Algorithmica (2022) 84:1132–1162
https://doi.org/10.1007/s00453-021-00915-7

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

Swapnam Bajpai[1] · Vaibhav Krishan[1] · Deepanshu Kush[2] · Nutan Limaye[1] · Srikanth Srinivasan[2]

Received: 30 June 2019 / Accepted: 10 December 2021 / Published online: 7 January 2022

Abstract
We show that there is a better-than-brute-force 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 at most $n^{1+\varepsilon}$ many wires. 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 with linearly many wires).We give two different algorithms for the case of a single PTF. The first uses a learning algorithm for learning degree-1 PTFs (or Linear Threshold Functions) using comparison queries due to Kane, Lovett and Moran (STOC 2018), and the second uses a proof of Hofmeister (COCOON 1996) for converting a degree-1 PTF to a depth-two threshold circuit with small weights. We show that both these ideas fit nicely into a memoization approach that yields the #SAT algorithms.

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 $f : \{-1, 1\}^n \to \{-1, 1\}$,[1] 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

[1] We work with the $\{-1, 1\}$ basis for Boolean functions, which is by now standard in the literature. (See for instance [24].) Here -1 stands for True and 1 stands for False.

Partially supported by Grant MTR/2017/000958 from SERB, Government of India.

Extended author information available on the last page of the article function 1 or not. In this paper, we are interested in computing #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 $|C| \le n^{O(1)}$.

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 $\mathcal{C}$.[2] There has been a large body of work in this area, a small sample of which can be found in [26,27,33,37]. A striking result of this type was proved by Williams [33] 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., [1–3,38]). 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 [4,10], compute conjectured families of pseudorandom functions [23] (and hence are not amenable to *Natural* lower bound proof techniques in the sense defined by Razborov and Rudich [28]) and are at the frontier of our current lower bound techniques [8,21].

It is natural, therefore, to try to come up with circuit-analysis algorithms for threshold circuits. Indeed, there has been a large body of work in the area (reviewed in the Previous Work paragraph later in the Introduction), 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 2. 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 $Q(a) < 0$. Surprisingly, no algorithm is known for this problem that is significantly better than 2^n time.[3]

Note that for a linear polynomial (i.e. $k = 1$), this problem is trivial. For $k = 2$ the problem is already non-trivial. While not noted explicitly in the literature, a better-than-brute-force algorithm for satisfiability of 2-PTFs is implied by the results from [32,35]. However, the stronger counting variant of this problem for 2-PTFs is open as far as we know.

[2] This intuition appears in Section 4 of [34].

[3] An algorithm was claimed for this problem in the work of Sakai, Seto, Tamaki and Teruyama [31]. Unfortunately, the proof of this claim only works when the weights are suitably small. See Footnote 1 on page 4 of [18].

In this paper, we solve the 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 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 $f(a) = \mathrm{sgn}(P(a))$. (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.[4]

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 3 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 $\tilde{O}(n^k)$ [22]. However, note that our algorithms work 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 4 *Fix any constant k. There is a deterministic algorithm that solves the #SAT problem for k-PTFs in time* 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 5 An anonymous ITCS 2019 referee pointed out to us that from two results of Williams [32,35], 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 seem to extend to $k \geq 3$.

Using a different approach, we give another algorithm for the same problem. This result is incomparable to Theorem 4. While the running time is better (and comparable

[4] It is known [22] that such a representation always exists.

to Williams' algorithm mentioned above when $k = 2$) as long as M is subexponential in n, the algorithm is *zero-error randomized*.[5]

Theorem 6 *Fix any constant k. There is a zero-error randomized algorithm that solves the #SAT problem for k-PTFs in time* $\mathrm{poly}(n, M)\cdot 2^{n-S}$ *where* $S = \Omega(n^{1/k})$ *and* (n, M) *are the parameters of the input k-PTF* f.

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 [17] who proved strong average case lower bounds for slightly superlinear-size constant-depth k-PTF circuits. Using these ideas, Kabanets and Lu [18] 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).[6] 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 [18], 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.

Definition 7 (*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.[7]

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

Definition 8 (*#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 #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 also implies a zero-error randomized version of Theorem 4.

[5] A previous version of this paper only gave a bounded-error randomized algorithm for this problem. However, an anonymous reviewer generously showed us how to modify our algorithm to get a a zero-error randomized algorithm.

[6] 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.

[7] 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.

Theorem 9 *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 $\text{poly}(n, M) \cdot 2^{n-S}$, where $S = n^{\varepsilon_{k,d}}$ and (n, s, d, M) are the parameters of the input k-PTF circuit.*

We note that in the Williams [33] framework of proving lower bounds via satisfiability algorithms, zero-error randomized algorithms are as good as deterministic algorithms (as noted already above, even co-non-deterministic algorithms are good enough). However, the above theorem does not imply any new lower bounds, as slightly superlinear-size k-PTF circuits already follows from the work of Kane, Kabanets and Lu [17].

Previous work Satisfiability algorithms for TC^0 have been widely investigated. Impagliazzo, Lovett, Paturi and Schneider [14,16] give algorithms for checking satisfiability of depth-2 threshold circuits with $O(n)$ gates. The former result was improved by Chen and Santhanam [6]. An incomparable result was proved by Williams [36] who obtained algorithms for subexponential-sized circuits from the class $\mathrm{ACC}^0 \circ \mathrm{LTF}$, which is a subclass of subexponential TC^0.[8] 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. [31].[9] 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 [15].) 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 [17]. In a follow-up paper, Kabanets and Lu [18] considered the satisfiability question for k-PTF circuits, and 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).

[8] $\mathrm{ACC}^0 \circ \mathrm{LTF}$ 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.

[9] More specifically, the algorithm of Sakai et al. [31] 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, $M \leq O(n^{1-\Omega(1)})$).

1.1 Proof outline

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

1.1.1 Satisfiability algorithm for k-PTFs

At a high level, we follow the same strategy as Sakai et al. [31]. Their algorithm uses *memoization*, which is a standard and very useful strategy for satisfiability algorithms (see, e.g. [29]). 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 $2^{O(m^{k+1})}$ [5]. 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 $2^{\text{poly}(m)}$ [22], 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. [31]. 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 [5]: 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 [25].)

We show two different ways of circumventing these problems, using two different ideas from the literature.

Using learning theory We use a beautiful recent result of Kane, Lovett and Moran [19], 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 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*.[10]

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 poly$(m, w(P''))$ time. Note, crucially, that the height of the linear decision tree implied by [19] construction is poly(m) and *independent* of the bit-complexity of the coefficients of the polynomial P'' (which may be as big as poly(n) in our algorithm). This concludes the description of the algorithm for k-PTF.

Using circuit complexity A famous result of Goldmann, Håstad and Razborov [9] shows that any linear threshold function (with possibly very large weights) can be simulated by a depth-2 threshold circuit with small weights. A simple proof of this was provided by Hofmeister [11]. The basic idea is to use the Chinese Remainder Theorem to reduce checking integer equalities involving very large integers to checking integer equalities with much smaller numbers (by going modulo small primes).

In our setting, this idea allows us to reduce (via a randomized procedure) the problem to be solved in Step 2 to solving satisfiability for k-PTFs[11] on m variables *with small coefficients*, as long as M is not too large. Since there are not many such PTFs, we can compute and store the answers to all such queries beforehand. This yields the algorithm.

1.1.2 Satisfiability algorithm for *k*-PTF circuits

For k-PTF circuits, we follow a template set up by the result of Kabanets and Lu [18] 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 [18], 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

[10] In the first draft of our work we used the linear decision tree designed by [20], which gave us a zero-error randomized algorithm for #SAT(k-PTF). An anonymous referee of ITCS 2019 suggested that we can replace it with the decision tree from [19] to derandomize our algorithm.

[11] Strictly speaking, we reduce to solving satisfiability for a variant of PTFs called *exact* PTFs but this is not important here.

few inputs. [18] 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 [30]. 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 k-PTF on m variables which is δ-close to an explicit constant.

Base case Here, we make the additional observation (which [18] 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 [18] 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 #SAT for *k*-PTFs: the first algorithm

2.1 A result of Kane, Lovett, and Moran [19]

In this subsection, we formally present the result from [19] which we use in the memoization step of our #SAT algorithm in the following subsection. We begin with the following couple of definitions.

Definition 10 (*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

$P(x_1, \ldots, x_m)$ of degree k 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 $\mathrm{coeff}_{m,k}(P)$ to denote this vector. When m, k are clear from context, we will simply use $\mathrm{coeff}(P)$ instead of $\mathrm{coeff}_{m,k}(P)$.

Definition 11 (*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, or query, 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 query, 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 from [19] will be crucial for us.

Theorem 12 *There is a deterministic algorithm, which on input a positive integer r and a subset $H \subseteq \{-1,1\}^r$, produces a linear decision tree of depth $\Delta = O(r \log^2 r \cdot \log |H|)$ that computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ and 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 the truth table of the LTF defined by w (with constant term 0) on inputs from $H \subseteq \{-1,1\}^r$.*

Moreover, the algorithm runs in time $2^{O(\Delta)}$.

Theorem 1.8 from [19] shows the *existence* of such a deterministic linear decision tree. However, as noted by an anonymous ITCS 2019 reviewer,[12] their proof can in fact be slightly modified to yield an algorithm to construct it in the claimed running time. For completeness, we give a proof in "Appendix A".

We will need a version of Theorem 12 for evaluating (tuples of) k-PTFs. It follows easily from Theorem 12.

Corollary 13 *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 deterministic algorithm which on input positive integers m and $\ell \leq m^c$ computes a function $F : \mathbb{R}^{r\cdot\ell} \to \mathbb{N}$ as follows: given as input any ℓ-tuple of coefficient vectors $\overline{w} = (\mathrm{coeff}_{m,k}(P_1), \ldots, \mathrm{coeff}_{m,k}(P_\ell)) \in \mathbb{R}^{r\cdot\ell}$ such that $P_i(a) \neq 0$ for all $a \in \{-1,1\}^m$, $F(\overline{w})$ is the number of common satisfying assignments to all the k-PTFs on $\{-1,1\}^m$ sign-represented by $P_1, \ldots, P_\ell$. Further, the algorithm runs in time $2^{O(\Delta)}$, where $\Delta = O(\ell \cdot m^{k+1} \log^2 m)$.*

Proof For each $b \in \{-1,1\}^m$, define $\mathrm{eval}_b \in \{-1,1\}^r$ 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 $\{\mathrm{eval}_b \mid b \in \{-1,1\}^m\}$. Clearly, $|H| \leq 2^m$. Further, note that given any polynomial $P(x_1, \ldots, x_m)$ of degree at most

[12] A preliminary version of this paper only claimed a zero-error randomized algorithm for the construction of such a linear decision tree, which is immediate from the work of [20].

k, the truth table of the k-PTF sign-represented by P is given by the evaluation of the LTF represented by $\mathrm{coeff}(P)$ at the points in H. Our aim, therefore, is to evaluate the LTFs corresponding to $\mathrm{coeff}(P_1), \ldots, \mathrm{coeff}(P_\ell)$ at all the points in H.

For each i, we use the deterministic algorithm from Theorem 12 to produce a decision tree $\mathcal{T}_i$ 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 $\mathrm{coeff}(P_i)$ at all points in H). Note that $\mathcal{T}_i$ has depth $O(m^k \log^2 m \cdot \log(2^m)) = O(m^{k+1} \log^2 m)$. The final tree $\mathcal{T}$ is obtained by simply running $\mathcal{T}_1, \ldots, \mathcal{T}_\ell$ in order, which is of depth $O(\ell \cdot m^{k+1} \log^2 m)$. Observe that the tree $\mathcal{T}$ outputs the number of common satisfying assignments to all the f_i.

The claim about the running time follows from the analogous claim in Theorem 12 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. □

2.2 The #SAT algorithm

We are now ready to prove Theorem 4. We first state the algorithm, which follows a standard memoization idea (see, e.g. [29]). 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) (recall from Definition 2 that $M = w(P)$ is the bit-complexity of the sum of the absolute values of all the coefficients of P). Set $m = n^{1/(k+1)}/\log n$.
Algorithm $\mathcal{A}$

1. Use the algorithm from Corollary 13 with $\ell = 1$ to construct a deterministic linear decision tree T such that on any input polynomial $Q(x_1, \ldots, x_m)$ (or more precisely $\mathrm{coeff}_{m,k}(Q)$) of degree at most k that sign-represents a k-PTF g on m variables, T computes the number of satisfying assignments to g.
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 T on $\mathrm{coeff}(P_\sigma)$ and compute its output N_σ, the number of satisfying assignments to P_σ. Add this to the current value of N.
4. Output N.

Correctness It is clear from Corollary 13 (invoked for $\ell = 1$) and step 3b that algorithm $\mathcal{A}$ outputs the correct number of satisfying assignments to f.

Running time We show that the running time of algorithm $\mathcal{A}$ is $\mathrm{poly}(n, M) \cdot 2^{n-m}$. First note that by Corollary 13, the construction of a linear decision tree T takes $2^{O(\Gamma)}$ time, where $\Gamma = m^{k+1} \log^2 m$, and hence, step 1 takes $2^{O(\Gamma)}$ 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 $\mathrm{coeff}(P_\sigma)$ takes only $\mathrm{poly}(n, M)$ time. Recall that the depth of T is $O(\Gamma)$ and thus, on input vector $\mathrm{coeff}(P_\sigma)$, each of whose entries has bit complexity at most M, it takes time $O(\Gamma) \cdot \mathrm{poly}(M, n)$ to run T and obtain the output N_σ. Therefore, step 3 takes $\mathrm{poly}(n, M) \cdot 2^{n-m}$ time. Finally, the claim about the total running time of algorithm $\mathcal{A}$ follows at once when we observe that for the setting $m = (n/\log^3 n)^{1/k+1}$, $\Gamma = O(n/\log n) = o(n)$.

3 #SAT for *k*-PTFs: the second algorithm

Here, we present an alternate approach to #SAT for k-PTFs. This approach uses memoization as before, but the idea now will be to first reduce the size of the coefficients, by going modulo small primes. A major hurdle with this is that PTFs use inequalities, which do not gel well with this operation. Hence, we will first transform our PTFs into a similar model which uses equalities, namely *Exact Polynomial Threshold Functions*.

Definition 14 (*Exact Polynomial Threshold Functions* [12,13]) A Boolean function $E : \{-1, 1\}^n \to \{-1, 1\}$ is called an *Exact Polynomial Threshold Function* of degree k, or a k-EPTF, if there exists a multilinear polynomial $P \in \mathbb{R}[x_1, \dots, x_n]$ of degree k such that for all $a \in \{-1, 1\}^n$, $E(a) = -1$ if and only if $P(a) = 0$. We refer to such a P as a representation of E. When $k = 1$, we call E an Exact Threshold Function, or ETF for short.

The main idea in this algorithm is to first convert the given PTF to a disjoint OR of EPTFs. To do this, we follow Hofmeister [11], who showed how to do this for degree one. His proof is constructive and can easily be adapted to higher degrees.

Lemma 15 (Implicit in [11]) *Let $f : \{-1, 1\}^n \to \{-1, 1\}$ be a k-PTF sign-represented by a polynomial $P \in \mathbb{Z}[x_1, \dots, x_n]$ with parameters (n, M). Then it can be written as,*

$$f = \bigvee_{i=1}^{h} E_i$$

where $h = O(Mn^{2k})$ and each E_i is a k-EPTF that can be represented by a degree k polynomial with weight $O(M + k \log n)$.

Moreover, the OR is a disjoint OR i.e. at most one of the E_is can evaluate to TRUE for a given input.

Finally, this transformation is constructive in the following sense. There is a deterministic algorithm running in $\mathrm{poly}(n, M)$ *time that, on input an integer polynomial $P \in \mathbb{Z}[x_1, \dots, x_n]$ with parameters (n, M) representing f, produces polynomials $P_1, \dots, P_h$ of weight $O(M + k \log n)$ where P_i represents E_i for each $i \in [h]$.*

We include a proof of this lemma in “Appendix B” for completeness.

One of the bottlenecks to a brute force approach to satisfiability directly using this lemma is the size of the coefficients. For this reason, instead of evaluating an EPTF as is, we evaluate it modulo many small primes. We first define this modular version of EPTFs.

Definition 16 (*modular EPTFs*) Let $P \in \mathbb{Z}[x_1, \ldots, x_n]$ be any integer polynomial of degree at most k. For a prime p, we define the Boolean function $E_P^p : \{-1, 1\}^n \to \{-1, 1\}$ such that for all a, $E_P^p(a) = -1$ if and only if $P(a) \equiv 0 \mod p$.
We call such a Boolean function E_P^p a p-modular k-EPTF.

Evaluating a polynomial modulo small enough primes will reduce the size of the coefficients but it will also introduce errors. However, if we evaluate modulo a random prime among sufficiently many primes, the error probability can be shown to be small. The underlying principle is the well-known Chinese Remainder Theorem, which we state below.

Theorem 17 (Chinese Remainder Theorem) *Let $\{p_1, \ldots, p_\ell\}$ be a set of distinct primes, and $a \in \mathbb{Z}$. Then the following are equivalent:*

- *for all* $1 \leq i \leq \ell$, $a \equiv 0 \mod p_i$.
- $a \equiv 0 \mod \prod_{i=1}^{\ell} p_i$.

In particular if $a \neq 0$ and $a \equiv 0 \pmod{p_i}$ for each $i \in [\ell]$, then $\ell \leq \log_2 |a|$.

Now we are ready to describe the algorithm in detail. The input, as earlier, will be 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 taken to be (n, M). We assume that all monomials of the same degree are ordered among themselves using a predetermined ordering. One such ordering is the lexicographic ordering.

Algorithm $\mathcal{MOD}p$:

Set $m = \delta n^{1/k}$ and $A = \beta 2^m$ for a small enough constant δ and a large enough constant β.

1. Using Lemma 15, decompose f as a disjoint OR of k-EPTFs

$$f(X) = \bigvee_{i=1}^{h} E_i(X).$$

   Here $h = O(Mn^{2k}) = \text{poly}(n, M)$ and each E_i is a k-EPTF represented by a degree-k polynomial P_i with weight at most $M' = O(M + \log n)$.
2. We now describe the memoization step. For each prime $p \in [1, M'A \log(M'A)]$, we do the following. For each $i \in [h]$, consider all degree-k integer polynomials in $x_1, \ldots, x_m$ such that the following holds.

- The coefficients of the monomials with degree exactly k are chosen by reducing the corresponding coefficients of the polynomial P_i modulo p (to obtain a non-negative integer less than p).
- The coefficients of the monomials of degree less than k are allowed to be any non-negative integers less than p. Let $\mathcal{P}_{i,p}$ be the set of all such polynomials.

Each polynomial $Q \in \mathcal{P}_{i,p}$ defines a p-modular k-EPTF E_Q^p of m variables. For each such Q, use brute force over all m input variables and count the number of satisfying assignments of E_Q^p. Store the results in a table.

3. Set $N = 0$. (N will ultimately be the number of satisfying assignments to f).
4. For each $\sigma : \{x_{m+1}, \dots, x_n\} \to \{-1, 1\}$, for each $i \in [h]$, do the following. For each $j \in [2n]$, do the following.
   (a) Choose a uniformly random prime $p_j \in [1, M'A \log(M'A)]$.
   (b) Compute $P_{i,\sigma}$, the restriction of P_i given by the partial assignment σ. (Note that $P_{i,\sigma}$ is a polynomial in $x_1, \dots, x_m$. Further, since P_i has degree at most k, the coefficient of any monomial of degree exactly k in $P_{i,\sigma}$ is the same as it is in P_i.)
   (C) Let Q be the polynomial in $\mathcal{P}_{i,p}$ obtained by reducing the coefficients of P_i modulo p_j. Look up the number of satisfying assignments of $E_Q^{p_j}$ in the table constructed in Step 2. Let this be $N_{i,\sigma,j}$.

   Arrange $N_{i,\sigma,1}, \dots, N_{i,\sigma,2n}$ in increasing order and let $N_{i,\sigma}$ be the smallest value. Add $N_{i,\sigma}$ to N.
5. Output N.

We now prove Theorem 6 from the introduction. Note that Theorem 6 is trivial when $M = 2^{\Omega(n^{1/k})}$ since #SAT for k-PTFs can be solved in time poly$(n, M)2^n$ by a trivial brute-force algorithm. Hence, from now on, we assume that $M \leq 2^{\varepsilon n^{1/k}}$ for a suitably small constant ε. The following statement now almost implies Theorem 6, except for the zero-error criterion.

Theorem 18 *The following holds for large enough constant β and small enough constants ε, δ. $\mathcal{MODp}$ is a randomized algorithm, which on input a polynomial P of degree k with parameters (n, M) with $M \leq 2^{\varepsilon n^{1/k}}$, outputs the number of satisfying assignments for $f = \operatorname{sgn}(P)$ with probability $1 - o(1)$. The algorithm runs in time at most $2^{n-\Omega(n^{1/k})}$.*

Proof *Correctness* Recall that f is decomposed as a disjoint OR of k-EPTFs $E_1, \dots, E_h$ represented by polynomials $P_1, \dots, P_h$ in Step 1. For any assignment σ considered in Step 4, we still have that the corresponding relation between the restrictions f_σ and $E_{1,\sigma}, \dots, E_{h,\sigma}$. It suffices to show that in Step 4, for each i and σ, $N_{i,\sigma}$ equals the number of satisfying assignments of $E_{i,\sigma}$ with high probability.

To this end, we claim that for any restriction σ on the last $n - m$ variables and any $i \in [h]$ the following holds.

$$\Pr[N_{i,\sigma} = \text{ number of satisfying assignments of } E_{i,\sigma}] \geq 1 - \frac{1}{4^n} \tag{1}$$

To show this we proceed as follows. Note that for each $j \in [2n]$, $N_{i,\sigma,j}$ is equal to the number of $a \in \{-1, 1\}^m$ such that $P_{i,\sigma}(a) \equiv 0 \pmod{p_j}$. We call p_j a bad prime for a if $P_{i,\sigma}(a)$ is non-zero but $P_{i,\sigma}(a) \equiv 0 \pmod{p_j}$, i.e a is not a satisfying assignment of $E_{i,\sigma}$, but under the modulo operation, it gets counted as a satisfying assignment. Further, we say that p_j is a bad prime if it is bad for some $a \in \{-1, 1\}^m$, i.e. modulo p_j some non-satisfying assignment gets counted as a satisfying assignment. Note that $N_{i,\sigma,j}$ is always at least the number of satisfying assignments of $E_{i,\sigma}$, with equality occurring if p_j is not a bad prime.

We now bound the probability that p_j is a bad prime. Fix any $a \in \{-1,1\}^m$. As P_i has weight at most M', we have $|P_{i,\sigma}(a)| \leq 2^{M'}$. Using Theorem 17, the number of primes that are bad for a is bounded by M'. Hence the total number of bad primes is at most $M'2^m$. On the other hand, the total number of primes in the range $[1, M'A\log(M'A)]$ is at least $\Omega(M'A)$ by the Prime Number theorem. For a large enough constant β, this is at least $4M'2^m$. Thus, the probability that p_j is a bad prime is at most $1/4$.

Since $N_{i,\sigma}$ is the smallest of all the $N_{i,\sigma,j}$ for $j \in [2n]$, we see that $N_{i,\sigma}$ is equal to the number of satisfying assignments of $E_{i,\sigma}$ unless every p_j ($j \in [2n]$) is a bad prime. The probability of this is at most $(1/4)^n$. This proves (1).

By a union bound, the probability that there exist σ and i such that $N_{i,\sigma}$ is not equal to the number of satisfying assignments of $E_{i,\sigma}$ is at most $h2^{n-m}/4^n = O(Mn^{2k}2^{n-m}/4^n) = o(1)$, where we have used the upper bound on M from the statement of the theorem. Hence with probability $1-o(1)$, the algorithm correctly computes the number of satisfying assignments of $E_{i,\sigma}$ for each σ, i. In this case the algorithm correctly returns the number of satisfying assignments of f.

Running time The running time of the algorithm is dominated by the running times of Step 2 and Step 4.

In Step 2, the number of primes in the specified range is $O(M'2^m)$. For a given prime p and $i \in [h]$, the set $\mathcal{P}_{i,p}$ has size at most $(c_1M'2^m)^{m^{k-1}}$ for some positive constant c_1. Hence the number of modular k-EPTFs that are considered is at most $O(hM'2^m) \cdot (c_1M'2^m)^{m^{k-1}} = 2^{O(m^k + m^{k-1}\log M')}$. For each such EPTF, it takes time $2^m \cdot \mathrm{poly}(n, M') = 2^m \cdot \mathrm{poly}(n, M)$ to brute force over all possible assignments. Hence the total time needed to execute this step is $2^{O(m^k + m^{k-1}\log M')} \cdot \mathrm{poly}(n, M) \leq 2^{n/2}$ for our choice of parameters and suitably small ε, δ.

For Step 4, the total running time is $2^{n-m}\,\mathrm{poly}(n, M)$. Hence, for the given choice of parameters, and using $M \leq 2^{\varepsilon n^{1/k}}$ for suitably small ε, the final running time is $2^{n-\Omega(n^{1/k})}$. □

Making the algorithm zero-error The above almost implies Theorem 6 with the only exception being that the algorithm is not zero-error. However, there is a simple and elegant fix for this, as was pointed out to us by a anonymous reviewer.

Note that the above algorithm always returns an estimate that is *at least* the number of satisfying assignments of f, as the only source of error is when an non-satisfying assignment is incorrectly counted as a satisfying assignment at the time of computing $N_{i,\sigma}$ for some $i \in [h], \sigma \in \{-1,1\}^{n-m}$.

So, to get the zero-error algorithm, we proceed as follows. We run the above algorithm on polynomials P and $-P$, which represent Boolean functions f and the negation of f respectively. The algorithm returns estimates N_1 and N_2, where $N_1 \geq N_f$ and $N_2 \geq N_{\neg f}$ and N_g denotes the number of satisfying assignments of g. Note that both inequalities are equalities precisely when $N_1 + N_2 = 2^n$, and this happens with probability $1-o(1)$. Hence, the algorithm simply checks that $N_1 + N_2 = 2^n$ and returns N_1 in this case. Otherwise, the algorithm returns ?.

4 Constant-depth circuits with PTF gates

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

Definition 19 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 20 For a Boolean function $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$, 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 a minority assignment otherwise.

Definition 21 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 [18] 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-close to an explicit constant. We denote this set of gates by B (as we will simplify them by setting them to the constant they are close to).
  – At most n^{β_d} gates are not δ-close to an explicit constant. 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 [18], 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 [7,18], 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 #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 [18].

In what follows we will prove Theorem 9. 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 the 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 about some structure in the circuit (this algorithm is from [18]). 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 PTFs that are δ-close to an explicit constant, 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 case of our algorithm, which will count satisfying assignments for AND of superlinearly 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 $2 \le i \le d$, 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 $\mathrm{poly}(n, M)$ in Sect. 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} which is a complete binary tree of depth $n - n^{1-2\beta}$ such that for a uniformly random leaf $\sigma \in \{-1, 1\}^{n-n^{1-2\beta}}$, the corresponding circuit C_σ is a *good* circuit 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 [18], such a subroutine $\mathcal{A}_2(C, d, n, M)$ was designed. Specifically, they proved the following theorem.

Theorem 22 *(Kabanets and Lu [18]) There is a zero-error randomized algorithm* $\mathcal{A}_2(C, d, n, M)$ *that runs in time* $\mathrm{poly}(n, M) \cdot O(2^{n-n^{1-2\beta}})$ *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* $\mathrm{poly}(n, M)$ *which computes* B_σ *and* G_σ.

Remark 23 In [18], 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 24 In designing the above subroutine in [18], 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.

Further, while the statement of the result in [18] does not guarantee that the decision tree T_{DT} obtained is a complete binary tree, it is easy to see that this follows from their analysis (and the analysis from [7] that is used as a black box).

In this sense, the above theorem statement is a slight restatement of [18, Lemma 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: The set of all $a \in \{-1, 1\}^m$ such that $\exists i \in [\ell]$ for which $P_i(a) > 0$.

Lemma 25 *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.

$\mathcal{A}_3$(m, ‘, , g$_1$, . . . , g‘)

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 $O(\ell).\mathrm{poly}(m, M) = \mathrm{poly}(m, M)$ 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 $2^q \cdot \ell \cdot \mathrm{poly}(m, M)$ time. Therefore, we can bound the total running time by

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

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

Lemma 26 $|\mathcal{T}| \leq \ell \cdot \sqrt{\delta} \cdot 2^{m-q}$.

Proof We define for $i \in [\ell], \mathcal{T}_i = \{\rho \in \{-1, 1\}^{m-q} | g_{i,\rho}$ 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 $\mathrm{poly}(m, M) \cdot \sqrt{\delta} \cdot 2^m$ 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 $\sum_{i \in [\tau]} \text{fan-in}(g_i) \leq n^{1+\varepsilon_1}$.

Oracle access to: $\mathcal{A}_1, \mathcal{A}_2$.

Output: $\#\{a \in \{-1, 1\}^n \mid \forall i \in [\tau], P_i(a) < 0\}$.

4.3.1 The details of the algorithm

```
$\mathcal{A}_4(\mathbf{n}, \mathbf{M}, \mathbf{g_1}, \ldots, \mathbf{g_\o})$

 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 $\mathrm{T_{DT}}$. Initialize $N$ to 0.
 3. For each leaf $\sigma$ of $\mathrm{T_{DT}}$, do the following:
    (A) If $C_\sigma$ is not *good*, count the number of satisfying assignments for $C_\sigma$ by brute-
        force and add to $N$.
    (B) If $C_\sigma$ is *good*, do the following:
        (i) $C_\sigma$ is now an AND of PTFs in $B_\sigma$ and $G_\sigma$, over $n' = n^{1-2\beta_1}$ variables,
            where all PTFs in $B_\sigma$ are $\delta$-close to an explicit constant, where
            $\delta = \exp(-n^{\beta_1/B\cdot k^2})$. Moreover, $|B_\sigma| \le n, |G_\sigma| \le n^{\beta_1}$. 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 $h'_\rho$ 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 assign-
                ments 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 $n^{\beta_1}$ 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 27 $\mathcal{A}_4$ *is a zero-error randomized algorithm that counts the number of satisfying assignments correctly. Further,* $\mathcal{A}_4$ *runs in time* $\mathrm{poly}(n, M) \cdot O(2^{n-n^\alpha})$ *and outputs the right answer with probability at least* $1/2$ *(and outputs ? otherwise).*

Proof *Correctness* For a leaf σ of $\mathrm{T_{DT}}$, 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 $m = n^\alpha \le n^{1/(2(k+1))}$ because of the setting of α and further, we have $n^{\beta_1} \le n^{0.1}$.

Finally, the claim about the error probability follows from the error probability of $\mathcal{A}_2$ (Theorem 22).

Running Time The time taken for constructing T_{DT} is $\mathrm{poly}(n, M) \cdot O(2^{n-n^{1-2\beta_1}})$, by Theorem 22. For a leaf σ of T_{DT}, we know that step (A) is executed with probability at most $2^{-n^{\varepsilon_1}}$. The total time for running step (A) is thus $\mathrm{poly}(n, M) \cdot O(2^{n-n^{\varepsilon_1}})$. We know that the oracle $\mathcal{A}_1$ answers calls in $\mathrm{poly}(n, M)$ time. Hence, the total time for running step (a) is $\mathrm{poly}(n, M) \cdot O(2^{n-n^{\alpha}})$. Next, note that if step (b) 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 (b) is $\mathrm{poly}(n, M) \cdot O(2^{n+n^{\alpha}-n^{\beta_1/Bk^2}})$. Recall that $\zeta = \min(1, A/2Bk^2)$, implying $\alpha = \frac{\zeta\varepsilon_1}{2(k+1)} = \frac{\zeta\beta_1}{2A(k+1)} \leq \frac{\beta_1}{4Bk^2(k+1)} < \frac{\beta_1}{Bk^2}$. Similar to the analysis of step (a), the total time for running step (c) is also $\mathrm{poly}(n, M) \cdot O(2^{n-n^{\alpha}})$. We conclude that the total running time is $\mathrm{poly}(n, M) \cdot O(2^{n-n^{\alpha}})$. 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 $\#\mathrm{SAT}(C, \mathcal{P})$ 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: $\#\mathrm{SAT}(C, \mathcal{P})$.

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.

$\mathcal{A}_5$**(n, d, M, n**$^{\mathbf{1+"_d}}$**, 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}. (If not, output ?.)
3. For each leaf $\sigma \in \{-1, 1\}^{n-n^{1-2\beta_d}}$ of T_{DT}.
   (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` $+ N_\sigma$.
   (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 $\delta = 2^{-n^{\beta_d/Bk^2}}$. 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 $B'_\sigma = \{Q'_1, \dots, Q'_\ell\}$.

(e) Run $\mathcal{A}_3(n^{1-2\beta_d}, \ell, \delta, h'_1, \dots, h'_\ell)$ 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 $(\forall i \in [\tau], P_{i,\sigma}(a) < 0))$, then $\texttt{count} = \texttt{count} + 1$.

(f) Let $G_\sigma = \{f_1, \dots, f_t\}$ be specified by polynomials $R_1, \dots, R_t$. 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 $G'_{\sigma,b} = \{R'_1, \dots, R'_t\}$.
  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 $\texttt{count}$.

4.4.2 The correctness argument and running time analysis

Lemma 28 *The algorithm $\mathcal{A}_5$ described above is a zero-error randomized algorithm which on input $(C, \mathcal{P})$ as described above, correctly solves #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 Sect. 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

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

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 29 $\mathcal{T}(n, d, M) \leq \operatorname{poly}(n, M) \cdot 2^{n - n^{\zeta\varepsilon_d/2(k+1)}}$.

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 $d \geq 2$.

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

The first term above accounts for the running time of $\mathcal{A}_2$ and all steps other than 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 (2) 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 9 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 $f_1, \dots, f_s$ specified by polynomials $P_1, \dots, P_s$ 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 9. 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:

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

1. (*Oracle Construction Step*) Construct the oracle $\mathcal{A}_1$ as follows. Use the algorithm from Corollary 13, with ℓ chosen to be $n^{0.1}$ and m to be $n^{1/2(k+1)}$, to construct a deterministic linear decision tree T such that on any input $\overline{w} = (\text{coeff}_{m,k}(Q_1), \dots, \text{coeff}_{m,k}(Q_\ell)) \in \mathbb{R}^{r\cdot\ell}$ (where Q_is are polynomials of degree at most k that sign-represent k-PTFs g_i, each on m variables), T computes the number of common satisfying assignments to $g_1, \dots, g_\ell$.
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 $(n', s, f_1, \dots, f_s)$ where $n' \leq m$ and $s \leq \ell$, run T on the input $\overline{w} = (\text{coeff}_{n',k}(P_1), \dots, \text{coeff}_{n',k}(P_s)) \in \mathbb{R}^{r\cdot s}$. (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.)

Lemma 30 *The construction of the 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 valid parameters) in* $\text{poly}(n, M)$ *time.*

Proof Substituting the parameters $\ell = n^{0.1}$ and $m = n^{1/(2(k+1))}$ in Corollary 13, we see that the construction of $\mathcal{A}_1$ (step 1) takes $2^{O(n^{0.6}\log^2 n)}$ time. Also, the claimed running time of answering a call follows from the bound on the depth of T given by the proof of Corollary 13. □

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 30, Theorem 22, Lemmas 25, 27, and 28 respectively). From the analysis of $\mathcal{A}_5$, we see that the probability of error in $\mathcal{B}$ is at most $1/2$.

Running Time By Lemma 28 and 30, the running time of $\mathcal{B}$ will be $2^{O(n^{0.6}\log^2 n)} + \text{poly}(n, M) \cdot 2^{n - n^{\zeta\varepsilon_d/2(k+1)}}$. 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 9.

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. Finally, we are very grateful to the anonymous reviewers of Algorithmica for their careful reading of the paper. In particular, we are grateful to an anonymous reviewer who generously told us how to make the algorithm in Theorem 6 a zero-error algorithm.

Funding This material is based upon work supported by Science and Engineering Research Board under the Grant Nos. MTR/2017/000909 and MTR/2017/000958.

A Proof of Theorem 12

We need the following definition from [20].

Definition 31 (*Inference from comparison queries* [20].) For a finite set $S \subseteq \{-1, 1\}^n$ and a vector $w \in \mathbb{R}^n$, let $\text{infer}(S, w)$ denote the set of all points $a \in \{-1, 1\}^n$ such that the sign of $\langle a, w\rangle$ can be inferred from label and comparison queries on S i.e. all queries of the form $\text{sgn}(\langle s, w\rangle)$ and $\text{sgn}(\langle s - t, w\rangle)$ respectively, where $s, t \in S$. Formally, this means that $a \in \text{infer}(S, w)$ if and only if for any $w' \in \mathbb{R}^n$ such that $\text{sgn}(\langle s, w\rangle) = \text{sgn}(\langle s, w'\rangle)$ and $\text{sgn}(\langle s - t, w\rangle) = \text{sgn}(\langle s - t, w'\rangle)$ for all $s, t \in S$, it also holds that $\text{sgn}(\langle a, w\rangle) = \text{sgn}(\langle a, w'\rangle)$. (Here, we also allow $\langle s, w\rangle$ to be 0 in which case we use the standard definition $\text{sgn}(\langle s, w\rangle) = 0$.)

Lemma 5.2 from [19] will be instrumental for us, which we restate (in a slightly weaker form) as follows.

Lemma 32 (Weaker version of Lemma 5.2 from [19]) *Suppose $T \subseteq \{-1, 1\}^r$. Then there exists a subset $S \subseteq T$ of size $|S| = \ell = O(r \log r)$ (which we shall term a 'universal' set for T) such that for every $w \in \mathbb{R}^r$, $|\mathit{infer}(S, w) \cap T| \geq \frac{|T|}{8}$.*

We now begin the proof of Theorem 12.

The proof of existence of the decision tree in [19, Theorem 1.8] now proceeds as follows: the tree consists of levels such that at each level, we have a subset T containing those elements t of H for which $\text{sgn}(\langle t, w\rangle)$ has not yet been inferred. But by Lemma 32, there is a subset $S \subseteq T$ such that by performing label and comparison queries on S, we can infer the signs of $\langle t, w\rangle$ for at least a constant fraction of $|T|$. Assume $S = \{s_1, \ldots, s_\ell\}$. The linear decision tree queries $\langle s_i, w\rangle \geq 0$ to determine $\text{sgn}(\langle s_i, w\rangle)$ for each $i \in [\ell]$, and then performs a comparison-based sorting algorithm to determine the signs of each $\langle s_i - s_j, w\rangle$ for each $i, j \in [\ell]$. Note that this can be done with $O(\ell \log \ell)$ many queries of the form $\langle s_i - s_j, w\rangle \geq 0$ and hence, the queries of the linear decision tree have coefficients from $\{-2, -1, 0, 1, 2\}$ as claimed. By the definition of S, we can now infer $\text{sgn}(\langle t, w\rangle)$ for at least $|T|/8$ many $t \in T$. We then recurse in this manner on the smaller set of elements whose signs are not inferred, which constitutes the next level. After $O(\log |H|)$ many such levels, we have inferred $\text{sgn}(\langle t, w\rangle)$ for each $t \in H$, which means we can output the truth table of the LTF defined by w on inputs from H. This gives us a linear decision tree of depth $O(\ell \log \ell \cdot \log |H|) = O(r \log^2 r \cdot \log |H|)$ as claimed.

The above description of the linear decision tree is constructive given the universal sets S at each level. Therefore, to give an algorithm for constructing the linear decision tree, it suffices to give an algorithm to find these 'universal' sets S at each level. We next define a subroutine $\mathcal{F}$ that on input $T \subseteq \{-1,1\}^r$, outputs a subset S satisfying the conclusion of lemma 32. The basic idea of the subroutine is to cycle through all sets S of size ℓ and for each S, check if it is universal for the set T; by an observation of [20],[13] this latter check can be performed efficiently (in the sense we need) by solving a linear program.

$\mathcal{F}(T)$:

1. Let $\ell = O(r \log r)$ as in Lemma 32. If $|T| < \ell$, simply output T. Otherwise, proceed as follows.
2. For all $S \subseteq T$ of size ℓ,
   (a) Let $S = \{s_1, \ldots, s_\ell\}$.
   (b) For each permutation $\pi : [\ell] \to [\ell]$, $b \in \{0,1\}^{\ell-1}$ and $j \in [\ell+1]$, do the following.
      i Solve a linear program to find a $w \in \mathbb{R}^r$ such that:

- $\langle s_{\pi(1)}, w\rangle \geq \langle s_{\pi(2)}, w\rangle \geq \cdots \geq \langle s_{\pi(\ell)}, w\rangle$,
- $\langle s_{\pi(i)}, w\rangle = \langle s_{\pi(i+1)}, w\rangle$ if and only if $b_i = 1$, and
- $\langle s_{\pi(i)}, w\rangle \geq 0$ if and only if $i < j$.

We call such a w (if it exists) (S, π, b, j)-*feasible*. Further, we say that (π, b, j) is S-*feasible* if such a w exists and S-*infeasible* otherwise. If (π, b, j) is S-infeasible, proceed to the next iteration of this loop.

ii Set $I(S, \pi, b, j) = S$.
iii For each $t \in T \backslash S$, let $\beta = \beta(S, \pi, b, j, t) = \mathrm{sgn}(\langle t, w\rangle) \in \{-1, 0, 1\}$. For each $\gamma \in \{-1, 0, 1\} \backslash \{\beta\}$, solve a linear program to check if there is a w^γ such that w^γ is (S, π, b, j)-feasible and furthermore satisfies $\mathrm{sgn}(\langle t, w^\gamma\rangle) = \gamma$. If there is no such w^γ (for either value of $\gamma \in \{-1, 0, 1\} \backslash \{\beta\}$), then add t to the set $I(S, \pi, b, j)$.

3. Output a set S for which $|I(S, \pi, b, j)| \geq |\frac{|T|}{8}$ for each S-feasible (π, b, j).

Claim 33 *The subroutine $\mathcal{F}$ outputs a set S that is universal for T.*

Proof The main observation is the following. Fix any $S \subseteq T$ of size ℓ and (π, b) as chosen in Step 2(b) above. For any $w, w' \in \mathbb{R}^r$ that are (S, π, b, j)-feasible, we have $\mathrm{infer}(S, w) = \mathrm{infer}(S, w')$ (this follows from the definition of $\mathrm{infer}(S, w)$ above). In particular, the set $I(S, \pi, b, j)$ computed by the algorithm in Step 2(b) is equal to $\mathrm{infer}(S, w)$ for *any* w that is (S, π, b, j)-feasible (and not just the w chosen in Step 2(b)i.).

Now, assume that $S \subseteq T$ of size ℓ is not universal for T. We argue that $\mathcal{F}$ cannot output S. We know there is a $w \in \mathbb{R}^n$ such that $\mathrm{infer}(S, w) < |T|/8$. Fix (π, b) such that w is (S, π, b, j)-feasible. When the algorithm considers this (π, b, j) in Step 2(b), it will find that $I(S, \pi, b, j) = \mathrm{infer}(S, w)$ has size less than $|T|/8$. Hence, this set S will not be output by $\mathcal{F}$.

[13] This is mentioned in a remark [20, Page 363] on the "Computational Complexity" of their procedure. We also thank Daniel Kane (personal communication) for telling us about this.

By a similar argument, it follows that if S is universal, then the algorithm will find that $|I(S,\pi,b)| > |T|/8$ for any (π,b,j) that is S-feasible. Since Lemma 32 guarantees the existence of a universal subset S of size ℓ, the algorithm outputs such a set. $\square$

Armed with the subroutine $\mathcal{F}$, we can now describe the algorithm to construct the linear decision tree for F exactly as in [19]. We give the details here for completeness. The algorithm takes as input $H \subseteq \{-1,1\}^r$ and outputs a linear decision tree $\mathcal{T}$ each of whose leaves is labelled by a function $v : H \to \{-1,1\}$.

$\mathcal{C}(H)$:

1. Run $\mathcal{F}(H)$ and let $S = \{s_1,\ldots,s_\ell\}$ be the universal set found by $\mathcal{F}$. If $S = H$, the output tree $\mathcal{T}$ is simply a bruteforce algorithm that queries $\langle t, w\rangle \geq 0$ for each $t \in H$. Otherwise, proceed as follows.
2. Compute a linear decision tree $\mathcal{T}'$ which makes $O(\ell\log\ell)$ queries of the form $\langle s_i, w\rangle \geq 0$ and $\langle s_i - s_j, w\rangle \geq 0$, and uses a comparison-based sorting algorithm to compute $(\pi,b,j) \in S_\ell \times \{0,1\}^{\ell-1} \times [\ell+1]$ such that w is (S,π,b,j)-feasible.
3. For each leaf λ' of $\mathcal{T}'$, which corresponds to some (π,b,j), do the following.
   (a) Compute $I(S,\pi,b,j)$ as in Step 2(b) of algorithm $\mathcal{F}$ above. Define a function $v_{\lambda'} : I(S,\pi,b,j) \to \{-1,1\}$ by $v_{\lambda'}(t) = \beta(S,\pi,b,j,t)$ where $\beta(S,\pi,b,j,t)$ is as computed in Step 2(b) of algorithm $\mathcal{F}$ above.
   (b) Recursively call $\mathcal{C}(H\backslash I(S,\pi,b,j))$ to obtain a linear decision tree $\mathcal{T}_{\pi,b,j}$ that at each leaf λ'' outputs a $v_{\lambda''} : H\backslash I(S,\pi,b,j) \to \{-1,1\}$.
   (c) Append $\mathcal{T}_{\pi,b,j}$ to $\mathcal{T}$ at leaf λ' and relabel each leaf λ'' of $\mathcal{T}_{\pi,b,j}$ by the function $v_{\lambda'} \cup v_{\lambda'} : H \to \{-1,1\}$.
4. Output the tree $\mathcal{T}$ thus constructed.

Correctness We start by arguing that the output function $v : H \to \{-1,1\}$ correctly computes the LTF defined by w at each point $t \in H$. The proof is by induction on the size of H. The base case, when $|H| < \ell$ is trivial, since $\mathcal{F}(H) = H$ in this case. We now argue the inductive case. In the proof of Claim 33 above, we observed that for any w that is (S,π,b,j)-feasible, the set $I(S,\pi,b,j)$ computed in $\mathcal{F}$ is exactly the set infer(S,w). The same argument also shows that for each $t \in I(S,\pi,b,j)$, $\beta(S,\pi,b,j,t) = \text{sgn}(\langle t,w\rangle)$ for each w that is (S,π,b,j)-feasible. It follows from this that the partial function $v_{\lambda'}$ defined in Step 3(a) of $\mathcal{C}$ computes $\text{sgn}(\langle t,w\rangle)$ correctly for each $t \in I(S,\pi,b,j)$. By induction, the recursive algorithm outputs a linear decision tree that computes $\text{sgn}(\langle t,w\rangle)$ correctly for each $t \in H\backslash I(S,\pi,b,j)$. Putting these together, we see that the output tree $\mathcal{T}$ computes $\text{sgn}(\langle t,w\rangle)$ correctly for all $t \in H$.

The bound claimed in Theorem 12 on the depth Δ of $\mathcal{T}$ follows easily, using the fact that S is universal and hence $|I(S,\pi,b,j)| \geq |T|/8$ for each possible (π,b,j) corresponding to a leaf of $\mathcal{T}'$.

Running Time First let us analyze the running time of the subroutine $\mathcal{F}$ on a set $T \subseteq \{-1,1\}^r$ of size N. There are at most $O(N^\ell)$ subsets of T of size ℓ and at most $O(\ell^\ell)$ triples $(\pi,b,j) \in S_\ell \times \{0,1\}^{\ell-1} \times [\ell+1]$. Further, for fixed S,π,b,j, we run a linear program that runs in time poly(ℓ) for each $t \in T$. Therefore, we can bound the total running time of the subroutine by $\mathsf{T}_{\mathcal{F}}(N) := O(N^{\ell+1} \cdot \ell^\ell \cdot \text{poly}(\ell))$.

The running time of $\mathcal{C}(H)$ can now be bounded by noting that $\mathcal{C}$ calls itself at most $2^{O(\Delta)}$ times and the running time per recursive call can be bounded by $\ell^{O(\ell)} \cdot \text{poly}(\mathsf{T}_{\mathcal{F}}(|H|)) = (|H|\ell)^{O(\ell)}$. This yields a running time of $2^{O(\Delta)} \cdot (|H|\ell)^{O(\ell)} = 2^{O(\Delta)}$.

B Proof of Lemma 15

Say f is represented by integer polynomial P with parameters (n, M). First, note that P can be thought of as a linear function L evaluated at the point $(y_1, \dots, y_m)$ where each y_i represents a monomial over x_is, of degree at most k and thus, $m = \sum_{i=0}^{k} \binom{n}{i} \le n^k$. Let this linear function be $L = \sum_{i=1}^{m} w_i y_i + w_0$. Let $\sum_i |w_i| + |w_0| = W$. Note that $\log_2(W) \le M$.

Now, for any $r \in \mathbb{R}$, define the following operation:

$$\text{trunc}(r) = \begin{cases} \lfloor r/2 \rfloor & r \ge 0 \\ \lceil r/2 \rceil & r < 0. \end{cases}$$

Also define the linear function $\text{half}(L)$ in the following way: $\text{half}(L) = \sum_{i=1}^{m} \text{trunc}(w_i) y_i$. Using this, define the following sequence of linear functions.

$$L^{(0)} = L \text{ and } L^{(\ell+1)} = \text{half}(L^{(\ell)})$$

Note that $L^{(M+1)} \equiv 0$ since all its coefficients are 0. The rounding procedure used for truncating introduces errors. Define the error as follows.

$$\text{err}(L) = \left\lceil \max_{y \in \{-1,1\}^m} \max_{\ell \ge 0} \left| \frac{L^{(\ell)}(y)}{2} - L^{(\ell+1)}(y) \right| \right\rceil$$

Trivially, we have $\text{err}(L) \le m$. Lemma 1 from [11] characterizes the behaviour of the series of linear functions just defined.

Lemma 34 (Lemma 1 from [11]) *Define the interval* $T = [-2 \cdot \text{err}(L), 2 \cdot \text{err}(L)]$. *Then the following holds for all* $y \in \{-1, 1\}^m$.

1. $L^{(\ell)}(y) \in T \implies L^{(\ell+1)}(y) \in T$.
2. $L^{(\ell)}(y) \notin T \implies \text{sgn}(L^{(\ell+1)}(y)) = \text{sgn}(L^{(\ell)}(y))$.

Note that if $L^{(\ell+1)}(y) \in T$ then $L^{(\ell)}(y) \ge -6 \cdot \text{err}(L)$. Thus, for any $y \in \{-1, 1\}^m$, we can write

$$\begin{aligned} \text{sgn}(L(y)) &= \bigvee_{l=0}^{M} \left(L^{(\ell+1)}(y) \in [-2 \cdot \text{err}(P), -1] \wedge L^{(\ell)}(y) \in [-6 \cdot \text{err}(P), -2 \cdot \text{err}(P) - 1] \right) \\ &= \bigvee_{l=0}^{M} \bigvee_{i=-2\cdot\text{err}(P)}^{-1} \bigvee_{j=-6\cdot\text{err}(P)}^{-2\cdot\text{err}(P)-1} \left(L^{(\ell+1)}(y) = i \wedge L^{(\ell)}(y) = j \right) \end{aligned}$$

The above represents the LTF $\operatorname{sgn}(L(y))$ as an OR of ANDs of ETFs. Moreover, the OR is a disjoint OR: this follows from Lemma 34 which implies that there can be at most one $\ell \leq M$ such that $L^{(\ell+1)}(y)$ lies in the interval $[-2\operatorname{err}(L), 2\operatorname{err}(L)]$ and $L^{(\ell)}(y)$ does not. Note that this size of the disjoint OR is at most $(M+1) \cdot O(m^2) = O(Mn^{2k})$.

To finish the proof, it suffices to show that $(L^{(\ell+1)}(y) = i) \wedge (L^{(\ell)}(y) = j)$ can be represented as a single ETF. Define $G(y) = L^{(\ell+1)}(y) - i$ and $H(y) = L^{(\ell)}(y) - j$. The total sum of absolute values of coefficients of both G and H is at most $W' = W + m + 1$, using the trivial bound for $\operatorname{err}(L)$. Consider the ETF $(W'+1)G(y) + H(y) = 0$. This is clearly equivalent to $G(y) = 0 \wedge H(y) = 0$. Also, the bit complexity of this ETF is at most $2\log_2 W' \leq O(M + k\log n)$. Now, each ETF is over the variables y_i, which themselves were monomials over x_i. Thus each ETF can be thought of as a k-EPTF over x_i. This completes the proof.

References

1. Abboud, A., Bringmann, K.: 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, pp. 8:1–8:18 (2018)
2. Abboud, A., Hansen, T.D., Williams, V.V., Williams, R.: 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, pp. 375–388 (2016)
3. Abboud, A., Rubinstein, A.: 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, pp. 35:1–35:14 (2018)
4. Beame, P., Cook, S.A., Hoover, H.J.: Log depth circuits for division and related problems. SIAM J. Comput. **15**(4), 994–1003 (1986)
5. Chow, C-K.: On the characterization of threshold functions. In: Switching Circuit Theory and Logical Design, 1961. SWCT 1961. Proceedings of the Second Annual Symposium on Switching Circuit Theory and Logical Design, pp. 34–38. IEEE (1961)
6. Chen, R., Santhanam, R.: Improved algorithms for sparse max-sat and max-k-csp. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 33–45. Springer, Berlin (2015)
7. Chen, R., Santhanam, R., Srinivasan, S.: Average-case lower bounds and satisfiability algorithms for small threshold circuits. Theory Comput. **14**(1), 1–55 (2018)
8. Chen, L., Williams, R.R.: Stronger connections between circuit analysis and circuit lower bounds, via pcps of proximity. In: 34th Computational Complexity Conference (CCC 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019)
9. Goldmann, M., Håstad, J., Razborov, A.A.: Majority gates VS. general weighted threshold gates. Comput. Complex. **2**, 277–300 (1992)
10. Hesse, W., Allender, E., Barrington, D.A.M.: Uniform constant-depth threshold circuits for division and iterated multiplication. J. Comput. Syst. Sci. **65**(4), 695–716 (2002)
11. Hofmeister, T.: A note on the simulation of exponential threshold weights. In: International Computing and Combinatorics Conference, pp. 136–141. Springer, Berlin (1996)
12. Hansen, K.A., Podolskii, V.V.: Exact threshold circuits. In: Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, USA, June 9–12, 2010, pp. 270–279 (2010)
13. Hansen, K.A., Podolskii, V.V.: Polynomial threshold functions and Boolean threshold circuits. Inf. Comput. **240**, 56–73 (2015)
14. Impagliazzo, R., Lovett, S., Paturi, R., Schneider, S.: 0–1 integer linear programming with a linear number of constraints. Electron. Colloquium Comput. Complex. (ECCC) **21**, 24 (2014)
15. Impagliazzo, R., Paturi, R., Saks, M.E.: Size-depth tradeoffs for threshold circuits. SIAM J. Comput. **26**(3), 693–707 (1997)

16. Impagliazzo, R., Paturi, R., Schneider, S.: A satisfiability algorithm for sparse depth two threshold circuits. In: IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS), 2013, pp. 479–488. IEEE (2013)
17. Kabanets, V., Kane, D.M., Lu, Z.: A polynomial restriction lemma with applications. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp. 615–628. ACM (2017)
18. Kabanets, V., Lu, Z.: Satisfiability and derandomization for small polynomial threshold circuits. In: Eric, B., Klaus, J., José D.P.R., David, S. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20–22, 2018 - Princeton, NJ, USA, volume 116 of LIPIcs, pp. 46:1–46:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
19. Kane, D.M., Lovett, S., Moran, S.: Near-optimal linear decision trees for k-sum and related problems. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25–29, 2018, pp. 554–563 (2018)
20. Kane, D.M., Lovett, S., Moran, S., Zhang, J.: Active classification with comparison queries. In: Chris, U. (eds.) 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15–17, 2017, pp. 355–366. IEEE Computer Society, New York (2017)
21. Kane, D.M., Williams, R.: 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, pp. 633–643 (2016)
22. Muroga, S.: Threshold Logic and Its Applications. Wiley, Hoboken (1971)
23. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM **51**(2), 231–262 (2004)
24. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press, Cambridge (2014)
25. O'Donnell, R., Servedio, R.A.: The chow parameters problem. SIAM J. Comput. **40**(1), 165–199 (2011)
26. Paturi, R., Pudlák, P., Saks, M.E., Zane, F.: An improved exponential-time algorithm for k-sat. J. ACM (JACM) **52**(3), 337–364 (2005)
27. Paturi, R., Pudlák, P., Zane, F.: Satisfiability coding lemma. Chic. J. Theor. Comput. Sci. **1999**, 11 (1999)
28. Razborov, A.A., Rudich, S.: Natural proofs. J. Comput. Syst. Sci. **55**(1), 24–35 (1997)
29. Santhanam, R.: 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, pp. 183–192 (2010)
30. Srinivasan, S.: On improved degree lower bounds for polynomial approximation. In: Anil, S., Nisheeth, K.V. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12–14, 2013, Guwahati, India, volume 24 of LIPIcs, pp. 201–212. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
31. Sakai, T., Seto, K., Tamaki, S., Teruyama, J.: Bounded depth circuits with weighted symmetric gates: satisfiability, lower bounds and compression. In: 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016), vol. 58, pp. 82:1–82:16 (2016)
32. Williams, R.: 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, pp. 1227–1237 (2004)
33. Williams, R.: Improving exhaustive search implies superpolynomial lower bounds. In: Proceedings of the Forty-Second ACM Symposium on Theory of Computing, pp. 231–240. ACM, London (2010)
34. Williams, R.: Guest column: a casual tour around a circuit complexity bound. SIGACT News **42**(3), 54–76 (2011)
35. Williams, R.: 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, pp. 664–673 (2014)
36. Williams, R.: New algorithms and lower bounds for circuits with linear threshold gates. In: Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, pp. 194–202. ACM, New York (2014)
37. Williams, R.: Nonuniform ACC circuit lower bounds. J. ACM **61**(1), 2:1–2:32 (2014)
38. Williams, V.V.: 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., pp. 17–29 (2015)

Authors and Affiliations

Swapnam Bajpai[1] · Vaibhav Krishan[1] · Deepanshu Kush[2] · Nutan Limaye[1] · Srikanth Srinivasan[2]

✉ Nutan Limaye
nutan@cse.iitb.ac.in

Swapnam Bajpai
swapnam@cse.iitb.ac.in

Vaibhav Krishan
vaibhkrishan@iitb.ac.in

Deepanshu Kush
deepkush@iitb.ac.in

Srikanth Srinivasan
srikanth@math.iitb.ac.in

[1] Department of Computer Science and Engineering, IIT Bombay, Mumbai, India

[2] Department of Mathematics, IIT Bombay, Mumbai, India