Electronic Colloquium on Computational Complexity, Report No. 150 (2013)

# An Improved Deterministic #SAT Algorithm for Small De Morgan Formulas

Ruiwen Chen* Valentine Kabanets† Nitin Saurabh‡

November 4, 2013

**Abstract**

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

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

**Keywords:** de Morgan formulas, random restrictions, shrinkage, SAT algorithms.

## 1 Introduction

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

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

While the original motivation for the shrinkage results of [Sub61, IN93, PZ93, Hås98] was to prove formula lower bounds, the same results turn out to be useful also for designing nontrivial SAT algorithms for small de Morgan formulas. Santhanam [San10] strengthened Subbotovskaya's *expected* shrinkage result to *concentrated* shrinkage, i.e., shrinkage with high probability, and used this to get a deterministic #SAT algorithm (counting the number of satisfying assignments) for

*School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada; ruiwenc@sfu.ca
†School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada; kabanets@cs.sfu.ca
‡Institute of Mathematical Sciences, Chennai, India; nitin@imsc.res.in



ISSN 1433-8092

linear-size de Morgan formulas, with the running time $2^{n-\Omega(n)}$. Santhanam's algorithm deterministically selects a most frequent variable in the current formula, and recurses on the two subformulas obtained by restricting the chosen variable to 0 and 1; after $n - \Omega(n)$ recursive calls, almost all obtained formulas depend on fewer than the actual number of free variables remaining, which leads to nontrivial savings over the brute-force SAT algorithm for the original formula. A similar algorithm works also for formulas of size less than $n^{2.5}$, with the running time $2^{n-n^{\Omega(1)}}$ [CKK+13].

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

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

## 1.1 Our main results and techniques

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

**Theorem 1.1** (Main). *There is a deterministic algorithm for counting the number of satisfying assignments in a given de Morgan formula on $n$ variables of size at most $n^{2.63}$ which runs in time at most $2^{n-n^{\delta}}$, for some constant $0 < \delta < 1$.*

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

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

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

Namely, once we have this constructive shrinkage result, based on restricting one variable at a time, we apply the martingale-based analysis of [KR13, CKK$^+$13] to derive a *concentrated* version of constructive shrinkage, showing that almost all random settings of the selected variables yield restricted formulas of reduced size, where the restricted formulas are simplified by our efficient procedure. The shrinkage exponent $\Gamma = (5 - \sqrt{3})/2 \approx 1.63$ is the same as in [PZ93]. Using [CKK$^+$13], we then get a deterministic #SAT algorithm, running in time $2^{n-n^{\Omega(1)}}$, that works for de Morgan formulas of size up to $n^{\Gamma+1-o(1)} \approx n^{2.63}$.

### 1.2 Related work

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

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

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

**Remainder of the paper.** We give basic definitions in Section 2. Section 3 contains our efficient formula-simplification procedures. We use these procedures in Section 4 to prove a constructive and concentrated shrinkage result for de Morgan formulas. This is then used in Section 5 to describe and analyze our #SAT algorithm from Theorem 1.1. Section 6 contains some open questions.

## 2 Preliminaries

A *(de Morgan) formula* is a binary tree where each leaf is labeled by a literal (a variable $x$ or its negation $\overline{x}$) or a constant (0 or 1), and each internal node is labeled by $\wedge$ or $\vee$. A formula naturally computes a boolean function on its input variables.

Let $F$ be a formula with no constant leaves. We define the *size* of $F$, denoted by $L(F)$, as number of leaves in $F$. Following [PZ93], we define a *twig* to be a subtree with exactly two leaves. Let $T(F)$ be the number of twigs in $F$. We define the *weight* of $F$ as $w(F) = L(F) + \alpha \cdot T(F)$,

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

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

where $\alpha = \sqrt{3} - 1 \approx 0.732$. For convenience, if $F$ is a constant, we define $L(F) = w(F) = 0$. We say $F$ is *trivial* if it is a constant or a literal. Note that we define the size and weight only for formulas which are either constants or with no constant leaves; this is without loss of generality since constants can always be eliminated using a simplification procedure below.

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

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

# 3 Formula simplification procedures

## 3.1 Basic simplification

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

**Simplify**($F$):

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

1. **Constant elimination.**
   (a) If a subformula is of the form $0 \wedge G$, replace it by 0.
   (b) If a subformula is of the form $1 \vee G$, replace it by 1.
   (c) If a subformula is of the form $1 \wedge G$ or $0 \vee G$, replace it by $G$.
2. **One-variable simplification.**
   (a) If a subformula is of the form $y \vee G$ and $y$ or $\overline{y}$ appears in $G$, replace the subformula by $y \vee G|_{y=0}$.
   (b) If a subformula is of the form $y \wedge G$ and $y$ or $\overline{y}$ appears in $G$, replace the subformula by $y \wedge G|_{y=1}$.
   (c) If a subformula $G$ is of the form $G_1 \vee G_2$ for non-trivial $G_1$ and $G_2$, and $G$ $\vee$-depends on a literal $y$, then replace $G$ by $y \vee G|_{y=0}$.
   (d) If a subformula $G$ is of the form $G_1 \wedge G_2$ for non-trivial $G_1$ and $G_2$, and $G$ $\wedge$-depends on a literal $y$, then replace $G$ by $y \wedge G|_{y=1}$.

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

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

The procedure **Simplify** reduces the formula size and does not increase the number of twigs.

**Lemma 3.1.** *Let $F$ be a formula with no constant leaves. Suppose we substitute $k$ leaves of $F$ by constants, and run* **Simplify** *which produces a new formula $F'$. Then $L(F') \leqslant L(F) - k$ and $T(F') \leqslant T(F)$.*

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

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

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

**Lemma 3.2.** **Simplify** *runs in polynomial time.*

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

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

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

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

### 3.2 Simplification under all one-variable restrictions

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

The idea behind the transformations in **RestrictSimplify** is the following. When a formula simplifies to a literal under some one-variable restriction, then the formula must be logically equivalent to some special form. For example, if we know that $F|_{x=1}$ simplifies to a literal $y$, then $F$ itself must be logically equivalent to $(x \wedge y) \vee (\overline{x} \wedge G)$ for some $G$. This logically equivalent form may help to simplify $F$ under other one-variable restrictions.

**RestrictSimplify**$(F)$:

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

If $F$ is $G \vee H$ or $G \wedge H$, recursively call **RestrictSimplify** to compute $\{G_y\}$ and $\{H_y\}$, and initialize each $F_y :=$ **Simplify**$(G_y \vee H_y)$ or $F_y :=$ **Simplify**$(G_y \wedge H_y)$, respectively. Then apply the following transformations whenever possible.

Suppose there are two literals $x$ and $y$ over distinct variables such that $F_x = y$.

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

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

**Remark 3.3.** It is possible to introduce more simplifications rules in **RestrictSimplify**, e.g., when $F_x$ is a constant for some literal $x$, or when, in case 5, $x$ or $\overline{x}$ appears in $F_z$[3]. However, such simplifications are not needed for our proof of constructive shrinkage.

Next we argue the efficiency of **RestrictSimplify**.

**Lemma 3.4. RestrictSimplify** *runs in polynomial time.*

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

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

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

**Claim 3.5.** *For $F = G \vee H$ or $F = G \wedge H$, we have $w(F_y) \leqslant w(G_y) + w(H_y)$, for all literals $y$ except those where $G_y$ and $H_y$ are literals over distinct variables.*

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

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

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

**Lemma 3.6.** *The solo structure of a non-trivial formula $F$ must be in one of the following forms:*

*(i) the empty relation,*

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

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

*(iv) $\{x \Rightarrow y, \ y \Rightarrow x, \ \overline{x} \Rightarrow \overline{y}, \ \overline{y} \Rightarrow \overline{x}\}$,*

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

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

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

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

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

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

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

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

# 4 Constructive and concentrated shrinkage

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

### 4.1 Average savings under one-variable restrictions

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

**Theorem 4.1.** *For any formula $F$, it holds that*

$$\frac{\sigma(F)}{w(F)} \geqslant 2\gamma,$$

*where $\gamma = (5 - \sqrt{3})/2 \approx 1.63$.*

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

For the base case, we analyze all possible formulas of size at most 4.

**Lemma 4.2.** *For any simplified formula $F$ of size at most 4, it holds that $\sigma(F)/w(F) \geqslant 2\gamma$.*

*Proof.* Table 1 lists all simplified formulas (or their duals) of size at most 4, together with the savings. The cases labeled by * were not consider in [PZ93] since they are not the smallest logically equivalent formulas.

Table 1: Savings for all formulas with $2 \leqslant L(F) \leqslant 4$

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

□

For formulas of size larger than 4, we consider whether one child of the root is trivial. Without loss of generality, we assume the root is labeled by $\vee$; the other case is dual.

**Lemma 4.3.** *If $F$ is a simplified formula of the form $x \vee G$ for some literal $x$ and subformula $G$, and $L(F) \geqslant 5$, then $\sigma(F)/w(F) \geqslant 2\gamma$.*

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

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

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

If $k \geqslant 5$, then

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

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

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

**Lemma 4.4.** *Suppose $F$ is of the form $G \vee H$ with $L(F) \geqslant 5$ and $G, H$ are non-trivial. Then $\sigma(F)/w(F) \geqslant 2\gamma$.*

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

*Proof.* We first prove that, for a literal $x$, if $G_x$ and $H_x$ are not literals over distinct variables, then $\sigma_x(F) \geqslant \sigma_x(G) + \sigma_x(H)$. Since $w(F) = w(G) + w(H)$, the claim follows from $w(F_x) \leqslant w(G_x) + w(H_x)$, which holds by Claim 3.5.

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

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

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

Since $G_x = y$, transformation 3(a)–(b) in **RestrictSimplify** guarantee that either $G_y$ is constant 1 or it $\vee$-depends on $x$. Similarly either $H_z$ is constant 1 or it $\vee$-depends on $x$. Since

$H_y|_{x=1} \equiv H_x|_{y=1} = z$, we get that $H_y$ is not a constant (it depends on $z$), and if it is a literal it must be $z$. Similarly $G_z$ is not a constant (it depends on $y$), and if it is a literal it must be $y$.

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

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

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

**Case $k \geqslant 3$:** By Lemma 3.6, the solo structure of $G$ and $H$ must be one of cases (ii), (iii), or (iv).

First assume that either $G$ or $H$ is in case (ii) of Lemma 3.6. Without loss of generality, suppose $G$ is in case (ii); then $G$ is logically equivalent to a literal $y$ but itself is non-trivial, which implies that $w(G) \geqslant 4 + \alpha$. (The smallest non-trivial, simplified formula equivalent to a literal has size at least 4). We have that $w(G_z) = 1$ for at least $k$ literals $z \notin \{y, \overline{y}\}$, and $w(G_y) = w(G_{\overline{y}}) = 0$. Then by the fact that $w(F) = w(G) + w(H)$ and the induction hypothesis on $H$, we have

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

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

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

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

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

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

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

*Proof of Theorem 4.1.* The proof is by combining the base case in Lemma 4.2 and the two inductive cases in Lemma 4.3 and Lemma 4.4. □

## 4.2 Concentrated shrinkage

Theorem 4.1 characterizes the average shrinkage of the weight of a formula when a randomly chosen literal is restricted. Given a formula $F$ on $n$ variables, if we randomly pick one variable and randomly assign it 0 or 1, the weight of the restricted formula (produced by **RestrictSimplify**) reduces by at least $\gamma \cdot w(F)/n$ on average.

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

**Lemma 4.5.** *Let $F$ be a formula on $n$ variables. Let $x$ be the variable such that $\sigma_x(F) + \sigma_{\overline{x}}(F)$ is maximized. Let $F'$ be $F_x$ or $F_{\overline{x}}$ with equal probability. Then we have $w(F') \leqslant w(F) - 1$ and*

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

*Proof.* Restricting a variable eliminates at least one leaf; therefore $w(F') \leqslant w(F) - 1$.

By Theorem 4.1, $n(\sigma_x(F) + \sigma_{\overline{x}}(F)) \geqslant \sigma(F) \geqslant 2\gamma \cdot w(F)$. Then we have

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

as required. □

Next we use the martingale-based analysis from [KR13, CKK+13] to derive a "high-probability shrinkage" result from Lemma 4.5. 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 $1 \leqslant i \leqslant n$. We need the following version of Azuma's inequality.

**Lemma 4.6** ([CKK+13])**.** *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*

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

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

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

We have the following.

**Lemma 4.7.** *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 $Z_i \leqslant c_i := -\gamma \log\left(1 - \frac{1}{n-i+1}\right)$.*

*Proof.* Since $w_i \leqslant w_{i-1}$, we have $Z_i \leqslant c_i$. By Jensen's inequality and Lemma 4.5, we get

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

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

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

**Lemma 4.8** (Concentrated weight shrinkage)**.** *Let $F$ be any given de Morgan formula on $n$ variables. For any $k > 10$, we have*

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

*Proof.* Let $\lambda$ be arbitrary, and let $c_i$'s be as defined in Lemma 4.7. By Lemmas 4.7 and 4.6, 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 $\sum_{j=1}^{i} Z_j = w_i - w_0 - \gamma \log \frac{n-i}{n}$. Hence,

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

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

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

Taking $i = n - k$, we get

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

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

Finally, by $w(F)/(1 + \alpha/2) \leqslant L(F) \leqslant w(F)$ for all $F$, we get from Lemma 4.8 the desired concentrated constructive shrinkage with respect to the restriction process defined above.

**Corollary 4.9** (Concentrated constructive shrinkage)**.** *Let $F$ be an arbitrary de Morgan formula. There exist constants $c, d > 1$ such that, for any $k > 10$,*

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

## 5 #SAT Algorithm for $n^{2.63}$-size de Morgan Formulas

Here we prove our main result.

**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 $n^{2.63}$ which runs in time $t(n) \leqslant 2^{n-n^{\delta}}$, for some constant $0 < \delta < 1$.*

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

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

Note that constructing this decision tree takes time $2^{n-k}\mathsf{poly}(n)$, since the procedure **Restrict-Simplify** runs in polynomial time. By Corollary 4.9, all but at most $2^{-k/d}$ fraction of the leaves have the formula size $L(F_{n-k}) < c \cdot L(F)\left(\frac{k}{n}\right)^{\gamma} = cn^{1-\epsilon+\gamma\alpha}$.

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

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

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

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

## 6 Open questions

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

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

## References


[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*.

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

[CKK+13] R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Electronic Colloquium on Computational Complexity*, 20(57), 2013.

[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.

[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*, pages 111–119, 2012.

[IN93] R. Impagliazzo and N. Nisan. The effect of random restrictions on formula size. *Random Structures and Algorithms*, 4(2):121–134, 1993.

[KR13] I. Komargodski and R. Raz. Average-case lower bounds for formula size. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, pages 171–180, 2013.

[KRT13] I. Komargodski, R. Raz, and A. Tal. Improved average-case lower bounds for DeMorgan formula size. *Electronic Colloquium on Computational Complexity*, 20(58), 2013.

[PPZ99] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 1999.

[PZ93] M. Paterson and U. Zwick. Shrinkage of de Morgan formulae under restriction. *Random Structures and Algorithms*, 4(2):135–150, 1993.

[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.

[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.

[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*.

[TX13] L. Trevisan and T. Xue. A derandomized switching lemma and an improved derandomization of $\mathrm{AC}^0$. In *Proceedings of the Twenty-Eighth Annual IEEE Conference on Computational Complexity*, pages 242–247, 2013.

[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.



ECCC ISSN 1433-8092

http://eccc.hpi-web.de