polynomial-time computable properties useful against $\mathcal{C}$).

Other related work: Perhaps the earliest example of a compression algorithm for a general class of Boolean functions is due to Yablonski [58], 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 [39], [30], [2], [16]. In particular, [2], [16] show that finding an approximately *minimal*-size DNF for a given truth table of an n-variate Boolean function is NP-hard, for the approximation factor n^γ for some constant $0 < \gamma < 1$.

Concurrent independent work: Raz, Komargodski, and Tal [34] improve the average-case de Morgan formula lower bounds of [33] to handle formulas of size about n^3. They also 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 [33]). Similarly to our paper but independently of our work, Komargodski et al. [34] also adapt Andreev's method to arbitrary (not necessarily completely random) restrictions by using appropriate randomness extractors.

The remainder of the paper: We prove our compression theorem in Section II, and the shrinkage result in Section III. We give our #SAT algorithms in Section IV. Average-case formula lower bounds are proved in Section V. We prove that compression implies circuit lower bounds in Section VI. We conclude with open questions in Section VII. Due to page limitations, some proofs are omitted in this conference version. For a full version of the paper, please see [11].

II. Compression from restriction-based circuit lower bounds

Here we prove the Compression Theorem.

A. Compression of DNFs via Set Cover

It is well-known that DNFs of almost minimum size can be computed from the truth table of $f\colon \{0,1\}^n \to \{0,1\}$ using a greedy Set Cover heuristic [28], [37], [13]. We recall this heuristic next.

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

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

For the analysis, observe that since ℓ subsets cover U, they also cover every subset of U. Hence, in each iteration of the algorithm, there exists a subset that covers at least $1/\ell$ fraction of the not-yet-covered points. After each iteration, the size of the set of points that are not covered reduces by the factor $(1 - 1/\ell)$. Thus, after t iterations, the number of points not yet covered is at most $|U| \cdot (1 - 1/\ell)^t \leqslant |U| \cdot e^{-t/\ell}$, 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 $f\colon \{0,1\}^n \to \{0,1\}$ be given by its truth table. Suppose that there exists a DNF computing f such that the DNF consists of ℓ terms (conjunctions). With each term a on n variables, we associate the set $S_a = a^{-1}(1)$ of points of $\{0,1\}^n$ where it evaluates to 1. We enumerate over all possible terms a on n variables, and keep only those sets S_a where $S_a \subseteq f^{-1}(1)$ (i.e., S_a does not cover any zero of f); note that all ℓ terms of the minimal DNF for f will be kept. Next we run the greedy set cover algorithm on the universe $U = f^{-1}(1)$ and the collection of sets S_a chosen above. By the analysis above, we get $O(\ell \cdot \log |U|)$ terms

such that their disjunction computes f. That is, we find a DNF for f of size at most $O(n)$ factor larger than that of the minimal DNF for f.

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

B. Compression of AC^0 functions via DNFs

The known lower bounds for AC^0 circuits are based on the fact that almost all random restrictions simplify a small AC^0 circuit to a function that depends on fewer than the remaining unrestricted variables. Intuitively, this means that there is a partitioning of the Boolean cube $\{0,1\}^n$ into not too many disjoint regions such that the original AC^0 circuit is constant over each region. This intuition can be made precise using the Switching Lemma [20], [45], [5], [25].

Lemma II.1 ([25])**.** *Every d-depth s-size Boolean circuit 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 Set-Cover algorithm, we get the following.

Theorem II.2. *There is a deterministic $\mathsf{poly}(2^n)$-time algorithm A satisfying the following. Let $f\colon \{0,1\}^n \to \{0,1\}$ be any Boolean function computable by an AC^0 circuit of depth d and size $s = s(n)$. Given the truth table of f, and 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 AC^0 circuits of size up to $2^{n^{1/(d-1)}}$, the size for which we know lower bounds against AC^0.

C. 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 [52] 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 [21]. However, these results are not strong enough to provide a kind of structure of easy functions that would be useful for compression. By analogy with the case of AC^0, we would like to say something like "for every small de Morgan formula, there is a partition of the Boolean cube into not too many regions such that the original formula is constant on each region". In particular, we need a "high probability" version of the classical shrinkage results of [52], [21].

Recently, there have been several such shrinkage results proved for different purposes. Santhanam [48] 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. [26] 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 [33] proved a shrinkage result for certain random restrictions (different from the ones in [48]), 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 [48], [33]. We use the same notion of random restrictions as in [48], 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 [33], 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 [33].

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 [48], 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 II.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 de Morgan formulas, and $\Gamma = 1$ for formulas over the complete basis and for branching programs.*

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

Corollary II.4. *Let $F(x_1, \ldots, x_n)$ be any formula (branching program) of size $O(n^d)$, where the constant d is such that $d < 2.5$ for de Morgan formulas, and $d < 2$ for formulas*

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

Proof: We consider the case of de Morgan formulas only; the case of formulas over the complete basis or branching programs can be argued analogously. Let $d = 2.5 - \nu$, for some constant $\nu > 0$. Set $\delta := \nu/3$, and $\gamma := 1 - \nu/2$. By Lemma II.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 $O(n^d/n^{1.5(1-\delta)}) = O(n^{1-\nu/2})$. ∎

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 $f\colon \{0,1\}^n \to \{0,1\}$ 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.

Theorem II.5. *There is a deterministic $\mathsf{poly}(2^n)$-time algorithm A satisfying the following. Let $f\colon \{0,1\}^n \to \{0,1\}$ 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 α of points in $\{0,1\}^n$, for some $0 \leqslant \alpha < 1$.*

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

Proof: Let $U = f^{-1}(1)$, and let $\beta = |U|/2^n$. If $\beta \leqslant 2\alpha$, 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[4] 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 $\mathcal{S}$.

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

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

Using this generalized algorithm, we get the following.

Theorem II.6. *There is an efficient compression algorithm that, given the truth table of a formula (branching program) F on n variables of size $L(F) \leqslant n^d$, produces an equivalent Boolean circuit of size at most 2^{n-n^ϵ}, for some constant $0 < \epsilon < 1$ (dependent on d), where $d < 2.5$ 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 II.4, this F can be computed by a decision tree of depth $m := n - n^\delta$ such that all but at most $\alpha := 2^{-n^\delta}$ fraction of the leaves correspond to restricted subformulas of F of size n^γ on $k := n^\delta$ variables, for some constants $0 < \delta, \gamma < 1$ dependent on d.

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

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^γ. Let us assume, without loss of generality, that all the first $\ell := 2^m - b$ formulas F_i are small. Define the circuits $C_i := (c_i \wedge F_i)$, for $1 \leqslant i \leqslant \ell$, 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^γ 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 II.5. Running the greedy algorithm of Theorem II.5, we get a circuit for F of total size at most $O(\ell n^2 + \alpha n 2^n) \leqslant \mathsf{poly}(n) \cdot 2^{n-n^\delta}$. ∎

D. 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 [49].

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.

Theorem II.7. *There is a deterministic* $\mathsf{poly}(2^n)$*-time algorithm A satisfying the following. Let $f\colon \{0,1\}^n \to \{0,1\}$ 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 $O(sn^3 \cdot 2^{n/2})$.*

III. Shrinkage of de Morgan Formulas

Here we prove the Shrinkage Lemma. We use the adaptive restrictions of [48] (each time randomly restricting the most frequent variable in the formula). Following [33], 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 [33], 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.

A. A variant of Azuma's Inequality

Lemma III.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 $t \geqslant 0$, $\mathbf{E}[e^{tY}] \leqslant e^{t^2c^2/2}$.*

Proof: Suppose Y takes two values a and b with equal probability, and $a \leqslant b \leqslant c$. If $b \leqslant 0$, then $e^{tY} \leqslant 1 \leqslant e^{t^2c^2/2}$. If $b > 0$, since $\mathbf{E}[Y] = \frac{1}{2}(a+b) \leqslant 0$, then $a \leqslant -b$ and $\mathbf{E}[e^{tY}] = \frac{1}{2}(e^{ta} + e^{tb}) \leqslant \frac{1}{2}(e^{-tb} + e^{tb}) \leqslant e^{t^2b^2/2} \leqslant e^{t^2c^2/2}$, by 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 III.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 λ, we have $\mathbf{Pr}[X_n - X_0 \geqslant \lambda] \leqslant \exp\left(-\frac{\lambda^2}{2\sum_{i=1}^n c_i^2}\right)$.*

The proof, omitted here, is by adapting the standard proof of Azuma's inequality to our "one-side bounded" variables.

B. Shrinkage lemma

A de Morgan formula can be simplified using the following *simplification rules*, which have been used in [21], [48]. We denote by ψ an arbitrary subformula, and y a literal. The rules are: (1) If $0 \wedge \psi$ or $1 \vee \psi$ appears, then replace it by 0 or 1, respectively. (2) If $0 \vee \psi$ or $1 \wedge \psi$ appears, then replace it by ψ. (3) If $y \vee \psi$ appears, then replace all occurrences of y in ψ by 0 and $\overline{y}$ by 1; if $y \wedge \psi$ appears, then replace all occurrences of y in ψ by 1 and $\overline{y}$ by 0. We say a de Morgan formula is *simplified* if none of the above rules are applicable. Note that in a simplified formula, by the rule 3, if a leaf is labeled with x or $\overline{x}$, then its sibling subtree does not contain the variable x.

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 simplified formula 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 III.3 (Shrinkage Lemma)**.** *Let F be any given de Morgan formula on n variables. For any $k \geqslant 4$, we have $\mathbf{Pr}\left[L(F_{n-k}) \geqslant 2 \cdot L(F) \cdot \left(\frac{k}{n}\right)^{3/2}\right] < 2^{-k}$.*

We need the following auxiliary lemma; its proof follows easily from the simplification rules (see also [48], [29]).

Lemma III.4. *$L(F_{i+1}) \leqslant L(F_i) \cdot (1 - 1/n)$, and $\mathbf{E}[L(F_{i+1})] \leqslant L(F_i) \cdot (1 - 1/n)^{3/2}$.*

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:

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

Conditioned on $R_1, \ldots, R_{i-1}$, the formula F_{i-1} is fixed, and Z_i assumes two values with equal probability. Using Lemma III.4 and Jensen's inequality, we get the following.

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

Now we can complete the proof of the Shrinkage Lemma.

Proof of Lemma III.3: Let λ be arbitrary, and let c_i's be as defined in Lemma III.5. By Lemma III.5 and Lemma III.2, we get $\mathbf{Pr}\left[\sum_{j=1}^i Z_j \geqslant \lambda\right] \leqslant \exp\left(-\frac{\lambda^2}{2\sum_{j=1}^i c_j^2}\right)$. The left-hand side, by the fact that $\sum_{j=1}^i Z_j = l_i - l_0 - \frac{3}{2}\log\frac{n-i}{n}$, is $\mathbf{Pr}\left[L_i \geqslant e^\lambda L_0 \left(\frac{n-i}{n}\right)^{3/2}\right]$.

For each $1 \leqslant j \leqslant i$, we have $c_j \leqslant 1/2(n-j)$, using the inequality $\log(1+x) \leqslant x$. Thus, $\sum_{j=1}^i c_j^2 \leqslant 1/4(n-i-1)$.

Taking $i = n - k$, we get $\mathbf{Pr}\left[L_{n-k} \geqslant e^\lambda L_0 \left(\frac{k}{n}\right)^{3/2}\right] \leqslant e^{-2\lambda^2(k-1)}$. Choosing $\lambda = \ln 2$ concludes the proof. ∎

IV. #SAT ALGORITHMS FOR FORMULAS

A. $n^{2.49}$-size de Morgan 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 IV.1. *There is a deterministic algorithm counting the number of satisfying assignments in a given formula on n variables of size $\leqslant n^d$ which runs in time $t(n) \leqslant 2^{n-n^\delta}$, for some constant $0 < \delta < 1$ (dependent on d), where $d < 2.5$ 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 Lemma III.3, all but 2^{-k} fraction of the leaves have the formula size $< 2L(F)\left(\frac{k}{n}\right)^{3/2} = 2 \cdot n^{2.5-\epsilon} \cdot n^{1.5(\alpha-1)} = 2n^{1-\epsilon+1.5\alpha}$.

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 $2^{n-k} \cdot \mathsf{poly}(n) + 2^{O(n^\gamma \log n)} \leqslant 2^{n-n^\alpha} \cdot \mathsf{poly}(n)$.

The overall runtime is at most 2^{n-n^δ} for some $\delta > 0$. ■

B. Linear-size formulas

We analyze Santhanam's $2^{n-\delta n}$-time satisfiability algorithm [48] for cn-size de Morgan formulas, using the "supermartingale approach", and get an explicit bound on δ.

Theorem IV.2 ([48]). *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 $\delta \geqslant 1/(32c^2)$.*

We also use the "supermartingale approach" to provide a different analysis of the #SAT algorithm for linear-size general formulas of [50].

Finally, we observe that the proof of Theorem IV.2 immediately yields an average-case lower bound for linear-size de Morgan formulas [48]. Indeed, by the proof of Theorem IV.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 $n^{2.5}$.

V. 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 [33] on average-case hardness for de Morgan formulas. Our proof is more modular than the original argument of [33], 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).

A. Andreev's original argument

We sketch the original idea of Andreev first. Andreev [3] 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 Γ).

The proof is by contradiction. Suppose we have a small de Morgan formula computing $A'(y)$. The argument relies on two observations. First, under a random restriction (with appropriate parameters), the restricted subformula of $A'(y)$ will have size considerably less than n. Secondly, a random restriction is likely to leave at least one variable free (unrestricted) in each of the blocks. When both these events happen, we get a small-size de Morgan formula that can be used to compute the bits of x_0, which contradicts the assumed hardness of x_0.

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

Andreev uses a very simple extractor for y (extracting one bit of randomness from each block in y), but this extractor works only for "sources of randomness" which have a "block structure", namely, every block contains at least one truly random bit. This dictates that the argument be constrained to use restrictions which in addition to leaving k unrestricted bits, also respect this "block structure" (at least with high probability). This is not an issue in Andreev's argument which uses random restrictions (that indeed respect the "block structure" with high probability). However, this creates difficulties if one wants to use other choices of restrictions as is the case in both [33] and the argument of this paper.

B. 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. 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.

C. Average-case hardness

We use the following extractor by Rao [43].

Theorem V.1 ([43])**.** *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.

Theorem V.2. *Let $r = n^\gamma$, for any given $0 < \gamma < 1$. There exists a binary code C mapping $(4n)$-bit message to a codeword of length 2^r, such that C is (ρ, L)-list decodable for $\rho = 1/2 - O(2^{-r/4})$ and $L \leqslant O(2^{r/2})$. Furthermore, there is a polynomial-time algorithm for computing $C(x)$ in position z, for any inputs $x \in \{0,1\}^{4n}$ and $z \in \{0,1\}^r$.*

Loosely speaking, as in [33], the code is used to perform "worst-case to average-case hardness amplification" in the spirit of [53]: 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 $f\colon \{0,1\}^{4n} \times \{0,1\}^n \to \{0,1\}$ be defined by $f(x,y) = C(x)_{E(y)}$, where C is the code from Theorem V.2 and E is Rao's extractor (from Theorem V.1) mapping n bits to $m = r = n^\gamma$ bits, for the min-entropy $k \geqslant 2m$. We prove the following.

Theorem V.3. *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, $\mathbf{Pr}_{y\in\{0,1\}^n}[F(y) = f'(y)] < \frac{1}{2} + \frac{1}{2^{n^\sigma}}$.*

By an averaging argument applied to Theorem V.3, we get the following corollary.

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

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

VI. Circuit lower bounds from compression

Since incompressibility of Boolean functions is a special case of a natural property in the sense of [46], the existence of compression algorithms for a circuit class $\mathcal{C}$ implies that there is no strong PRG in $\mathcal{C}$. Here we argue that such compression algorithms would also yield circuit lower bounds against $\mathcal{C}$ for a language in NEXP.

A. Arbitrary subclass of polynomial-size circuits

It was shown in [24] 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. 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 VI.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 $\mathsf{NEXP} \not\subseteq \mathcal{C}$.*

It is easy to get an analogue of Theorem VI.1 also for deterministic *lossy* compression algorithms.

Remark VI.2. If we could show that ACC^0-computable functions are compressible, we would get an alternative proof of Williams's lower bound $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$ [56]. Interestingly, while such a compression algorithm would yield a natural property for ACC^0, the overall lower bound proof would still use non-natural arguments and non-relativizing arguments that come from the use of [24].

B. Other function classes that are hard to compress

Large AC^0 circuits: Compressing functions computable by "large" AC^0 circuits (of size 2^{n^ϵ} with $\epsilon \gg 1/d$, where d is the depth of the circuit) is difficult since every function computable by a polynomial-size NC^1 circuit has an equivalent AC^0 circuit of size 2^{n^ϵ} (and some depth d dependent on ϵ). The existence of a compression algorithm for such large AC^0 circuits would imply a *natural property* in the sense of [46] useful against NC^1. The latter implies that no strong enough PRG can be computed by NC^1 circuits [46], [2]. Also, using Theorem VI.1, we get that such compression would imply that $\mathsf{NEXP} \not\subset \mathsf{NC}^1$.

Theorem VI.3. *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 $O(2^n/n^{1.5})$ [42], [47]. Using the well-known connection between non-monotone functions and monotone slice functions [7], we argue that compressing polynomial-size monotone functions is as hard as compressing arbitrary functions in $\mathsf{P/poly}$.

Theorem VI.4. *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 $2^n/n$.*

VII. Open questions

Can we extend our compressibility results 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 [44], [51]? Can we compress functions computable by ACC^0 circuits? More generally, do 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. Can we achieve better compression for the circuit classes considered?

Is there a general connection between compression and SAT algorithms?

Using the recent independent work by Komargodski et al. [34] 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 [34] is only *randomized* (due to the notion of random restrictions used in [34]). It is an interesting open question to get a *deterministic* such algorithm for n^3-size de Morgan formulas. (A similar problem is also open for AC^0-SAT algorithms, where there is a quantitative gap between the AC^0 circuit size that can be handled by the randomized algorithm of [25] and the deterministic algorithm of [6].) Very recently, [12] give a deterministic #SAT-algorithm for de Morgan formulas of size $n^{2.63}$, with the running time $2^{n-n^{\Omega(1)}}$.

For small AC^0 circuits and small AC^0 circuits with few threshold gates, one can get nontrivial *lossy* compression using the Fourier transform [36], [19]. What about lossy compression for other circuit classes?

For example, for polynomial-size AC^0 circuits with parity-gates, we know by the results of Razborov and Smolensky [44], [51] that every such function can be approximated by a $(\mathsf{poly}\log n)$-degree polynomial over $GF(2)$ to within error $1/n$. This 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 [54] for the case of binary Reed-Muller codes of order 2.)

Acknowledgment

We thank Ran Raz, Dieter van Melkebeek, and Avi Wigderson for helpful discussions. The first three authors were supported by NSERC; the fourth author by BSF grant 2010120, ISF grant 864/11, and ERC starting grant 279559; the fifth author by NSF Grants CCF-0916160 and CCF-1218723 and BSF Grant 2010120.

References

[1] M. Agrawal, "Proving lower bounds via pseudo-random generators," in *FSTTCS*, 2005, pp. 92–105.

[2] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. Saks, "Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table," *SICOMP*, vol. 38, no. 1, pp. 63–84, 2008.

[3] A. Andreev, "On a method of obtaining more than quadratic effective lower bounds for the complexity of π-schemes," *Vest. Moskov. Univer. Mat.*, vol. 42, no. 1, pp. 70–73, 1987.

[4] A. 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*, 1999, pp. 179–189.

[5] P. Beame, "A switching lemma primer," Dept. CSE, UW, Tech. Rep., 1994.

[6] P. Beame, R. Impagliazzo, and S. Srinivasan, "Approximating AC^0 by small height decision trees and a deterministic algorithm for $\#\mathsf{AC}^0\mathsf{SAT}$," in *CCC*, 2012, pp. 117–125.

[7] S. Berkowitz, "On some relationships between monotone and non-monotone circuit complexity," UofT, Tech. Rep., 1982.

[8] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SICOMP*, vol. 13, pp. 850–864, 1984.

[9] M. Braverman, "Polylogarithmic independence fools AC^0 circuits," *JACM*, vol. 57, pp. 28:1–28:10, 2010.

[10] C. Calabro, R. Impagliazzo, and R. Paturi, "The complexity of satisfiability of small depth circuits," in *IWPEC*, 2009, pp. 75–85.

[11] R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman, "Mining circuit lower bound proofs for meta-algorithms," *ECCC*, vol. 20, no. 57, 2013.

[12] R. Chen, V. Kabanets, and N. Saurabh, "An improved deterministic #SAT algorithm for small de Morgan formulas," *ECCC*, vol. 20, no. 150, 2013.

[13] V. Chvátal, "A greedy heuristic for the set covering problem," *Math. Oper. Res.*, vol. 4, pp. 233–235, 1979.

[14] S. Cook, "The complexity of theorem-proving procedures," in *STOC*, 1971, pp. 151–158.

[15] E. Dantsin and E. Hirsch, "Worst-case upper bounds," in *Handbook of Satisfiability*, 2009, pp. 403–424.

[16] V. Feldman, "Hardness of approximate two-level logic minimization and PAC learning with membership queries," *JCSS*, vol. 75, no. 1, pp. 13–26, 2009.

[17] L. Fortnow and A. Klivans, "Linear advice for randomized logarithmic space," in *STOC*, 2006, pp. 469–476.

[18] M. Furst, J. Saxe, and M. Sipser, "Parity, circuits, and the polynomial-time hierarchy," *Math. Systems Theory*, vol. 17, no. 1, pp. 13–27, Apr. 1984.

[19] P. Gopalan and R. A. Servedio, "Learning and lower bounds for AC^0 with threshold gates," in *APPROX-RANDOM*, 2010, pp. 588–601.

[20] J. Håstad, "Almost optimal lower bounds for small depth circuits," in *STOC*, 1986, pp. 6–20.

[21] ——, "The shrinkage exponent of de morgan formulae is 2," *SICOMP*, vol. 27, pp. 48–64, 1998.

[22] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SICOMP*, vol. 28, pp. 1364–1396, 1999.

[23] J. Heintz and C.-P. Schnorr, "Testing polynomials which are easy to compute," *L'Ens. Math.*, vol. 30, pp. 237–254, 1982.

[24] R. Impagliazzo, V. Kabanets, and A. Wigderson, "In search of an easy witness: Exponential time vs. probabilistic polynomial time," *JCSS*, vol. 65, no. 4, pp. 672–694, 2002.

[25] R. Impagliazzo, W. Matthews, and R. Paturi, "A satisfiability algorithm for AC^0," in *SODA*, 2012, pp. 961–972.

[26] R. Impagliazzo, R. Meka, and D. Zuckerman, "Pseudorandomness from shrinkage," in *FOCS*, 2012.

[27] R. Impagliazzo and A. Wigderson, "P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma," in *STOC*, 1997, pp. 220–229.

[28] D. Johnson, "Approximation algorithms for combinatorial problems," *JCSS*, vol. 9, pp. 256–278, 1974.

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

[30] V. Kabanets and J.-Y. Cai, "Circuit minimization problem," in *STOC*, 2000, pp. 73–79.

[31] V. Kabanets and R. Impagliazzo, "Derandomizing polynomial identity tests means proving circuit lower bounds," *Comp. Compl.*, vol. 13, no. 1–2, pp. 1–46, 2004.

[32] R. Kannan, "Circuit-size lower bounds and non-reducibility to sparse sets," *Info. Contr.*, vol. 55, pp. 40–56, 1982.

[33] I. Komargodski and R. Raz, "Average-case lower bounds for formula size," in *STOC*, 2013, pp. 171–180.

[34] I. Komargodski, R. Raz, and A. Tal, "Improved average-case lower bounds for DeMorgan formula size," *ECCC*, vol. 20, no. 58, 2013.

[35] L. Levin, "Universal sorting problems," *Problems of Information Transmission*, vol. 9, pp. 265–266, 1973.

[36] N. Linial, Y. Mansour, and N. Nisan, "Constant depth circuits, Fourier transform and learnability," *JACM*, vol. 40, no. 3, pp. 607–620, 1993.

[37] L. Lovász, "On the ratio of optimal integral and fractional covers," *Discrete Mathematics*, vol. 13, pp. 383–390, 1975.

[38] O. Lupanov, "On the synthesis of switching circuits," *Doklady Akademii Nauk SSSR*, vol. 119, no. 1, pp. 23–26, 1958.

[39] W. Masek, "Some NP-complete set covering problems," 1979, manuscript.

[40] E. Nechiporuk, "On a Boolean function," *Doklady Akademii Nauk SSSR*, vol. 169, no. 4, pp. 765–766, 1966.

[41] N. Nisan and A. Wigderson, "Hardness vs. randomness," *JCSS*, vol. 49, pp. 149–167, 1994.

[42] N. Pippenger, "The complexity of monotone boolean functions," *Theory of Computing Systems*, vol. 11, pp. 289–316, 1977.

[43] A. Rao, "Extractors for low-weight affine sources," in *CCC*, 2009, pp. 95–101.

[44] A. Razborov, "Lower bounds on the size of bounded depth circuits over a complete basis with logical addition," *Mathematical Notes*, vol. 41, pp. 333–338, 1987.

[45] ——, "Bounded arithmetic and lower bounds in boolean complexity," in *Feasible Mathematics II*. Birkhauser, 1993, pp. 344–386.

[46] A. Razborov and S. Rudich, "Natural proofs," *JCSS*, vol. 55, pp. 24–35, 1997.

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

[48] R. Santhanam, "Fighting perebor: New and improved algorithms for formula and QBF satisfiability," in *FOCS*, 2010, pp. 183–192.

[49] P. Savický and S. Zák, "A large lower bound for 1-branching programs," *ECCC*, vol. TR96-036, 1996.

[50] K. Seto and S. Tamaki, "A satisfiability algorithm and average-case hardness for formulas over the full binary basis," in *CCC*, 2012, pp. 107–116.

[51] R. Smolensky, "Algebraic methods in the theory of lower bounds for boolean circuit complexity," in *STOC*, 1987, pp. 77–82.

[52] B. Subbotovskaya, "Realizations of linear function by formulas using $\vee$, $\&$, $^-$," *DAN SSSR*, vol. 136, no. 3, pp. 553–555, 1961.

[53] M. Sudan, L. Trevisan, and S. Vadhan, "Pseudorandom generators without the XOR lemma," *JCSS*, vol. 62, no. 2, pp. 236–266, 2001.

[54] M. Tulsiani and J. Wolf, "Quadratic Goldreich-Levin theorems," in *FOCS*, 2011, pp. 619–628.

[55] R. Williams, "Improving exhaustive search implies superpolynomial lower bounds," in *STOC*, 2010, pp. 231–240.

[56] ——, "Non-uniform ACC circuit lower bounds," in *CCC*, 2011, pp. 115–125.

[57] ——, "Natural proofs versus derandomization," in *STOC*, 2013, pp. 21–30.

[58] S. Yablonski, "On the impossibility of eliminating perebor in solving some problems of circuit theory," *DAN SSSR*, vol. 124, no. 1, pp. 44–47, 1959.

[59] A. Yao, "Theory and applications of trapdoor functions," in *FOCS*, 1982, pp. 80–91.

[60] ——, "Separating the polynomial-time hierarchy by oracles," in *FOCS*, 1985, pp. 1–10.

[61] F. Zane, "Circuits, CNFs, and satisfiability," Ph.D. dissertation, UCSD, 1998.