Reconstruction Algorithms for Low-Rank Tensors and Depth-3 Multilinear Circuits

Vishwas Bhargava
Rutgers University
USA
vishwas1384@gmail.com

Shubhangi Saraf
Rutgers University
USA
shubhangi.saraf@gmail.com

Ilya Volkovich
Boston College
USA
ilya.volkovich@bc.edu

ABSTRACT

We give new and efficient black-box reconstruction algorithms for some classes of depth-3 arithmetic circuits. As a consequence, we obtain the first efficient algorithm for computing the tensor rank and for finding the optimal tensor decomposition as a sum of rank-one tensors when then input is a *constant-rank* tensor. More specifically, we provide efficient learning algorithms that run in randomized polynomial time over general fields and in deterministic polynomial time over $\mathbb{R}$ and $\mathbb{C}$ for the following classes: 1) *Set-multilinear depth-3 circuits of constant top fan-in ($\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits).* As a consequence of our algorithm, we obtain the first polynomial time algorithm for tensor rank computation and optimal tensor decomposition of constant-rank tensors. This result holds for d dimensional tensors for any d, but is interesting even for $d = 3$. 2) *Sums of powers of constantly many linear forms ($\Sigma \wedge \Sigma(k)$ circuits).* As a consequence we obtain the first polynomial-time algorithm for tensor rank computation and optimal tensor decomposition of constant-rank symmetric tensors. 3) *Multilinear depth-3 circuits of constant top fan-in (multilinear $\Sigma\Pi\Sigma(k)$ circuits).* Our algorithm works over all fields of characteristic 0 or large enough characteristic. Prior to our work the only efficient algorithms known were over polynomially-sized finite fields (see. Karnin-Shpilka 09'). Prior to our work, the only polynomial-time or even subexponential-time algorithms known (deterministic or randomized) for subclasses of $\Sigma\Pi\Sigma(k)$ circuits that also work over large/infinite fields were for the setting when the top fan-in k is at most 2 (see Sinha 16' and Sinha 20').

CCS CONCEPTS

• **Theory of computation → Algebraic complexity theory**; **Pseudorandomness and derandomization**; *Circuit complexity*.

KEYWORDS

Tensor Rank, Tensor Decomposition, Derandomization, Arithmetic Circuit Reconstruction,

ACM Reference Format:
Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. 2021. Reconstruction Algorithms for Low-Rank Tensors and Depth-3 Multilinear Circuits. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3406325.3451096

STOC '21, June 21–25, 2021, Virtual, Italy

ACM ISBN 978-1-4503-8053-9/21/06…$15.00
https://doi.org/10.1145/3406325.3451096

1 INTRODUCTION

Arithmetic circuits are directed acyclic graphs (DAG) computing multivariate polynomials succinctly, building up from variables using (+) addition and (×) multiplication operations. *Reconstruction* of arithmetic circuits is the following problem: given black-box (a.k.a oracle/ membership query) access to a polynomial computed by a circuit C of size s from some class of circuits $\mathcal{C}$, give an efficient algorithm (deterministic or randomized) for recovering C or some circuit C' that computes the same polynomial as C. This problem is the algebraic analogue of exact learning in Boolean circuit complexity [5]. If one additionally requires that the output circuit belongs to the same class $\mathcal{C}$ as the input circuit, then it is called *proper learning*.

Reconstruction of arithmetic circuits is an extremely natural problem, but also a really hard problem. Thus in the past few years, much attention has focused on reconstruction algorithms for various interesting subclasses of arithmetic circuits [7, 17, 38, 39]. In particular, much attention has focused on depth-3 and depth-4 arithmetic circuits [9, 23, 30, 55, 56]. Depth-3 and depth-4 circuits have been intensely studied for the problem of proving lower bounds, deterministic polynomial identity testing as well as polynomial reconstruction (which is probably the hardest of the three). Given the depth reduction results of [3, 21, 42, 57], we know that depth-3 and depth-4 arithmetic circuits are very expressive, and good enough reconstruction algorithms (or even lower bounds or polynomial identity testing) for these models would have major implications for general circuits. Thus perhaps not surprisingly, we are quite far from obtaining efficient reconstruction algorithms even for depth-3 circuits.

In this work, we will focus on some interesting subclasses of depth-3 circuits with bounded top fan-in ($\Sigma\Pi\Sigma(k)$ circuits) and give efficient *proper learning* algorithms for them. A setting of particular interest for us (and which motivated much of this work) is when the underlying field is large or infinite (such as $\mathbb{R}$ or $\mathbb{C}$), since in that setting we have even fewer reconstruction algorithms. Though we state many of our results over all fields, for concreteness it will be convenient to imagine the underlying field being $\mathbb{R}$ or $\mathbb{C}$ or $\mathbb{F}_p$.

The subclasses of $\Sigma\Pi\Sigma(k)$ circuits that we study, already capture some very interesting models, and our result for one of these subclasses implies the first efficient polynomial-time algorithm for tensor rank computation and optimal tensor decomposition of *constant-rank tensors*. Before describing the connection to tensors

and stating our results, we first give some background on polynomial reconstruction.

There is substantial evidence supporting the hardness of arithmetic circuit reconstruction. Deterministic algorithms for reconstruction are at least as hard as deterministic black-box algorithms for polynomial identity testing, which is equivalent to proving lowering bounds for general arithmetic circuits [2, 28]. Randomized reconstruction is also believed to be a hard problem and there are a number of results showing hardness of reconstruction under various complexity-theoretic and cryptographic assumptions [18, 25, 40, 52]. (For more details see the section on hardness-results in [9]).

Despite reconstruction being a very hard problem, there has been a lot of research focused on efficient reconstruction for restricted classes of arithmetic circuits. Yet, the progress has still been quite slow. Even among the class of constant-depth arithmetic circuits, we only understand reconstruction well for a handful of restricted cases [9, 23, 30, 39, 55]. If one studies *average case reconstruction* (a model that has received increased attention in recent years) then we know a number of additional results and they hold for richer circuit classes [19, 22, 24, 33–35]. However we will not discuss this setting much since the focus of this work will be on the worst case setting.

Before describing the status of what we know about reconstruction for some of the relevant circuit classes, we first define some natural classes of arithmetic circuits that will play an important role in our discussion.

Some Definitions of Relevant Circuit Classes. The model of depth-3 arithmetic circuits with top fan-in k, which we refer as $\Sigma\Pi\Sigma(k)$ circuits, has three layers of alternating Σ and Π gates and computes a polynomial of the form $C(\bar{x}) = \sum_{i=1}^{k} T_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} l_{ij}(\bar{x})$ where the $l_{ij}(\bar{x})$-s are linear polynomials.

A multilinear polynomial is a polynomial with individual degree of each variable bounded by 1. We say that a circuit C is multilinear (or syntactically multilinear) if every gate in C computes a multilinear polynomial. Thus, a *multilinear* $\Sigma\Pi\Sigma(k)$ circuit is a $\Sigma\Pi\Sigma(k)$ circuit in which each multiplication gate T_i computes a multilinear polynomial.

A more refined subclass of multilinear polynomial is that of *set-multilinear* polynomials. Let $\sqcup_{j\in[d]}X_j$ be a partition of the set X of input variables. Then a polynomial is set-multilinear under partition $\sqcup_{j\in[d]}X_j$ if each monomial of the polynomial picks up *exactly* one variable from each part in the partition.

A *set-multilinear* $\Sigma\Pi\Sigma(k)$ circuit under partition $\sqcup_{j\in[d]}X_j$ (which we denote as $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit) is a $\Sigma\Pi\Sigma(k)$ circuit in which each multiplication gate T_i computes a set-multilinear polynomial respecting the partition $\sqcup_{j\in[d]}X_j$. In the Section 1.1 we will discuss this model and its connection to tensor decomposition.

The final subclass of $\Sigma\Pi\Sigma(k)$ circuits that we discuss is the innocuous looking class of sum of power of k linear forms, also referred to as *diagonal* depth-3 circuits with bounded top fan-in ($\Sigma\wedge\Sigma(k)$ circuits). These are a subclass of $\Sigma\Pi\Sigma(k)$ circuits where instead of using multiplication gates, we are just allowed powering gates which raise an input linear polynomial to some power. In Section 1.1 we will discuss this model and its connection to *symmetric* tensor decomposition.

Proper Learning. The focus of this work will be on *proper learning* algorithms for subclasses of $\Sigma\Pi\Sigma(k)$ circuits.

Note that in the setting of proper learning, if C' is a subclass of C, then an efficient proper learning algorithm for C does not imply an efficient proper learning algorithm for C'. Indeed, as some evidence towards this, note that there are efficient algorithms for proper learning of read-once algebraic branching programs (ROABPs) [7, 17, 38], but we do not know proper learning algorithms for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits and $\Sigma\wedge\Sigma$ circuits (with no bound on the top fan-in), which are both subclasses of ROABPs. In fact, it is known that properly learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits or $\Sigma\wedge\Sigma$ circuits with an optimal bound for the top fan-in is NP-hard [25, 52].

Reconstruction algorithms for $\Sigma\Pi\Sigma(k)$ circuits and for subclasses of $\Sigma\Pi\Sigma(k)$ circuits have been studied in the past a fair bit. The only proper reconstruction algorithms that we are aware of are for the model of multilinear $\Sigma\Pi\Sigma(k)$ circuits by Karnin and Shpilka [30] and for $\Sigma\Pi\Sigma(2)$ circuits by Sinha [55, 56]. In the case of $\Sigma\Pi\Sigma(2)$ circuits, the algorithms are proper (i.e. the output is also a $\Sigma\Pi\Sigma(2)$ circuit) only if the "rank" of the linear forms in the underlying circuit is large enough.

All three of these results are highly nontrivial and they introduce several beautiful techniques which give insight into the structure of these models. The Karnin-Shpilka result is in fact more general and gives reconstruction algorithms for $\Sigma\Pi\Sigma(k)$ circuits without the multilinearity constraint, but in this setting the learning algorithms aren't proper (and they do not work over large fields) and we will not discuss it here. For multilinear $\Sigma\Pi\Sigma(k)$ circuits as well, the running time of the Karnin-Shpilka algorithm has a polynomial dependence on the field size $|\mathbb{F}|$. Thus it works only over polynomially-sized finite fields, and in particular it does not work over large or infinite fields (which is the primary focus of this work). We discuss the algorithm from [30] in a little more detail in Section 1.2.

Our goal is to obtain algorithms that work over infinite fields ($\mathbb{R}$, $\mathbb{C}$) with polynomial dependence on the input bit complexity, and that work over finite fields $\mathbb{F}_q$ with poly($\log q$) dependence on the field size. In this setting, the only subclasses of $\Sigma\Pi\Sigma(k)$ circuits for which we know proper learning algorithms is for $\Sigma\Pi\Sigma(2)$ circuits, if the "rank" of the linear forms in the underlying circuit is large enough [55, 56]. Both these results use fairly sophisticated tools, and really show why even for the seemingly simple case of $k = 2$, reconstruction can be fairly complex.

Some additional classes of bounded depth circuits for which we do know proper learning algorithms that work over large fields are depth-2 ($\Sigma\Pi$) arithmetic circuits (a.k.a sparse polynomials) which have efficient polynomial-time algorithms [8, 39], and multilinear depth-4 circuits with top fan-in 2 (multilinear $\Sigma\Pi\Sigma\Pi(2)$ circuits) [23].

1.1 Connection to the Tensor Rank Problem

Tensors, higher dimensional analogues of matrices, are multi-dimensional arrays with entries from some field $\mathbb{F}$. For instance, a 3-dimensional tensor can be written as $\mathcal{T} = (\alpha_{i,j,k}) \in$

$\mathbb{F}^{n_1 \times n_2 \times n_3}$. We will work with general d-dimensional tensors $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. The *rank* of a tensor $\mathcal{T}$ can be defined as the smallest r for which $\mathcal{T}$ can be written as a sum of r tensors of rank 1, where a rank-1 tensor is a tensor of the form $v_1 \otimes \cdots \otimes v_d$ with $v_i \in \mathbb{F}^{n_i}$. Here $\otimes$ is the Kronecker (outer) product a.k.a *tensor product*. The expression of $\mathcal{T}$ as a sum of such rank-1 tensors, over the field $\mathbb{F}$ is called $\mathbb{F}$-*tensor decomposition* or just tensor decomposition, for short. The notion of tensor rank/decomposition has become a fundamental tool in different branches of modern science with applications in machine learning, statistics, signal processing, computational complexity, psychometrics, linguistics and chemometrics. We refer the reader to the detailed monograph by Landsberg [43] and the references therein for more details on applications of tensor decomposition.

For a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ consider the following polynomial

$$f_{\mathcal{T}}(X) \stackrel{\Delta}{=} \sum_{(j_1,\ldots,j_d) \in [n_1] \times \cdots \times [n_d]} \alpha_{j_1,j_2,\ldots,j_d} x_{1,j_1} x_{2,j_2} \cdots x_{d,j_d}.$$

Let $C(X) = \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$ be a set-multilinear depth-3 circuit over $\mathbb{F}$ respecting the partition $\sqcup_{j \in [d]} X_j$, and computing $f_{\mathcal{T}}(X)$. Then observe that $\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_{i,1}) \otimes \cdots \otimes \bar{v}(\ell_{i,d})$ where $\bar{v}(\ell_{i,j})$ corresponds to the linear form $\ell_{i,j}$ as an n_j-dimensional vector over $\mathbb{F}$. Indeed, it is easy to see that a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ has rank at most r if and only if $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(r)$ circuit. Therefore, rank of $\mathcal{T}$ is the smallest k for which $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit.

Consider the following question. **Question 1:** Given as input a 3-dimensional tensor $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$, is there an efficient algorithm for computing its tensor rank? This problem is known to be NP-hard in general [25]. Now consider the following variant of the question. **Question 1′:** Given as input a 3-dimensional tensor $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$ such that the tensor rank is at most some fixed constant. Does the problem still remain hard, or is the rank efficiently computable? One could also ask these same questions for d-dimensional tensors where d is large. Let $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. In such a setting, one might not even be able to efficiently store the entire tensor as an array. However, if the tensor rank is small (say a constant), then there is still a small "implicit" representation of $\mathcal{T}$ a sum of rank one tensors. In this setting, one has black-box access to measurements of $\mathcal{T}$. In particular, given $\bar{\alpha}_i \in \mathbb{F}^{n_i}$ for all $i \in [d]$, the measurement of $\mathcal{T}$ at $(\bar{\alpha}_1, \ldots, \bar{\alpha}_d)$ equals $\langle \mathcal{T}, \bar{\alpha}_1 \otimes \cdots \otimes \bar{\alpha}_d \rangle$. The d-dimensional question is strictly harder than the three dimensional question, and again we can ask (**d-dimensional analog of Question 1′**)- suppose the tensor rank of $\mathcal{T}$ is at most some fixed constant. Is there an efficient algorithm for computing the tensor rank of $\mathcal{T}$?

Observe that each measurement of $\mathcal{T}$ at $(\bar{\alpha_1}, \ldots, \bar{\alpha_d})$ corresponds to a black-box evaluation of the polynomial $f_{\mathcal{T}}$ at $(\bar{\alpha_1}, \ldots, \bar{\alpha_d})$. Moreover, finding the optimal decomposition of $\mathcal{T}$ as a sum of rank-1 tensors is equivalent to the following: Given black-box access to $f_{\mathcal{T}}$, reconstruct it as a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit with the smallest possible top fan-in.

The three dimensional version was asked as an open question in the work of Schaefer and Stefankovic [51]. In a related setting, a version of the d-dimensional variant (efficiently learning an optimal decomposition of a constant-rank tensor by black-box access to the measurements) was also raised in the recent work of Chen and Meka [13]. It turns out that the answer to the above question is extremely sensitive to the underlying field. For instance, if the underlying field is the rationals ($\mathbb{Q}$), then even if the tensor rank is a constant, computing the exact value of the tensor rank over $\mathbb{Q}$ is not known to be decidable (and is, in fact, believed to be undecidable) [51, 52].

In this paper, we give the first randomized polynomial-time algorithm for computing the tensor rank of a constant-rank, d-dimensional tensor $\mathcal{T}$ [1]. Over the fields $\mathbb{R}$ and $\mathbb{C}$ we also show how to obtain deterministic polynomial time algorithms. Moreover, our algorithm finds the optimal decomposition of $\mathcal{T}$ as a sum of rank-1 tensors. Our algorithm works over fields such as $\mathbb{R}$, large enough finite fields, $\mathbb{C}$, and any other algebraically closed fields. Over other fields, we are only able to compute the tensor rank when we view the entries of the tensor as elements of some extension field.

Theorem 1 (Informal). *Let k be any constant. There exists a randomized polynomial-time algorithm that given black-box access to a polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit over $\mathbb{F}$, and the partition $\sqcup X_i$ of the set of variables X, outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f. When $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$ then our algorithm is deterministic.*

This implies a polynomial-time algorithm to compute the optimal tensor decomposition (and hence also the tensor rank) of constant-rank tensors for various fields. The formal version of the result is given in Theorem 1.1

Our proof uses various ingredients such as a variable reduction procedure, and setting up and solving a system of polynomial equations. Another important ingredient used is the *rank bounds* that were developed in the study of polynomial identity testing for $\Sigma\Pi\Sigma(k)$ circuits [15, 36, 37, 46, 47]. These are structural results for identically zero $\Sigma\Pi\Sigma(k)$ circuits, and essentially show that under some mild conditions, any $\Sigma\Pi\Sigma(k)$ circuit which computes the identically zero polynomial must have its linear forms contained in a "low-dimensional" space. This understanding led to very efficient deterministic polynomial identity testing results for this class, and then eventually were used in efficient reconstruction algorithms for subclasses of $\Sigma\Pi\Sigma(k)$ circuits as well.

Symmetric Tensors: Just as we asked the question of tensor rank computation for general tensors, we can also ask the analogous questions for *symmetric tensors*.

A tensor $\mathcal{T}$ is called symmetric if $X_1 = X_2 = \cdots = X_d$ and we have $\mathcal{T}(i_1, i_2, \ldots, i_d) = \mathcal{T}(j_1, j_2, \ldots, j_d)$ whenever $(i_1, i_2, \ldots, i_d)$ is a permutation of $(j_1, j_2, \ldots, j_d)$. Thus, a symmetric tensor is a

[1] It is possible that the algorithm of Karnin and Shpilka [30] for learning multilinear $\Sigma\Pi\Sigma(k)$ circuits can be adapted to also properly learn set-multilinear $\Sigma\Pi\Sigma(k)$ circuits. The Karnin-Shpilka algorithm has a polynomial dependence on field size $|\mathbb{F}|$. If there algorithm can be adapted then it would give a polynomial-time algorithm over small finite fields. The algorithms in this paper work over infinite fields as well, and that setting was the primary motivation for this work.

higher order generalization of a symmetric matrix. Analogous to tensor rank, *symmetric rank* is obtained when the constituting rank-1 tensors are imposed to be themselves symmetric, that is $\bar{v} \otimes \bar{v} \cdots \otimes \bar{v}$.

Just like in the case of general tensors, computing the symmetric rank reduces to finding the optimal top fan-in of a special class of arithmetic circuits, which is sum of power of linear forms ($\Sigma \wedge \Sigma$) circuits. The class of $\Sigma \wedge \Sigma(k)$ circuits computes polynomials of the form $f = \ell_1^d + \cdots \ell_k^d$ where each ℓ_i is a linear polynomial over the underlying n variables.

Let $C(X) = \sum_{i=1}^{k} \ell_i^d$ be a $\Sigma\wedge\Sigma(k)$ circuit over $\mathbb{F}$ computing $f_{Sym,\mathcal{T}}(X)$ for a symmetric tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. Then $\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_i) \otimes \cdots \otimes \bar{v}(\ell_i)$ where $\bar{v}(\ell_i)$ is a n-dimensional vector corresponding to the linear form $\ell_{i,j}$.

Just as in the case of tensor rank, determining the symmetric rank of tensors is also known to be NP-hard [52]. One could still ask if there are efficient algorithms for determining the symmetric rank when the rank is constant. In this paper, we give (what we believe to be) the first randomized polynomial-time algorithm for computing the symmetric tensor rank of a constant-rank d-dimensional symmetric tensor $\mathcal{T}$.

Theorem 2 (Informal). *Let k be any constant. Let $\mathbb{F}$ be any field of characteristic 0 or sufficiently large characteristic. There exists a randomized polynomial-time algorithm that given black-box access to a polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\wedge\Sigma(k)$ circuit with constant k over $\mathbb{F}$, outputs a $\Sigma \wedge \Sigma(k)$ circuit computing f. When $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$ then our algorithm is deterministic.*

This implies a polynomial-time algorithm to compute the optimal symmetric tensor decomposition (and hence also the symmetric tensor rank) of constant-rank symmetric tensors over various fields. The formal version of the result is given in Theorem 1.4.

Our proof in this case also uses a variable reduction procedure, and setting up and solving a system of polynomial equations. However the proof is overall way simpler than that for general tensors (and actually fits in about half a page!).

1.2 Multilinear $\Sigma\Pi\Sigma(k)$ Circuits

Multilinear $\Sigma\Pi\Sigma(k)$ circuits are a more general class of circuits than $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits. In the proper learning setting however, a proper learning algorithm for multilinear $\Sigma\Pi\Sigma(k)$ circuits does not imply a proper learning algorithm for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.

In this paper we also study reconstruction algorithms for multilinear $\Sigma\Pi\Sigma(k)$ circuit. Multilinear $\Sigma\Pi\Sigma(k)$ circuits were studied by by Karnin and Shpilka [30] and they give the first polynomial-time algorithm for this class of circuits. However the running time of the Karnin-Shpilka algorithm has a polynomial dependence on the field size $|\mathbb{F}|$. Thus it works only over polynomially sized finite fields, and in particular it does not work over infinite fields [2].

At a very high level, the way the algorithm works in [30] is as follows. It finds a suitable projection of the input circuit where only constantly many variables are kept "alive" and the rest are set to field constants. The new circuit in constantly many variables has only constantly many field elements appearing as coefficients, and hence in time poly($|\mathbb{F}|$) one can efficiently "guess" it by going over all possibilities for what the projected circuit looks like. Once the algorithm hits upon the correct guess of the projected circuit, then it "lifts" the projected circuit to recover the original circuit. The implementation of the lifting procedure is quite clever and uses a very nice clustering procedure. The only place where the prohibitive dependence on the field size comes up is in guessing the projected circuit.

In this work we give the first randomized polynomial-time proper learning algorithm for this model that works over large fields (and in particular infinite fields). Our algorithm works over all fields of characteristic 0 or characteristic greater than d (where d is the degree of the circuit). Over $\mathbb{R}$ and $\mathbb{C}$ we show how to derandomize the above algorithm and to obtain *deterministic* polynomial time algorithms. Several of the ideas in our algorithm are inspired by the algorithm from [30] but we need several new ideas as well.

One similarity we have with [30] is that we also project to constantly many variables and try to learn the projected circuit. Instead of "guessing" or iterating to find the projected circuit, we reduce the problem to solving a suitable system of polynomial equations. The problem is that the projected circuit may not have a unique representation as a multilinear $\Sigma\Pi\Sigma(k)$ circuit, and hence the representation learnt by polynomial system solving might be just some representation (not the original representation) and it might not be liftable. This leads to some subtleties and the rest of the algorithm and how we implement the lift is quite different. We give a more detailed overview in Section 2.3.

Theorem 3 (Informal). *Let k be a constant. Let $\mathbb{F}$ be any field of characteristic 0 or sufficiently large characteristic. There exists a randomized polynomial-time algorithm that given black-box access to a polynomial $f \in \mathbb{F}[X]$ computable by a multilinear $\Sigma\Pi\Sigma(k)$ circuit over $\mathbb{F}$, outputs a multilinear $\Sigma\Pi\Sigma(k)$ circuit computing f. Over $\mathbb{R}$ and $\mathbb{C}$ the algorithms we obtain are in deterministic polynomial time.*

This implies a polynomial-time algorithm for learning multilinear $\Sigma\Pi\Sigma(k)$ circuits over infinite fields. The formal version of the result is given in Theorem 1.6.

1.3 Our Results

We now state our results. All our algorithms will be randomized algorithms over general fields, and hence algorithms will output the correctly reconstructed circuit with high (say ≥ 0.9) probability. This probability can boosted to $1-o(1)$ by simply doing independent repetitions. Over $\mathbb{R}$ and $\mathbb{C}$, all our algorithms are deterministic.

Our first main result is a polynomial-time algorithm for proper learning of the class of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.

Theorem 1.1 (Proper learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits). *Given black-box access to a degree d, n variate polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit over $\mathbb{F}$, and given the partition $\sqcup X_i$ of the set of variables X, there is a randomized* $\mathrm{poly}(d^{k^3}, k^{k^{10}}, n, c)$ *time algorithm for computing a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c*

[2] The Karnin-Shpilka [30] result is in fact more general and gives reconstruction algorithms for $\Sigma\Pi\Sigma(k)$ circuits without the multilinearity constraint, but in this setting the learning algorithms aren't proper and we will not discuss it.

is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd \cdot 2^{k+1}$ or algebraically closed, then the output circuit is over $\mathbb{F}$ as well. Otherwise the output circuit is over a degree $\mathrm{poly}(k^{k^{10}})$ *extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time* $\mathrm{poly}(d^{k^3}, k^{k^{10}}, n, c)$.

We would like to remark that this is the first proper learning algorithm for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits, and it works over all fields. We feel this result is particularly interesting in the setting of large or infinite fields such as $\mathbb{R}$ or $\mathbb{C}$, and understanding reconstruction algorithms in that setting was the goal of this work. If we didn't require the learning to be "proper" and were okay with letting the output be a polynomial from a bigger class, then such algorithms were already known (even without the restriction of top fan-in) [7, 38].

By the equivalence described in Section 1.1 we obtain the following immediate corollary to Theorem 1.1 which for constant-rank tensors gives us an efficient tensor decomposition algorithm for expressing the input tensor as sum of rank one tensors.

When $\mathcal{T}$ is a d dimensional tensor, as described in Section 1.1, even storing all of $\mathcal{T}$ as an array is too inefficient. However if the rank is small, there is still a small implicit description of $\mathcal{T}$. We consider the setting when we have black-box access to measurements of $\mathcal{T}$ (as described in Section 1.1). This exactly corresponds to having black-box access to the associated polynomial $f_{\mathcal{T}}$.

Corollary 1.2 (Decomposing fixed rank tensors). *Let $\mathcal{T} \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ be a d-dimensional tensor of rank at most k. Let $n = \sum_{i=1}^{d} n_i$. Given black-box access to measurements of $\mathcal{T}$ (equivalently to evaluations of $f_{\mathcal{T}}$), there exists a randomized* $\mathrm{poly}(d^{k^3}, k^{k^{10}}, n, c)$ *time algorithm for computing a decomposition of $\mathcal{T}$ as a sum of at most k rank 1 tensors, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd \cdot 2^{k+1}$ or algebraically closed, then the decomposition is over $\mathbb{F}$ as well. Otherwise the decomposition will be over (a degree* $\mathrm{poly}(k^{k^{10}})$*) extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time* $\mathrm{poly}(d^{k^3}, k^{k^{10}}, n, c)$.

Notice that we can use the above result to obtain an efficient algorithm for computing the exact value of the tensor rank of the input tensor (at least over $\mathbb{R}$, $\mathbb{C}$, large finite fields and other algebraically closed fields). Over other fields we can only compute the tensor rank over an extension field. The way one can compute the tensor rank is as follows: run the above algorithm for all values of k starting from $k = 1$, and the smallest k for which the algorithm successfully outputs a tensor decomposition will be the tensor rank of $\mathcal{T}$. (Note that one can test when the output is successful by a simple randomized polynomial identity test.)

Remark 1.3. *The dependence on k (exponential tower of size 2) is not optimized in the above theorem and corollary and can be improved to a single exponential in k when $\mathbb{F} = \mathbb{C}, \mathbb{R}$. However, the single exponential dependence on k is expected as tensor decomposition is NP-hard in general [25, 51] and not even known to be computable for $\mathbb{Q}$, thus justifying our need to go to extension fields. See Section 3.4 for more details on hardness of tensor decomposition.*

Note that in the case of constant dimensional tensors (i.e. when one can actually efficiently look at all the entries), we can simulate black-box access to the polynomial $f_{\mathcal{T}}$, given access to the entries of the tensor and vice versa. Thus in the constant dimensional setting our algorithm also gives a way for computing tensor rank and obtaining the optimal tensor decomposition given access to the entries of the tensor. This in particular answers an open question asked by Schaefer and Stefankovic [51], who asked as an open question the complexity of computing the tensor-rank when the rank is constant. Our proof in the constant dimensional setting is simpler than that for the setting of growing d. In the setting of d dimensional tensors (for large or growing d) the question of whether one can get improved efficiency when the rank of $\mathcal{T}$ is constant was raised in the work of Chen and Meka [13] (in a slightly different context). Our work addresses and resolves this question in the black-box query setting for worst case tensors.

Analogous to the result above for tensor decomposition of general tensors, we also obtain efficient algorithms for optimal symmetric tensor decomposition of constant-rank symmetric tensors. The setting of constant-rank symmetric tensors ends up being much simpler than general tensors, and our proofs for this model are much simpler. This result will follow as a corollary of the next result, which is a randomized polynomial-time algorithm for proper learning of $\Sigma \wedge \Sigma(k)$ circuits.

Theorem 1.4. *(Proper learning $\Sigma \wedge \Sigma(k)$ circuits) Given black-box access to a degree d, n variate polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma \wedge \Sigma(k)$ circuit over $\mathbb{F}$, such thar $\mathrm{char}(\mathbb{F}) > d$ or 0, there is a randomized* $\mathrm{poly}\left((dk)^{k^{10}}, n, c\right)$ *time algorithm for computing a $\Sigma \wedge \Sigma(k)$ circuit computing f, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd2^k$ or algebraically closed, then the output circuit is over $\mathbb{F}$ as well. Otherwise the output circuit is over a degree* $\mathrm{poly}((dk)^{k^{10}})$ *extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time* $\mathrm{poly}\left((dk)^{k^{10}}, n, c\right)$.

By the equivalence described in Section 1.1, we obtain the following immediate corollary to Theorem 1.4 which for constant-rank tensors gives us an efficient symmetric tensor decomposition algorithm for expressing the input tensor as sum of rank one symmetric tensors.

Corollary 1.5 (Decomposing fixed symmetric rank tensors). *Let $\mathcal{T}$ be a symmetric d-dimensional tensor of side length n, with $\mathbb{F}$-entries and symmetric rank at most k, such that $\mathrm{char}(\mathbb{F}) > d$ or 0. Given black-box access to $f_{Sym,\mathcal{T}}$, there is a randomized* $\mathrm{poly}\left((dk)^{k^{10}}, n, c\right)$ *time algorithm for computing a decomposition of $\mathcal{T}$ as a sum of at most k rank 1 symmetric tensors, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd2^k$ or algebraically closed, then the decomposition is over $\mathbb{F}$ as well. Otherwise the decomposition will be over (a degree* $\mathrm{poly}\left((dk)^{k^{10}}\right)$ *extension of*

$\mathbb{F}$. *Moreover when* $\mathbb{F}$ *is* $\mathbb{R}$ *or* $\mathbb{C}$*, then we show that the above algorithm can be made to run in deterministic time* $\text{poly}\left((dk)^{k^{k^{10}}}, n, c\right)$.

Again, like in the case of general tensor decomposition, Remark 1.3 holds here as well.

We next state our result on proper learning of multilinear $\Sigma\Pi\Sigma(k)$ circuits.

Theorem 1.6 (Proper learning multilinear-$\Sigma\Pi\Sigma(k)$ circuits). *Given black-box access to a degree* d*,* n *variate polynomial* $f \in \mathbb{F}[X]$ *computable by a multilinear* $\Sigma\Pi\Sigma(k)$ *circuit over* $\mathbb{F}$*, such that* $char(\mathbb{F}) > d$ *or* 0*, there is a randomized* $\text{poly}\left(n^{k^{k^{k^{10}}}}, d, k^{k^{k^{k^{10}}}}, c\right)$ *time algorithm for computing a multilinear* $\Sigma\Pi\Sigma(k)$ *circuit computing* f*, where* $c = \log q$ *if* $\mathbb{F} = \mathbb{F}_q$ *and* c *is the maximum bit complexity of any coefficient of* f *if* $\mathbb{F}$ *is infinite. When the underlying field* $\mathbb{F}$ *is* $\mathbb{R}$ *or* $\mathbb{F}_q$ *with* $q \geq nd \cdot 2^{k+1}$ *or algebraically closed, then the output circuit is over* $\mathbb{F}$ *as well. Otherwise the output circuit is over a degree* $\text{poly}\left((k)^{k^{k^{k^{10}}}}\right)$ *extension of* $\mathbb{F}$*. Moreover when* $\mathbb{F}$ *is* $\mathbb{R}$ *or* $\mathbb{C}$*, then we show that the above algorithm can be made to run in deterministic time* $\text{poly}\left(n^{k^{k^{k^{10}}}}, d, k^{k^{k^{k^{10}}}}, c\right)$.

This is the first efficient proper learning algorithm for multilinear-$\Sigma\Pi\Sigma(k)$ circuits that works over large fields, and in particular infinite fields such as $\mathbb{R}$ and $\mathbb{C}$. Even here, the dependence on k (exponential tower of size 3) is not optimized in the above theorem and can be improved to a tower of size 2 in k when $\mathbb{F} = \mathbb{C}, \mathbb{R},$.

Deterministic vs Randomized Reconstruction Algorithms: The algorithms we give in this paper are randomized over general fields and deterministic over $\mathbb{R}$ and $\mathbb{C}$. Indeed, derandomizing them in general, will be highly nontrivial for the following reason. In the reconstruction problem for all three subclasses of $\Sigma\Pi\Sigma(k)$ circuits being studied, we can embed within them the problem of solving a system of polynomial equations. (See Theorem 3.14 and the discussion in Section 3.4.) The only efficient algorithms we know for solving systems of polynomial equations over large finite fields (i.e with running time polynomial in $\log q$ for a field $\mathbb{F}_q$) are randomized and it is a very interesting open question to derandomize them. A derandomized solution to our reconstruction algorithms over large finite fields would have very interesting algorithmic implications for polynomial system solving, see [1, Problem 15].

Interestingly, the large characteristic case is the only case when low-variate polynomial system solving is hard to derandomize. That is, if the underlying field is not a finite field with large characteristic, then there do exist efficient deterministic algorithms for low-variate polynomial system solving. See Section 3.3.1 for details. Also, this turns out to be the only bottleneck for derandomizing our learning/decomposition algorithms. That is, if the underlying field is not a finite field with large characteristic, then the algorithms underlying Theorems 1.1, 1.4, 1.6 can be derandomized efficiently. Though we do not mention this explicitly, it is easy to see that, when $\mathbb{F} = \mathbb{F}_{p^t}$ then the algorithms mentioned in Theorems 1.1, 1.4, 1.6 can be made deterministic with an additional polynomial in p (characteristic) dependence in time complexity. See derandomization remarks in respective sections for details.

When we present our proofs, for simplicity we will first present the randomized algorithms and then later point out the changes that need to be made in order to derandomize them.

1.4 Related/Previous Work

Reconstruction of $\Sigma\Pi\Sigma(k)$ circuits has received a fair amount of attention. The case of $k = 1$ is resolved by the black-box factoring algorithm of Kaltofen and Trager [29]. The case of $k = 2$ is already highly nontrivial and very interesting and thus needed quite a few new ideas. This case was first studied by Shpilka [53], who designed a reconstruction algorithm for $k = 2$ which was later improved by Karnin and Shpilka [30] who gave efficient reconstruction algorithms for $(\Sigma\Pi\Sigma(k))$ circuits for any constant top fan-in k. When the input is an n-variate, degree d polynomial computed by a size s circuit, both algorithms run in time quasipoly$(n, d, |\mathbb{F}|, s)$. The algorithms are not 'proper learning' algorithms, and the output is from a larger class of "generalized" depth-3 circuit. Moreover given the dependence of the running time on the field size, these algorithms aren't efficient over large/infinite fields.

Over fields of characteristic 0, the only efficient reconstruction algorithm we know for $\Sigma\Pi\Sigma(k)$ circuits is the randomized algorithm by [55] which works for $k = 2$, and uses lots of new ideas such as quantitative/robust Sylvester-Gallai theorems for high dimensional points. Very recently, in [56], Sinha studied the case of $k = 2$ for finite fields and gave the first algorithm in this setting with poly log dependence in field size. These algorithms are mostly proper, but not always. When the rank of the linear forms in the input polynomial is not high dimensional, then the output circuit might not be a $\Sigma\Pi\Sigma(2)$ circuit.

When the input is a *multilinear* $\Sigma\Pi\Sigma(k)$ circuit, the works of Shpilka [53] and Karnin-Shpilka [30] give polynomial-time *proper learning* algorithms. The dependence on the field size is still poly$(|\mathbb{F}|)$, and hence these algorithms do not work over large/infinite fields. Inspired by the work of Karnin and Shpilka, in [9] similar results were obtained for multilinear depth-4 circuits with bounded top fan-in $(\Sigma\Pi\Sigma\Pi(k)$ circuits). The running time is however still at least poly$(|\mathbb{F}|)$, and hence it does not work over large/infinite fields. When the top fan-in is 2, i.e. for $\Sigma\Pi\Sigma\Pi(2)$ circuits, we do know such efficient polynomial-time reconstruction algorithms by the work of Gupta, Kayal and Lokam [23].

Other Results: The class of circuits for which we understand reconstruction really well is the class of depth-2 $(\Sigma\Pi)$ arithmetic circuits (a.k.a sparse polynomials). We can properly learn sparse polynomials in *deterministic* poly(s, n, d) time over any field [8, 39]. Another class for which we understand reconstruction *reasonably* well is the class of read-once oblivious branching programs (ROABPs). Klivans and Shpilka [38] gave a randomized reconstruction (proper learning) algorithm for ran in time poly(n, d, s). This was later derandomized in [17] with time complexity quasipoly(n, d, s). For depth-3 circuits, reconstruction algorithms for various other restricted classes have been studied. For instance, for set-multilinear depth-3 circuits [7, 38] gave a randomized poly(n,d,s) (improper) learning algorithm which outputs an ROABP.

Recently, there has been a flurry of activity in average case learning algorithms for various arithmetic circuit classes [19, 22, 24, 33–35]. These results can be thought of as worst case reconstruction,

given some non-degeneracy condition holds for some implicit polynomials (which are usually computed by intermediate gates). Interestingly, these results fall under the umbrella of learning from natural lower bounds which is an exciting area of research in arithmetic as well as Boolean circuit complexity [12, 35].

2 PROOF OVERVIEW

We have three main results in the paper:(1) reconstruction of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits (equivalent to low rank tensor decomposition), (2) reconstruction of $\Sigma \wedge \Sigma(k)$ circuits (equivalent to low rank symmetric tensor decomposition), and (3) reconstruction of multilinear $\Sigma\Pi\Sigma(k)$ circuits.

Our algorithms are randomized over general fields and we show how to derandomize then over $\mathbb{R}$ and $\mathbb{C}$. For simplicity, in the proof overview we will only discuss the randomized algorithms. Later in the paper when we give the formal proof we will show how to derandomize the algorithms.

A common theme in the proof of each of these results is that all proofs involve a *variable reduction procedure* and setting up and *solving a suitable system of polynomial equations*, where a solution to the system gives some important information about the circuit being reconstructed. In the case of reconstruction for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits and multilinear $\Sigma\Pi\Sigma(k)$ circuits, the proofs are considerably more involved and also use "rank bound" techniques that give structural information about $\Sigma\Pi\Sigma(k)$ circuits that are identically 0.

For simplicity, we start with a proof overview of the result that was (in hindsight) quite easy to prove, which is coming up with an efficient reconstruction algorithm for $\Sigma \wedge \Sigma(k)$ circuits.

2.1 Reconstruction of $\Sigma \wedge \Sigma(k)$ Circuits

Let f be a polynomial which has a $\Sigma \wedge \Sigma(k)$ representation, and let

$$C_f \equiv \sum_{i=1}^{k} (a_{i,1}x_1 + a_{i,2}x_2 + a_{i,3}x_3 + \ldots + a_{i,n}x_n)^d$$

be the $\Sigma \wedge \Sigma(k)$ circuit computing f.

An important observation is that if f can be represented by a $\Sigma \wedge \Sigma(k)$ circuit, then f has only k "essential variables". In particular one can apply an invertible linear transformation to the variables of f so that the transformed f only depends on k variables.

What is nice is that such a linear transformation can actually be computed without actually looking at C_f and its linear forms, but only with black-box access to f. This follows from result of Kayal [32], and which built upon a result by Carlini [11]. (The original result by Kayal was not stated or used in the black-box setting, but it is easy to see that the proof an be adapted to black-box setting as well.) Let $g_A(\bar{x}) = f(A \cdot \bar{x})$, where $g_A(\bar{x})$ depends only on k variables. Since the algorithm can compute A, hence given black-box access to f, it can efficiently simulate black-box access to g_A. Moreover, observe that g_A also has a $\Sigma \wedge \Sigma(k)$ representation. Thus if we can learn a $\Sigma \wedge \Sigma(k)$ representation of g_A, then by simply applying the inverse linear transform, one can recover a $\Sigma \wedge \Sigma(k)$ representation of f.

Thus the new goal is to learn a $\Sigma\wedge\Sigma(k)$ representation of g_A given black-box access to it. We will do this be reducing the problem of learning the $\Sigma\wedge\Sigma(k)$ representation of g_A to solving a suitable system of polynomial equations. Recall that g_A only depends on k variables. Thus the monomial representation of g_A only has $\binom{k+d}{d}$ monomials. Since k is small, this quantity is not too big, and one can invoke black-box reconstruction algorithms for sparse polynomials [8, 39] to learn g_A as a sum of monomials. Let $g_A = \sum_{\bar{e}} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}$ be the monomial representation of g_A.

Let $C_{g_A} \equiv \sum_{i=1}^{k} (b_{i,1}x_1 + b_{i,2}x_2 + b_{i,3}x_3 + \ldots + b_{i,k}x_k)^d$ be a $\Sigma\wedge\Sigma(k)$ representation of g_A.

Then notice that $\sum_{i=1}^{k} (b_{i,1}x_1 + b_{i,2}x_2 + b_{i,3}x_3 + \ldots + b_{i,k}x_k)^d = \sum_{\bar{e}} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}$.

Now for each monomial $\bar{x}^{\bar{e}}$ that appears in g_A, we can compare the coefficient of $\bar{x}^{\bar{e}}$ on both sides of the above expression to get a polynomial equation in the variables $b_{i,j}$. Doing this for all monomials gives us a system of at most $\binom{k+d}{d}$ polynomial equations in k^2 variables, with $b_{i,j}$ as the unknown variables. Observe that any solution to the system of equations would give a $\Sigma \wedge \Sigma(k)$ representation of g_A an vice versa. By Theorem 3.13, this system can be solved in polynomial time if k is a constant.

2.2 Reconstruction of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ Circuits

We now show how to efficiently reconstruct $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits. Again, variable reduction and setting up and solving polynomial systems of equations play an important role, but several other ingredients (such as rank bound techniques) also go into the proof and the proof is more involved.

We are given as input black-box access to a degree d, n variate polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit over $\mathbb{F}$, and we are also given the partition $\sqcup X_i$ of the set of variables X. Let $C_f \equiv \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$, be a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f, where each $\ell_{i,j}$ is a linear polynomial in X_j variables.

2.2.1 Variable Reduction. As a first step, we show how to reduce the number of variables in each part to at most k. Here we cannot directly invoke the result by Kayal [32] and Carlini [11] for the following reasons. The total number of essential variables is $k \times d$ which is quite large. Though the number of essential variables in every part is at most k, there seems to be no straightforward way to apply the result separately to each part[3]. Even if kd was small, after applying the linear transformation given by the Carlini-Kayal result, the new circuit might not be set-multilinear, and we need to crucially maintain set-multilinearity in order for the other steps of the algorithm to be carried out.

Instead, we use the structural properties of set-multilinear circuits to come up with a a different black-box algorithm for performing the variable reduction. We essentially come up with d different invertible linear transformations, one for each set of variables in the partition, that reduces the variables in each set to at most k. In the full version of the paper we elaborate more on how we find these transformations using some properties of the underlying class of circuits. After this step is performed, one can essentially assume that the input circuit is such that each set of the partition has at most k variables.

2.2.2 Reconstructing Low-Degree ($d \leq 2k^2$) $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ Circuits. Once we have the variable reduction established, we proceed along

[3] Since the linear maps might then end up being over the field of rational functions in the remaining variables.

the same lines as the algorithm for reconstructing $\Sigma\wedge\Sigma(k)$ circuits. Since the degree is small, the number of monomials appearing in f is small, and the total number of variables appearing in f is small. (Unlike the symmetric case where the number of monomials was small even for high degree circuits). One can invoke black-box reconstruction algorithms for sparse polynomials [8, 39] to learn f as a sum of monomials. Then, similar to the $\Sigma\wedge\Sigma(k)$ case, we set up a system of polynomial equations in poly(k) variables such that every solution to the system corresponds to a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f.

2.2.3 Reconstructing High-Degree ($d > 2k^2$) $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ Circuits. The high level plan for reconstructing general high degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits is to use induction on k. When $k = 1$, then the algorithm just invokes a black-box factoring algorithm such as [29]. Now assume $k \geq 2$.

Our first step will be to just learn any one linear form appearing in C_f. (Actually as a first step it will be convenient to learn two distinct linear forms such that each multiplication gate contains at most one of them.) In the next step we will use that linear form to learn most of the linear forms of C_f. In the final step we will try to learn all the linear forms and obtain a full $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f.

Learning one (or two) linear forms appearing in C_f: The algorithm chooses k^2 sets of variables in the partition $X = \sqcup X_i$ to keep "alive" and sets the variables in the remaining sets to random values. Let the resulting restricted polynomial be f_R and the resulting constant degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit be C_{f_R}.

Now, we already know reconstruction algorithms (from the previous case) for low degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits which we could invoke. If we could learn C_{f_R}, then in particular we would have learnt several linear forms of C_f. However note that all we have is black-box access to f_R, which might not have a unique $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit representation. In fact it might have exponentially many $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit representations, and our reconstruction algorithm would learn one of these representations. Thus it is possible that we do not learn C_{f_R}, but some other $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit representation of f_R, call it C'_{f_R} . Now a priori it may seem that the linear forms in C'_{f_R} might not have anything in common with the linear forms of C_{f_R} or C_f. However using rank bound arguments that have been used extensively in the past to analyze identically 0 $\Sigma\Pi\Sigma(k)$ circuits (for polynomial identity testing and polynomial reconstruction), one can show two distinct $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representations of the same polynomial must indeed have many linear forms in common (as long as the degree is large enough, which it is in our case). Thus we get that C_{f_R} and C'_{f_R} (which we learnt) must have many linear forms in common. Though we may not know exactly which linear form of C'_{f_R} also appears in C_f, we can come up with a small list of candidate options and then iterate over these options. Any wrong candidate will not lead to a successful output of the final algorithm and we will be able to detect it by a later testing phase. Thus we can effectively assume we know a linear form in C_f. In fact if we do things more carefully we can ensure that we know two linear forms ℓ_1 and ℓ_2 appearing in C_f such that they are supported on the same subset of variables.

Learning most of the linear forms from each multiplication gate of C_f: Once we learn ℓ_1 and ℓ_2 appearing in C_f, we try to learn more linear forms as follows. (We don't need f_R any more or C'_{f_R})

The algorithm applies a suitable random setting of the variables of ℓ_1 in the polynomial f, that makes ℓ_1 evaluate to 0, and results in a circuit with $< k$ multiplication gates. Call the restricted polynomial f_{R_1} and let $C_{f_{R_1}}$ be the restricted version of C_f. By the inductive hypothesis, we can learn a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ representation of f_{R_1}. Call this $C'_{f_{R_1}}$. If we could actually learn the representation $C_{f_{R_1}}$ then we would have learnt most of the linear forms in all the multiplication gates of C_f that did not get set to zero under the restriction. However we can only learn some other representation, which we called $C'_{f_{R_1}}$. Using rank bound arguments, we will however still be able to argue that $C'_{f_{R_1}}$ and $C_{f_{R_1}}$ have a lot in common. In fact we show that each multiplication gate of $C_{f_{R_1}}$ overlaps almost entirely (in all but k linear forms) with some multiplication gate of $C'_{f_{R_1}}$. Repeating this procedure for the other linear form ℓ_2 as well gives us another restricted circuit $C_{f_{R_2}}$ and the version of it that is learnt which is $C'_{f_{R_2}}$. It is now easy to see that each multiplication gate of C_f overlaps almost entirely (in all but k linear forms) with some multiplication gate of $C'_{f_{R_1}}$ or $C'_{f_{R_1}}$.

Once we have this, by iterating over all ways of matching up the multiplication gates and choices of overlap, we can make generate a polynomial sized list of k-tuples $(G_1, G_2, \ldots G_k)$ which has the following property. One of the k-tuples $(G_1, G_2, \ldots G_k)$ from the list will have the property that $f = G_1H_1 + \cdots G_kH_k$ and $G_iH_i = T_i$ where T_i was one of the multiplication gates in the original $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f, C_f. Each G_i has degree $d - k^2$ and hence each H_i has degree k^2. By a little bit of more effort we can also ensure that all the H_i depend on the same sets of the underlying variable partition. The final algorithm will go over all possible k-tuples $(G_1, G_2, \ldots G_k)$ from the list in order to find the correct one. All the wrong ones will not lead to a successful reconstruction, and will get eliminated by a later testing phase.

Learning the full $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f: We now assume that we have learnt k polynomials $G_1, G_2, \ldots, G_k$ such that $f = G_1H_1 + \cdots G_kH_k$. $G_iH_i = T_i$ where T_i was one of the multiplication gates in the original $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f. Each H_i is a polynomial in k^3 variables of degree at most k^2 (since after variable reduction each part had at most k variables) and all the H_i depend on the same sets of the underlying variable partition. We need to now learn the H_i, or even some variation of them which will eventually lead to a full $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f.

We demonstrate how we do this with some simple examples. As a simple case, suppose that the G_i are linearly independent polynomials. By substituting random values into the variables of the G_i, we obtain black-box access to a random linear combination of $H_1, \ldots H_k$. Call this linear combination P_1. From black-box access to P_1, we can actually obtain the monomial representation of P_1 using black-box interpolation for sparse polynomials. We can repeat this process k times to get k different random linear combinations of $H_1, \ldots H_k$. The linear independence of $G_1, G_2, \ldots G_k$ implies

that these random linear combinations will be linearly independent with high probability. Since we know the G_i, we actually know to coefficients of the random linear combinations. Thus once we learn these combinations, we can invert the transformation and actually get black-box access to each H_i individually. Once we have black-box access to each H_i, we can factorize them in a black-box way and hence recover the full underlying circuit.

Here is a slightly more general case. Imagine that $k = 3$, G_1 and G_2 are independent, but $G_3 = G_1 + G_2$. Since we actually know the G_is, we can learn their linear dependency structure (for instance by taking enough random evaluations of them and learning the linear dependence structure of the evaluations. Then,

$$C_f = G_1H_1 + G_2H_2 + (G_1 + G_2)H_3 = G_1(H_1 + H_3) + G_2(H_2 + H_3)$$

Let $H_1 + H_3 = K_1$ and $H_2 + H_3 = K_2$. Now just as in the simple case when all the G_is were independent, we can again learn the monomial representation of two distinct random linear combinations of K_1 and K_2, and then use this to recover the monomial representations of K_1 and K_2. What remains is to find a representation of K_1 which looks like $H_1 + H_3$ and a representation of K_2 which looks like $H_2 + H_3$. Individually, each looks like a case of finding a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(2)$ representation for low degree polynomials, but these two $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(2)$ representations are entangled since they must share a multiplication gate. However we can set up one big system of polynomial equations for solving both these reconstruction problems at the same time that takes into account the shared multiplication gate.

This more general case that we just described contains most of the ideas for the fully general case. For more details, refer to the full version of the paper.

2.3 Reconstruction of Multilinear $\Sigma\Pi\Sigma(k)$ Circuits

We now give a proof overview and describe our algorithm for efficiently learning multilinear $\Sigma\Pi\Sigma(k)$ circuits. The main goal of this result is to find a procedure which also works over large and infinite fields.

Variable reduction and setting up and solving polynomial systems of equations again play an important role, especially for the case of low degree multilinear $\Sigma\Pi\Sigma(k)$ circuits. However the implementation of this technique and how to set up and solve the system of equations is more subtle. For general high degree multilinear $\Sigma\Pi\Sigma(k)$ circuits, we need several other tools such as a clustering procedure (inspired by the work of [30], rank bounds, the notion of rank preserving subspaces, black-box factoring algorithms and an error correcting procedure.

2.3.1 Reconstruction of Low-Degree Multilinear $\Sigma\Pi\Sigma(k)$ Circuits. Think of k (the top fan-in) and d (the degree) to be constants, and the number of variables, n, to be growing. Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial computed by a degree d, multilinear $\Sigma\Pi\Sigma(k)$ circuit C of the form

$$\sum_{i=1}^{k} T_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(\bar{x}) \tag{1}$$

where for each fixed i, the different $\ell_{i,j}$ are supported on disjoint variables.

Let m be the number of essential variables in f. Since there at most kd linear forms appearing in C, it is easy to see that the number of essential variables in f, i.e. m, is at most kd.

We now apply a variable reduction procedure, and for this we invoke the result by Kayal[32] and Carlini [11] to efficiently compute an invertible linear transformation $A \in \mathbb{F}^{n\times n}$ such that $f(A \cdot \bar{x})$ only depends on the first m variables.

Let $g(\bar{x}) = f(A\cdot\bar{x})$. Observe that given black-box access to f, one can easily simulate black-box access to g. Also since $g(A^{-1} \cdot \bar{x}) = f(\bar{x})$, any algorithm that can efficiently learn g can also efficiently learn f in the following way. For each $i \in [n]$, suppose that R_i denote the ith row of A^{-1}. Then in the ith input to g we simply input the linear polynomial $L_i = \langle R_i, \bar{x}\rangle$, which is the inner product of R_i and the vector $\bar{x}$ of formal input variables. Since g only depends on the first m variables, we only really need to do this operation for $i \in [m]$.

Since f is computed by a degree d multilinear $\Sigma\Pi\Sigma(k)$ circuit, hence $g(X) = f(A \cdot \bar{x})$ also has a natural degree d $\Sigma\Pi\Sigma(k)$ circuit representation, where the linear forms of that representation are obtained by applying the transformation A to corresponding linear forms of C. Let us call this circuit C_g. Notice that C_g *may not be multilinear*. However, if were somehow able to learn the precise circuit C_g, then by substituting each variable x_i to L_i then we would recover the circuit C which is indeed multilinear.

Thus our goal is now the following. We have black-box access to g which only depends on m variables. We would like to devise an algorithm for reconstructing C_g. Now here is a slight issue. C_g is a *particular* degree d $\Sigma\Pi\Sigma(k)$ representation of g. It has the nice property that when we plug in $x_i = L_i$ (for all $i \in [m]$) in this representation, then we recover a multilinear $\Sigma\Pi\Sigma(k)$ representation of f. Let us call the new circuit obtained by plugging in $x_i = L_i$ for each i, the "lift" of C_g. Observe that g might have multiple (perhaps exponentially many) representations as a degree d $\Sigma\Pi\Sigma(k)$ circuit. If given black-box access to g, the reconstruction algorithm finds some other degree d $\Sigma\Pi\Sigma(k)$ representation of g, call it C'_g, then there is no guarantee that when we plug in $x_i = L_i$ in this representation, then we recover a multilinear $\Sigma\Pi\Sigma(k)$ representation of f. In other words, the lift of C'_g in general *may not be multilinear.*

Although in our algorithm we will not actually be able to guarantee that we learn precisely C_g, however the existence of C_g tells us that *there exists* a $\Sigma\Pi\Sigma(k)$ representation of g whose lift is a multilinear $\Sigma\Pi\Sigma(k)$ circuit. We will use this existence to actually find a suitable $\Sigma\Pi\Sigma(k)$ representation of g whose lift is multilinear.

In order to learn a degree d $\Sigma\Pi\Sigma(k)$ representation of g we will set up a system of polynomial equations such that any solution to it will give as a degree d $\Sigma\Pi\Sigma(k)$ representation of g. (We do this in a very similar manner to how we did it for $\Sigma\wedge\Sigma(k)$ circuits and $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.) We then show how to impose several additional polynomial constraints to this system that will further ensure that whatever $\Sigma\Pi\Sigma(k)$ representation is learnt will be such that its lift will be a multilinear $\Sigma\Pi\Sigma(k)$ circuit.

2.3.2 Reconstructing General (High-Degree) Multilinear $\Sigma\Pi\Sigma(k)$ Circuits. We now describe our algorithm for reconstructing general multilinear $\Sigma\Pi\Sigma(k)$ circuits. What we describe here is a bit of a simplification and it avoids some technical issues, but we hope that it provides a high level picture of the algorithm.

Clustering of gates: We use a very nice and elegant clustering procedure devised in the work of Karnin and Shpilka [31] (which they used for reconstructing $\Sigma\Pi\Sigma(k)$ circuits over small fields). We will not describe the algorithm here, but describe some nice properties that the clustering satisfies. Given as input $C = \sum_i T_i$ where the T_i are the multiplication gates of a degree d multilinear $\Sigma\Pi\Sigma(k)$ circuit C, the clustering algorithm looks at the T_i and outputs a partition of the the k multiplication gates into a set of clusters $C_1, C_2, \ldots C_r$ (for some $r \in [k]$). Each cluster C_i is some subset of the multiplication gates of C, and has the property that any two multiplication gates in a cluster are very "close" to each other. Suppose that $C_i = \{T_{i_1}, T_{i_2}, T_{i_3}\}$. Then consider the associated circuit $C'_i = T_{i_1} + T_{i_2} + T_{i_3}$. The closeness of every two of the multiplication gates will imply that one can write C'_i as

$$C'_i = T_{i_1} + T_{i_2} + T_{i_3} = \gcd(T_{i_1}, T_{i_2}, T_{i_3}) \times \left(T'_{i_1} + T'_{i_2} + T'_{i_3}\right)$$

where $T'_{i_1} + T'_{i_2} + T'_{i_3}$ is a *low degree* multilinear $\Sigma\Pi\Sigma(3)$ circuit. Now notice that we don't know what C is (that is what we are trying to learn) and hence we cannot apply any clustering procedure to it. However this clustering exists, and it is canonical. We only have black-box access to the original circuit C. Suppose that we could somehow obtain black-box access to each of the clusters (or rather to the circuits corresponding to the clusters). We would then actually be done! Here is why. Suppose we had black-box access to C'_i, then we would first apply a black-box factoring algorithm (such as that given by [29]) to compute all the linear factors of C'_i (thus we would obtain $\gcd(T_{i_1}, T_{i_2}, T_{i_3})$) and divide them out. We would then be left with black-box access to $T'_{i_1} + T'_{i_2} + T'_{i_3}$ is a *low degree* multilinear $\Sigma\Pi\Sigma(3)$ circuit. But we already saw how to reconstruct low degree multilinear $\Sigma\Pi\Sigma(3)$ circuits! By multiplying it with its linear factors, we we would be able to recover a multilinear $\Sigma\Pi\Sigma(3)$ circuit for C_i. We would repeat this procedure for each cluster and then put it all together to obtain a multilinear $\Sigma\Pi\Sigma(k)$ representation for C.

Thus the goal from now on will be to somehow obtain black-box access to the clusters. The clustering output by the clustering algorithm also has some additional nice properties. It is a "robust" clustering, that is, if two multiplication gates got assigned to different clusters, then they are quite "far" from each other (in some well defined sense). This nice property ends up implying the following. We start with the circuit C in n variables. Then there is some constant number (about k^k) of variables one can keep "alive" (call these the $\bar{y}$ variables) such that if we set the remaining variables (call these the $\bar{z}$ variables) to random values ($\bar{z} = \bar{\alpha}$), then the new restricted circuit $C|_{\bar{z}=\bar{\alpha}}$ has the following property. Suppose we applied the clustering algorithm to $C|_{\bar{z}=\bar{\alpha}}$, then the clusters obtained would exactly match up with the clusters output by the clustering algorithm applied to the circuit C, and each cluster of $C|_{\bar{z}=\bar{\alpha}}$ would be obtained by the same restriction procedure being applied to the corresponding cluster of C.

Obtaining access to evaluations of the clusters at random inputs: Notice that though we do not know what C is, we can know what $C|_{\bar{z}=\bar{\alpha}}$ is. This is because $C|_{\bar{z}=\bar{\alpha}}$ has only about k^k variables and hence is a low degree multilinear $\Sigma\Pi\Sigma(k)$ circuit. Hence we can reconstruct it. We have to be a bit careful here since our reconstruction algorithm might not output the precise circuit $C|_{\bar{z}=\bar{\alpha}}$ but some other multilinear $\Sigma\Pi\Sigma(k)$ circuit representation of the same polynomial, call it $C'|_{\bar{z}=\bar{\alpha}}$. However the clustering procedure turns out to be robust enough that the clusters of $C|_{\bar{z}=\bar{\alpha}}$ and the clusters of $C'|_{\bar{z}=\bar{\alpha}}$ match up to compute the same polynomials. Hence we can essentially assume that we know what $C|_{\bar{z}=\bar{\alpha}}$ is and hence we can cluster its gates as well. By the properties of clustering, the clusters of $C|_{\bar{z}=\bar{\alpha}}$ match up with the clusters of C (after we set the $\bar{z} = \bar{\alpha}$). Thus though we do not as yet have black-box access to the clusters of C, we can indeed recover what the clusters look like after setting $\bar{z} = \bar{\alpha}$. Thus if $C'_1, C'_2, \ldots C'_r$ are circuits corresponding to the clusters of C, then we can recover their restrictions to $\bar{z} = \bar{\alpha}$. Notice that α was any random sample from $\mathbb{F}^m$, where m is the number of Z variables. Thus we can essentially recover black-box evaluations of the clusters at *randomly chosen inputs*. If we could do the same for the Z variables being set to any arbitrary adversarially chosen $\beta \in \mathbb{F}^m$ then we would be done.

There is one issue we have swept under the rug, which is the following. The clusters of $C|_{\bar{z}=\bar{\alpha}}$ match up with the clusters of C, but *we don't know what this matching is*. In particular, we might be able to learn $C|_{\bar{z}=\bar{\alpha}}$ as well as $C|_{\bar{z}=\bar{\alpha}'}$ for two distinct $\bar{\alpha}, \bar{\alpha}' \in \mathbb{F}^m$, and we might be able to cluster both of them, and these clusters correspond to the clusters of C, but since we don't know the correspondence we cannot really say that we know the value of $C'_i|\bar{z} = \bar{\alpha}$ as well as $C'_i|_{\bar{z}=\bar{\alpha}'}$ for the same C'_i. We will refer to this as "ambiguity issue".

Obtaining the corresponding between two clusterings: We now address the ambiguity issue. Suppose we know what $C'_i|_{\bar{z}=\bar{\alpha}}$ looks like. We would like to be able to compute $C'_i|_{\bar{z}=\bar{\alpha}'}$ for any other randomly chosen $\alpha' \in \mathbb{F}^m$. Note that we can reconstruct $C|\bar{z} = \bar{\alpha}'$ and cluster it and that would give us the set $\{C'_1|_{\bar{z}=\bar{\alpha}'}, C'_1|_{\bar{z}=\bar{\alpha}'}, \ldots C'_r|_{\bar{z}=\bar{\alpha}'}\}$, but we may not know which element of the set corresponds to $C'_i|_{\bar{z}=\bar{\alpha}'}$. In order to do this identification, we first show how to do this when $\bar{\alpha}$ and $\bar{\alpha}'$ differ in only one coordinate, and then we use a hybrid argument to stitch it together for general $\bar{\alpha}$ and $\bar{\alpha}'$ (by considering a sequence of different αs going from α to α' and with consecutive elements differing in one coordinate). When $\bar{\alpha}$ and $\bar{\alpha}'$ differ in only one coordinate, we observe that $C'_i|_{\bar{z}=\bar{\alpha}}$ and $C'_i|_{\bar{z}=\bar{\alpha}'}$ are very similar or very "near each other" in a suitably defined metric. Then using the robustness property of the clustering we show that the identification of $C'_i|_{\bar{z}=\bar{\alpha}'}$ can be done.

From evaluations at random points to evaluations at worst case points: Let C'_i be the circuit corresponding to cluster C_i. Let us assume we know how to compute $C'_i|_{\bar{z}=\bar{\alpha}}$ for any randomly chosen $\alpha \in \mathbb{F}^m$. Now let $\bar{\beta}$ be some arbitrary point in $\mathbb{F}^m$. We would like to compute $C'_i|_{\bar{z}=\bar{\beta}}$. We use Reed-Solomon decoding for this. We consider the line $t \cdot \bar{\alpha} + (1-t) \cdot \bar{\beta}$ passing through $\bar{\alpha}$ and $\bar{\beta}$ in $\mathbb{F}^m$. In order to learn $C'_i|_{\bar{z}=\bar{\beta}}$, we will learn the restriction of C'_i to the full line, which is a polynomial in the Y variables and the additional t variable. Then setting $t = 0$ would give us the value at β. To learn the restriction to the line, it suffices to learn the restriction on at least $d+1$ points on the line, where d is the degree of the t variable. By evaluating at $d+1$ random points (which can be done since these points look random) on the line, we can accomplish this.

Derandomization: The result can be derandomized over $\mathbb{R}$ and $\mathbb{C}$. For more details, see the full version of the paper.

3 DEPTH-3 CIRCUITS

In this section we formally introduce the general model of depth-3 circuits and specialization of set-multilinear depth-3 circuits, which is the focus of our paper. It is to be noted that depth-3 circuits were a subject for a long line of study [6, 15, 31, 36, 37, 48–50, 54].

Definition 3.1. *A depth-3 $\Sigma\Pi\Sigma(k)$ circuit C computes a polynomial of the form*

$$C(X) = \sum_{i=1}^{k} T_i(X) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(X),$$

where the $\ell_{i,j}$-s are linear functions; $\ell_{i,j}(X) = \sum_{t=1}^{n} a_{i,j}^{t} x_t + a_{i,j}^{0}$ with $a_{i,j}^{t} \in \mathbb{F}$.

A multilinear $\Sigma\Pi\Sigma(k)$ *circuit is a $\Sigma\Pi\Sigma(k)$ circuit in which each T_i is a multilinear polynomial. In particular, each such T_i is a product of variable-disjoint linear functions.*

Given a partition $X = \sqcup_{j\in[d]} X_j$ of X, a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ *circuit is a further specialization of a multilinear circuit to the case when each $\ell_{i,j}$ is a linear* form *in $\mathbb{F}[X_j]$. That is, each $\ell_{i,j}$ is defined over the variables in X_j and $a_{i,j}^{0} = 0$.*

We say that C is minimal *if no subset of the multiplication gates sums to zero. We define* $\gcd(C)$ *as the linear product of all the non-constant linear functions that belong to all the T_i-s. I.e.* $\gcd(C) = \gcd(T_1, \ldots, T_k)$. *We say that C is* simple *if* $\gcd(C) = 1$. *The simplification of C, denoted by* $\mathrm{sim}(C)$, *is defined as $C/\gcd(C)$. In other words, the circuit resulting upon the removal of all the linear functions that appears in* $\gcd(C)$. *Finally, we say that a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit has* width w, *if $|X_j| \le w$ for all j.*

Throughout the paper, we will be referring to this quantity as the *width* of a polynomial, width of a circuit, since our model is is $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits, it all essentially means the same.

3.1 Tensors and Set-Multilinear Depth-3 Circuits

Tensors, higher dimensional analogues of matrices, are multi-dimensional arrays with entries from some field $\mathbb{F}$. For instance, a 3-dimensional tensor can be written as $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$ and 2-dimensional tensors simply corresponds to traditional matrices. We will work with general d-dimensional tensors $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$, here $[n_1] \times \cdots \times [n_d]$ refers to the shape of the tensor and n_i as length of tensor in i-th dimension. Just like any matrix has a natural definition of rank, there is an analogue for tensors as well.

The rank of a tensor $\mathcal{T}$ can be defined as the smallest r for which $\mathcal{T}$ can be written as a sum of r tensors of rank 1, where a rank-1 tensor is a tensor of the form $v_1 \otimes \cdots \otimes v_d$ with $v_i \in \mathbb{F}^{n_i}$. Here $\otimes$ is the Kronecker (outer) product a.k.a *tensor product*. The expression of $\mathcal{T}$ as a sum of such rank-1 tensors, over the field $\mathbb{F}$ is called $\mathbb{F}$*-tensor decomposition* or just tensor decomposition, for short. The notion of Tensor rank/decomposition has become a fundamental tool in different branches of modern science with applications in statistics, signal processing, complexity of computation, psychometrics, linguistics and chemometrics. We refer the reader to a monograph by Landsberg [43] and the references therein for more details on application of tensor decomposition.

For our application, it would be useful to think of tensors as a restricted form of multilinear polynomials that are called *set-multilinear* polynomials. To this end, let us fix the following notation throughout the paper.

Let $d \in \mathbb{N}$. We will refer to d as the *dimension*. For $j \in [d]$ let $X_j = \{x_{j,1}, x_{j,2}, \ldots, x_{j,n_j}\}$, where $n_j = |X_j|$. Finally, let $X = \sqcup_{j\in[d]} X_j$. That is, $\{X_j\}_{\{j\in[d]\}}$ form a partition of X.

Definition 3.2 (Set-Multilinear polynomial). *A polynomial $P \in \mathbb{F}[X]$ is called* set-multilinear *w.r.t (the partition) X, if every monomial that appears in P is of the form $x_{i_1} x_{i_2} \cdots x_{i_d}$ where $x_{i_j} \in X_j$.*

In other words, each monomial of a set-multilinear polynomial picks up exactly one variable from each part in the partition. These polynomial have been well studied in the past [4, 16, 45] in particular since many natural polynomials like the Determinant, the Permanent, Nisan-Wigderson and others are set-multilinear w.r.t appropriate partitions of variables. Furthermore, each tensor can be regraded as a set-multilinear polynomial.

Definition 3.3. *For a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ consider the following polynomial*

$$f_{\mathcal{T}}(X) \stackrel{\Delta}{=} \sum_{(j_1,\ldots,j_d)\in[n_1]\times\cdots\times[n_d]} \alpha_{j_1,j_2,\ldots,j_d} x_{1,j_1} x_{2,j_2} \cdots x_{d,j_d}.$$

Observe that $f_{\mathcal{T}}(X)$ is a set-multilinear polynomial w.r.t X. More interestingly, there is a direct correspondence between tensor decomposition and computing the polynomial $f_{\mathcal{T}}(X)$ in the model of set-multilinear depth-3 circuits. We first define the model formally.

Definition 3.4 (Set-Multilinear Depth-3 Circuits). *A set-multilinear depth-3 circuit w.r.t to (a partition) X with top fan-in k, denoted by $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ computes a (set-multilinear) polynomial of the form*

$$C(X) \equiv \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}(X_j)$$

where $\ell_{i,j}(X_j)$ is a linear form in $\mathbb{F}[X_j]$.

To gain some intuition, suppose that $f_{\mathcal{T}}(X) = \ell_{i,1}(X_1) \cdot \ell_{i,2}(X_2) \cdots \ell_{i,d}(X_d)$ for some tensor $\mathcal{T}$. We can observe that in this case $\mathcal{T}$ is a rank-1 tensor. Extending this observation, the following provides a formal connection between tensor decomposition and computing the polynomial $f_{\mathcal{T}}(X)$ by set-multilinear depth-3 circuits.

Observation 3.5. *Let $C(X) = \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$ be a set-multilinear depth-3 circuit over $\mathbb{F}$ computing $f_{\mathcal{T}}(X)$ for a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. Then*

$$\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_{i,1}) \otimes \cdots \otimes \bar{v}(\ell_{i,d})$$

where $\bar{v}(\ell_{i,j})$ corresponds to the linear form $\ell_{i,j}$ as an n_j-dimensional vector over $\mathbb{F}$.

Note that this connection is, in fact, a correspondence: any $\mathbb{F}$-tensor decomposition of $\mathcal{T}$ gives a circuit over $\mathbb{F}$. This leads to the following important lemma:

Lemma 3.6. *A tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$ has rank at most r if and only if $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_X(r)$ circuit. Therefore, rank of $\mathcal{T}$ is the smallest k for which $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_X(k)$ circuit.*

Proof. The proof is straightforward. Note that, $\ell_{i,1}(X_1)\cdot\ell_{i,2}(X_2)\cdots\ell_{i,d}(X_d)$ exactly corresponds to a rank-1 tensors. Thus, C_f gives a rank k $\mathbb{F}$-tensor decomposition of $\mathcal{T}$ and any $\mathbb{F}$-tensor decomposition gives a circuit over $\mathbb{F}$. □

3.2 Symmetric Tensors and Sums of Powers of Linear Forms

A tensor $\mathcal{T}$ is called symmetric if $X = X_1 = X_2 = \cdots = X_d$ and we have $\mathcal{T}(i_1, i_2, \ldots, i_d) = \mathcal{T}(j_1, j_2, \ldots, j_d)$ whenever $(i_1, i_2, \ldots, i_d)$ is a permutation of $(j_1, j_2, \ldots, j_d)$. Thus, a symmetric tensor is a higher order generalization of a symmetric matrix. Analogous to tensor rank, *symmetric rank* is obtained when the constituting rank-1 tensors are imposed to be themselves symmetric, that is $\bar{v}\otimes\bar{v}\cdots\otimes\bar{v}$.

Definition 3.7. *For a symmetric tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n\times\cdots\times n}$ consider the following polynomial*

$$f_{Sym,\mathcal{T}}(X) \triangleq \sum_{(j_1,\ldots,j_d)\in[n]\times\cdots\times[n]} \alpha_{j_1,j_2,\ldots,j_d} x_{j_1}x_{j_2}\cdots x_{j_d}.$$

Just like in case of general tensors, computing the symmetric rank reduces to finding the optimal top fan-in of a special class of arithmetic circuits, which is sum of power of linear forms ($\Sigma\wedge\Sigma$) circuits defined below.

Definition 3.8 (Sum of power of linear forms). *The Sum of power of linear forms with top fan-in k computes a polynomial of the form $f = \ell_1^d + \cdots \ell_k^d$ where each ℓ_i is a linear polynomial over the n variables.*

Observation 3.9. *Let $C(X) = \sum_{i=1}^{k} \ell_i^d$ be a $\Sigma\wedge\Sigma(k)$ circuit over $\mathbb{F}$ computing $f_{Sym,\mathcal{T}}(X)$ for a symmetric tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1\times\cdots\times n_d}$. Then*

$$\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_i)\otimes\cdots\otimes\bar{v}(\ell_i)$$

where $v(\ell_i)$ is a n-dimensional vector corresponding to the linear form $\ell_{i,j}$.

Remark 3.10. *Both Tensor rank and Symmetric rank are dependent on the underlying field, that is Tensor rank of a tensor $\mathcal{T}$ over $\mathbb{F}$ and $\mathbb{G}$, an extension of $\mathbb{F}$ can be different, see [51, 52] for details. The correspondence discussed above, among Tensor rank(symmetric rank) and top fan-in of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits($\Sigma\wedge\Sigma$ circuits), respects the dependence of rank on underlying field. That is, in order to find rank of $\mathcal{T}$ over $\mathbb{G}$ we have to find an optimal top fan-in of a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit over $\mathbb{G}$ computing $f_{\mathcal{T}}$.*

3.3 Complexity of Solving a System of Polynomial Equations

Solving a system of polynomial equations is the following problem: For a field $\mathbb{F}$, we are given m polynomials $f_1, f_2, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$, each of degree at most d. We want to test if there exist a solution (this is the decision version) to $f_1 = 0, f_2 = 0, \ldots, f_m = 0$ in $\mathbb{F}^n$, or find a solution if it exists (this is the search version). A straightforward reduction from 3-SAT shows that polynomial system solving is NP-hard in general. This is a fundamental problem in computational algebra, and it has received lot of attention over various fields. To mention a few, system solving is NP-complete for finite fields, in PSPACE over $\mathbb{R}$ [10] and in Polynomial Hierarchy (Σ_2), assuming GRH [41].

Interestingly, for $\mathbb{F} = \mathbb{Q}$ system solving is not even known to be decidable! In fact, if we restrict the question to integral domains (like $\mathbb{Z}$) then the problem is undecidable. This was the well-known Hilbert's tenth problem, which asks if a given Diophantine equation has an integral solution, and was famously proved to be undecidable in the 70's, see [44].

In this work, we are mainly concerned with polynomial system solving when the number of variables involved is small (such as a constant). In this case, polynomial system solving turns out is efficient under various settings. We will use the following definitions for describing the complexity of solving a system of equations under various settings.

Definition 3.11 ($\mathrm{Sys}_{\mathbb{F}}(n,m,d)$). *Let $\mathrm{Sys}_{\mathbb{F}}(n,m,d)$ denote the randomized time complexity of finding a solution $\in \mathbb{F}^n$ to a system of m polynomial equations $\in \mathbb{F}[x_1,\ldots,x_n]$ of total degree d (if one exists). Also, consider a weaker version of the above problem, let $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n,m,d)$ denote the randomized time complexity of finding a solution (could be in an extension of $\mathbb{F}$) to a system of m polynomial equations $\in \mathbb{F}[x_1,\ldots,x_n]$ of total degree d (if one exists).*

Definition 3.12 (Det-$\mathrm{Sys}_{\mathbb{F}}(n,m,d)$). *Let Det-$\mathrm{Sys}_{\mathbb{F}}(n,m,d)$ denote the deterministic time complexity of finding a solution $\in \mathbb{F}^n$ to a system of m polynomial equations $\in \mathbb{F}[x_1,\ldots,x_n]$ of total degree d (if one exists).*

We will now mention various known upper bounds on $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n,m,d)$ and $\mathrm{Sys}_{\mathbb{F}}(n,m,d)$ for various fields. In all these bounds, we have suppressed a $\mathrm{poly}(c)$ dependence in the running time, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite.

Theorem 3.13. *Let $f_1, f_2, \ldots f_m \in \mathbb{F}[x_1,\ldots,x_n]$ be n-variate polynomials of degree at most d. Then, the complexity of finding a single solution to the system $f_1(x) = 0, \ldots, f_m(x) = 0$ (if one exists) over various fields is as follows:*

1. *For all fields $\mathbb{F}$, $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n,m,d) = \mathrm{poly}((nmd)^{3^n})$. This follows from standard techniques in elimination theory, see [14] for details. For a detailed sketch of the argument and a bound on the size of the extension.*
2. *[26][4] For $\mathbb{F} = \mathbb{F}_q$, $\mathrm{Sys}_{\mathbb{F}}(n,m,d) = O(d^{n^n}\cdot(m\log q^{O(1)}))$.*
3. *[20] For $\mathbb{F} = \mathbb{R}$, $\mathrm{Sys}_{\mathbb{F}}(n,m,d) =$ Det-$\mathrm{Sys}_{\mathbb{F}}(n,m,d) = \mathrm{poly}((md)^{n^2})$. Note that*

[4] the main results of this work is written for the case when q is prime, but the authors observe that it works for general q as well.

in this case the assumption is that the coefficients are integers or rationals[5]*. However, the output might be a tuple of algebraic numbers over* $\mathbb{R}$ *where the degree of the extension is polynomially bounded when* n *is a constant. See [20] for details. Note that all algebraic algorithms used in this paper will continue to hold when the inputs are algebraic numbers of low/polynomial degree, and we deal with algebraic extensions in the standard way.*

(4) *[27] For* $\mathbb{F} = \mathbb{C}$ *(or any algebraically closed field),* $\mathrm{Sys}_{\mathbb{F}}(n, m, d) =$ *Det-*$\mathrm{Sys}_{\mathbb{F}}(n, m, d) = (mn)^{O(n)} \cdot d^{O(n^2)}$.

Note that, for all cases described above, both $\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ and $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n, m, d)$ are bounded by $\mathrm{poly}((nmd)^{n^n})$. Thus, when $n = O(1)$, $\mathrm{Sys}(n, m, d) = \mathrm{poly}(m, d)$.

For clarity in presentation, we *artificially* define $\mathrm{Sys}(n, m, d)$ as the complexity of finding a solution to a system of m polynomial equations $\in \mathbb{F}[x_1, \ldots, x_n]$ of total degree d s.t. the solution has to lie in $\mathbb{F}$ if $\mathbb{F} = \mathbb{R}, \mathbb{C}, \mathbb{F}_q$ or an algebraically closed field, and it could be over an algebraic extension for other fields. Clearly, as discussed above $\mathrm{Sys}(n, m, d) = \mathrm{poly}((nmd)^{n^n})$.

3.3.1 Derandomizing Solving System of Equations. Derandomizing solving system of equation in general is considered a hard problem for the following reason. Just solving a univarite quadratic equation over $\mathbb{F}_p$ in *deterministic* $\mathrm{poly}(\log p)$ time is a notoriously hard open problem, See [1, Problem 15]. Interestingly, this is the only case when low-variate plynomial system solving is hard to derandomize. That is, if the underlying field is not a finite field with large characteristic, then there do exist efficient deterministic algorithms for low-variate system solving.

Indeed solving systems of polynomial equations is the only place in the paper where randomness is utilized. Thus, all our algorithms can be derandomized over $\mathbb{R}, \mathbb{C}$, since the algorithms mentioned in Theorem 3.13, for polynomial system solving over $\mathbb{F} = \mathbb{R}$ and $\mathbb{C}$ are already deterministic. Though we did not mention it, polynomial system solving (and hence our algorithms) can also be derandomized over $\mathbb{F}_{p^d}$(in time $\mathrm{poly}(p, d)$ time).

3.4 Hardness of Computing Tensor Rank

The first step towards understanding the computational complexity was by Håstad [25] who showed that determining the tensor rank is an NP-hard over $\mathbb{Q}$ and NP-complete over finite fields. A better way to understand hardness results for computing tensor rank is to study its connection to solving system of polynomial equations.

Theorem 3.14. *[51] For any field* $\mathbb{F}$*, given a system of* m *algebraic equations* S *over* $\mathbb{F}$*, we can in polynomial time construct a 3 dimension tensor* $\mathcal{T}_S$ *of shape* $[3m] \times [3m] \times [n+1]$ *and an integer* $k = 2m + n$ *such that* S *has a solution* $\in \mathbb{F}$ *iff* $\mathcal{T}$ *has rank atmost* $2m + n$ *over* $\mathbb{F}$.

This shows equivalence between system solving and computing tensor rank. This along with complexity of system solving (discussed in the previous section) shows that computing tensor rank is NP-complete over finite fields, over $\mathbb{R}$ it is in PSPACE [10] and is in the Polynomial Hierarchy (Σ_2), assuming the GRH [41].

[5]Here the authors assumed that the constants appearing in the system are integers (or rationals). Note that for all computational applications we can WLOG assume this by simply approximating/truncating a given real number at some number of bits.

Similar, reductions also hold for integral domains (e.g. $\mathbb{Z}$) [52], thus showing that computing Tensor rank is *undecidable* over $\mathbb{Z}$ and not known to be decidable over $\mathbb{Q}$. Due to the equivalence between tensor rank computation and learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits with optimal top fan-in, we get the corresponding hardness consequences for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$-circuit reconstruction as well.

Such results also hold for symmetric rank computation, see [52]. Concretely, for 3-dimensional tensors of length n, Shitov showed that we can convert general tensors $\mathcal{T}$ to symmetric tensors $\mathcal{T}_{sym}$ s.t. $rank(\mathcal{T}) + 4.5(n^2 + n) = \text{symmetric-rank}(\mathcal{T}_{sym})$, thus transferring the results mentioned above for general tensors to symmetric tensors as well. Again, these hardness results along with equivalence between symmetric tensor rank computation and reconstructing optimal (w.r.t top fan-in) $\Sigma\wedge\Sigma$ circuits implies that proper learning (with optimal top-fan-in) for $\Sigma\wedge\Sigma$ circuits is as hard as polynomial system solving. In particular, it is NP-hard for most fields and maybe even undecidable over $\mathbb{Q}$.

4 ACKNOWLEDGMENTS

Research supported in part by NSF grants CCF-1350572, CCF-1540634, CCF-1909683, BSF grant 2014359, a Sloan research fellowship and the Simons Collaboration on Algorithms and Geometry.

REFERENCES

[1] L. M. Adleman and K. S. McCurley. 1994. Open problems in number theoretic complexity, II. In *International Algorithmic Number Theory Symposium*. Springer, 291–322.
[2] M. Agrawal. 2005. Proving Lower Bounds Via Pseudo-random Generators. In *Proceedings of the 25th FSTTCS (LNCS, Vol. 3821)*. 92–105.
[3] M. Agrawal and V. Vinay. 2008. Arithmetic circuits: A chasm at depth four. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 67–75.
[4] N. Alon, M. Kumar, and B. L. Volk. 2020. Unbalancing Sets and An Almost Quadratic Lower Bound for Syntactically Multilinear Arithmetic Circuits. *Comb.* 40, 2 (2020), 149–178. https://doi.org/10.1007/s00493-019-4009-0
[5] D. Angluin. 1988. Queries and Concept Learning. *Machine Learning* 2 (1988), 319–342.
[6] V. Arvind and P. Mukhopadhyay. 2010. The Monomial Ideal Membership Problem and Polynomial Identity Testing. *Information and Computation* 208, 4 (2010), 351–363.
[7] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. 2000. Learning functions represented as multiplicity automata. *J. ACM* 47, 3 (2000), 506–530.
[8] M. Ben-Or and P. Tiwari. 1988. A Deterministic Algorithm for Sparse Multivariate Polynominal Interpolation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*. 301–309.
[9] V. Bhargava, S. Saraf, and I. Volkovich. 2020. Reconstruction of Depth-4 Multilinear Circuits. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Shuchi Chawla (Ed.). SIAM, 2144–2160. https://doi.org/10.1137/1.9781611975994.132
[10] J. Canny. 1988. Some algebraic and geometric computations in PSPACE. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. 460–467.
[11] E. Carlini. 2006. Reducing the number of variables of a polynomial. In *Algebraic Geometry and Geometric Modeling*, Mohamed Elkadi, Bernard Mourrain, and Ragni Piene (Eds.). Springer, 237–247. https://doi.org/10.1007/978-3-540-33275-6_15
[12] M. L. Carmosino, R. Impagliazzo, V. Kabanets, and A. Kolokolova. 2016. Learning Algorithms from Natural Proofs. In *Proceedings of the 31st Conference on Computational Complexity, CCC*. 1–24. https://doi.org/10.4230/LIPIcs.CCC.2016.10
[13] S. Chen and R. Meka. 2020. Learning Polynomials in Few Relevant Dimensions. In *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria] (Proceedings of Machine Learning Research, Vol. 125)*, Jacob D. Abernethy and Shivani Agarwal (Eds.). PMLR, 1161–1227. http://proceedings.mlr.press/v125/chen20a.html
[14] D. A. Cox, J. Little, and D. O'Shea. 2015. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (4. ed.)*. Springer.

[15] Z. Dvir and A. Shpilka. 2007. Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. *SIAM J. on Computing* 36, 5 (2007), 1404–1434.

[16] M. A. Forbes, R. Saptharishi, and A. Shpilka. 2014. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Symposium on Theory of Computing, STOC*. 867–875. https://doi.org/10.1145/2591796.2591816

[17] M. A. Forbes and A. Shpilka. 2012. Quasipolynomial-time Identity Testing of Non-Commutative and Read-Once Oblivious Algebraic Branching Programs. *Electronic Colloquium on Computational Complexity (ECCC)* 19 (2012), 115.

[18] L. Fortnow and A. R. Klivans. 2009. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.* 75, 1 (2009), 27–36.

[19] A. Garg, N. Kayal, and C. Saha. 2020. Learning sums of powers of low-degree polynomials in the non-degenerate case. *arXiv preprint arXiv:2004.06898* (2020).

[20] D. Yu. Grigor'ev and N.N. Vorobjov. 1988. Solving systems of polynomial inequalities in subexponential time. *Journal of Symbolic Computation* 5, 1 (1988), 37 – 64. https://doi.org/10.1016/S0747-7171(88)80005-1

[21] A. Gupta, P. Kamath, N. Kayal, and R. Saptharishi. 2013. Arithmetic Circuits: A Chasm at Depth Three. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 578–587. https://doi.org/10.1109/FOCS.2013.68

[22] A. Gupta, N. Kayal, and S. V. Lokam. 2011. Efficient Reconstruction of Random Multilinear Formulas. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*. 778–787. https://doi.org/10.1109/FOCS.2011.70

[23] A. Gupta, N. Kayal, and S. V. Lokam. 2012. Reconstruction of Depth-4 Multilinear Circuits with Top fanin 2. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*. 625–642. Full version at https://eccc.weizmann.ac.il/report/2011/153.

[24] A. Gupta, N. Kayal, and Y. Qiao. 2014. Random arithmetic formulas can be reconstructed efficiently. *Computational Complexity* 23, 2 (2014), 207–303. https://doi.org/10.1007/s00037-014-0085-0

[25] Johan Håstad. 1990. Tensor Rank is NP-Complete. *J. Algorithms* 11, 4 (1990), 644–654.

[26] M. D. Huang and Y. C. Wong. 1999. Solvability of systems of polynomial congruences modulo a large prime. *computational complexity* 8, 3 (1999), 227–257.

[27] D. Ierardi. 1989. Quantifier Elimination in the Theory of an Algebraically-Closed Field. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing* (Seattle, Washington, USA) *(STOC '89)*. Association for Computing Machinery, New York, NY, USA, 138–147. https://doi.org/10.1145/73007.73020

[28] V. Kabanets and R. Impagliazzo. 2003. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*. 355–364.

[29] E. Kaltofen and B. M. Trager. 1990. Computing with Polynomials Given by Black Boxes for Their Evaluations: Greatest Common Divisors, Factorization, Separation of Numerators and Denominators. *J. of Symbolic Computation* 9, 3 (1990), 301–320.

[30] Z. S. Karnin and A. Shpilka. 2009. Reconstruction of Generalized Depth-3 Arithmetic Circuits with Bounded Top Fan-in. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*. 274–285. Full version at http://www.cs.technion.ac.il/ shpilka/publications/KarninShpilka09.pdf.

[31] Z. S. Karnin and A. Shpilka. 2011. Black box polynomial identity testing of generalized depth-3 arithmetic circuits with bounded top fan-in. *Combinatorica* 31, 3 (2011), 333–364. https://doi.org/10.1007/s00493-011-2537-3

[32] N. Kayal. 2011. Efficient algorithms for some special cases of the polynomial equivalence problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 1409–1421.

[33] N. Kayal, V. Nair, and C. Saha. 2018. Average-case linear matrix factorization and reconstruction of low width Algebraic Branching Programs. *Electronic Colloquium on Computational Complexity (ECCC)* 25 (2018), 29. https://eccc.weizmann.ac.il/report/2018/029

[34] N. Kayal, V. Nair, C. Saha, and S. Tavenas. 2017. Reconstruction of Full Rank Algebraic Branching Programs. In *32nd Computational Complexity Conference, CCC 2017*. 21:1–21:61. https://doi.org/10.4230/LIPIcs.CCC.2017.21

[35] N. Kayal and C. Saha. 2019. Reconstruction of non-degenerate homogeneous depth three circuits. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*. 413–424. https://doi.org/10.1145/3313276.3316360

[36] N. Kayal and S. Saraf. 2009. Blackbox Polynomial Identity Testing for Depth 3 Circuits. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 198–207. Full version at https://eccc.weizmann.ac.il/report/2009/032.

[37] N. Kayal and N. Saxena. 2007. Polynomial Identity Testing for Depth 3 Circuits. *Computational Complexity* 16, 2 (2007), 115–138.

[38] A. Klivans and A. Shpilka. 2006. Learning restricted models of arithmetic circuits. *Theory of computing* 2, 10 (2006), 185–206.

[39] A. Klivans and D. Spielman. 2001. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*. 216–223.

[40] A. R. Klivans and A. A. Sherstov. 2009. Cryptographic hardness for learning intersections of halfspaces. *J. Comput. Syst. Sci.* 75, 1 (2009), 2–12.

[41] P. Koiran. 1996. Hilbert's Nullstellensatz is in the polynomial hierarchy. *Journal of complexity* 12, 4 (1996), 273–286.

[42] P. Koiran. 2010. Arithmetic circuits: the chasm at depth four gets wider. *CoRR* abs/1006.4700 (2010).

[43] J. Landsberg. 2012. Tensors: geometry and applications. *Representation theory* 381, 402 (2012), 3.

[44] Y. Matijasevič and J. Robinson. 1975. Reduction of an arbitrary Diophantine equation to one in 13 unknowns. *Acta Arithmetica* 27, 1 (1975), 521–553.

[45] R. Raz. 2013. Tensor-Rank and Lower Bounds for Arithmetic Formulas. *J. ACM* 60, 6 (2013), 40:1–40:15. https://doi.org/10.1145/2535928

[46] N. Saxena and C. Seshadhri. 2009. An Almost Optimal Rank Bound for Depth-3 Identities. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*. 137–148.

[47] N. Saxena and C. Seshadhri. 2010. From Sylvester-Gallai Configurations to Rank Bounds: Improved Black-Box Identity Test for Deph-3 Circuits. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 21–30.

[48] N. Saxena and C. Seshadhri. 2011. An Almost Optimal Rank Bound for Depth-3 Identities. *SIAM J. Comput.* 40, 1 (2011), 200–224.

[49] N. Saxena and C. Seshadhri. 2012. Blackbox Identity Testing for Bounded Top-Fanin Depth-3 Circuits: The Field Doesn't Matter. *SIAM J. Comput.* 41, 5 (2012), 1285–1298.

[50] N. Saxena and C. Seshadhri. 2013. From sylvester-gallai configurations to rank bounds: Improved blackbox identity test for depth-3 circuits. *J. ACM* 60, 5 (2013), 33.

[51] M. Schaefer and D. Stefankovic. 2016. The Complexity of Tensor Rank. *CoRR* abs/1612.04338 (2016). arXiv:1612.04338 http://arxiv.org/abs/1612.04338

[52] Y. Shitov. 2016. How hard is the tensor rank? *arXiv preprint arXiv:1611.01559* (2016).

[53] A. Shpilka. 2009. Interpolation of depth-3 arithmetic circuits with two multiplication gates. *SIAM J. on Computing* 38, 6 (2009), 2130–2161.

[54] A. Shpilka and I. Volkovich. 2015. Read-Once Polynomial Identity Testing. *Computational Complexity* 24, 3 (2015), 477–532.

[55] G. Sinha. 2016. Reconstruction of Real Depth-3 Circuits with Top Fan-In 2. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*. 31:1–31:53. https://doi.org/10.4230/LIPIcs.CCC.2016.31

[56] G. Sinha. 2020. Efficient reconstruction of depth three circuits with top fan-in two. *Electron. Colloquium Comput. Complex.* 27 (2020), 125. https://eccc.weizmann.ac.il/report/2020/125

[57] S. Tavenas. 2013. Improved Bounds for Reduction to Depth 4 and Depth 3. In *MFCS*. 813–824.