A #SAT Algorithm for Small Constant-Depth Circuits with PTF Gates

Swapnam Bajpai
Indian Institute of Technology, Bombay, Mumbai, India
swapnam@cse.iitb.ac.in

Vaibhav Krishan
Indian Institute of Technology, Bombay, Mumbai, India
vaibhkrishan@iitb.ac.in

Deepanshu Kush
Indian Institute of Technology, Bombay, Mumbai, India
kush.deepanshu@gmail.com

Nutan Limaye[1]
Indian Institute of Technology, Bombay, Mumbai, India
nutan@cse.iitb.ac.in

Srikanth Srinivasan[2]
Department of Mathematics, Indian Institute of Technology Bombay, Mumbai, India
srikanth@math.iitb.ac.in

Abstract

We show that there is a zero-error randomized algorithm that, when given a small constant-depth Boolean circuit C made up of gates that compute constant-degree Polynomial Threshold functions or PTFs (i.e., Boolean functions that compute signs of constant-degree polynomials), counts the number of satisfying assignments to C in significantly better than brute-force time.

Formally, for any constants d, k, there is an $\varepsilon > 0$ such that the zero-error randomized algorithm counts the number of satisfying assignments to a given depth-d circuit C made up of k-PTF gates such that C has size at most $n^{1+\varepsilon}$. The algorithm runs in time $2^{n-n^{\Omega(\varepsilon)}}$.

Before our result, no algorithm for beating brute-force search was known for counting the number of satisfying assignments even for a single degree-k PTF (which is a depth-1 circuit of linear size).

The main new tool is the use of a learning algorithm for learning degree-1 PTFs (or Linear Threshold Functions) using comparison queries due to Kane, Lovett, Moran and Zhang (FOCS 2017). We show that their ideas fit nicely into a memoization approach that yields the #SAT algorithms.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases SAT, Polynomial Threshold Functions, Constant-depth Boolean Circuits, Linear Decision Trees, Zero-error randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.8

Acknowledgements We would like to thank Valentine Kabanets for telling us about this problem and Paul Beame, Daniel Kane, Valentine Kabanets, Ryan O'Donnell, Rahul Santhanam, Madhu Sudan and Ryan Williams for helpful discussions. We would also like to thank the anonymous referees of ITCS 2019 for their helpful comments and suggestions.

[1] Funded by Mathematical Research Impact Centric Support (MATRICS), SERB.
[2] Funded by Mathematical Research Impact Centric Support (MATRICS), SERB. Work partially done while visiting the Simons Institute for the Theory of Computing, Berkeley.

10th Innovations in Theoretical Computer Science (ITCS 2019).
Editor: Avrim Blum; Article No. 8; pp. 8:1–8:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

This paper adds to the growing line of work on *circuit-analysis algorithms*, where we are given as input a Boolean circuit C from a fixed class $\mathcal{C}$ computing a function $f : \{-1,1\}^n \to \{-1,1\}$,[3] and we are required to compute some parameter of the function f. A typical example of this is the question of satisfiability, i.e. whether f is the constant function 1 or not. In this paper, we are interested in computing $\#\mathrm{SAT}(f)$, which is the number of satisfying assignments of f (i.e. $|\{a \in \{-1,1\}^n \mid f(a) = -1\}|$).

Problems of this form can always be solved by "brute-force" in time $\mathrm{poly}(|C|) \cdot 2^n$ by trying all assignments to C. The question is can this brute-force algorithm be significantly improved, say to time $2^n/n^{\omega(1)}$ when C is small, say $|C| \leq n^{O(1)}$.

Such algorithms, intuitively are able to distinguish a small circuit $C \in \mathcal{C}$ from a "black-box" and hence find some structure in C. This structure, in turn, is useful in answering other questions about $\mathcal{C}$, such as proving lower bounds against the class $\mathcal{C}$.[4] There has been a large body of work in this area, a small sample of which can be found in [21, 20, 26, 27]. A striking result of this type was proved by Williams [26] who showed that for many circuit classes $\mathcal{C}$, even *co-non-deterministic* satisfiability algorithms running in better than brute-force time yield lower bounds against $\mathcal{C}$.

Recently, researchers have also uncovered tight connections between many combinatorial problems and circuit-analysis algorithms, showing that even modest improvements over brute-force search can be used to improve long-standing bounds for these combinatorial problems (see, e.g., [30, 2, 3, 1]). This yields further impetus in improving known circuit-analysis algorithms.

This paper is concerned with #SAT algorithms for constant depth threshold circuits, denoted as TC^0, which are Boolean circuits where each gate computes a linear threshold function (LTF); an LTF computes a Boolean function which accepts or rejects based on the sign of a (real-valued) linear polynomial evaluated on its input. Such circuits are surprisingly powerful: for example, they can perform all integer arithmetic efficiently [4, 9], and are at the frontier of our current lower bound techniques [16, 5].

It is natural, therefore, to try to come up with circuit-analysis algorithms for threshold circuits. Indeed, there has a large body of work in the area (reviewed below), but some extremely simple questions remain open.

An example of such a question is the existence of a better-than-brute-force algorithm for satisfiability of degree-k PTFs where k is a constant greater than 1. Informally, the question is the following: we are given a degree-k polynomial $Q(x_1, \ldots, x_n)$ in n Boolean variables and we ask if there is any Boolean assignment $a \in \{-1,1\}^n$ to $x_1, \ldots, x_n$ such that $Q(a) < 0$. (Note that for a linear polynomial (i.e. $k = 1$), this problem is trivial.)

Surprisingly, no algorithm is known for this problem that is significantly better than 2^n time.[5] In this paper, we solve the stronger counting variant of this problem for any constant-degree PTFs. We start with some definitions and then describe this result.

▶ **Definition 1** (Polynomial Threshold Functions). A *Polynomial Threshold Function (PTF) on n variables of degree-k* is a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ such that there is

[3] We work with the $\{-1,1\}$ basis for Boolean functions, which is by now standard in the literature. (See for instance [18].) Here -1 stands for True and 1 stands for False.

[4] This intuition was provided to us by Ryan Williams.

[5] An algorithm was claimed for this problem in the work of Sakai, Seto, Tamaki and Teruyama [22]. Unfortunately, the proof of this claim only works when the weights are suitably small. See Footnote 1 on page 4 of [14].

a degree-k multilinear polynomial $P(x_1, \ldots, x_n) \in \mathbb{R}[x_1, \ldots, x_n]$ that, for all $a \in \{-1, 1\}^n$, satisfies $f(a) = \text{sgn}(P(a))$. (We assume that $P(a) \neq 0$ for any $a \in \{-1, 1\}^n$.)

In such a scenario, we call f a *k-PTF*. In the special case that $k = 1$, we call f a *Linear Threshold function (LTF)*. We also say that the polynomial P *sign-represents* f.

When $P \in \mathbb{Z}[x_1, \ldots, x_n]$, we define the weight of P, denoted $w(P)$, to be the *bit-complexity* of the sum of the absolute values of all the coefficients of P. In particular, the coefficients of P are integers in the range $[-2^{w(P)}, 2^{w(P)}]$.

We now formally define the #SAT problem for k-PTFs. Throughout, we assume that k is a constant and not a part of the input.

▶ **Definition 2** (#SAT problem for k-PTFs)**.** The problem is defined as follows.

Input: A k-PTF f, specified by a degree-k polynomial $P(x_1, \ldots, x_n)$ with integer coefficients.[6]

Output: The number of satisfying assignments to f. That is, the number of $a \in \{-1, 1\}^n$ such that $P(a) < 0$.

We use #SAT(f) to denote this output. We say that the input instance has parameters (n, M) if n is the number of input variables and $w(P) \leq M$.

▶ Remark. An interesting setting of M is poly(n) since any k-PTF can be represented by an integer polynomial with coefficients of bit-complexity at most $\tilde{O}(n^k)$ [17]. However, note that our algorithms are even when M is $\exp(n^{o(1)})$, i.e. when the weights are slightly short of doubly exponential in n.

We give a better-than-brute-force algorithm for #SAT(k-PTF). Formally we prove the following theorem.

▶ **Theorem 3.** *Fix any constant k. There is a zero-error randomized algorithm that solves the #SAT problem for k-PTFs in time* $\text{poly}(n, M) \cdot 2^{n-S}$ *where $S = \tilde{\Omega}(n^{1/(k+1)})$ and (n, M) are the parameters of the input k-PTF f. (The $\tilde{\Omega}(\cdot)$ hides factors that are inverse polylogarithmic in n.)*

▶ Remark. An anonymous ITCS 2019 referee pointed out to us that from two results of Williams [25, 28], it follows that satisfiability for 2-PTFs can be solved in $2^{n-\Omega(\sqrt{n})}$ time. Note that this is better than the runtime of our algorithm. However, the method does not extend to $k \geq 3$.

We then extend this result to a powerful model of circuits called *k-PTF circuits*, where each gate computes a k-PTF. This model was first studied by Kane, Kabanets and Lu [13] who proved strong average case lower bounds for slightly superlinear-size constant-depth k-PTF circuits. Using these ideas, Kabanets and Lu [14] were able to give a #SAT algorithm for a restricted class of k-PTF circuits, where each gate computes a PTF with a subquadratically many, say $n^{1.99}$, monomials (while the size remains the same, i.e. slightly superlinear).[7] A reason for this restriction on the PTFs was that they did not have an algorithm to handle even a single degree-2 PTF (which can have $\Omega(n^2)$ many monomials).

Building on our #SAT algorithm for k-PTFs and the ideas of [14], we are able to handle general k-PTF circuits of slightly superlinear size. We state these results formally below.

We first define k-PTF circuits formally.

[6] It is known [17] that such a representation always exists.

[7] Their result also works for the slightly larger class of PTFs that are subquadratically sparse in the $\{0, 1\}$-basis with *no restriction* on degree. Our result can also be stated for the larger class of *polynomially sparse* PTFs, but for the sake of simplicity, we stick to constant-degree PTFs.

▶ **Definition 4** (k-PTF circuits)**.** A *k-PTF circuit* on n variables is a Boolean circuit on n variables where each gate g of fan-in m computes a fixed k-PTF of its m inputs. The size of the circuit is the *number of wires* in the circuit, and the depth of the circuit is the longest path from an input to the output gate.[8]

The problems we consider is the #SAT problem for k-PTF circuits, defined as follows.

▶ **Definition 5** (#SAT problem for k-PTF circuits)**.** The problem is defined as follows.
Input: A k-PTF circuit C, where each gate g is labelled by an integer polynomial that sign-represents the function that is computed by g.
Output: The number of satisfying assignments to C.
We use $\#\mathrm{SAT}(C)$ to denote this output. We say that the input instance has parameters (n, s, d, M) where n is the number of input variables, s is the size of C, d is the depth of C and M is the maximum over the weights of the degree-k polynomials specifying the k-PTFs in C. We will say that M is the weight of C, denoted by $w(C)$.

We now state our result on #SAT for k-PTF circuits. The following result implies Theorem 3, but we prove them separately.

▶ **Theorem 6.** *Fix any constants k, d. Then the following holds for some constant $\varepsilon_{k,d} > 0$ depending on k, d. There is a zero-error randomized algorithm that solves the #SAT problem for k-PTF circuits of size at most $s = n^{1+\varepsilon_{k,d}}$ with probability at least $1/4$ and outputs ? otherwise. The algorithm runs in time $\mathrm{poly}(n, M) \cdot 2^{n-S}$, where $S = n^{\varepsilon_{k,d}}$ and (n, s, d, M) are the parameters of the input k-PTF circuit.*

Previous work. Satisfiability algorithms for TC^0 have been widely investigated. Impagliazzo, Lovett, Paturi and Schneider [12, 10] give algorithms for checking satisfiability of depth-2 threshold circuits with $O(n)$ gates. An incomparable result was proved by Williams [29] who obtained algorithms for subexponential-sized circuits from the class $\mathrm{ACC}^0 \circ \mathrm{LTF}$, which is a subclass of subexponential TC^0.[9] For the special case of k-PTFs (and generalizations to sparse PTFs over the $\{0, 1\}$ basis) with *small weights*, a #SAT algorithm was devised by Sakai et al. [22].[10] The high-level idea of our algorithm is the same as theirs.

For general constant-depth threshold circuits, the first satisfiability algorithm was given by Chen, Santhanam and Srinivasan [7]. In their paper, Chen et al. gave the first average case lower bound for TC^0 circuits of slightly super linear size $n^{1+\varepsilon_d}$, where ε_d depends on the depth of the circuit. (These are roughly the strongest size lower bounds we know for general TC^0 circuits even in the worst case [11].) Using their ideas, they gave the first (zero-error randomized) improvement to brute-force-search for satisfiability algorithms (and indeed even #SAT algorithms) for constant depth TC^0 circuits of size at most $n^{1+\varepsilon_d}$.

The lower bound results of [7] were extended to the much more powerful class of k-PTF circuits (of roughly the same size as [7]) by Kane, Kabanets and Lu [13]. In a follow-up paper, Kabanets and Lu [14] considered the satisfiability question for k-PTF circuits, and

[8] Note, crucially, that only the fan-in of a gate counts towards its size. So any gate computing a k-PTF on m variables only adds m to the size of the circuit, though of course the polynomial representing this PTF may have $\approx m^k$ monomials.

[9] $\mathrm{ACC}^0 \circ \mathrm{LTF}$ is a subclass of TC^0 where general threshold gates are allowed only just above the variables. All computations above these gates are one of AND, OR or Modular gates (that count the number of inputs modulo a constant). It is suspected (but not proved) that subexponential-sized ACC^0 circuits cannot simulate even a single general threshold gate. Hence, it is not clear if the class of subexponential-sized $\mathrm{ACC}^0 \circ \mathrm{LTF}$ circuits contains even depth-2 TC^0 circuits of linear size.

[10] More specifically, the algorithm of Sakai et al. [22] works as long as the weight of the input polynomial $P \in \mathbb{Z}[x_1, \ldots, x_n]$ is bounded by $\exp(n^{1-\Omega(1)})$ (or equivalently, $M \leq O(n^{1-\Omega(1)})$).

could resolve this question in the special case that each PTF is subquadratically sparse, i.e. has $n^{2-\Omega(1)}$ monomials. One of the reasons for this sparsity restriction is that their strategy does not seem to yield a SAT algorithm for a single degree-2 PTF (which is a depth-1 2-PTF circuit of *linear* size).

1.1 Proof outline

For simplicity we discuss SAT algorithms instead of #SAT algorithms.

Satisfiability algorithm for k-PTFs

At a high level, we follow the same strategy as Sakai et al. [22]. Their algorithm uses *memoization*, which is a standard and very useful strategy for satisfiability algorithms (see, e.g. [23]). Let $\mathcal{C}$ be a circuit class and $\mathcal{C}_n$ be the subclass of circuits from $\mathcal{C}$ that have n variables. Memoization algorithms for $\mathcal{C}$-SAT fit into the following two-step template.

- Step 1: Solve by brute-force all instances of $\mathcal{C}$-SAT where the input circuit $C' \in \mathcal{C}_m$ for some suitable $m \ll n$. (Typically, $m = n^{\varepsilon}$ for some constant ε.) Usually this takes $\exp(m^{O(1)}) \ll 2^n$ time.
- Step 2: On the input $C \in \mathcal{C}_n$, set all input variables $x_{m+1}, \ldots, x_n$ to Boolean values and for each such setting, obtain $C'' \in \mathcal{C}_m$ on m variables. Typically C'' is a circuit for which we have solved satisfiability in Step 1 and hence by a simple table lookup, we should be able to check if C'' is satisfiable in $\text{poly}(|C|)$ time. Overall, this takes time $O^*(2^{n-m}) \ll 2^n$.

At first sight, this seems perfect for k-PTFs, since it is a standard result that the number of k-PTFs on m variables is at most $2^{O(m^{k+1})}$ [8]. Thus, Step 1 can be done in $2^{O(m^{k+1})} \ll 2^n$ time.

For implementing Step 2, we need to ensure that the lookup (for satisfiability for k-PTFs on m variables) can be done quickly. Unfortunately how to do this is unclear. The following two ways suggest themselves.

- Store all polynomials $P' \in \mathbb{Z}[x_1, \ldots, x_m]$ with small coefficients. Since every k-PTF f can be sign-represented by an integer polynomial with coefficients of size $2^{\text{poly}(m)}$ [17], this can be done with a table of size $2^{\text{poly}(m)}$ and in time $2^{\text{poly}(m)}$. When the coefficients are small (say of bit-complexity $\leq n^{o(1)}$), then this strategy already yields a #SAT algorithm, as observed by Sakai et al. [22]. Unfortunately, in general, given a restriction $P'' \in \mathbb{Z}[x_1, \ldots, x_m]$ of a polynomial $P \in \mathbb{Z}[x_1, \ldots, x_n]$, its coefficients can be much larger (say $2^{\text{poly}(n)}$) and it is not clear how to efficiently find a polynomial with small coefficients that sign-represents the same function.
- It is also known that every k-PTF on m variables can be uniquely identified by $\text{poly}(m)$ numbers of bit-complexity $O(m)$ each [8]: these are called the "Chow parameters" of f. Again for this representation, it is unclear how to compute efficiently the Chow parameters of the function represented by the restricted polynomial P''. (Even for an LTF, computing the Chow parameters is as hard as Subset-sum [19].)

The way we solve this problem is by using a beautiful recent result of Kane, Lovett, Moran and Zhang [15], who show that there is a simple decision tree that, when given as input the coefficients of any degree-k polynomial $P' \in \mathbb{Z}[x_1, \ldots, x_m]$, can determine the sign of the polynomial P' at all points in $\{-1, 1\}^m$ using only $\text{poly}(m)$ queries to the coefficients of P. Here, each query is a linear inequality on the coefficients of P; such a decision tree is called a *linear decision tree*.

Our strategy is to replace Step 1 with the construction of this linear decision tree (which can be done in $\exp(m^{O(1)})$ time). At each leaf of the linear decision tree, we replace the truth table of the input polynomial P' by a single bit that indicates whether $f' = \mathrm{sgn}(P')$ is satisfiable or not.

In Step 2, we simply run this decision tree on our restricted polynomial P'' and obtain the answer to the corresponding satisfiability query in $\mathrm{poly}(m, w(P''))$ time. Note, crucially, that the height of the linear decision tree implied by [15] construction is $\mathrm{poly}(m)$ and *independent* of the bit-complexity of the coefficients of the polynomial P'' (which may be as big as $\mathrm{poly}(n)$ in our algorithm). This concludes the description of the algorithm for k-PTF.

Satisfiability algorithm for k-PTF circuits

For k-PTF circuits, we follow a template set up by the result of Kabanets and Lu [14] on sparse-PTF circuits. We start by describing this template and then describe what is new in our algorithm.

The Kabanets-Lu algorithm is an induction on the depth d of the circuit (which is a fixed constant). Given as input a depth d k-PTF circuit C on n variables, Kabanets and Lu do the following:

Depth-reduction: In [14], it is shown that on a random restriction that sets all *but* $n^{1-2\beta}$ variables (here, think of β as a small constant, say 0.01) to random Boolean values, the bottom layer of C simplifies in the following sense.

All but $t \leq n^{\beta}$ gates at the bottom layer become *exponentially* biased, i.e. on all but $\delta = \exp(-n^{\Omega(1)})$ fraction of inputs they are equal to a fixed $b \in \{-1, 1\}$. Now, for each such biased gate g, there is a minority value $b_g \in \{-1, 1\}$ that it takes on very few inputs. [14] show how to enumerate this small number of inputs in $\delta \cdot 2^n$ time and check if there is a satisfying assignment among these inputs. Having ascertained that there is no such assignment, we replace these gates by their majority value and there are only t gates at the bottom layer. At this point, we "guess" the output of these t "unbiased" gates and for each such guess $\sigma \in \{-1, 1\}^t$, we check if there is an assignment that simultaneously satisfies:

(a) the depth $d - 1$ circuit C', obtained by setting the unbiased gates to the guess σ, is satisfied.

(b) each unbiased gate g_i evaluates to the corresponding value σ_i.

Base case: Continuing this way, we eventually get to a base case which is an AND of sparse PTFs for which there is a satisfiability algorithm using the polynomial method.

In the above algorithm, there are two steps where subquadratic sparsity is crucially used. The first is the minority assignment enumeration algorithm for PTFs, which uses ideas of Chen and Santhanam [6] to reduce the problem to enumerating biased LTFs, which is easy [7]. The second is the base case, which uses a non-trivial polynomial approximation for LTFs [24]. Neither of these results hold for even degree-2 PTFs in general. To overcome this, we do the following.

Enumerating minority assignments. Given a k-PTF on m variables that is $\delta = \exp(-n^{\Omega(1)})$ -close to $b \in \{-1, 1\}$, we enumerate its minority assignments as follows. First, we set up a linear decision tree as in the k-PTF satisfiability algorithm. Then we set all but $q \approx \log \frac{1}{\delta}$ variables of the PTF. On most such settings, the resulting PTF becomes the constant function and we can check this using the linear decision tree we created earlier. In this setting, there is nothing to do. Otherwise, we brute-force over the remaining variables to find the minority

assignments. Setting parameters suitably, this yields an $O(\sqrt{\delta} \cdot 2^m)$ time algorithm to find the minority assignments of a δ-biased k-PTF on m variables.

Base case. Here, we make the additional observation (which [14] do not need) that the AND of PTFs that is obtained further is *small* in that it only has slightly superlinear size. Hence, we can apply another random restriction in the style of [14] and using the minority assignment enumeration ideas, reduce it to an AND of a small (say $n^{0.1}$) number of PTFs on $n^{0.01}$ (say) variables. At this point, we can again run the linear decision tree (in a slightly more generalized form) to check satisfiability.

2 A result of Kane, Lovett, Moran, and Zhang [15]

▶ **Definition 7** (Coefficient vectors.)**.** Fix any $k, m \geq 1$. There are exactly $r = \sum_{i=0}^{k} \binom{m}{i}$ many multilinear monomials of degree at most k. Any multilinear polynomial $P(x_1, \ldots, x_m)$ can be identified with a list of the coefficients of its monomials in lexicographic order (say) and hence with some vector $w \in \mathbb{R}^r$. We call w the *coefficient vector* of P and use $\text{coeff}_{m,k}(P)$ to denote this vector. When m, k are clear from context, we will simply use $\text{coeff}(P)$ instead of $\text{coeff}_{m,k}(P)$.

▶ **Definition 8** (Linear Decision Trees)**.** A *Linear Decision Tree* for a function $f : \mathbb{R}^r \to S$ (for some set S) is a decision tree where each internal node is labelled by a linear inequality of the form $\sum_{i=1}^{r} w_i z_i \geq \theta$ (here $z_1, \ldots, z_n$ denote the input variables). Depending on the answer to this linear inequality, computation proceeds to the left or right child of this node, and this process continues until a leaf is reached, which is labelled with an element of S that is the output of f on the given input.

The following construction of linear decision trees due to Kane, Lovett, Moran and Zhang [15] will be crucial for us.

▶ **Theorem 9.** *There is a randomized algorithm, which on input a positive integer r, a subset $H \subseteq \{-1, 1\}^r$, and an error parameter ε, produces a (random) linear decision tree T of depth $\Delta = O(r \log r \cdot \log(|H|/\varepsilon))$ that computes a (random) function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|} \cup \{?\}$ that has the following properties.*

1. *Each linear query has coefficients in $\{-2, -1, 0, 1, 2\}$.*
2. *Given as input any $w \in \mathbb{R}^r$ such that $\langle w, a \rangle \neq 0$ for all $a \in \{-1, 1\}^r$, $F(w)$ is either the truth table of the LTF defined by w (with constant term 0) on inputs from $H \subseteq \{-1, 1\}^r$, or is equal to ?. Further, we have $\Pr_F[F(w) = ?] \leq \varepsilon$.*

The randomized algorithm runs in time $2^{O(\Delta)}$.

▶ Remark. The last statement in the above theorem is not formally stated in [15] but can easily be inferred from their proof and a remark [15, Page 363] on the "Computational Complexity" of their procedure.[11]

We will need a generalization of this theorem for evaluating (tuples of) k-PTFs. However, it is a simple corollary of this theorem.

▶ **Corollary 10.** *Fix positive constants k and c. Let $r = \sum_{i=0}^{k} \binom{m}{i} = \Theta(m^k)$ denote the number of coefficients in a degree-k multilinear polynomial in m variables. There is a randomized algorithm which on input positive integers m and $\ell \leq m^c$ produces a (random)*

[11] We also thank Daniel Kane (personal communication) for telling us about this.

linear decision tree T of depth $\Delta = O(\ell \cdot m^{k+1} \log m)$ that computes a (random) function $F : \mathbb{R}^{r \cdot \ell} \to \mathbb{N} \cup \{?\}$ that has the following properties.

1. *Each linear query has coefficients in $\{-2, -1, 0, 1, 2\}$.*
2. *Given as input any ℓ-tuple of coefficient vectors $\overline{w} = (\text{coeff}_{m,k}(P_1), \ldots, \text{coeff}_{m,k}(P_\ell)) \in \mathbb{R}^{r \cdot \ell}$ such that $P_i(a) \neq 0$ for all $a \in \{-1, 1\}^m$, $F(\overline{w})$ is either the number of common satisfying assignments to all the k-PTFs on $\{-1, 1\}^m$ sign-represented by $P_1, \ldots, P_\ell$, or is equal to ?. Further, we have $\Pr_F[F(\overline{w}) =?] \leq (1/2)$.*

The randomized algorithm runs in time $2^{O(\Delta)}$.

Proof. For each $b \in \{-1, 1\}^m$, define $\text{eval}_b \in \{-1, 1\}^r$ to be the vector of all evaluations of multilinear monomials of degree at most k, taken in lexicographic order, on the input b. Define $H \subseteq \{-1, 1\}^r$ to be the set $\{\text{eval}_b \mid b \in \{-1, 1\}^m\}$. Clearly, $|H| \leq 2^m$. Further, note that given any polynomial $P(x_1, \ldots, x_m)$ of degree at most k, the truth table of the k-PTF sign-represented by P is given by the evaluation of the LTF represented by $\text{coeff}(P)$ at the points in H. Our aim, therefore, is to evaluate the LTFs corresponding to $\text{coeff}(P_1), \ldots, \text{coeff}(P_\ell)$ at all the points in H.

For each i, we use the randomized algorithm from Theorem 9 to produce a decision tree T_i that evaluates the Boolean function $f_i : \{-1, 1\}^m \to \{-1, 1\}$ sign-represented by P_i (or equivalently, evaluating the LTF corresponding to $\text{coeff}(P_i)$ at all points in H) with error $\varepsilon = 1/2\ell$. Note that T_i has depth $O(m^k \log m \cdot \log(2^m/\ell)) = O(m^{k+1} \log m)$ as $\ell \leq m^c$. The final tree T is obtained by simply running $T_1, \ldots, T_\ell$ in order, which is of depth $O(\ell m^{k+1} \log m)$. The tree T outputs the number of common satisfying assignments to all the f_i if all the T_is succeed and ? otherwise. Since each T_i outputs ? with probability at most $1/2\ell$, the tree T outputs ? with probability at most $(1/2\ell) \cdot \ell = 1/2$.

The claim about the running time follows from the analogous claim in Theorem 9 and the fact that the number of common satisfying assignments to all the f_i can be computed from the truth tables in $2^{O(m)}$ time. This completes the proof. ◀

3 The PTF-SAT algorithm

We are now ready to prove Theorem 3. We first state the algorithm, which follows a standard memoization idea (see, e.g. [23]). We assume that the input is a polynomial $P \in \mathbb{Z}[x_1, \ldots, x_n]$ of degree at most k that sign-represents a Boolean function f on n variables. The parameters of the instance are assumed to be (n, M). Set $m = n^{1/(k+1)} / \log n$.

Algorithm $\mathcal{A}$.

1. Use $n_1 = 10n$ *independent runs* of the algorithm from Corollary 10 with $\ell = 1$ to construct independent random linear decision trees $T_1, \ldots, T_{n_1}$ such that on any input polynomial $Q(x_1, \ldots, x_m)$ (or more precisely $\text{coeff}_{m,k}(Q)$) of degree at most k that sign-represents an k-PTF g on m variables, each T_i computes the number of satisfying assignments to g with error at most $1/2$.
2. Set $N = 0$. (N will ultimately be the number of satisfying assignments to f.)
3. For each setting $\sigma \in \{-1, 1\}^{n-m}$ to the variables $x_{m+1}, \ldots, x_n$, do the following:
   a. Compute the polynomial P_σ obtained by substituting the variables $x_{m+1,\ldots,x_n}$ accordingly in P.

b. Run the decision trees $T_1, \ldots, T_{n_1}$ on coeff(P_σ) and compute their outputs. If all the outputs are ?, output ?. Otherwise, some T_i outputs N_σ, the number of satisfying assignments to P_σ. Add this to the current estimate to N.

4. Output N.

Correctness. It is clear from Corollary 10 and step 3b that algorithm $\mathcal{A}$ outputs either ? or the correct number of satisfying assignments to f. Further, we claim that with probability at least $1 - 1/2^{\Omega(n)}$, the output is indeed the number of satisfying assignments to f. To see this, observe that it follows from Corollary 10 that for each setting $\sigma \in \{-1,1\}^{n-m}$ to the variables $x_{m+1}, \ldots, x_n$, each linear decision tree T_i produced in step 1 errs on coeff(P_σ) (i.e. outputs ?) with probability at most $1/2$. The probability of each T_i doing so is thus at most $1/2^{n_1}$, as they are constructed independently. So the probability that the algorithm fails to determine N_σ is at most $1/2^{n_1}$. Finally, taking a union bound over all σ, which are 2^{n-m} in number, we conclude that the probability of algorithm $\mathcal{A}$ outputting ? is at most $2^{n-m}/2^{n_1} \leq 1/2^{\Omega(n)}$.

Running time. We show that the running time of algorithm $\mathcal{A}$ is $\text{poly}(n, M) \cdot 2^{n-m}$. First note that by Corollary 10, the construction of a single linear decision tree T_i takes $2^{O(\Gamma)}$ time, where $\Gamma = m^{k+1} \log m$, and hence, step 1 takes $n_1 \cdot 2^{O(\Gamma)}$ time. Next, for a setting $\sigma \in \{-1,1\}^{n-m}$ to the variables $x_{m+1}, \ldots, x_n$, computing P_σ and constructing the vector coeff(P_σ) takes only $\text{poly}(n, M)$ time. Recall that the depth of each linear decision tree T_i is $O(\Gamma)$ and thus, on input vector coeff(P_σ), each of whose entries has bit complexity at most M, it takes time $O(\Gamma) \cdot \text{poly}(M, n)$ to run all T_i and obtain the output N_σ or ?. Therefore, step 3 takes $\text{poly}(n, M) \cdot 2^{n-m}$ time. Finally, the claim about the total running time of algorithm $\mathcal{A}$ follows at once when we observe that for the setting $m = n^{1/(k+1)}/\log n$, $\Gamma = o(n/(\log n)^k) = o(n)$.

4 Constant-depth circuits with PTF gates

In this section we give an algorithm for counting the number of satisfying assignment for a k-PTF circuit of constant depth and slightly superlinear size. We begin with some definitions.

▶ **Definition 11.** Let $\delta \leq 1$ be any parameter. Two Boolean functions f, g are said to be δ-close if $\Pr_x[f(x) \neq g(x)] \leq \delta$.

A k-PTF f specified by a polynomial P is said to be δ-close to an explicit constant if it is δ-close to a constant and such a constant can be computed efficiently, i.e. $\text{poly}(n, M)$, where n is the number of variables in P and $w(P)$ is at most M.

▶ **Definition 12.** For a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$, the majority value of f is the bit value $b \in \{-1,1\}$ which maximizes $\Pr_x[f(x) = b]$ and the bit value $-b$ is said to be its minority value.

For a Boolean function f with majority value b, an assignment $x \in \{-1,1\}^n$ is said to be a majority assignment if $f(x) = b$ and minority assignment otherwise.

▶ **Definition 13.** Given a k-PTF f on n variables specified by a polynomial P, a parameter $m \leq n$ and a partial assignment $\sigma \in \{-1,1\}^{n-m}$ on $n-m$ variables, let P_σ be the polynomial obtained by substituting the variables in P according to σ. If P has parameters (n, M) then P_σ has parameters (m, M). For a k-PTF circuit C, C_σ is defined similarly. If C has parameters (n, s, d, M) then C_σ has parameters (m, s, d, M).

Outline of the #SAT procedure. For designing a #SAT algorithm for k-PTF circuits, we use the generic framework developed by Kabanets and Lu [14] with some crucial modifications.

Given a k-PTF circuit C on n variables of depth d we want to count the number of satisfying assignments $a \in \{-1,1\}^n$ such that $C(a) = -1$. We in fact solve a slightly more general problem. Given $(C, \mathcal{P})$, where C is a small k-PTF circuit of depth d and $\mathcal{P}$ is a set of k-PTF functions, such that $\sum_{f\in\mathcal{P}} \text{fan-in}(f)$ is small, we count the number of assignments that simultaneously satisfy C and all the function in $\mathcal{P}$.

At the core of the algorithm that solves this problem, Algorithm $\mathcal{B}$, is a recursive procedure $\mathcal{A}_5$, which works as follows: on inputs $(C, \mathcal{P})$ it first applies a simplification step that outputs $\ll 2^n$ instances of the form $(C', \mathcal{P}')$ such that

- Both C' and functions in $\mathcal{P}'$ are on $m \ll n$ variables.
- The sets of satisfying assignments of these instances "almost" partition the set of satisfying assignments of $(C, \mathcal{P})$.
- With all but very small probability the bottom layer of C' has the following nice structure.
  - At most n gates are δ-biased. We denote this set of gates by B (as we will simplify them by setting them to the values they are biased towards).
  - At most n^{β_d} gates are not δ-biased. We denote these gates by G (as we will simplify them by "guessing" their values).
- There is a small set of satisfying assignments that are not covered by the satisfying assignments of $(C', \mathcal{P}')$ but we can count these assignments with a brute-force algorithm that does not take too much time.

For each C' with this nice structure, then we try to use this structure to create C'' which has depth $d-1$. Suppose we reduce the depth as follows:

- Set all the gates in B to the values that they are biased towards.
- Try all the settings of the values that the gates in G can take, thereby from C' creating possibly $2^{n^{\beta_d}}$ instances $(C'', \mathcal{P}')$.

$(C'', \mathcal{P}')$ now is an instance where C'' has depth $d-1$. Unfortunately, by simply setting biased gates to the values they are biased towards, we may miss out on the minority assignments to these gates which could eventually satisfy C'. We design a subroutine $\mathcal{A}_3$ to precisely handle this issue, i.e. to keep track of the number of minority assignments, say $N_{C'}$. This part of our algorithm is completely different from that of [14], which only works for subquadratically sparse PTFs.

Once $\mathcal{A}_3$ has computed $N_{C'}$, i.e. the number of satisfying assignments among the minority assignments, we now need to only count the number of satisfying assignments among the rest of the assignments.

To achieve this we use an idea similar to that in [7, 14], which involves appending $\mathcal{P}'$ with a few more k-PTFs (this forces the biased gates to their majority values). This gives say a set $\tilde{\mathcal{P}}'$. Similarly, while setting gates in G to their guessed values, we again use the same idea to ensure that we are counting satisfying assignments consistent with the guessed values, once again updating $\tilde{\mathcal{P}}'$ to a new set $\mathcal{P}''$. This creates instances of the form $(C'', \mathcal{P}'')$, where the depth of C'' is $d-1$.

This way, we iteratively decrease the depth of the circuit by 1. Finally, we have instances $(C'', \mathcal{P}'')$ such that the depth of C'' is 1, i.e. it is a single k-PTF, say h. At this stage we solve $\#\text{SAT}(\tilde{C})$, where $\tilde{C} = h \wedge \bigwedge_{f\in\mathcal{P}''} f$. This is handled in a subroutine $\mathcal{A}_4$. Here too our algorithm differs significantly from [14].

In what follows we will prove Theorem 6. In order to do so, we will set up various subroutines $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5$ designed to accomplish certain tasks and combine them together at the end to finally design algorithm $\mathcal{B}$ for the #SAT problem for k-PTF circuits.

$\mathcal{A}_1$ will be an oracle, used in other routines, which will compute number of common satisfying assignments for small AND of PTFs on few variables (using the same idea as in the algorithm for #SAT for k-PTFs). $\mathcal{A}_2$ will be a simplification step, which will allow us to argue to argue about some structure in the circuit (this algorithm is from [14]). It will make many gates at the bottom of the circuit δ-close to a constant, thus simplifying it. $\mathcal{A}_3$ will be used to count minority satisfying assignments for a bunch of δ-biased PTFs, i.e. assignments which cause at least one of the PTFs to evaluate to its minority value. $\mathcal{A}_4$ will be a general base of case of our algorithm, which will count satisfying assignments for AND of superlinear many PTFs, by first using $\mathcal{A}_2$ to simplify the circuit, then reducing it to the case of small AND of PTFs and then using $\mathcal{A}_1$. $\mathcal{A}_5$ will be a recursive procedure, which will use $\mathcal{A}_2$ to first simplify the circuit, and then convert it into a circuit of lower depth, finally making a recursive call on the simplified circuit.

Parameter setting. Let d be a constant. Let A, B be two fixed absolute large constants. Let $\zeta = \min(1, A/2Bk^2)$. For each $2 \leq i \leq d$, let $\beta_i = A \cdot \varepsilon_i$ and $\varepsilon_i = (\frac{\zeta}{10A(k+1)})^i$. Choose $\beta_1 = 1/10$ and $\varepsilon_1 = 1/10A$.

Oracle access to a subroutine. Let $\mathcal{A}_1(n', s, f_1, \ldots, f_s)$ denote a subroutine with the following specification. Here, n is the number of variables in the original input circuit.

> **Input:** AND of k-PTFs, say $f_1, \ldots, f_s$ specified by polynomials $P_1, \ldots, P_s$ respectively, such that $s \leq n^{0.1}$ and for each $i \in [s]$, f_i is defined over $n' \leq n^{1/(2(k+1))}$ variables and $w(P_i) \leq M$.
> **Output:** $\#\{a \in \{-1,1\}^{n'} \mid \forall i \in [s], P_i(a) = -1\}$.

In what follows, we will assume that we have access to the above subroutine $\mathcal{A}_1$. We will set up such an oracle and show that it answers any call to it in time $\text{poly}(n, M)$ in Section 4.5.

4.1 Simplification of a k-PTF circuit

For any $1 > \varepsilon \gg (\log n)^{-1}$, let $\beta = A\varepsilon$ and $\delta = \exp(-n^{\beta/B \cdot k^2})$, where A and B are constants. Note that it is these constants A, B we use in the parameter settings paragraph above. Let $\mathcal{A}_2(C, d, n, M)$ be the following subroutine.

> **Input:** k-PTF circuit C of depth d on n variables with size $n^{1+\varepsilon}$ and weight M.
> **Output:** A decision tree T_{DT} of depth $n - n^{1-2\beta}$ such that for a uniformly random leaf $\sigma \in \{-1,1\}^{n-n^{1-2\beta}}$ it outputs a *good* circuit C_σ with probability $1 - \exp(-n^\varepsilon)$, where C_σ is called good if its bottom layer has the following structure:
> - there are at most n gates which are δ-close to an explicit constant. Let B_σ denote this set of gates.
> - there are at most n^β gates that are not δ-close to an explicit constant. Let us denote this set of gates by G_σ.

In [14], such a subroutine $\mathcal{A}_2(C, d, n, M)$ was designed. Specifically, they proved the following theorem.

▶ **Theorem 14** (Kabanets and Lu [14]). *There is a zero-error randomized algorithm $\mathcal{A}_2(C, d, n, M)$ that runs in time $\text{poly}(n, M) \cdot O(2^{n-n^{1-2\beta}})$ and outputs a decision tree as described above with probability at least $1 - 1/2^{10n}$ (and outputs ? otherwise). Moreover, given a good C_σ, there is a deterministic algorithm that runs in time $\text{poly}(n, M)$ which computes B_σ and G_σ.*

▶ Remark. In [14], it is easy to see that the probability of outputting ? is at most $1/2$. To bring down this probability to $1/2^{10n}$, we run their procedure in parallel $10n$ times, and output the first tree that is output by the algorithm. The probability that no such tree is output is $1/2^{10n}$.

▶ Remark. In designing the above subroutine in [14], they consider a more general class of polynomially sparse-PTF circuits (i.e. each gate computes a PTF with polynomially many monomials) as opposed to the k-PTF circuits we consider here. Under this weaker assumption, they get that $\delta = \exp(-n^{\Omega(\beta^3)})$. However, by redoing their analysis for degree k-PTFs, it is easy to see that δ could be set to $\exp(-n^{\beta/B\cdot k^2})$ for some constant B. Under this setting of δ, we get exactly the same guarantees. In this sense, the above theorem statement is a slight restatement of [14, Theorem 3.11].

4.2 Enumerating the minority assignments

We now design an algorithm $\mathcal{A}_3(m, \ell, \delta, g_1, \ldots, g_\ell)$, which has the following behaviour.

Input: parameters $m \leq n, \ell, \delta$ such that $\delta \in \left[\exp(-m^{1/10(k+1)}), 1\right]$, $\ell \leq m^2$, k-PTFs $g_1, g_2, \ldots, g_\ell$ specified by polynomials $P_1, \ldots, P_\ell$ on m variables $(x_1, \ldots, x_m)$ each of weight at most M and which are δ-close to -1.

Oracle access to: $\mathcal{A}_1$.

Output: $a \in \{-1, 1\}^m$ such that $\exists i \in [\ell]$ for which $P_i(a) > 0$.

▶ **Lemma 15.** *There is a deterministic algorithm $\mathcal{A}_3(m, \ell, \delta, g_1, \ldots, g_\ell)$ as specified above that runs in time* $\mathrm{poly}(m, M) \cdot \sqrt{\delta} \cdot 2^m$.

Proof. We start with the description of the algorithm.

$\boldsymbol{\mathcal{A}_3(\mathrm{m}, \ell, \delta, \mathrm{g}_1, \ldots, \mathrm{g}_\ell)}$.

1. Set $q = \frac{1}{2}\log\frac{1}{\delta} \leq \frac{m}{2}$ and let $\mathcal{N} = \emptyset$. ($\mathcal{N}$ will eventually be the collection of minority assignments i.e. all $a \in \{-1, 1\}^m$ such that $\exists i \in [\ell]$ for which $P_i(a) > 0$.)
2. For each setting $\rho \in \{-1, 1\}^{m-q}$ to the variables $x_{q+1}, \ldots, x_m$, do the following:
   a. Construct the restricted polynomials $P_{1,\rho}, \ldots, P_{\ell,\rho}$. Let $g_{i,\rho} = \mathrm{sgn}(P_{i,\rho})$ for $i \in [\ell]$.
   b. Using oracle $\mathcal{A}_1(q, 1, -g_{i,\rho})$, check for each $i \in [\ell]$ if $g_{i,\rho}$ is the constant function -1 by checking if the output of the oracle on the input $-g_{i,\rho}$ is zero.
   c. If there is an $i \in [\ell]$ such that $g_{i,\rho}$ is not the constant function -1, try all possible assignments χ to the remaining q variables $x_1, \ldots, x_q$. This way, enumerate all assignments $b = (\chi, \rho)$ to $x_1, \ldots, x_m$ for which there is an $i \in [\ell]$ such that $P_i(b) > 0$. Add such an assignment to the collection $\mathcal{N}$.
3. Output $\mathcal{N}$.

Correctness. If $a \in \{-1, 1\}^m$ is a minority assignment (i.e. $\exists i_0 \in [\ell]$ so that $P_{i_0}(a) < 0$) and if $a = (\chi, \rho)$ where ρ is an assignment to the last $m - q$ variables, and χ to the first q, a will get added to $\mathcal{N}$ in the loop of step 2 corresponding to ρ and that of χ in step 2c, because of i_0 being a witness. Conversely, observe that we only add to the collection $\mathcal{N}$ when we encounter a minority assignment.

Running time. For each setting $\rho \in \{-1, 1\}^{m-q}$ to the variables $x_{q+1}, \ldots, x_m$, step 2a takes $\mathrm{poly}(m, M)$ time and step 2b takes $O(\ell) = O(m^2)$ time and so combined, they take only $\mathrm{poly}(m, M)$ time. Let $\mathcal{T}$ be the set consisting of all assignments ρ to the last $m - q$ variables such that the algorithm enters the loop described in step 2c i.e.

$$\mathcal{T} = \{\rho \in \{-1, 1\}^{m-q} | \exists i \in [\ell] : g_{i,\rho} \text{ is not the constant function} - 1\}$$

and let $\mathcal{T}^c$ denote its complement. Also note that for a $\rho \in \mathcal{T}$, enumeration of minority assignments in step 2c takes $2^q \cdot \ell \cdot \text{poly}(m, M)$ time. Therefore, we can bound the total running time by

$$\text{poly}(m, M)(2^q \cdot |\mathcal{T}| + |\mathcal{T}^c|).$$

Next, we claim that the size of $\mathcal{T}$ is small:

▶ **Lemma 16.** $|\mathcal{T}| \leq \ell \cdot \sqrt{\delta} \cdot 2^{m-q}$.

Proof. We define for $i \in [\ell]$, $\mathcal{T}_i = \{\rho \in \{-1,1\}^{m-q} | g_{i,\rho} \text{ is not the constant function } -1\}$. By the union bound, it is sufficient to show that $|\mathcal{T}_i| \leq \sqrt{\delta} \cdot 2^{m-q}$ for a fixed $i \in [\ell]$. Let D_m denote the uniform distribution on $\{-1,1\}^m$ i.e. on all possible assignments to the variables $x_1, \ldots, x_m$. Then from the definition of δ-closeness, we know

$$\Pr_{a \sim D_m}[g_i(a) = 1] \leq \delta.$$

Writing LHS in the following way, we have

$$\mathop{\mathbf{E}}_{\rho \sim D_{m-q}} \left[\Pr_{\chi \sim D_q}[g_{i,\rho}(\chi) = 1] \right] \leq \delta$$

where D_{m-q} and D_q denote uniform distributions on assignments to the last $m-q$ variables and the first q variables respectively. By Markov's inequality,

$$\Pr_{\rho \sim D_{m-q}}[\Pr_{\chi \sim D_q}[g_{i,\rho}(\chi) = 1] \geq \sqrt{\delta}] \leq \sqrt{\delta}$$

Consider a ρ for which this event does not occur i.e. for which $\Pr_{\chi \sim D_q}[g_{i,\rho}(\chi) = 1] < \sqrt{\delta}$. For such a ρ, $g_{i,\rho}$ has only $2^q = 1/\sqrt{\delta}$ many inputs and therefore, $g_{i,\rho}$ must be the constant function -1. Thus, we conclude that

$$\Pr_{\rho \sim D_{m-q}}[g_{i,\rho} \text{ is not the constant function } -1] \leq \sqrt{\delta}$$

or in other words, $|\mathcal{T}_i| \leq \sqrt{\delta} \cdot 2^{m-q}$. ◀

Finally, by using the trivial bound $|\mathcal{T}^c| \leq 2^{m-q}$ and the above claim, we obtain a total running time of $\text{poly}(m, M) \cdot \sqrt{\delta} \cdot 2^m$ and this concludes the proof of the lemma. ◀

4.3 #SAT for AND of k-PTFs

We design an algorithm $\mathcal{A}_4(n, M, g_1, \ldots, g_\tau)$ with the following functionality.

Input: A set of k-PTFs $g_1, \ldots, g_\tau$ specified by polynomials $P_1, \ldots, P_\tau$ on n variables such that $w(p_i) \leq M$ for each $i \in [\tau]$ and $\sum_{i \in [\tau]} \text{fan-in}(g_i) \leq n^{1+\varepsilon_1}$.
Output: $\#\{a \in \{-1,1\}^n \mid \forall i \in [\tau], P_i(a) < 0\}$.

4.3.1 The details of the algorithm

$\mathcal{A}_4(\mathbf{n}, \mathbf{M}, \mathbf{g_1}, \ldots, \mathbf{g}_\tau)$.

1. Let $m = n^\alpha$ for $\alpha = \frac{\zeta\varepsilon_1}{2(k+1)}$. Let C denote the AND of $g_1, \ldots, g_\tau$.
2. Run $\mathcal{A}_2(C, 2, n, M)$ to obtain the decision tree T_{DT}. Initialize N to 0.
3. For each leaf σ of T_{DT}, do the following:
   - **(A)** If C_σ is not *good*, count the number of satisfying assignments for C_σ by brute-force and add to N.
   - **(B)** If C_σ is *good*, do the following:
     - **(i)** C_σ is now an AND of PTFs in B_σ and G_σ, over $n' = n^{1-2\beta_1}$ variables, where all PTFs in B_σ are δ-close to an explicit constant, where $\delta = \exp(-n^{\beta_1/B\cdot k^2})$. Moreover, $|B_\sigma| \leq n, |G_\sigma| \leq n^{\beta_1}$.
       Let $B_\sigma = \{h_1, \ldots, h_\ell\}$ be specified by $Q_1, \ldots, Q_\ell$.
       Suppose for $i \in [\ell]$, h_i is close to $a_i \in \{-1, 1\}$. Then let $Q'_i = -a_i \cdot Q_i$ and $h'_i = \mathrm{sgn}(Q'_i)$. Let $B'_\sigma = \{Q'_1, \ldots, Q'_\ell\}$.
     - **(ii)** For each restriction $\rho : \{x_{m+1}, \ldots, x_{n'}\} \to \{-1, 1\}$, do the following:
       - **(a)** Check if there exists $h' \in B'_\sigma$ such that h'_ρ is not the constant function -1 using $\mathcal{A}_1(m, 1, h'_\rho)$.
       - **(b)** If such an $h' \in B'_\sigma$ exists, then count the number of satisfying assignments for $C_{\sigma\rho}$ by brute-force and add to N.
       - **(c)** If the above does not hold, we have established that for each $h_i \in B_\sigma$, $h_{i,\rho}$ is the constant function a_i. If $\exists i \in [\ell]$ such that $a_i = 1$, it means $C_{\sigma\rho}$ is also a constant 1 . Then simply halt. Else set each h_i to a_i.
         Thus, $C_{\sigma\rho}$ has been reduced to an AND of n^{β_1} many PTFs over m variables. Call this set $G'_{\sigma\rho}$, use $\mathcal{A}_1(m, n^{\beta_1}, G'_{\sigma\rho})$ to calculate the number of satisfying assignments and add the output to N.
4. Finally, output N.

4.3.2 The correctness argument and running time analysis

▶ **Lemma 17.** *$\mathcal{A}_4$ is a zero-error randomized algorithm that counts the number of satisfying assignments correctly. Further, $\mathcal{A}_4$ runs in time* $\mathrm{poly}(n, M) \cdot 2^{n-n^\alpha}$ *and outputs the right answer with probability at least* $1/2$ *(and outputs ? otherwise).*

Proof.

Correctness. For a leaf σ of T_{DT}, when C_σ is not *good*, we simply use brute-force, which is guaranteed to be correct. Otherwise,

- If h'_ρ not the constant function -1 for some $h' \in B'_\sigma$, then we again use brute-force, which is guaranteed to work correctly.
- Otherwise, for each $h' \in B'_\sigma$, h'_ρ is the constant function -1. Here we only need to consider the satisfying assignments for the gates in $G_{\sigma\rho}$. For this we use $\mathcal{A}_1$, that works correctly by assumption.

Further, we need to ensure that the parameters that we call $\mathcal{A}_1$ on, are valid. To see this, observe that $m = n^\alpha \leq n^{1/(2(k+1))}$ because of the setting of α and further, we have $n^{\beta_1} \leq n^{0.1}$.

Finally, the claim about the error probability follows from the error probability of $\mathcal{A}_2$ (Theorem 14).

Running Time. The time taken for constructing T_{DT} is $O^*(2^{n-n^{1-2\beta_1}})$, by Theorem 14. For a leaf σ of T_{DT}, we know that step 3A is executed with probability at most $2^{-n^{\varepsilon_1}}$. The total time for running step 3A is thus $O^*(2^{n-n^{\varepsilon_1}})$. We know that the oracle $\mathcal{A}_1$ answers

calls in poly(n, M) time. Hence, the total time for running step 3(B)iia is $O^*(2^{n-n^\alpha})$. Next, note that if step 3(B)iib is executed, then all PTFs in B_σ are δ-close to -1. So, the number of times it runs is at most $\delta \cdot 2^{n'}$. Therefore, the total time for running step 3(B)iib is $O^*(2^{n+n^\alpha-n^{\beta_1/Bk^2}})$. Similar to the analysis of step 3(B)iia, the total time for running step 3(B)iic is also $O^*(2^{n-n^\alpha})$.
Summing them up, we conclude that total running time is $O^*(2^{n-n^\alpha})$, as due to our choice of various parameters, $n - n^\alpha$ is the dominating power of 2. This completes the proof. ◀

4.4 #SAT for larger depth k-PTF circuits

Let C be a k-PTF circuit of depth $d \geq 1$ on n variables and let $\mathcal{P}$ be a set of k-PTFs $g_1, \ldots, g_\tau$, which are specified by n-variate polynomials $P_1, \ldots, P_\tau$. Let #SAT$(C, \mathcal{P})$ denote $\#\{a \in \{-1,1\}^n \mid C(a) < 0 \text{ and } \forall i \in [\tau], P_i(a) < 0\}$. We now specify our depth-reduction algorithm $\mathcal{A}_5(n, d, M, n^{1+\varepsilon_d}, C, \mathcal{P})$.

Input: $(C, \mathcal{P})$ as follows:
- k-PTF circuit C with parameters $(n, n^{1+\varepsilon_d}, d, M)$.
- a set $\mathcal{P}$ of k-PTFs $g_1, \ldots, g_\tau$ on n variables, which are specified by polynomials $P_1, \ldots, P_\tau$ such that $\sum_{i=1}^{\tau} \text{fan-in}(g_i) \leq n^{1+\varepsilon_d}$ and for each $i \in [\tau]$, $w(P_i) \leq M$.

Oracle access to: $\mathcal{A}_1, \mathcal{A}_4$.
Output: #SAT$(C, \mathcal{P})$.

We start by describing the algorithm.

4.4.1 The details of the algorithm

Let `count` be a global counter initialized to 0 before the execution of the algorithm.

$\mathcal{A}_5(\mathbf{n, d, M, n^{1+\varepsilon_d}, C, \mathcal{P}})$.

1. If $d = 1$, output $\mathcal{A}_4(n, M, \{C\} \cup \mathcal{P})$ and halt.
2. Run $\mathcal{A}_2(C, d, n, M)$, which gives us a T_{DT}.
3. For each leaf $\sigma \in \{-1,1\}^{n-n^{1-2\beta_d}}$ of T_{DT}. (If not, output ?.)
   a. For each $i \in [\tau]$ compute $P_{i,\sigma}$, the polynomial obtained by substituting σ in its variables. Let $\mathcal{P}_\sigma = \{P_{1,\sigma}, \ldots, P_{\tau,\sigma}\}$.
   b. Obtain C_σ. If C_σ is not a good circuit, then brute-force to find the number of satisfying assignments of $(C_\sigma, \mathcal{P}_\sigma)$, say N_σ, and set `count` = `count` + N_σ.
   c. If C_σ is good then obtain B_σ and G_σ.
   d. Let $B_\sigma = \{h_1, \ldots, h_\ell\}$ be specified by $Q_1, \ldots, Q_\ell$. We know that each $h \in B_\sigma$ is δ-close to an explicit constant, for $\delta = \exp(-n^{\beta_d/Bk^2})$.
   Suppose for $i \in [\ell]$, h_i is close to $a_i \in \{-1, 1\}$. Then let $Q'_i = -a_i \cdot Q_i$ and $h'_i = \mathrm{sgn}(Q'_i)$. Let $B'_\sigma = \{Q'_1, \ldots, Q'_\ell\}$.
   e. Run $\mathcal{A}_3(n^{1-2\beta_d}, \ell, \delta, h'_1, \ldots, h'_\ell)$ to obtain the set $\mathcal{N}_\sigma$ of all the minority assignments of B_σ. (Note that this uses oracle access to $\mathcal{A}_1$.)
   for each $a \in \mathcal{N}_\sigma$, if $((C(a) < 0)$ AND $(\forall i \in [\ell], P_i(a) < 0))$, then `count` = `count` + 1.
   f. Let $G_\sigma = \{f_1, \ldots, f_t\}$ be specified by polynomials $R_1, \ldots, R_t$. We know that $t \leq n^{\beta_d}$. For each $b \in \{-1,1\}^t$,
      i. Let $R'_i = -b_i \cdot R_i$ for $i \in [t]$. Let $G'_{\sigma,b} = \{R'_1, \ldots, R'_t\}$.
      ii. Let $C_{\sigma,b}$ be the circuit obtained from C_σ by replacing each h_i by a_i $1 \leq i \leq \ell$ and each f_j by b_j for $1 \leq j \leq t$.

iii. $\mathcal{P}_{\sigma,b} = \mathcal{P}_\sigma \cup B'_\sigma \cup G'_{\sigma,b}$.
iv. If $d > 2$ then run $\mathcal{A}_5(n^{1-2\beta_d}, d-1, M, n^{1+\varepsilon_d}, C_{\sigma,b}, \mathcal{P}_{\sigma,b})$ $n_1 = 10n$ times and let N_σ be the output of the first run that does not output ?. Set $\texttt{count} = \texttt{count} + N_\sigma$. (If all runs of $\mathcal{A}_5$ output ?, then output ?.)
v. If $d = 2$ then run $\mathcal{A}_4(n^{1-2\beta_d}, M, C_{\sigma,b} \cup \mathcal{P}_{\sigma,b})$ $n_1 = 10n$ times and let N_σ be the output of the first run that does not output ?. Set $\texttt{count} = \texttt{count} + N_\sigma$. (If all runs of $\mathcal{A}_5$ output ?, then output ?.)

4. Output count.

4.4.2 The correctness argument and running time analysis

▶ **Lemma 18.** *The algorithm $\mathcal{A}_5$ described above is a zero-error randomized algorithm which on input $(C, \mathcal{P})$ as described above, correctly #SAT$(C, \mathcal{P})$. Moreover, the algorithm outputs the correct answer (and not ?) with probability at least $1/2$. Finally, $\mathcal{A}_5(n, d, M, n^{1+\varepsilon_d}, C, \emptyset)$ runs in time $\mathrm{poly}(n, M) \cdot 2^{n - n^{\zeta\varepsilon_d/2(k+1)}}$, where parameters ε_d, ζ are as defined at the beginning of Section 4.*

Proof. We argue correctness by induction on the depth d of the circuit C.

Clearly, if $d = 1$, correctness follows from the correctness of algorithm $\mathcal{A}_4$. This takes care of the base case.

If $d \geq 2$, we argue first that if the algorithm does not output ?, then it does output #SAT$(C, \mathcal{P})$ correctly. Assume that the algorithm $\mathcal{A}_2$ outputs a decision tree T_{DT} as required (otherwise, the algorithm outputs ? and we are done). Now, it is sufficient to argue that for each σ, the number of satisfying assignments to $(C_\sigma, \mathcal{P}_\sigma)$ is computed correctly (if the algorithm does not output ?).

Fix any σ. If C_σ is not a good circuit, then the algorithm uses brute-force to compute #SAT$(C_\sigma, \mathcal{P}_\sigma)$ which yields the right answer. So we may assume that C_σ is indeed good.

Now, the satisfying assignments to $(C_\sigma, \mathcal{P}_\sigma)$ break into two kinds: those that are minority assignments to the set B_σ and those that are majority assignments to B_σ. The former set is enumerated in Step 3e (correctly by our analysis of $\mathcal{A}_3$) and hence we count all these assignments in this step.

Finally, we claim that the satisfying assignments to $(C_\sigma, \mathcal{P}_\sigma)$ that are majority assignments of all gates in B_σ are counted in Step 3f. To see this, note that each such assignment $a \in \{-1, 1\}^{n^{1-2\beta_d}}$ forces the gates in G_σ to some values $b_1, \ldots, b_t \in \{-1, 1\}$. Note that for each such $b \in \{-1, 1\}^t$, these assignments are exactly the satisfying assignments of the pair $(C_{\sigma,b}, \mathcal{P}_{\sigma,b})$ as defined in the algorithm. In particular, the number satisfying assignments to $(C_\sigma, \mathcal{P}_\sigma)$ that are majority assignments of all gates in B_σ can be written as

$$\sum_{b \in \{-1,1\}^t} \#\mathrm{SAT}(C_{\sigma,b}, \mathcal{P}_{\sigma,b}).$$

We now want to apply the induction hypothesis to argue that all the terms in the sum are computed correctly. To do this, we need to argue that the size of $C_{\sigma,b}$ and the total fan-in of the gates in $\mathcal{P}_{\sigma,b}$ are bounded as required (note that the total size of C remains the same, while the total fan-in of $\mathcal{P}$ increases by the total fan-in of the gates in $B'_\sigma \cup G'_{\sigma,b}$ which is at most $n^{1+\varepsilon_d}$). It can be checked that this boils down to the following two inequalities

$$n^{(1-2\beta_d)(1+\varepsilon_{d-1})} \geq n^{1+\varepsilon_d} \text{ and } n^{(1-2\beta_d)(1+\varepsilon_{d-1})} \leq 2n^{1+\varepsilon_d}$$

both of which are easily verified for our choice of parameters (for large enough n). Thus, by the induction hypothesis, all the terms in the sum are computed correctly (unless we get ?). Hence, the output of the algorithm is correct by induction.

Now, we analyze the probability of error. If $d = 1$, the probability of error is at most $1/2$ by the analysis of $\mathcal{A}_4$. If $d > 2$, we get an error if either $\mathcal{A}_2$ outputs ? or there is some σ such that the corresponding runs of $\mathcal{A}_5$ or $\mathcal{A}_4$ output ?. The probability of each is at most $1/2^{10n}$. Taking a union bound over at most 2^n many σ, we see that the probability of error is at most $1/2^{\Omega(n)} \leq 1/2$.

Finally, we analyze the running time. Define $\mathcal{T}(n, d, M)$ to be the running time of the algorithm on a pair $(C, \mathcal{P})$ as specified in the input description above. We need the following claim.

▶ **Lemma 19.** $\mathcal{T}(n, d, M) \leq \text{poly}(n, M) \cdot 2^{n - n^{\zeta \varepsilon_d / 2(k+1)}}$.

To see the above, we argue by induction. The case $d = 1$ follows from the running time of $\mathcal{A}_4$. Further from the description of the algorithm, we get the following inequality for $d \geq 2$.

$$\mathcal{T}(n, d, M) \leq \text{poly}(n, M) \cdot (2^{n - n^{1-2\beta_d}} + 2^{n - n^{\varepsilon_d}} + 2^{n - \frac{1}{2} \cdot n^{-\beta_d/(Bk^2)}} + 2^{n - n^{(1-2\beta_d)\zeta\varepsilon_{d-1}/2(k+1)}}) \quad (1)$$

The first term above accounts for the running time of $\mathcal{A}_2$ and all steps other Steps 3b,3e and 3f. The second term accounts for the brute force search in Step 3b since there are only a $2^{-n^{\varepsilon_d}}$ fraction of σ where it is performed. The third term accounts for the minority enumeration algorithm in Step 3e (running time follows from the running time of that algorithm). The last term is the running time of Step 3f and follows from the induction hypothesis.

It suffices to argue that each term in the RHS of (1) can be bounded by $2^{n - n^{\zeta \varepsilon_d / 2(k+1)}}$. This is an easy verification from our choice of parameters and left to the reader. This concludes the proof. ◀

4.5 Putting it together

In this subsection, we complete the proof of Theorem 6 using the aforementioned subroutines. We also need to describe the subroutine $\mathcal{A}_1$, which is critical for all the other subroutines. We shall do so *inside* our final algorithm for the #SAT problem for k-PTF circuits, algorithm $\mathcal{B}$. Recall that $\mathcal{A}_1$ has the following specifications:

Input: AND of k-PTFs, say $f_1, \ldots, f_s$ specified by polynomials $P_1, \ldots, P_s$ respectively, such that $s \leq n^{0.1}$ and for each $i \in [s]$, f_i is defined over $n' \leq n^{1/(2(k+1))}$ variables and $w(P_i) \leq M$.
Output: $\#\{a \in \{-1, 1\}^{n'} \mid \forall i \in [s], f_i(a) = -1\}$.

We are now ready to complete the proof of Theorem 6. Suppose C is the input k-PTF circuit with parameters $(n, n^{1+\varepsilon_d}, d, M)$. On these input parameters $(C, n, n^{1+\varepsilon_d}, d, k, M)$, we finally have the following algorithm for the #SAT problem for k-PTF circuits:

$\mathcal{B}(\mathbf{C, n, n^{1+\varepsilon_d}, d, k, M})$.

1. (*Oracle Construction Step*) Construct the oracle $\mathcal{A}_1$ as follows. Use $n_1 = 10n$ *independent runs* of the algorithm from Corollary 10, with ℓ chosen to be $n^{0.1}$ and m to be $n^{1/2(k+1)}$, to construct independent random linear decision trees $T_1, \ldots, T_{n_1}$ such that on any input $\overline{w} = (\text{coeff}_{m,k}(Q_1), \ldots, \text{coeff}_{m,k}(Q_\ell)) \in \mathbb{R}^{r \cdot \ell}$ (where Q_is are polynomials of degree at most k that sign-represent k-PTFs g_i, each on m variables), each T_i computes the number of common satisfying assignments to $g_1, \ldots, g_\ell$ with error at most $1/2$.
2. Run $\mathcal{A}_5(n, d, M, n^{1+\varepsilon_d}, C, \emptyset)$. For an internal call to $\mathcal{A}_1$, say on parameters $(n', s, f_1, \ldots, f_s)$ where $n' \leq m$ and $s \leq \ell$, do the following:

a. Run each T_i on the input $\overline{w} = (\text{coeff}_{n',k}(P_1), \ldots, \text{coeff}_{n',k}(P_s)) \in \mathbb{R}^{r \cdot s}$. (We expand out the coefficient vectors with dummy variables so that they depend on exactly m variables. Similarly, using some dummy polynomials, we can assume that there are exactly ℓ polynomials.)
b. If some T_i outputs the number of common satisfying assignments to $f_1, \ldots, f_s$, then output that. Otherwise, if all T_i output ?, then output ?.

▶ **Lemma 20.** *The construction of the zero-error randomized oracle $\mathcal{A}_1$ in the above algorithm takes $2^{O(n^{0.6})}$ time. Once constructed, the oracle $\mathcal{A}_1$ answers any call (with the correct parameters) in $\text{poly}(n, M)$ time with error at most $1/2^{10n}$.*

Proof.
Correctness. It is clear from Corollary 10 that algorithm $\mathcal{A}_1$ outputs either ? or the correct number of common satisfying assignments to $f_1, \ldots, f_s$. Further, as the T_is in step 1 are constructed independently, it follows that with probability at least $1 - 1/2^{10n}$, the algorithm indeed outputs the number of common satisfying assignments to $f_1, \ldots, f_s$.
Running Time. Substituting the parameters $\ell = n^{0.1}$ and $m = n^{1/(2(k+1))}$ in Corollary 10, we see that the construction of $\mathcal{A}_1$ (step 1) takes $n_1 \cdot 2^{n^{0.6}}$ time. Also, the claimed running time of answering a call follows upon observing that steps 2a and 2b combined take only $\text{poly}(n, M)$ time to execute. ◀

With the correctness of $\mathcal{A}_1$ now firmly established, we finally argue the correctness and running time of algorithm $\mathcal{B}$.

Correctness. The correctness of $\mathcal{B}$ follows from that of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$, and $\mathcal{A}_5$ (see Lemma 20, Theorem 14, Lemmas 15, 17, and 18 respectively). Furthermore, if the algorithm $\mathcal{A}_1$ is assumed to have no error at all, then from the analysis of $\mathcal{A}_5$, we see that the probability of error in $\mathcal{B}$ is at most $1/2$. However, as algorithm $\mathcal{A}_1$ is itself randomized, we still need to bound the probability that any of the calls made to $\mathcal{A}_1$ produce an undesirable output (i.e. an output of ?). To this end, first note that as the running time of $\mathcal{A}_5$ is bounded by 2^n, the number of calls to $\mathcal{A}_1$ is also bounded by 2^n. But by Theorem 14 and Lemma 20, the probability of $\mathcal{A}_1$ outputting ? is bounded by $1/2^{10n}$. Therefore, by the union bound, algorithm $\mathcal{B}$ correctly outputs the number of satisfying assignments to the input circuit C with probability at least $1/2 - 1/2^{\Omega(n)} \geq 1/4$.

Running Time. By Lemma 18 and 20, the running time of $\mathcal{B}$ will be $2^{O(n^{0.6})} + \text{poly}(n, M) \cdot 2^{n - n^{\zeta\varepsilon_d/2(k+1)}}$. Thus, the final running time is $\text{poly}(n, M) \cdot 2^{n-S}$ where $S = n^{\zeta\varepsilon_d/2(k+1)}$ and where $\varepsilon_d > 0$ is a constant depending only on k and d. Setting $\varepsilon_{k,d} = \zeta\varepsilon_d/2(k+1)$ gives the statement of Theorem 6.

References

1 Amir Abboud and Karl Bringmann. Tighter Connections Between Formula-SAT and Shaving Logs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 8:1–8:18, 2018.
2 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388, 2016.

3 Amir Abboud and Aviad Rubinstein. Fast and Deterministic Constant Factor Approximation Algorithms for LCS Imply New Circuit Lower Bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 35:1–35:14, 2018.

4 Paul Beame, Stephen A. Cook, and H. James Hoover. Log Depth Circuits for Division and Related Problems. *SIAM J. Comput.*, 15(4):994–1003, 1986.

5 Lijie Chen. Toward Super-Polynomial Size Lower Bounds for Depth-Two Threshold Circuits. *CoRR*, abs/1805.10698, 2018. URL: `http://arxiv.org/abs/1805.10698`.

6 Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse MAX-SAT and MAX-k-CSP. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 33–45. Springer, 2015.

7 Ruiwen Chen, Rahul Santhanam, and Srikanth Srinivasan. Average-Case Lower Bounds and Satisfiability Algorithms for Small Threshold Circuits. *Theory of Computing*, 14(9):1–55, 2018. `doi:10.4086/toc.2018.v014a009`.

8 Chao-Kong Chow. On the characterization of threshold functions. In *Switching Circuit Theory and Logical Design, 1961. SWCT 1961. Proceedings of the Second Annual Symposium on*, pages 34–38. IEEE, 1961.

9 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002.

10 Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 Integer Linear Programming with a Linear Number of Constraints. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:24, 2014.

11 Russell Impagliazzo, Ramamohan Paturi, and Michael E. Saks. Size-Depth Tradeoffs for Threshold Circuits. *SIAM J. Comput.*, 26(3):693–707, 1997.

12 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 479–488. IEEE, 2013.

13 Valentine Kabanets, Daniel M Kane, and Zhenjian Lu. A polynomial restriction lemma with applications. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 615–628. ACM, 2017.

14 Valentine Kabanets and Zhenjian Lu. Satisfiability and Derandomization for Small Polynomial Threshold Circuits. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 116, 2018.

15 Daniel M Kane, Shachar Lovett, Shay Moran, and Jiapeng Zhang. Active classification with comparison queries. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 355–366. IEEE, 2017.

16 Daniel M. Kane and Ryan Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016.*, pages 633–643, 2016.

17 S Muroga. Threshold logic and its applications. 1971.

18 Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

19 Ryan O'Donnell and Rocco A. Servedio. The Chow Parameters Problem. *SIAM J. Comput.*, 40(1):165–199, 2011.

20 Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.

21 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 566–574. IEEE, 1997.

22 Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. Bounded Depth Circuits with Weighted Symmetric Gates: Satisfiability, Lower Bounds and Compression. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58, pages 82:1–82:16, 2016.

23 Rahul Santhanam. Fighting Perebor: New and Improved Algorithms for Formula and QBF Satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010*, pages 183–192, 2010.

24 Srikanth Srinivasan. On improved degree lower bounds for polynomial approximation. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 24. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

25 Ryan Williams. A New Algorithm for Optimal Constraint Satisfaction and Its Implications. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 1227–1237, 2004.

26 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 231–240. ACM, 2010.

27 Ryan Williams. Non-uniform ACC Circuit Lower Bounds. In *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, pages 115–125. IEEE, 2011.

28 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673, 2014.

29 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 194–202. ACM, 2014.

30 Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015.*, pages 17–29, 2015.