On the Complexity of Symmetric Polynomials

Markus Bläser
Department of Computer Science, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
mblaeser@cs.uni-saarland.de

Gorav Jindal[1]
Department of Computer Science, Aalto University, Espoo, Finland
gorav.jindal@gmail.com

Abstract

The fundamental theorem of symmetric polynomials states that for a symmetric polynomial $f_{\text{Sym}} \in \mathbb{C}[x_1, x_2, \ldots, x_n]$, there exists a unique "witness" $f \in \mathbb{C}[y_1, y_2, \ldots, y_n]$ such that $f_{\text{Sym}} = f(e_1, e_2, \ldots, e_n)$, where the e_i's are the elementary symmetric polynomials.

In this paper, we study the arithmetic complexity $L(f)$ of the witness f as a function of the arithmetic complexity $L(f_{\text{Sym}})$ of f_{Sym}. We show that the arithmetic complexity $L(f)$ of f is bounded by $\mathsf{poly}(L(f_{\text{Sym}}), \deg(f), n)$. To the best of our knowledge, prior to this work only exponential upper bounds were known for $L(f)$. The main ingredient in our result is an algebraic analogue of Newton's iteration on power series. As a corollary of this result, we show that if $\mathsf{VP} \neq \mathsf{VNP}$ then there exist symmetric polynomial families which have super-polynomial arithmetic complexity.

Furthermore, we study the complexity of testing whether a function is symmetric. For polynomials, this question is equivalent to arithmetic circuit identity testing. In contrast to this, we show that it is hard for Boolean functions.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Symmetric Polynomials, Arithmetic Circuits, Arithmetic Complexity, Power Series, Elementary Symmetric Polynomials, Newton's Iteration

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.47

1 Introduction

Lipton and Regan [10] ask the question whether understanding the arithmetic complexity of symmetric polynomials is enough to understand the arithmetic complexity of all polynomials. We here answer this question in the affirmative. The fundamental theorem of symmetric polynomials establishes a bijection between symmetric polynomials and arbitrary polynomials. It states that for every symmetric polynomial $f_{\text{Sym}} \in \mathbb{C}[x_1, x_2, \ldots, x_n]$, there exists a unique polynomial $f \in \mathbb{C}[y_1, y_2, \ldots, y_n]$ such that $f_{\text{Sym}} = f(e_1, e_2, \ldots, e_n)$, where the e_i's are the elementary symmetric polynomials. We prove that the arithmetic circuit complexity of f and f_{Sym} are polynomially related.

An arithmetic circuit C is a directed acyclic graph with the following kind of nodes (gates):

[1] Supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 759557) and by Academy of Finland, under grant number 310415. This work was done while the author was a graduate student at Saarland University and the Max-Planck-Institut für Informatik.

- Nodes with in-degree zero labeled by variables or scalars, these are called input gates.
- Nodes labeled by addition $(+)$, subtraction $(-)$ or multiplication $(\times)$ gates, these gates have in-degree two and unbounded out-degree.
- Nodes with out-degree zero (it can be an input, $+, -$ or $\times$ gate), these are called output gates.

Each gate of such a circuit computes a multivariate polynomial in the following way:

- Input gates compute the polynomial by which they are labeled.
- A $\circ$ gate g computes the polynomial $g_1 \circ g_2$, if the children gates of g compute the polynomials g_1 and g_2, here $\circ \in \{+, -, \times\}$.

If the output gates of C compute the polynomials $g_1, g_2, \ldots, g_t$ then we say that C computes the set $\{g_1, g_2, \ldots, g_t\}$ of polynomials. In the literature, it is usually assumed that any arithmetic circuit C has a unique output gate and thus C computes a single multivariate polynomial. The *size* of an arithmetic circuit C is defined as the number of gates in C.

We can naturally model computations over a field by arithmetic circuits and thus the study of arithmetic circuits is essential in studying the computational complexity in algebraic models of computation. Valiant [15] defined the complexity classes VP and VNP as algebraic analogues of the classes P and NP. In algebraic complexity theory, complexity classes such as VP and VNP are defined as sets of polynomial families. For the precise definitions of VP and VNP, see section A in the appendix. For a polynomial f, $L(f)$ is defined as the size of the smallest circuit computing f. We use the same notation $L(S)$ to denote the size of the smallest circuit computing a set of polynomials S. There is also a notion of oracle complexity $L^g(f)$ of a polynomial f with respect to some polynomial g, see Definition 9.

Let $\mathfrak{S}_n$ be the symmetric group defined as the set of all permutations of the set $\{1, 2, \ldots, n\}$. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to be *symmetric* if $f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ for all $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n, \sigma \in \mathfrak{S}_n$. It is easy to see that all symmetric Boolean functions can be computed by constant depth threshold circuits, that is, are contained in the class TC^0. The notion of symmetric polynomials can also be defined similarly. It is natural to ask whether symmetric polynomials can also be computed efficiently, *i.e.*, whether the arithmetic complexity of symmetric polynomials is also small? This is the question we study in this paper.

1.1 Previous Work

It is well known that for every symmetric polynomial $g \in \mathbb{C}[x_1, x_2, \ldots, x_n]$, there exists a unique polynomial $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ such that $g = f(e_1, e_2, \ldots, e_n)$. Here, $e_1, e_2, \ldots, e_n$ denote the elementary symmetric polynomials in $x_1, x_2, \ldots, x_n$. For an arbitrary polynomial $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$, let f_{Sym} defined by $f_{\mathrm{Sym}} \overset{\mathrm{def}}{=\!=} f(e_1, e_2, \ldots, e_n)$ be the symmetric polynomial corresponding to f (see Section 2). We want to study the relation between the complexities $L(f)$ and $L(f_{\mathrm{Sym}})$. The question was studied and partially solved in [7, 6, 10]. More specifically, the following theorems were proved in [7, 6, 10].

▶ **Theorem 1** (Theorem 1 in [7]). *For any polynomial $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$, $L(f) \leq \Delta(n)L(f_{\mathrm{Sym}}) + 2$, where $\Delta(n) \leq 4^n(n!)^2$.*

Whereas [7] showed the bound on $L(f)$ for exact computation, [10] investigated a related problem of approximating the value of f at a given point by using an arithmetic circuit computing f_{Sym}.

▶ **Theorem 2** ([10]). *For any polynomial $f \in \mathbb{Q}[x_1, x_2, \ldots, x_n]$, there is an algorithm that computes the value $f(a)$ within ϵ in time $L(f_{\mathrm{Sym}}) + \mathsf{poly}(\log ||a||, n, \log \frac{1}{\epsilon})$ for any $a \in \mathbb{Q}^n$.*

Note that Theorem 2 does not compute a circuit for f but only gives an algorithm to approximate the value of f at a given point. The results in [6] were in a much more general setting. [6] studied the in-variance under general finite matrix groups, not just under $\mathfrak{S}_n$ as we do in this paper. By specializing the theorems in [6] for the finite matrix group $\mathfrak{S}_n$, we get the following result.

▶ **Theorem 3** ([6]). *For any polynomial $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$, we have $L(f) \leq ((n+1)!)^6 L(f_{\mathrm{Sym}})$.*

The upper bound in [6] (Theorem 3) is worse than that of [7] (Theorem 1) but this is to be expected because [6] solves a more general problem.

1.2 Our results

It is easy to see that $L(f_{\mathrm{Sym}}) \leq L(f) + n^{O(1)}$ (see [10]). All the exact bounds (with respect to $L(f_{\mathrm{Sym}})$) on $L(f)$ above are exponential. It was left as an open question in [10] whether $L(f)$ can be bounded polynomially with respect to $L(f_{\mathrm{Sym}})$. In this paper, we demonstrate that $L(f)$ can be polynomially bounded in terms of $L(f_{\mathrm{Sym}})$. In whatever follows, the complexity notation $\tilde{O}$ hides poly-logarithmic factors. The following Theorem 4 is the main contribution of this paper.

▶ **Theorem 4.** *For any polynomial $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ of degree d with $d_{\mathrm{Sym}} \overset{def}{=\!=} \deg(f_{\mathrm{Sym}})$, we have the following upper bounds on $L^{f_{\mathrm{Sym}}}(f)$ and $L(f)$:*

$$L^{f_{\mathrm{Sym}}}(f) \leq \tilde{O}\left(n^3 \cdot d^2 \cdot d_{\mathrm{Sym}}\right),$$
$$L(f) \leq \tilde{O}\left(d^2 L(f_{\mathrm{Sym}}) + d^2 n^2\right).$$

▶ Remark. It can be shown that $d_{\mathrm{Sym}} \leq dn$ and $d \leq d_{\mathrm{Sym}}$. Thus Theorem 4 implies that $L^{f_{\mathrm{Sym}}}(f) \leq \tilde{O}\left(n^4 \cdot d^3\right)$.

From Theorem 4, it easily follows that there exist families of symmetric polynomials of super-polynomial arithmetic complexity (assuming VP $\neq$ VNP).

In addition, we also consider the following problems:

1. SFT (symmetric function testing)
2. SPT (symmetric polynomial testing)

▶ **Problem 5** (SFT). *Given a Boolean circuit C computing the Boolean function $f(x_1, x_2, \ldots, x_n)$, check if f is symmetric, that is, is $f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ for all $\sigma \in \mathfrak{S}_n$?*

▶ **Problem 6** (SPT). *Given an arithmetic circuit C computing the polynomial $f(x_1, x_2, \ldots, x_n)$, check if f is a symmetric polynomial?*

Let CSAT be the problem of deciding whether a given Boolean circuit has a satisfying assignment, that is, computes a non-zero function (see [8]). ACIT is the problem of deciding whether the polynomial computed by a given arithmetic circuit is zero (see [1]). We prove the following results on the complexity of SPT and SFT.

▶ **Lemma 7.** SFT *and* CSAT *are polynomial time Turing reducible to each other, i.e.,* SFT $\leq_{\mathrm{P}}^{\mathrm{T}}$ CSAT *and* CSAT $\leq_{\mathrm{P}}^{\mathrm{T}}$ SFT.

▶ **Lemma 8.** SPT *and* ACIT *are polynomial time many one reducible to each other, i.e.,* SPT $\leq_{\mathrm{P}}$ ACIT *and* ACIT $\leq_{\mathrm{P}}$ SPT.

In light of above results, we notice the following contrasting situations in Boolean and algebraic models of computation:

- All symmetric Boolean functions are easy to compute but the problem of deciding the symmetry of a Boolean function is hard.
- There exist families of symmetric polynomials which are hard to compute (assuming $\mathsf{VP} \neq \mathsf{VNP}$) but deciding the symmetry of a polynomial is easy.

1.3 Proof ideas

The main proof idea is much easier to demonstrate in the case of $n = 2$. Let $B(y)$ be the following uni-variate polynomial in y with coefficients in $\mathbb{C}[x_1, x_2]$:

$$B(y) \overset{\text{def}}{=\!=} y^2 - (x_1 + x_2)y + x_1x_2.$$

Note that the roots of $B(y)$ are x_1, x_2. Hence we have the following equalities:

$$x_1 = \frac{x_1 + x_2 + \sqrt{(x_1 + x_2)^2 - 4x_1x_2}}{2},$$
$$x_2 = \frac{x_1 + x_2 - \sqrt{(x_1 + x_2)^2 - 4x_1x_2}}{2}.$$

We use the symbols e_1, e_2 for the elementary symmetric polynomials: $e_1 \overset{\text{def}}{=\!=} (x_1 + x_2)$ and $e_2 \overset{\text{def}}{=\!=} x_1x_2$. Thus we have the following equalities:

$$x_1 = \frac{e_1 + \sqrt{e_1^2 - 4e_2}}{2}, \tag{1}$$
$$x_2 = \frac{e_1 - \sqrt{e_1^2 - 4e_2}}{2}. \tag{2}$$

Let $f_{\text{Sym}} \in \mathbb{C}[x_1, x_2]$ be a symmetric polynomial and $\deg(f) = d$.

If we substitute the above radical expressions (in Equation 1 and Equation 2) for x_1 and x_2 in $f_{\text{Sym}}(x_1, x_2)$, then we obtain $f(e_1, e_2)$. But unfortunately, we can not perform these kind of substitutions in our model of computation. This is because we cannot compute expressions of the form $\sqrt{e_1^2 - 4e_2}$ with arithmetic circuits.

If we use the substitution $e_2 \leftarrow e_2 - 1$ in Equation 1 and Equation 2 and thereafter substitute x_1 and x_2 in $f_{\text{Sym}}(x_1, x_2)$, we shall obtain $f(e_1, e_2 - 1)$. The degree of $f(e_1, e_2 - 1)$ is also bounded by d. Even by using this substitution, the expressions in Equation 1 and Equation 2 cannot be computed by arithmetic circuits. But this substitution allows us to use Taylor expansion on $\sqrt{e_1^2 - 4(e_2 - 1)}$ to obtain a power series in e_1, e_2. Since $f(e_1, e_2 - 1)$ has degree at most d, we only need to substitute truncations of degree d of these Taylor series to obtain $f(e_1, e_2 - 1)$ (also some additional junk terms which can be removed efficiently) and subsequently use the substitution $e_2 \leftarrow e_2 + 1$ to obtain $f(e_1, e_2)$.

This method works for two variables. It can be extended to work for at most four variables because it is well known that polynomials of degree more than four are not solvable by radicals (see e.g. Section 15.9 in [2]). To make this idea work in general, we shall substitute e_n by $e_n + (-1)^{n-1}$ and then compute degree d truncation of roots of $B(y)$ using Newton's iteration.

1.4 Organization

Section 2 introduces elementary symmetric polynomials and also formally states the fundamental theorem of symmetric polynomials. We also state a folklore result about computing the homogeneous components of a polynomial. Section 3 describes the complexity of the problems SFT and SPT. Section 4 describes the main contribution of this paper, we use the classical Newton's iteration to prove Theorem 4 in this section. As an easy consequence of Theorem 4, Section 5 proves that there exist hard symmetric polynomial families (assuming $\mathsf{VP} \neq \mathsf{VNP}$).

2 Preliminaries

2.1 Notation and background

For a positive integer n, we use the notation $[n]$ to denote the set $\{1, 2, \dots, n\}$. Similarly, $[[n]]$ is used to denote the set $\{0, 1, 2, \dots, n\}$. Now we formally define the notion of oracle computations [5].

▶ **Definition 9** ([5]). The oracle complexity $L^g(f)$ of a polynomial $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ with respect to the oracle polynomial g is the minimum number of arithmetic operations $+, -, \times$ and evaluations of g (at previously computed values) that are sufficient to compute f from the indeterminates x_i and constants in $\mathbb{F}$.

▶ **Definition 10.** The i^{th} elementary symmetric polynomial e_i^n in n variables $x_1, x_2, \dots, x_n$ is defined as the following polynomial:

$$e_i^n \overset{\text{def}}{=\!=} \sum_{1 \le j_1 < j_2 < \cdots < j_i \le n} x_{j_1} \cdot x_{j_2} \cdot \dots \cdot x_{j_i}.$$

For an arbitrary polynomial $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$, we define the polynomial f_{Sym} as:

$$f_{\text{Sym}} \overset{\text{def}}{=\!=} f(e_1^n, e_2^n, \dots, e_n^n). \tag{3}$$

Whenever n is clear from the context, we use the notation e_i to denote the i^{th} elementary symmetric polynomial e_i^n. Note that f_{Sym} is a symmetric polynomial. So Equation 3 is a method to create symmetric polynomials. The fundamental theorem of symmetric polynomials states that Equation 3 is the only way to create symmetric polynomials.

▶ **Theorem 11** (see [3]). *If $g \in \mathbb{C}[x_1, x_2, \dots, x_n]$ is a symmetric polynomial, then there exists a unique polynomial $f \in \mathbb{C}[y_1, y_2, \dots, y_n]$ such that $g = f(e_1^n, e_2^n, \dots, e_n^n)$.*

Theorem 11 states that every symmetric polynomial g can be uniquely written as f_{Sym} for some f. Thus in whatever follows, we always use the notation of the kind f_{Sym} to denote a symmetric polynomial.

2.2 Basic tools

Suppose we have a circuit C computing a polynomial $f \in \mathbb{C}[x_1, x_2, \dots, x_n]$ of degree d. It might be the case that f is not homogeneous. For some applications, it might be better to work with homogeneous polynomials. So we want to know if there exist "small" circuits also for the homogeneous components of f. For a polynomial f, $f^{[m]}$ is used to denote the degree m homogeneous component of f. The following Lemma 12 proves that the homogeneous components of f also have "small" arithmetic circuits.

▶ **Lemma 12** (Folklore). *Let $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ be a polynomial with $d = \deg(f)$. For any $0 \le m \le d$, we have: $L^f(f^{[m]}) \le O(nd)$.*

Proof. For a fresh indeterminate y, consider the polynomial $f(yx_1, yx_2, \ldots, yx_n)$. We consider $f(yx_1, yx_2, \ldots, yx_n)$ as a uni-variate polynomial in y of degree d, with coefficients in $\mathbb{C}[x_1, x_2, \ldots, x_n]$. We observe that for any $0 \le m \le d$, the coefficient of y^m in $f(yx_1, yx_2, \ldots, yx_n)$ is $f^{[m]}$. Let $\alpha_1, \alpha_2, \ldots, \alpha_{d+1}$ be $d+1$ distinct constants in $\mathbb{C}$. By interpolation, we know that for any $0 \le m \le d$, the coefficient $f^{[m]}$ of y^m is a $\mathbb{C}$-linear combination of $d+1$ evaluations $f(\alpha_i x_1, \alpha_i x_2, \ldots, \alpha_i x_n)$ of $f(yx_1, yx_2, \ldots, yx_n)$ at $\alpha_i \in \{\alpha_1, \alpha_2, \ldots, \alpha_{d+1}\}$. Formally, for any $0 \le m \le d$ we have:

$$f^{[m]} \in \langle \{ f(\alpha_i x_1, \alpha_i x_2, \ldots, \alpha_i x_n) \mid \alpha_i \in \{\alpha_1, \alpha_2, \ldots, \alpha_{d+1}\}\}\rangle. \tag{4}$$

It is easy to observe that $L^f(f(\alpha_i x_1, \alpha_i x_2, \ldots, \alpha_i x_n)) = O(n)$. Equation 4 implies that $L^f(f^{[m]}) \le O(nd)$. ◀

Lemma 12 implies that $L(f^{[m]}) \le O(L(f) \cdot n \cdot d)$, this bound depends on the degree d of f. But if we do not care about the oracle complexity, the following bound (independent of degree of f) on $L(f^{[m]})$ is known.

▶ **Lemma 13** ([13, 14]). *Let $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ be a polynomial and m be a non-negative integer. Then we have:*

$$L(\{f^{[0]}, f^{[1]}, \ldots, f^{[m]}\}) \le O(m^2 L(f)).$$

3 Complexity of SFT and SPT

Here we prove Lemma 7 and Lemma 8.

Proof of Lemma 7. Given a Boolean circuit C, we want to check if the function $f(x_1, x_2, \ldots, x_n)$ computed by C is symmetric. As the permutation group $\mathfrak{S}_n$ is generated by two permutations $\sigma \stackrel{\text{def}}{=} (1, 2)$ and $\pi \stackrel{\text{def}}{=} (1, 2, \ldots, n)$ [4], it is necessary and sufficient to check if the given function f is invariant under these two permutations of variables. Thus we define the following Boolean functions:

$$g(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=} f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}),$$
$$h(x_1, x_2, \ldots, x_n) \stackrel{\text{def}}{=} f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}).$$

Now note that the equality of two Boolean variables x, y can be checked by the following equality gadget:

$$(x \stackrel{?}{=} y) = (\neg x \vee y) \wedge (x \vee \neg y).$$

Thus we only need to check if both $(\neg f \vee g) \wedge (f \vee \neg g)$ and $(\neg f \vee h) \wedge (f \vee \neg h)$ are tautologies (always equal to 1). This can be checked by two oracles calls to CSAT. Thus SFT $\le_{\mathrm{P}}^{\mathrm{T}}$ CSAT.

Now we prove the other direction. Given a Boolean circuit C, we want to check if the function $f(x_1, x_2, \ldots, x_n)$ computed by C is always zero. First we make an oracle call to SFT to check if f is symmetric. If f is not symmetric then obviously f is a non-zero function because the zero function is trivially symmetric. Thus we can assume f to be symmetric. Now we ask the SFT oracle if the function $h \stackrel{\text{def}}{=} f \wedge x_1$ is symmetric? If f was the zero function then so is h, therefore SFT oracle will answer that h is symmetric. So if SFT oracle answers h to be non-symmetric then obviously f was non-zero. If h also turns out to be symmetric then we know that:

$$\forall (a_1, a_2, \ldots, a_n) \in \{0,1\}^n : f \wedge a_1 = f \wedge a_2 = \cdots = f \wedge a_n. \tag{5}$$

Suppose f evaluated to 1 on a point $(a_1, a_2, \ldots, a_n) \notin \{(0,0,\ldots,0),(1,1,\ldots,1)\}$. This means that there exists $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ such that $f(a_1, a_2, \ldots, a_n) = 1$ with $a_i = 1, a_j = 0$ for some $i, j \in [n]$. Then obviously we have $f(a_1, a_2, \ldots, a_n) \wedge a_i = 1$ and $f(a_1, a_2, \ldots, a_n) \wedge a_j = 0$. Hence Equation 5 can not to be true. Thus f can only be non-zero on the set $\{(0,0,\ldots,0),(1,1,\ldots,1)\}$. The value of f at both these points can be checked manually to check whether f is the zero function or not. Therefore CSAT $\leq_{\mathrm{P}}^{\mathrm{T}}$ SFT. ◀

Proof of Lemma 8. Given an arithmetic circuit C, we want to check if the polynomial $f(x_1, x_2, \ldots, x_n)$ computed by C is symmetric.

As in the proof of the Lemma 7, we use the fact that permutation group $\mathfrak{S}_n$ is generated by two permutations $\sigma \overset{\text{def}}{=\!=} (1,2)$ and $\pi \overset{\text{def}}{=\!=} (1,2,\ldots,n)$. It is necessary and sufficient to check if the given polynomial f is invariant under these two permutations of variables. Analogous to the proof of the Lemma 7, we define the following polynomials:

$$g(x_1, x_2, \ldots, x_n) \overset{\text{def}}{=\!=} f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}),$$
$$h(x_1, x_2, \ldots, x_n) \overset{\text{def}}{=\!=} f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}).$$

Thus f is symmetric iff $f - g = f - h = 0$. Consider the polynomial $F = y(f-g) + z(f-h)$, here y, z are fresh variables. Thus f is symmetric iff F is the zero polynomial. Hence SPT $\leq_{\mathrm{P}}$ ACIT.

Now we prove the reverse direction. Given an arithmetic circuit C, we want to check if the polynomial $f(x_1, x_2, \ldots, x_n)$ computed by C is the zero polynomial or not. Consider the polynomial $G \overset{\text{def}}{=\!=} f(x_1^2, x_2^2, \ldots, x_n^2) \cdot x_1$. We know that f is non-zero iff G is non-zero. Suppose that $G \neq 0$. Now observe that in every monomial $\mathcal{M}$ of G, the degree of x_1 in $\mathcal{M}$ is odd and the degrees of the other variables $x_2, \ldots, x_n$ in $\mathcal{M}$ are even. Now consider the polynomial $H \overset{\text{def}}{=\!=} G(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ where $\sigma \overset{\text{def}}{=\!=} (1,2)$. In every monomial $\mathcal{M}'$ of H, the degree of x_2 in $\mathcal{M}'$ is odd and the degrees of the other variables $x_1, x_3, \ldots, x_n$ in $\mathcal{M}'$ are even. Thus $H \neq G$. Hence if G is non-zero then G is not symmetric because it is not invariant under the permutation $\sigma \overset{\text{def}}{=\!=} (1,2)$. Thus G is symmetric iff $f = 0$. Hence ACIT $\leq_{\mathrm{P}}$ SPT. ◀

4 Main algorithm

4.1 Roots as power series

Let $F(y) = F(y, u_1, u_2, \ldots, u_n) = y^n + f_1(u_1, u_2, \ldots, u_n)y^{n-1} + \ldots + f_n(u_1, u_2, \ldots, u_n)$ be a monic square-free polynomial in variables y and $u_1, u_2, \ldots, u_n$, here $f_i \in \mathbb{C}[u_1, u_2, \ldots, u_n]$. Let $A(u_1, u_2, \ldots, u_n)$ be a root of F with respect to y. The root is usually an algebraic function in $u_1, u_2, \ldots, u_n$ but not a power series. The following Lemma 14 formalizes a sufficient condition when roots of $F(y)$ can be expressed as power series in $u_1, u_2, \ldots, u_n$.

▶ **Lemma 14** (Condition A in [12]). *Let $F(y, u_1, u_2, \ldots, u_n)$ be square free and monic with respect to y. If $F(y, 0, 0, \ldots, 0)$ has no multiple root (as a uni-variate polynomial in y) then the roots $A_i(u_1, u_2, \ldots, u_n)$ of $F(y, u_1, u_2, \ldots, u_n)$ can be expanded into power series in $u_1, u_2, \ldots, u_n$.*

Algorithm 1 Newton's Method.

```
Input: A square free monic polynomial $F(y) = F(y, u_1, u_2, \ldots, u_n) \in \mathbb{C}[u_1, u_2, \ldots, u_n][y]$
    with respect to $y$ of degree $n$ such that $F(y, 0, 0, \ldots, 0)$ has $n$ simple roots. A positive
    integer $d$ with $d = 2^\ell$ for some $\ell \in \mathbb{N}$. We assume that $A_1, A_2, \ldots, A_n \in \mathbb{C}[[u_1, u_2, \ldots, u_n]]$
    are the roots of $F(y)$.
Output: Degree $d$ truncations $A_1^{(\ell)}, A_2^{(\ell)}, \ldots, A_n^{(\ell)}$ of the $n$ roots $(A_1, A_2, \ldots, A_n)$ of $F(y)$,
    that is, $A_i^{(\ell)} \equiv A_i \bmod I^d$ with $I \overset{\text{def}}{=\!=} (u_1, u_2, \ldots, u_n)$ for all $i \in [n]$.
1: $\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \leftarrow$ Roots of $F(y, 0, 0, \ldots, 0)$.
2: for $1 \le i \le n$ do
3:     $A_i^{(0)} \leftarrow \alpha_i$.
4:     for $0 \le k \le \ell - 1$ do
5:         $A_i^{(k+1)} \leftarrow A_i^{(k)} - \frac{F(A_i^{(k)})}{F'(A_i^{(k)})}$.
6:     end for
7: end for
8: return $A_1^{(\ell)}, A_2^{(\ell)}, \ldots, A_n^{(\ell)}$.
```

As stated above, we are interested in the following special case:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n e_n.$$

This F is being considered as a uni-variate polynomial in y over the power series ring $\mathbb{C}[[e_1, e_2, \ldots, e_n]]$. In this case, the roots of $F(y, e_1, e_2, \ldots, e_n)$ are $x_1, x_2, \ldots, x_n$. We want to express roots of this F as power series in $e_1, e_2, \ldots, e_n$. For this purpose, we consider a slightly modified version of F. More specifically, consider:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n (e_n + (-1)^{n-1}). \tag{6}$$

Notice that $F(y, 0, 0, \ldots, 0)$ has n distinct roots, namely the n^{th} roots of unity. Thus the roots of this $F(y)$ (Equation 6) can be expressed as power series in $e_1, e_2, \ldots, e_n$, this follows from Lemma 14. Let us record this as corollary 15.

▶ **Corollary 15.** *If F is as in Equation 6, then there exist n power series $A_1, A_2, \ldots, A_n \in \mathbb{C}[[e_1, e_2, \ldots, e_n]]$ such that $F(A_i) = 0$ for all $i \in [n]$.*

Now we show how to compute the degree d truncations of such roots $A_1, A_2, \ldots, A_n$. This already follows from [9]. For the reader's convenience, we describe the algorithm here and a proof of correctness can be found in Appendix A.

4.2 Newton's Method

For the analysis of Algorithm 1, define the ideal I as:

$$I \overset{\text{def}}{=\!=} (u_1, u_2, \ldots, u_n).$$

▶ **Theorem 16.** *In Algorithm 1, $A_i^{(k)} \equiv A_i \bmod I^{2^k}$ for all $0 \le k \le \ell$ and for all $i \in [n]$.*

In Algorithm 1, we need to compute the inverse of $F'(A^{(k)})$, since we want to compute $A^{(k+1)}$, it is enough to compute the inverse of $F'(A^{(k)}) \bmod I^{2^{k+1}}$. This also follows from [9]. We explicitly describe this in Algorithm 2.

▶ **Lemma 17.** *Algorithm 2 computes a polynomial p such that $p \equiv g^{-1} \bmod I^d$.*

Algorithm 2 Inverse computation.

Input: A circuit C computing the polynomial $g(u_1, u_2, \ldots, u_n)$ such that $g(0, 0, \ldots, 0) \neq 0$ and a positive integer d with $d = 2^\ell$ for some $\ell \in \mathbb{N}$.

Output: A circuit D for computing a polynomial $p(u_1, u_2, \ldots, u_n)$ such that $p \equiv g^{-1} \bmod I^d$, here $I = (u_1, u_2, \ldots, u_n)$ and g^{-1} is the inverse of g in $\mathbb{C}[[u_1, u_2, \ldots, u_n]]$.

```
1: $p_0 \leftarrow \frac{1}{g(0,0,\ldots,0)}$.
2: for $0 \leq k \leq \ell - 1$ do
3:     $p_{k+1} \leftarrow p_k \cdot (2 - g \cdot p_k)$.
4: end for
5: return $p_\ell$.
```

We can also prove that there is a "small" circuit for p in Lemma 17.

▶ **Lemma 18.** *Let $g(u_1, u_2, \ldots, u_n)$ be a polynomial such that $g(0, 0, \ldots, 0) \neq 0$. For any positive integer d with $d = 2^\ell$ for some $\ell \in \mathbb{N}$, there is a polynomial $p \in \mathbb{C}[u_1, u_2, \ldots, u_n]$ such that $p \equiv g^{-1} \bmod I^d$. Moreover, $L(p) \leq L(g) + O(\ell)$.*

Proof. In Algorithm 2, we need three arithmetic operations to compute p_{k+1} from p_k. It follows from Lemma 17 that $p_\ell = g^{-1} \bmod I^d$. Thus there exists a circuit of size $L(g) + 3 \cdot \ell = L(g) + O(\ell)$ computing $p \equiv g^{-1} \bmod I^d$. ◀

Now the following Theorem 19 follows by applying Lemma 18 and Theorem 16.

▶ **Theorem 19.** *Let $F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n (e_n + (-1)^{n-1})$ and let $A_1, A_2, \ldots, A_n \in \mathbb{C}[[e_1, e_2, \ldots, e_n]]$ such that $F(A_i, e_1, e_2, \ldots, e_n) = 0$ for all $i \in [n]$. Let d be a positive integer with $d = 2^\ell$ for some $\ell \in \mathbb{N}$ and let $I = (e_1, e_2, \ldots, e_n)$ be the ideal generated by $e_1, e_2, \ldots, e_n$ in the polynomial ring $\mathbb{C}[e_1, e_2, \ldots, e_n]$. Let polynomials D_i be such that $D_i \equiv A_i \bmod I^d$. Then $L(\{D_1, D_2, \ldots, D_n\}) \leq O(n^2 \ell + n\ell^2)$.*

Proof. We construct a circuit D whose outputs are $D_1, D_2, \ldots, D_n$. We construct the desired circuit D by using Algorithm 1 on $F(y, e_1, e_2, \ldots, e_n)$ and $s = (0, 0, \ldots, 0)$. It is enough to describe a circuit computing each D_i such that $D_i \equiv A_i \bmod I^d$. The circuit for $A_i^{(0)}$ in Algorithm 1 is trivially of size one. By Step 5 of Algorithm 1, a circuit for $A_i^{(k+1)}$ can be constructed given any circuits for $A_i^{(k)}$, $F(A_i^{(k)})$ and $F'(A_i^{(k)})$. Note that there are circuits of size $O(n)$ computing $F(y, e_1, e_2, \ldots, e_n)$ and $F'(y, e_1, e_2, \ldots, e_n) \overset{\text{def}}{=\!=} \frac{\partial F(y, e_1, e_2, \ldots, e_n)}{\partial y}$. Thus if $A_i^{(k)}$ has a circuit of size s, then there exists a size $s + O(n + \log d)$ circuit computing $A_i^{(k+1)}$, this follows from Lemma 18. In particular, there exists a circuit computing $D_i \overset{\text{def}}{=\!=} A_i^{(\lceil \log d \rceil)}$ of size $O(n \log d + \log^2 d)$. By Theorem 16, it follows that $D_i \equiv A_i \bmod I^d$. We combine the circuits computing D_i's to construct the desired circuit D of size $O(n \cdot n \log d + n \log^2 d) = O(n^2 \log d + n \log^2 d)$. ◀

Now we are ready to prove Theorem 4.

Proof of Theorem 4. The main idea is what we have hinted above. Namely, let $F(y, e_1, e_2, \ldots, e_n)$ be the following polynomial:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n (e_n). \tag{7}$$

Here $e_i = e_i^n$ is the i^{th} elementary symmetric polynomial. We know that the roots (as a uni-variate polynomial in y) of F are $x_1, x_2, \ldots, x_n$. Therefore, $x_1, x_2, \ldots, x_n$ are algebraic functions in $e_1, e_2, \ldots, e_n$. Thus $x_i = A_i(e_1, e_2, \ldots, e_n)$ for some algebraic function A_i.

Let $C_{\text{Sym}}(x_1, x_2, \ldots, x_n)$ be a circuit of size $L(f_{\text{Sym}})$ computing $f_{\text{Sym}}(x_1, x_2, \ldots, x_n)$. If we could substitute the x_i's by the A_i's in $C_{\text{Sym}}(x_1, x_2, \ldots, x_n)$, we would obtain a circuit for f. But we cannot compute algebraic functions using arithmetic circuits. Now replace e_n by $e_n + (-1)^{n-1}$ in Equation 7. Thus the new $F(y, e_1, e_2, \ldots, e_n)$ is:

$$F(y, e_1, e_2, \ldots, e_n) = y^n - e_1 y^{n-1} + \ldots + (-1)^n(e_n + (-1)^{n-1}). \tag{8}$$

We call the roots of $F(y)$ in Equation 8 again $A_1, A_2, \ldots, A_n$. By using Lemma 14, we know that the A_i's are in $\mathbb{C}[[e_1, e_2, \ldots, e_n]]$. The following Equation 9 follows from the above discussion:

$$C_{\text{Sym}}(A_1, A_2, \ldots, A_n) = f(e_1, e_2, \ldots, e_n + (-1)^{n-1}). \tag{9}$$

To compute f, it is enough to substitute the degree d truncations of the A_i's in Equation 9, instead of the exact infinite power series A_i. Let $D_1, D_2, \ldots, D_n$ be the outputs of the circuit D obtained by applying Theorem 19 with degree $2^{\lceil \log d \rceil}$. We substitute the $x_i \to D_i$ in the circuit $C_{\text{Sym}}(x_1, x_2, \ldots, x_n)$. We obtain the following equality:

$$h \overset{\text{def}}{=\!=} C_{\text{Sym}}(D_1, D_2, \ldots, D_n) = f(e_1, e_2, \ldots, e_n + (-1)^{n-1}) + g. \tag{10}$$

In the above Equation 10, g is a polynomial with all its monomials of degree at least $d+1$, *i.e.*, $g \in I^{d+1}$ with $I = (e_1, e_2, \ldots, e_n)$. Hence it follows that:

$$f(e_1, e_2, \ldots, e_n + (-1)^{n-1}) = \sum_{i=0}^{d} h^{[i]}.$$

By applying Theorem 19, we know that $L^{f_{\text{Sym}}}(h) \leq (n^2 \log d + n \log^2 d)$. Note that the degree of each D_i is at most $2^{\lceil \log d \rceil}$, which is at most $2d$. Thus the degree of h is at most $2dd_{\text{Sym}}$. By using Lemma 12, we conclude that for any $0 \leq i \leq \deg(h)$:

$$L^h(h^{[i]}) \leq O(n \cdot d \cdot d_{\text{Sym}}). \tag{11}$$

Equation 11 implies that $L^h(\sum_{i=0}^{d} h^{[i]}) \leq O(n \cdot d^2 \cdot d_{\text{Sym}})$. By using $L^{f_{\text{Sym}}}(h) \leq (n^2 \log d + n \log^2 d)$, we obtain that:

$$\begin{aligned} L^{f_{\text{Sym}}}(\sum_{i=0}^{d} h^{[i]}) &\leq O(n \cdot d^2 \cdot d_{\text{Sym}} \cdot (n^2 \log d + n \log^2 d)) \\ &= \tilde{O}\left(n^3 \cdot d^2 \cdot d_{\text{Sym}}\right). \end{aligned}$$

By using the substitution $e_n \to e_n - (-1)^{n-1}$, we obtain that:

$$L^{f_{\text{Sym}}}(f) \leq \tilde{O}\left(n^3 \cdot d^2 \cdot d_{\text{Sym}}\right).$$

If we use Lemma 13 instead of Lemma 12 in the above argument, we obtain that:

$$\begin{aligned} L(f) &\leq O(d^2(L(f_{\text{Sym}}) + n^2 \log d + n \log^2 d))) \\ &= \tilde{O}\left(d^2 L(f_{\text{Sym}}) + d^2 n^2\right). \end{aligned}$$

This concludes the proof. ◀

▶ Remark. In contrast to results in [7, 6], our results do depend on the degree d. But if the degree d is $\mathsf{poly}(n)$ then our upper bound on $L(f)$ is polynomial in n and $L(f_{\text{Sym}})$. This upper bound was exponential in [7, 6].

5 Hard Symmetric Polynomials

By using Theorem 4, we are ready to prove that there exist *hard* symmetric polynomials. To this end, the following Theorem 20 suffices.

▶ **Theorem 20.** *Let $(f_n)_{n\in\mathbb{N}}$ be a* VNP*-complete family. Then the corresponding symmetric polynomial family $((f_n)_{\mathrm{Sym}})_{n\in\mathbb{N}}$ is* VNP*-complete under c-reductions.*

Proof. Let $d = \deg(f_n)$ and $d_{\mathrm{Sym}} = \deg((f_n)_{\mathrm{Sym}})$. Since $(f_n)_{n\in\mathbb{N}} \in \mathsf{VNP}$, we know that both d and d_{Sym} are polynomially bounded in n. By using Theorem 4, we know that $L^{f_{\mathrm{Sym}}}(f) \leq \tilde{O}\left(n^3 \cdot d^2 \cdot d_{\mathrm{Sym}}\right) = \tilde{O}\left(\mathsf{poly}(n)\right)$. Thus $((f_n)_{\mathrm{Sym}})_{n\in\mathbb{N}}$ is VNP-hard under c-reductions. It is also easy to see that $((f_n)_{\mathrm{Sym}})_{n\in\mathbb{N}}$ is in VNP. Therefore $((f_n)_{\mathrm{Sym}})_{n\in\mathbb{N}}$ is VNP-complete under c-reductions. ◀

▶ **Corollary 21.** *The polynomial family $(q_n)_{n\in\mathbb{N}}$ defined by $q_n \overset{def}{=\!=} (\mathrm{per}_n)_{\mathrm{Sym}}$ is* VNP*-complete under c-reductions. Therefore if* VP$\neq$ VNP*, then the polynomial family $(q_n)_{n\in\mathbb{N}}$ is not in* VP*.*

References

1 E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. Miltersen. On the Complexity of Numerical Analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009. doi:10.1137/070697926.

2 Garrett Birkhoff and Saunders Mac Lane. *A survey of modern algebra*. New York : Macmillan, 4th ed edition, 1977. URL: http://www.gbv.de/dms/hbz/toc/ht000038471.pdf.

3 Ben Blum-Smith and Samuel Coskey. The Fundamental Theorem on Symmetric Polynomials: History's First Whiff of Galois Theory. *The College Mathematics Journal*, 48(1):18–29, 2017. URL: http://www.jstor.org/stable/10.4169/college.math.j.48.1.18.

4 J. N. Bray, M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien. Short presentations for alternating and symmetric groups. *Trans. Amer. Math. Soc.*, 363(6):3277–3285, 2011. doi:10.1090/S0002-9947-2011-05231-1.

5 Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7. Springer Science & Business Media, 2013.

6 Xavier Dahan, Éric Schost, and Jie Wu. Evaluation properties of invariant polynomials. *Journal of Symbolic Computation*, 44(11):1592–1604, 2009. In Memoriam Karin Gatermann. doi:10.1016/j.jsc.2008.12.002.

7 Pierrick Gaudry, Eric Schost, and Nicolas M. Thiéry. Evaluation properties of symmetric polynomials. *International Journal of Algebra and Computation*, 16(3):505–523, 2006. doi:10.1142/S0218196706003128.

8 Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

9 H. T. Kung and J. F. Traub. All Algebraic Functions Can Be Computed Fast. *J. ACM*, 25(2):245–260, April 1978. doi:10.1145/322063.322068.

10 Dick Lipton and Ken Regan. Arithmetic Complexity and Symmetry, July 2009. URL: https://rjlipton.wordpress.com/2009/07/10/arithmetic-complexity-and-symmetry/.

11 Meena Mahajan. Algebraic Complexity Classes. In Manindra Agrawal and Vikraman Arvind, editors, *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, pages 51–75. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-05446-9_4.

12 Tateaki Sasaki and Fujio Kako. Solving multivariate algebraic equation by Hensel construction. *Japan Journal of Industrial and Applied Mathematics*, 16(2):257–285, June 1999. doi:10.1007/BF03167329.

13 Amir Shpilka and Amir Yehudayoff. Arithmetic Circuits: A Survey of Recent Results and Open Questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4):207–388, 2010. doi:10.1561/0400000039.

14 Volker Strassen. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973.

15 L. G. Valiant. Completeness Classes in Algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 249–261, New York, NY, USA, 1979. ACM. doi:10.1145/800135.804419.

A Appendix

A.1 Algebraic complexity theory

Analogous to the idea of classical complexity classes, we can also define algebraic complexity classes. We refer the reader to [5, 11] for a more comprehensive introduction to algebraic complexity classes. In this section, a p-bounded function is simply a polynomially bounded function.

▶ **Definition 22** (Arithmetic Circuit Complexity)**.** For a polynomial $p \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, the (arithmetic) circuit complexity $L(p)$ of p is defined as the size of smallest arithmetic circuit computing p, that is

$$L(p) \overset{\text{def}}{=\!=} \min\{s \mid \exists \text{ size } s \text{ arithmetic circuit computing } p\}.$$

▶ **Definition 23** (p-family)**.** A family (or a sequence) $(f_n)_{n\in\mathbb{N}}$ of (multivariate) polynomials over the field $\mathbb{F}$ is said to be a p-family iff the number of variables as well as the degree of f_n are p-bounded functions of n.

Now we define the notion of *efficient* polynomial families.

▶ **Definition 24** (p-computable)**.** A p-family $(f_n)_{n\in\mathbb{N}}$ is called p-computable iff the arithmetic complexity $L(f_n)$ is a p-bounded function of n.

p-computable polynomial families define the algebraic analogue of the P, called VP.

▶ **Definition 25** (Class VP)**.** The (algebraic complexity) class VP is the set of all p-computable polynomial families.

▶ **Definition 26** (Class VNP)**.** A p-family $(f_n)_{n\in\mathbb{N}}$ is said to be in the (algebraic complexity) class VNP if there exists a polynomial family $(g_n)_{n\in\mathbb{N}} \in \mathsf{VP}$ with $g_n \in \mathbb{F}[x_1, x_2, \ldots, x_{q(n)}]$ such that:

$$f_n(x_1, x_2, \ldots, x_{p(n)}) = \sum_{e\in\{0,1\}^{q(n)-p(n)}} g_n(x_1, x_2, \ldots, x_{p(n)}, e_1, e_2, \ldots, e_{q(n)-p(n)}).$$

Similar to the Boolean case, there is an algebraic notion of reduction also, called the p-projections.

▶ **Definition 27** (Projection)**.** A polynomial $f(x_1, x_2, \ldots, x_n) \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ is said to be a projection of a polynomial $g(y_1, y_2, \ldots, y_m) \in \mathbb{F}[y_1, y_2, \ldots, y_m]$, if there exists a map $\alpha : \{y_1, y_2, \ldots, y_m\} \to \{x_1, x_2, \ldots, x_n\} \cup \mathbb{F}$ such that $f = g$ under the substitution map α. We write $f \leq g$ to denote that f is a projection of g.

▶ **Definition 28** (p-projection). A polynomial family $(f_n)_{n\in\mathbb{N}}$ is said to be a p-projection of a polynomial family $(g)_{n\in\mathbb{N}}$ if there is a p-bounded function $\beta : \mathbb{N} \to \mathbb{N}$ and $n_0 \in \mathbb{N}$ such that:

$\forall n \geq n_0 : f_n \leq g_{\beta(n)}$.

We denote $(f_n)_{n\in\mathbb{N}}$ being a p-projection of $(g)_{n\in\mathbb{N}}$ by $f \leq_p g$.

Definition 9 naturally lends to the following definition:

▶ **Definition 29** ([5]). Let $f = (f_n)$, $g = (g_n)$ be two polynomial families. We say that f is a c-reduction of g, denoted by $f \leq_c g$, iff there is a p-bounded function $t : \mathbb{N} \to \mathbb{N}$ such that $L^{g_{t(n)}}(f_n)$ is a p-bounded function of n.

Now the idea of completeness and hardness can be defined as in the case of Boolean case.

▶ **Definition 30** (Hardness and Completeness). For an algebraic complexity classic $\mathcal{C}$, a p-family $f = (f_n)_{n\in\mathbb{N}}$ is said to be a $\mathcal{C}$-hard if $g \leq_p f$ for all $g \in \mathcal{C}$, f is called $\mathcal{C}$-complete if f is $\mathcal{C}$-hard and $f \in \mathcal{C}$. Similarly, we can define the notion of hardness under c-reductions.

▶ **Theorem 31** ([15], see also [5]). *Over the fields $\mathbb{F}$ with $\mathrm{char}(\mathbb{F}) \neq 2$, the p-family (per_n) is* VNP*-complete.*

The holy grail of algebraic complexity theory is to show that VP $\neq$ VNP. For this it is enough to show that $(\mathrm{per}_n) \notin$ VP over fields $\mathbb{F}$ with $\mathrm{char}(\mathbb{F}) \neq 2$. Note that per_n and $\det_n$ are the same polynomials if $\mathrm{char}(\mathbb{F}) = 2$. Thus if $\mathrm{char}(\mathbb{F}) = 2$ then $(\mathrm{per}_n) \in$ VP hence (per_n) is unlikely to VNP-complete over fields of characteristic two.

A.2 Missing proofs

We provide some proofs of known results used in this work for the reader's convenience.

Proof of Lemma 13. Let C be a circuit of size $L(f)$ computing f. We create $m+1$ copies of each arithmetic gate in C, *i.e.*, each$\{+,-,\times\}$-gate G has $m+1$ copies $G_0, G_1, \ldots, G_m$. If the gate G computes the polynomial g then G_i computes the polynomial $g^{[i]}$. This can be trivially done for input and constant gates. Suppose $G = G_1 + G_2$ is a "$+$" gate and g_1, g_2 are the polynomials computed by gates G_1 and G_2 respectively. Now we know that $g^{[i]} = g_1^{[i]} + g_2^{[i]}$ for all $i \in [[m]]$. A similar statement is true for "$-$" gates also. If $G = G_1 \times G_2$ is a "$\times$" gate, then we have the following equality:

$$g^{[i]} = \sum_{j=0}^{i} g_1^{[j]} \cdot g_2^{[i-j]}. \tag{12}$$

Suppose we already have the gates for $g_1^{[j]}, g_2^{[j]}$ for all $j \in [[m]]$. Then one $g^{[i]}$ in Equation 12 can be computed using $2(i+1)$ additional gates. Thus the gates $G_0, G_1, \ldots, G_m$ can be constructed using $\sum_{k=0}^{m} 2(k+1) = O(m^2)$ gates. Hence every gate in C corresponds to at most $O(m^2)$ new gates. Therefore:

$$L(\{f^{[0]}, f^{[1]}, \ldots, f^{[m]}\}) \leq O(m^2 L(f)).$$ ◀

Proof of Theorem 16. Our claim is obviously true for $k = 0$. We prove the theorem by induction on k. We prove it simultaneously for all $i \in [n]$, so for the sake of brevity we use $A^{(k)}$ to denote $A_i^{(k)}$ and α to denote α_i. Consider the following equalities for a root $A = A_i$ of $F(y)$:

$$\begin{aligned} 0 &= F(A) = F(A^{(k)} + (A - A^{(k)})) \\ &= F(A^{(k)}) + (A - A^{(k)})F'(A^{(k)}) + \sum_{j>1} \frac{(A - A^{(k)})^j}{j!} F^{(j)}(A^{(k)}). \end{aligned} \tag{13}$$

Here $F^{(j)} \overset{\text{def}}{=\!=} \frac{\partial^j F(y,u_1,u_2,\ldots,u_n)}{\partial y^j}$ is the j^{th} derivative of $F(y)$ with respect to y. Since α is a simple root of $F(y,0,0,\ldots,0)$, we know that:

$$\text{constant term of } F'(A^{(k)}) = F'(\alpha) \neq 0.$$

Thus $F'(A_k)$ is invertible in the ring $\mathbb{C}[[u_1,u_2,\ldots,u_n]]$ of power series. Therefore we have:

$$\begin{aligned} A - A^{(k+1)} &= A - \left(A^{(k)} - \frac{F(A^{(k)})}{F'(A^{(k)})}\right) \\ &= -\sum_{j>1} \frac{(A-A^{(k)})^j F^{(j)}(A^{(k)})}{j!\cdot F'(A^{(k)})}. && \text{(by using Equation 13)} \end{aligned}$$

Since $A - A^{(k)} \in I^{2^k}$, the right hand side of the above equation is in $I^{2^{k+1}}$. Thus $A^{(k+1)} \equiv A \bmod I^{2^{k+1}}$. ◀

Proof of Lemma 17. We again prove it by induction on k, the induction hypothesis is that $p_k \equiv g^{-1} \bmod I^{2^k}$. This induction hypothesis is trivially true for $k=0$. Consider:

$$\begin{aligned} \frac{1}{g} - p_{k+1} &= \frac{1}{g} - p_k(2 - g\cdot p_k) \\ &= g\cdot(\frac{1}{g^2} - \frac{2p_k}{g} + p_k^2) \\ &= g\cdot(\frac{1}{g} - p_k)^2. \end{aligned}$$

By using the induction hypothesis, we know that $\frac{1}{g} - p_k \in I^{2^k}$. Therefore it implies that $\frac{1}{g} - p_{k+1} \in I^{2^{k+1}}$. Now the lemma follows from the fact that $\ell = \lceil \log d \rceil$. ◀