Electronic Colloquium on Computational Complexity, Report No. 125 (2007)

The Black-Box Query Complexity of Polynomial Summation

Ali Juma*
Department of Computer Science
University of Toronto
Toronto, Canada
ajuma@cs.toronto.edu

Valentine Kabanets†
School of Computing Science
Simon Fraser University
Vancouver, Canada
kabanets@cs.sfu.ca

Charles Rackoff
Department of Computer Science
University of Toronto
Toronto, Canada
rackoff@cs.toronto.edu

Amir Shpilka‡
Faculty of Computer Science
Technion
Haifa, Israel
shpilka@cs.technion.ac.il

October 11, 2007

Abstract

For any given Boolean formula $\phi(x_1, \ldots, x_n)$, one can efficiently construct (using *arithmetization*) a low-degree polynomial $p(x_1, \ldots, x_n)$ that agrees with ϕ over all points in the Boolean cube $\{0,1\}^n$; the constructed polynomial p can be interpreted as a polynomial over an arbitrary field $\mathbb{F}$. The problem $\#SAT$ (of counting the number of satisfying assignments of ϕ) thus reduces to the polynomial summation $\sum_{x \in \{0,1\}^n} p(x)$. Motivated by this connection, we study the *query complexity* of the polynomial summation problem: Given (oracle access to) a polynomial $p(x_1, \ldots, x_n)$, compute $\sum_{x \in \{0,1\}^n} p(x)$. Obviously, querying p at all 2^n points in $\{0,1\}^n$ suffices. Is there a field $\mathbb{F}$ such that, for every polynomial $p \in \mathbb{F}[x_1, \ldots, x_n]$, the sum $\sum_{x \in \{0,1\}^n} p(x)$ can be computed using fewer than 2^n queries from $\mathbb{F}^n$? We show that the simple upper bound 2^n is in fact *tight* for *any* field $\mathbb{F}$ in the *black-box model* where one has only oracle access to the polynomial p. We prove these lower bounds for the *adaptive* query model where the next query can depend on the values of p at previously queried points. Our lower bounds hold even for polynomials that have degree at most 2 in each variable. In contrast, for polynomials that have degree at most 1 in each variable (i.e., multilinear polynomials), we observe that a *single* query is sufficient over any field of characteristic other than 2. We also give query lower bounds for certain extensions of the polynomial summation problem.

1 Introduction

One of the biggest challenges in complexity theory is to determine the circuit complexity of functions such as *SAT* or *Permanent*. While both of these functions are commonly believed to require

*Partially supported by an NSERC postgraduate scholarship.

†Part of this research was done while the author was a postdoctoral fellow at the University of California, San Diego, supported by NSERC.

‡Part of the research was done while the author was a postdoctoral fellow at Harvard and M.I.T. Part of the research was supported by the Israel Science Foundation (grant number 439/06).

ISSN 1433-8092

superpolynomial circuit complexity, not even superlinear lower bounds are known for the general Boolean or arithmetic computational model; for restricted circuit models, superpolynomial lower bounds for *Permanent* are known (for the most recent results see [Raz04] and the references therein). Could it be that these problems have "small" (say, subexponential) circuit complexity? If they do, what would these small circuits look like?

In this paper, we consider one natural approach to constructing small circuits for the problem $\#SAT$, and prove that (the most naive version of) this approach fails.

1.1 Circuits for $\#SAT$

Recall that $\#SAT$ is the problem of computing the number of satisfying assignments to an input Boolean formula, say given in a conjunctive normal form with each clause of size at most 3 (i.e., 3-cnf). A possible approach to designing a small circuit for $\#SAT$ is as follows.

First, arithmetize the given Boolean formula $\phi(x_1, \ldots, x_n)$; arithmetization is a standard technique that has been successfully applied in the design of efficient interactive proof systems for such complexity classes as PSPACE and NEXP [LFKN92, Sha92, BFL91]. Arithmetizing a 3cnf formula $\phi(x_1, \ldots, x_n)$ is done inductively. For a variable x, the polynomial p_x is also x; for $\neg x$, the polynomial $p_{\neg x}$ is $1 - x$. For a clause $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ of literals ℓ_1, ℓ_2, ℓ_3, the polynomial p_c is $1-(1-p_{\ell_1})(1-p_{\ell_2})(1-p_{\ell_3})$. Finally, for the conjunction ϕ of clauses $c_1, \ldots, c_m$, the polynomial p_ϕ is $\prod_{i=1}^m p_{c_i}$. Thus we efficiently convert a given Boolean formula $\phi(x_1, \ldots, x_n)$ into a multivariate polynomial $p_\phi(x_1, \ldots, x_n)$ such that, for every $b = (b_1, \ldots, b_n) \in \{0,1\}^n$, $p_\phi(b) = 1$ if b is a satisfying assignment for ϕ, and $p_\phi(b) = 0$ otherwise.

We make several simple observations about the arithmetization. First, note that the arithmetization of a given 3cnf formula ϕ yields an arithmetic formula for p_ϕ of size *linear* in the size of ϕ. Also note that this arithmetic formula uses only constants 1 and -1, and so can be evaluated over *any* field $\mathbb{F}$; i.e., we can view the resulting polynomial p_ϕ as a polynomial from $\mathbb{F}[x_1, \ldots, x_n]$ for any field $\mathbb{F}$. Finally, note that the polynomial p_ϕ has total degree at most *linear* in the size of the formula ϕ.

Once we have the polynomial p_ϕ, our task of counting the number of satisfying assignments of the formula ϕ is reduced to that of the polynomial summation of p_ϕ over all $\{0,1\}$ values for the variables. Can this task be accomplished by circuits of subexponential size?

We do not know the answer to this question. The work on interactive proofs shows that an unlimited-power prover can prove the value of this summation to a (probabilistic) polynomial time verifier, and the verifier only has to evaluate p_ϕ at one point. In fact, this proof system works even if the polynomial is arbitrarily complex, as long as it is of low degree and the verifier has access to an oracle for evaluating it. This suggests the question: can we replace interaction by nonuniformity? Are there circuits with oracle access to an arbitrary low-degree polynomial in n variables, such that the circuits have polynomial (in n) size and compute the sum of the polynomial over all $0, 1$ values of the variables?

Here we allow the circuits to evaluate an oracle polynomial over an *arbitrary* field $\mathbb{F}$; as remarked above, evaluation over any field is possible for polynomials obtained by arithmetizing 3cnf formulas. For example, we can take $\mathbb{F}$ to be the field $\mathbb{Q}$ of rational numbers, and ask for the sum $\sum_{x\in\{0,1\}^n} p(x)$ when a polynomial $p(x_1, \ldots, x_n)$ is viewed as a polynomial from $\mathbb{Q}[x_1, \ldots, x_n]$. Here the hope may be that querying p at few "special" points in $\mathbb{Q}^n$ (outside of the Boolean cube $\{0,1\}^n$) would suffice for determining the sum $\sum_{x\in\{0,1\}^n} p(x)$. Is that possible? In general, is there a field $\mathbb{F}$ such that querying $p \in \mathbb{F}[x_1, \ldots, x_n]$ at significantly fewer than 2^n points from $\mathbb{F}^n$ allows one to determine the sum of p over the Boolean cube?

If the answer were "yes", this could yield very special kinds of small circuits for $\#SAT$. We show here that the answer is "no". In fact, 2^n oracle queries are *necessary over any field* $\mathbb{F}$.

1.2 Polynomial summation problem and our results

To obtain our negative results, we establish an exponential lower bound on the black-box query complexity of the following problem.

Problem 1. Polynomial Summation Problem
Given: Oracle access to a multivariate polynomial $p(x_1, \ldots, x_n)$ over a field $\mathbb{F}$.
Compute: $\sum_{x \in \{0,1\}^n} p(x)$.

We prove that the trivial upper bound of 2^n queries is actually tight for any field $\mathbb{F}$. Independently from our work, this result was also proved by Scott Aaronson and Avi Wigderson [S. Aaronson, personal communication, October 2007].

Theorem 2 (Main). *Let $\mathbb{F}$ be any field. Let A be an algorithm that, after asking k (possibly adaptive) queries to a given n-variate polynomial $p(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ of degree at most 2 in each variable, outputs the value $\sum_{x \in \{0,1\}^n} p(x)$. Then $k \geqslant 2^n$.*

Our lower bound 2^n holds even for polynomials of degree at most 2 in each variable. This should be contrasted with the case of *multilinear* polynomials (of degree at most 1 in each variable), for which we prove the following.

Observation 3. *Let $\mathbb{F}$ be any field of characteristic other than 2. Let $p(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be any multilinear polynomial. Then*

$$\sum_{x \in \{0,1\}^n} p(x) = 2^n \cdot p(1/2, \ldots, 1/2).$$

Thus, going from input polynomials of degree at most 1 in each variable to those of degree at most 2 in each variable produces an exponential jump in the query complexity of polynomial summation.

Observe that if the polynomial p_ϕ obtained by arithmetizing a Boolean formula ϕ were always multilinear, then $\#SAT$ would be easily solvable in polynomial time, using Theorem 3. However, even starting from a 3cnf ϕ where no variable occurs in more than three clauses, the best known polynomial that can be efficiently obtained from ϕ is not multilinear, but rather of degree at most 2 in each variable. Getting degree at most 3 is straightforward, just by arithmetizing ϕ. With an additional simple trick, we can achieve degree at most 2 in each variable; see Theorem 19 in the Appendix for details.

Our proofs rely on some basic tools from linear algebra. For a large number of beautiful applications of such tools to computer science and combinatorics, see [BF92, Juk01].

Remainder of the paper. Section 2 contains some basic facts of linear algebra. In Section 3, we prove Theorem 3. Our main query complexity lower bounds (yielding Theorem 2) are proved in Section 4. In Section 5, we consider an extension of our query model, and prove some lower bounds for this extension. We conclude with open questions in Section 6.

2 Preliminaries

2.1 Notation

For a natural number n, we will often denote by $[n]$ the set $\{1, \ldots, n\}$. For a subset $S \subseteq [n]$, we denote by $\bar{S}$ the complement of S in $[n]$, i.e., $\bar{S} = [n] \setminus S$.

Let $\mathbb{F}$ be any field. We denote by $\mathrm{char}(\mathbb{F})$ the *characteristic* of $\mathbb{F}$.

2.2 Linear algebra basics

We will need some basics of linear algebra (see, e.g., [Str88]). For two n-dimensional column vectors $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ from $\mathbb{F}^n$, for some field $\mathbb{F}$, their *inner product*[1] is defined as $\langle u, v\rangle = u^T v = \sum_{i=1}^n u_i \cdot v_i$, where u^T denotes the transpose of u. More generally, given any symmetric and invertible $n \times n$ matrix $W \in \mathbb{F}^{n\times n}$, the inner product of u and v with respect to W is defined as $\langle u, v\rangle_W = u^T W v$; when $W = I$ is the identity matrix, we obtain the original definition of inner product.

Clearly for any vector u, and any linear combination $\sum_{i=1}^k \alpha_i \cdot v^i$ of vectors v^i,

$$\langle u, \sum_{i=1}^k \alpha_i \cdot v^i\rangle_W = \sum_{i=1}^k \alpha_i \cdot \langle u, v^i\rangle_W.$$

2.3 Polynomials

A *monomial* over $\mathbb{F}$ in variables $x_1, \ldots, x_n$ is the product $m(x_1, \ldots, x_n) = \prod_{i=1}^n x_i^{d_i}$, where $d_i \geqslant 0$ is the degree of variable x_i in the monomial m. A *polynomial* is a linear combination of distinct monomials m_j, i.e., $p(x_1, \ldots, x_n) = \sum_{j=1}^k c_j \cdot m_j(x_1, \ldots, x_n)$ for $c_j \in \mathbb{F}$. For each $1 \leqslant i \leqslant n$, the *degree* of variable x_i in the polynomial p is the maximal degree of x_i over all monomials of p. A monomial $m(x_1, \ldots, x_n) = \prod_{i=1}^n x_i^{d_i}$ is called *multilinear* if $d_i \leqslant 1$ for every $1 \leqslant i \leqslant n$; note that there are exactly 2^n distinct multilinear monomials. A polynomial is multilinear if all of its monomials are multilinear.

For every subset $S \subseteq [n]$, we define the multilinear monomial $m_S = \prod_{i\in S} x_i$. We have the following simple observations.

Lemma 4. *Let $S \subseteq [n]$ be arbitrary. Over the field of characteristic 2, we have*

$$\sum_{\bar{b}\in\{0,1\}^n} m_S(\bar{b}) = \begin{cases} 0 & \text{if } S \neq [n] \\ 1 & \text{if } S = [n] \end{cases}.$$

Proof. If for some i we have $i \notin S$, then $\sum_{\bar{b}\in\{0,1\}^n} m_S(\bar{b}) = 2\sum_{\bar{b}\in\{0,1\}^{n-1}} m_{S\setminus\{i\}}(\bar{b}) \equiv_2 0$. ☐

Corollary 5. *Let $S, T \subseteq [n]$ be arbitrary. Over the field of characteristic 2, we have*

$$\sum_{\bar{b}\in\{0,1\}^n} m_S(\bar{b}) \cdot m_T(\bar{b}) = \begin{cases} 1 & \text{if } \bar{S} \subseteq T \\ 0 & \text{if } \bar{S} \not\subseteq T \end{cases}.$$

Proof. First note that for $b \in \{0, 1\}$, we have $b^2 = b$, and so $m_S(\bar{b}) \cdot m_T(\bar{b}) = m_{S\cup T}(\bar{b})$. Next observe that $S \cup T = [n]$ iff $\bar{S} \subseteq T$. The conclusion now follows from Lemma 4. ☐

[1] More precisely, when $\mathrm{char}(\mathbb{F}) > 0$ this is not an inner product but rather just a bilinear map.

Over any field $\mathbb{F}$ of characteristic other than 2, simple linear transformations of variables can take us between $\{0,1\}^n$ and $\{1,-1\}^n$. The mapping $x \mapsto 1-2x$ applied to each variable x_i takes us from $\{0,1\}^n$ to $\{1,-1\}^n$, and $x \mapsto (1-x)/2$ takes us back. Since both of these transformations are linear, they do not change the degree of any variable x_i in a given polynomial $p(x_1,\dots,x_n)$. In particular, a multilinear polynomial will remain multilinear after such transformations.

For fields of characteristic other than 2, we re-define the polynomial summation problem as follows.

Problem 6. Polynomial Summation Problem over fields of characteristic other than 2
Given: Oracle access to a multivariate polynomial $p(x_1,\dots,x_n)$ over a field $\mathbb{F}$ with $\mathrm{char}(\mathbb{F}) \neq 2$.
Compute: $\sum_{x\in\{1,-1\}^n} p(x)$.

The advantage of working over $\{1,-1\}^n$ stems from the following.

Lemma 7. *Let $m(x_1,\dots,x_n)$ be a monomial such that the degree of some variable x_i in m is exactly* 1*. Over the field of characteristic other than 2, we have $\sum_{\bar{b}\in\{1,-1\}^n} m(\bar{b}) = 0$.*

Proof. Write $m = x_i \cdot m'$, where the monomial m' does not contain x_i. Then $\sum_{\bar{b}\in\{1,-1\}^n} m(\bar{b}) = \sum_{a\in\{1,-1\}} \sum_{\bar{b}'\in\{1,-1\}^{n-1}} a \cdot m'(\bar{b}') = \sum_{a\in\{1,-1\}} a \cdot \sum_{\bar{b}'\in\{1,-1\}^{n-1}} m'(\bar{b}') = 0$. ☐

Corollary 8. *Let $S, T \subseteq [n]$ be arbitrary. Over the field of characteristic other than 2, we have*

$$\sum_{\bar{b}\in\{1,-1\}^n} m_S(\bar{b}) \cdot m_T(\bar{b}) = \begin{cases} 0 & \text{if } S \neq T \\ 2^n & \text{if } S = T \end{cases}.$$

Proof. If $S \neq T$, then the monomial $m_S \cdot m_T$ will contain at least one variable of degree 1, and the conclusion follows by Lemma 7. If $S = T$, then the monomial $m_S \cdot m_T$ has degree 0 or 2 in each variable, and hence it is equal to 1 at each point $\bar{b} \in \{1,-1\}^n$. ☐

3 The one-query upper bound for multilinear polynomials

Here we prove Observation 3 stated in the Introduction. For convenience, we re-state it below.

Observation 3. *Let $p(x_1,\dots,x_n)$ be any multilinear polynomial over a field of characteristic other than 2. Then*

$$\frac{1}{2^n} \sum_{\bar{b}\in\{0,1\}^n} p(\bar{b}) = p(1/2,\dots,1/2).$$

Proof. Let $p = \sum_{j=1}^{k} c_j \cdot m_j$ for distinct multilinear monomials m_j, where $k = 2^n$. Then

$$\frac{1}{2^n} \sum_{\bar{b}\in\{0,1\}^n} p(\bar{b}) = \mathbf{Exp}[p],$$

where the expectation $\mathbf{Exp}$ is taken with respect to the uniform distribution on the Boolean cube $\{0,1\}^n$. By linearity of expectation, $\mathbf{Exp}[p] = \sum_{j=1}^{k} c_j \cdot \mathbf{Exp}[m_j]$. For each multilinear monomial $m = x_1^{d_1} \dots x_n^{d_n}$, where $d_j \in \{0,1\}$, $\mathbf{Exp}[m] = \prod_{j=1}^{n} \mathbf{Exp}[x_j^{d_j}]$. For $d_j = 0$, $\mathbf{Exp}[x_j^{d_j}] = 1$; for $d_j = 1$, $\mathbf{Exp}[x_j^{d_j}] = 1/2$. Hence, $\mathbf{Exp}[m_j] = m_j(1/2,\dots,1/2)$, and so $\mathbf{Exp}[p] = p(1/2,\dots,1/2)$.

An alternative proof is as follows. First, change from the domain $\{0,1\}$ to $\{1,-1\}$ by applying the linear transformation $x \mapsto 1-2x$ to each variable of p. The resulting polynomial p' remains multilinear. Since the sum over $\{1,-1\}^n$ of any non-constant multilinear monomial is 0 by Lemma 7, all but the constant monomial of p' will disappear in polynomial summation. Thus, $\sum_{\bar{b}\in\{1,-1\}^n} p'(\bar{b}) = 2^n p'(0,\ldots,0)$. But, by the linear transformation $x \mapsto (1-x)/2$ from the domain $\{1,-1\}$ to $\{0,1\}$, we have $p'(0,\ldots,0) = p(1/2,\ldots,1/2)$. □

4 Lower bounds

4.1 The case of non-adaptive algorithms

Here we prove a lower bound for the polynomial summation problem for a special kind of non-adaptive algorithms. This will be used in the next section for the case of general, adaptive algorithms.

For a sequence Q of k points $q_1,\ldots,q_k$ from $\mathbb{F}^n$, and a sequence R of k non-zero field elements $r_1,\ldots,r_k \in \mathbb{F}$, define the algorithm $A_{Q,R}$ as follows: Given oracle access to a polynomial $p(x_1,\ldots,x_n)$, output $\sum_{i=1}^k r_i \cdot p(q_i)$.

Theorem 9. *Let $\mathbb{F}$ be any field. Let $Q = (q_1,\ldots,q_k) \in (\mathbb{F}^n)^k$ and let $R = (r_1,\ldots,r_k) \in (\mathbb{F}\setminus\{0\})^k$. Suppose that*

$$\sum_{\bar{b}\in\{0,1\}^n} m(\bar{b}) = \sum_{i=1}^k r_i \cdot m(q_i)$$

for every monomial $m(x_1,\ldots,x_n)$ of degree at most 2 in each variable. Then $k \geqslant 2^n$.

Proof. First we introduce some notation. For every monomial $m(x_1,\ldots,x_n)$, let μ_m be the vector of evaluations of m over the points in Q, i.e.,

$$\mu_m = (m(q_1),\ldots,m(q_k)).$$

For each subset $S \subseteq [n]$, let $m_S = \prod_{i\in S} x_i$, and let $\mu_S = \mu_{m_S}$.

We will show that the vectors μ_S, for $S \subseteq [n]$, are linearly independent over $\mathbb{F}$. Since there are exactly 2^n distinct subsets $S \subseteq [n]$, we get that the dimension of each μ_S must be at least 2^n, i.e., $k \geqslant 2^n$.

Our proof of linear independence will be different for the case of fields of characteristic 2 and those of characteristic other than 2. We first give a proof for the slightly simpler case of $\mathrm{char}(\mathbb{F}) \neq 2$.

By the discussion in Section 2.3, we can assume w.l.o.g. that

$$\sum_{\bar{b}\in\{1,-1\}^n} m(\bar{b}) = \sum_{i=1}^k r_i \cdot m(q_i)$$

for every monomial $m(x_1,\ldots,x_n)$ of degree at most 2 in each variable. Let W be the $k\times k$ diagonal matrix with the vector R on the diagonal; note that W is symmetric and invertible (since all $r_i \neq 0$). For any multilinear monomials m and m', we get from the assumption of the theorem and from Corollary 8 that

$$\langle \mu, \mu' \rangle_W = \begin{cases} 0 & \text{if } m \neq m' \\ 2^n & \text{if } m = m' \end{cases}. \tag{1}$$

Now we show that the vectors μ_S defined above are linearly independent. Suppose $\sum_{S\subseteq[n]} \alpha_S \cdot \mu_S = 0$, for some field elements α_S's. Then for every $T \subseteq \{1,\dots,n\}$, we have

$$0 = \langle \mu_T, \sum_{S\subseteq[n]} \alpha_S \cdot \mu_S\rangle_W = \sum_{S\subseteq[n]} \alpha_S \cdot \langle \mu_T, \mu_S\rangle_W = 2^n \cdot \alpha_T,$$

where the last equality is by Equation (1). Since $\mathrm{char}(\mathbb{F}) \neq 2$, we conclude that all $\alpha_T = 0$. Hence the vectors $\{\mu_S\}_{S\subseteq[n]}$ are linearly independent.

Next we prove the linear independence for the case of $\mathrm{char}(\mathbb{F}) = 2$. Again, let W be the $k \times k$ diagonal matrix with the vector R on the diagonal. Using the assumption of the theorem and Corollary 5, we get that for any $S, T \subseteq [n]$,

$$\langle \mu_S, \mu_T\rangle_W = \begin{cases} 1 & \text{if } \bar{S} \subseteq T \\ 0 & \text{if } \bar{S} \not\subseteq T \end{cases}. \tag{2}$$

Suppose $\sum_{S\subseteq[n]} \alpha_S \cdot \mu_S = 0$, for some field elements α_S's not all of which are zero. Let $S_0 \subseteq [n]$ be any subset such that

1. $\alpha_{S_0} \neq 0$, and
2. $\alpha_S = 0$ for every S such that $S_0 \subsetneq S$.

Note that such an S_0 exists unless all $\alpha_S = 0$. Condition (2) means that α_S can be nonzero only for $S = S_0$ or for S such that $S_0 \not\subseteq S$. It follows that

$$0 = \langle \mu_{\bar{S}_0}, \sum_{S\subseteq[n]} \alpha_S \cdot \mu_S\rangle_W = \sum_{S\subseteq[n]} \alpha_S \cdot \langle \mu_{\bar{S}_0}, \mu_S\rangle_W = \alpha_{S_0} \cdot \langle \mu_{\bar{S}_0}, \mu_{S_0}\rangle_W + \sum_{S\subseteq[n]: S_0 \not\subseteq S} \alpha_S \cdot \langle \mu_{\bar{S}_0}, \mu_S\rangle_W = \alpha_{S_0},$$

where the last equality is by Equation (2). This contradicts our choice of $\alpha_{S_0} \neq 0$. Hence we get that all $\alpha_S = 0$ in this case as well. □

4.2 The case of adaptive algorithms

In the previous subsection, we gave the lower bound 2^n on the number k of queries that any restricted algorithm $A_{Q,C}$ must make in order to solve polynomial summation for all n-variate *monomials* of degree at most 2 in each variable. Here we prove the same lower bound for general algorithms, but only in the case where the general algorithm is supposed to solve polynomial summation for any *polynomial* of degree at most 2 in each variable.

Theorem 10. *Let A be an algorithm that, after asking k queries to a given n-variate polynomial $p(x_1,\dots,x_n)$, outputs some value a. If $k < 2^n$, then there will exist an n-variate polynomial p of degree at most 2 in each variable such that the algorithm A makes a mistake on this p, i.e., $\sum_{\bar{b}\in\{0,1\}^n} p(\bar{b}) \neq a$.*

The proof of this theorem will follow from the following lemma.

Lemma 11. *Let A be an algorithm that, after asking k queries to a given n-variate polynomial $p \in \mathbb{F}[x_1,\dots,x_n]$ of degree at most 2 in each variable, outputs $\sum_{\bar{b}\in\{0,1\}^n} p(\bar{b})$. Then there exist sequences $Q = (q_1,\dots,q_k) \in (\mathbb{F}^n)^k$ and $R = (r_1,\dots,r_k) \in \mathbb{F}^k$ such that the algorithm $A_{Q,R}$ solves the polynomial summation problem for all polynomials $p \in \mathbb{F}[x_1,\dots,x_n]$ of degree at most 2 in each variable.*

Proof. Let $p(x_1, \dots, x_n) = \sum_{j=1}^{N} c_j \cdot m_j(x_1, \dots, x_n)$, where c_j's are yet unspecified coefficients and m_j's are monomials of degree at most 2 in each variable; the number of these monomials is $N = 3^n$. For each query point $q \in \mathbb{F}^n$ used by the algorithm A on p, it gets as an answer the linear combination of coefficients of p, namely, $\sum_{j=1}^{N} m_j(q) \cdot c_j$. The algorithm A may be adaptive and choose its next query point depending on the answers it has received so far.

Imagine that each of the k queries asked by A is answered by 0. Let $q_1, \dots, q_k \in \mathbb{F}^n$ be the sequence of queries asked by A in that case, and let a be the output produced by A. For each point q_i, $1 \leqslant i \leqslant k$, define the vector $m_{q_i} = (m_1(q_i), \dots, m_N(q_i))$. Denoting the vector $(c_1, \dots, c_N)$ by c, we can write the answer $p(q_i)$ obtained by the algorithm A to its query q_i as the inner product $\langle m_{q_i}, c\rangle$. On the other hand, the sum of p over the cube $\{0,1\}^n$ is also a linear combination of the coefficients of p, i.e., $\sum_{\bar{b} \in \{0,1\}^n} p(\bar{b}) = \sum_{j=1}^{N} s_j \cdot c_j$ for some field elements $s_1, \dots, s_N$. Denote the vector $(s_1, \dots, s_N)$ by s. Then the correct answer for polynomial summation of p can be written as $\langle s, c\rangle$.

To summarize, after k queries the algorithm A knows k inner products $\langle m_{q_i}, c\rangle$, for $1 \leqslant i \leqslant k$, and claims that the inner product $\langle s, c\rangle$ equals a. There are two cases to consider:

1. the vector s is *not* in the vector space spanned by $m_{q_1}, \dots, m_{q_k}$, and

2. the vector s is in that vector space.

Without loss of generality we may assume that the vectors $m_{q_1}, \dots, m_{q_k}$ are linearly independent. Otherwise, we can simply consider a smaller subset of linearly independent vectors $m'_1, \dots, m'_{k'}$ corresponding to fewer queries $q'_1, \dots, q'_{k'}$; the inner products $\langle m'_i, c\rangle$ would also give us the inner products $\langle m_q, c\rangle$ for all vectors m_q that are linear combinations of m'_js.

In Case 1, we know that the $k+1$ vectors $s, m_{q_1}, \dots, m_{q_k}$ are linearly independent. Let us write these vectors as rows of a $(k+1) \times N$ matrix M. Let $r = (a+1, 0, \dots, 0)$ be a $(k+1)$-dimensional column vector with $a+1$ in the first coordinate and 0 in all the rest. Clearly, the system of equations $Mc = r$ in unknowns $c_1, \dots, c_N$ will always have at least one solution (as the rows of M are linearly independent). Such a solution determines an n-variate polynomial p of degree at most 2 in each variable whose sum over the cube $\{0,1\}^n$ is $a+1$, but the algorithm A on input p will output $a \neq a+1$. Hence this case cannot happen, and we are left with the second case.

In Case 2, we have k query points $q_1, \dots, q_k$ such that $s = \sum_{j=1}^{k} r_j \cdot m_{q_j}$ for some sequence of field elements $r_1, \dots, r_k$. Without loss of generality, we may assume that all $r_j \neq 0$; otherwise, we can consider the summation over only those j where $r_j \neq 0$.

It follows that $\langle s, c\rangle = \sum_{j=1}^{k} r_j \cdot \langle m_{q_j}, c\rangle$ for *every* vector of coefficients c defining some polynomial p. By the definition of $\langle s, c\rangle$ and $\langle m_{q_j}, c\rangle$, we have a fixed sequence R of queries $q_1, \dots, q_k$ and a fixed sequence C of field elements $r_1, \dots, r_k$ such that, for *every* n-variate polynomial p of degree at most 2 in each variable,

$$\sum_{\bar{b} \in \{0,1\}^n} p(\bar{b}) = \sum_{i=1}^{k} r_i \cdot p(q_i),$$

as required. □

Proof of Theorem 10. The proof is immediate from Lemma 11 and Theorem 9. □

5 Allowing more non-uniformity

5.1 Polynomial summation with advice

Theorem 10 implies that there cannot be a family of polynomial-sized circuits solving the polynomial summation problem in the black-box model. In this section, we relax our requirement somewhat. More precisely, we allow a *list* of L algorithms $A_1, \dots, A_L$, for some finite L. The requirement now is the following: For every low-degree polynomial $p(x_1, \dots, x_n)$, there is at least one algorithm A_i on the list such that A_i, with oracle access to p, correctly solves the polynomial summation problem for that p.

Equivalently, this relaxation of the query model can be viewed as allowing a small amount of *advice* that can depend on the input polynomial p. Namely, given p, we allow about $\log L$ bits of advice that specify which of the L algorithms on the list should be used for this particular input polynomial p.

How does the query complexity of polynomial summation change in this new model? We have the following.

Theorem 12. *Let $\mathbb{F}$ be any finite field. Suppose there is a list of $L < |\mathbb{F}|$ algorithms $A_1, \dots, A_L$ such that the following holds. For every polynomial $p \in \mathbb{F}[x_1, \dots, x_n]$ of degree at most 2 in each variable, there is an algorithm A_i on the list that given oracle access to p, queries p (possibly adaptively) at most k times and outputs $\sum_{\bar{b} \in \{0,1\}^n} p(\bar{b})$. Then $k \geqslant 2^n$.*

Proof. Suppose $k < 2^n$. Consider all possible sequences $\tau = ((q_1, a_1), \dots, (q_k, a_k))$, where each $q_i \in \mathbb{F}^n$ and each $a_i \in \mathbb{F}$. We think of q_is as oracle queries to a given polynomial, and a_is as the answers to these queries.

Fix an algorithm $A = A_i$ from our list of algorithms. For each sequence τ, let P_τ be the set of polynomials in $\mathbb{F}[x_1, \dots, x_n]$ of degree at most 2 in each variable such that, given oracle access to any polynomial $p \in P_\tau$, the algorithm A asks queries and receives answers as specified by the sequence τ. After making these k queries, the algorithm A produces an answer $a \in \mathbb{F}$, which is the same for all polynomials $p \in P_\tau$.

We will argue as in the proof of Lemma 11. Let $p(x_1, \dots, x_n) = \sum_{j=1}^{N} c_j \cdot m_j(x_1, \dots, x_n)$, for unspecified coefficients c_j, where m_j's are monomials of degree at most 2 in each variable, and $N = 3^n$. For each point q_i, $1 \leqslant i \leqslant k$, define the vector $m_{q_i} = (m_1(q_i), \dots, m_N(q_i))$. Denoting the vector $(c_1, \dots, c_N)$ by c, we can write the answer $a_i = p(q_i)$ obtained by the algorithm A to its query q_i as $\langle m_{q_i}, c\rangle$. Also, $\sum_{\bar{b} \in \{0,1\}^n} p(\bar{b}) = \sum_{j=1}^{N} s_j \cdot c_j$ for some $s_1, \dots, s_N \in \mathbb{F}$. Denote the vector $(s_1, \dots, s_N)$ by s. Then the correct answer for the polynomial summation of p can be written as $\langle s, c\rangle$. We also assume, without loss of generality, that the vectors $m_{q_1}, \dots, m_{q_k}$ are linearly independent.

There are two cases to consider:

1. the vector s is *not* in the vector space spanned by $m_{q_1}, \dots, m_{q_k}$, and
2. the vector s is in that vector space.

In the second case, we get (as in the proof of Lemma 11) the existence of a special non-adaptive algorithm $A_{Q,R}$, for some sequence R of k field elements, such that $A_{Q,R}$ solves polynomial summation problem over $\mathbb{F}$ for all polynomials of degree at most 2 in each variable. This, however, contradicts Theorem 9.

Thus, we must have the first case, where the vectors $m_{q_1}, \dots, m_{q_k}, s$ are linearly independent over $\mathbb{F}$. Setting up a system of linear equations as in the proof of Lemma 11, we have for every

$\alpha \in \mathbb{F}$ that the following system of linear equations in unknowns c

$$\langle m_{q_j}, c\rangle = a_j, \quad 1 \leqslant j \leqslant k$$
$$\langle s, c\rangle = \alpha$$

has a solution space of dimension $N-(k+1)$, and in particular, the number of polynomials satisfying the system of linear equations above is the *same* for every $\alpha \in \mathbb{F}$. The polynomials in P_τ where A's output a is correct are exactly all solutions to the above system of equations for $\alpha = a$. So these polynomials have weight exactly $1/|\mathbb{F}|$ in the set P_τ.

Since the argument above holds for every choice of τ, we conclude that the set of polynomials $p \in \mathbb{F}[x_1, \ldots, x_n]$, of degree at most 2 in each variable, where A is correct has weight exactly $1/|\mathbb{F}|$ in the set of all polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ of degree at most 2 in each variable. Finally, the fraction of polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ of degree at most 2 in each variable which are correctly solved by at least one algorithm A_j, $1 \leqslant j \leqslant L$, is at most $L/|\mathbb{F}|$, which is less than 1 by our choice of L. Hence there is at least one polynomial p such that each algorithm A_j makes a mistake on this p. □

Remark 13. Note that Theorem 12 is *optimal* in terms of the allowed value L. If $L = |\mathbb{F}|$, then by letting the ith algorithm output the ith element of the field $\mathbb{F}$, we trivially obtain a list of algorithms such that every polynomial summation instance is solved by one of the algorithms on our list.

5.2 Polynomial summation for a family of polynomials

Our motivation for studying the query complexity of polynomial summation was the connection with $\#SAT$. So, in reality, we are interested in the problem of polynomial summation for only those polynomials that result from arithmetizing 3cnf formulas.

In this section, we consider the following way of representing a family of polynomials for which we want to solve polynomial summation. The input is now a $2n$-variate polynomial p of degree at most 2 in each variable, over some field $\mathbb{F}$. We view a $\{0,1\}$-valued assignment to the first n variables of such a polynomial as specifying (the arithmetization of) a propositional formula on n variables. So, given a $\{0,1\}$-valued assignment $\bar{x}$ to the first n variables, we would like to sum p over all $\{0,1\}$-valued assignments to the remaining n variables.

In this model, we also allow some advice that may depend on the given $2n$-variate polynomial p. That is, we allow a list of L algorithms such that, for every $2n$-variate polynomial p, at least one of the algorithms on the list solves polynomial summation for each polynomial resulting from p by fixing the first n variables to some $\{0,1\}$-valued assignment.

We show that even given an exponential amount of advice, no algorithm can solve polynomial summation with fewer than an exponential number of queries.

Theorem 14. *Let $\mathbb{F}$ be any finite field. Suppose there is a list of $L < |\mathbb{F}|^{2^{n/2}}$ algorithms such that, for every polynomial $p \in \mathbb{F}[x_1, \ldots x_n, y_1, \ldots, y_n]$ of degree at most 2 in each variable, there is at least one algorithm on the list that given oracle access to p, and given any input $\bar{x} \in \{0,1\}^n$, makes at most k (possibly adaptive) oracle queries and outputs $\sum_{\bar{y} \in \{0,1\}^n} p(\bar{x}, \bar{y})$. Then $k \geqslant 2^{n/2}$.*

The proof of this theorem will rely on the following generalization of Theorem 12.

Theorem 15. *Let $\mathbb{F}$ be a finite field. Let $m \geqslant 1$ be any integer. Suppose there is a list of $L < |\mathbb{F}|^m$ algorithms $A_1, \ldots, A_L$ such that the following holds. For every m-tuple of polynomials $p_1, \ldots, p_m \in \mathbb{F}[x_1, \ldots, x_n]$ of degree at most 2 in each variable, there is an algorithm A_i on the list that given oracle access to $p_1, \ldots, p_m$, queries each p_j (possibly adaptively) at most k times and outputs $\sum_{\bar{b} \in \{0,1\}^n} p_j(\bar{b})$, for every $1 \leqslant j \leqslant m$. Then $k \geqslant 2^n$.*

Proof. The proof is a straightforward extension of the proof of Theorem 12 to the case of m polynomials, for an arbitrary $m \geqslant 1$. $\square$

Proof of Theorem 14. Assume towards a contradiction that $k < 2^{n/2}$. We will express several n-variate polynomials of degree at most 2 in each variable as a single $2n$-variate polynomial of degree at most 2 in each variable. For $i \in \mathbb{N}$ such that $i < 2^n$, let $\bar{i} = \bar{i}_1 \dots \bar{i}_n$ denote the n-bit binary representation of i in the $\{0, 1\}$ basis. For any $2^{n/2}$ polynomials $p_0, p_1, \dots, p_{2^{n/2}-1} \in \mathbb{F}[x_1, \dots x_n]$ of degree at most two in each variable, we define a polynomial $q \in \mathbb{F}[x_1, \dots x_n, y_1, \dots, y_n]$ as follows:

$$q(x_1, \dots, x_n, y_1, \dots, y_n) = \sum_{i=0}^{2^{n/2}-1} \left(\prod_{\bar{i}_k = 1} x_k \right) \left(\prod_{\bar{i}_k = 0} (1 - x_k) \right) p_i(y_1, \dots, y_n).$$

Note that for all $0 \leqslant i < 2^{n/2}$ and $\bar{y} \in \mathbb{F}^n$, we have $q(\bar{i}_1, \bar{i}_2, \dots, \bar{i}_n, \bar{y}) = p_i(\bar{y})$. Also, note that q is of degree at most two in each variable.

By assumption, there will exist an algorithm A on our list of L algorithms such that A, on a Boolean assignment to x-s corresponding to i, makes at most $k < 2^{n/2}$ queries to q, and correctly computes the sum of p_i over the Boolean cube. Observe that a single query to q can be evaluated by querying each of the polynomials p_j once. Thus, to compute the sum of p_i over the Boolean cube, we need to make at most $k < 2^{n/2}$ queries to each p_j. It follows that we can compute the sum of every p_j, $1 \leqslant j \leqslant 2^{n/2}$, over the Boolean cube, by making at most $2^{n/2} \cdot k < 2^n$ queries to each p_j. But this contradicts Theorem 15 for $m = 2^{n/2}$. $\square$

5.3 The case of infinite fields

In the previous subsections, we considered a relaxation of the polynomial summation problem where we allow a list of L algorithms. In the case of a finite field $\mathbb{F}$, Theorem 12 is tight. In the case of an infinite field $\mathbb{F}$, it is intuitively obvious that no list of L algorithms should be able to solve the polynomial summation problem over $\mathbb{F}$ for any *finite* value of L. Here we shall prove that this is indeed the case, assuming a (very weak) restriction on algorithms.

Our restriction on algorithms (solving the polynomial summation problem) is that they "act in a measurable way", i.e., each step of the algorithm consists in computing some *measurable* function of the input values and the values computed so far. Recall that a function g is called *measurable* if for every measurable set B, we have that the set $g^{-1}(B)$ is also measurable. Here we shall consider only the case of the field $\mathbb{R}$ of reals, with the measure being the standard Lebesgue measure (for a background on measure theory see the excellent book of Rudin [Rud87]).

Next we describe more formally what we mean by an algorithm acting in a measurable way, and show that, for every such algorithm A, the set of n-variate polynomials over $\mathbb{R}$ for which A correctly solves the polynomial summation problem will be a measurable set.

Consider an algorithm A making k queries. Let q be its first query. For $1 \leqslant i < k$, let f_i be the function that, given the first i queries and the answers to them, computes the $(i+1)$st query. Let f_k be the function that, given the first k queries and the answers to them, outputs the final answer (which is supposed to be equal to the summation of a given polynomial over the Boolean cube). We say that such an algorithm A acts in a *measurable way* if all f_is, $1 \leqslant i \leqslant k$, are measurable functions.

Lemma 16. *Let A be an algorithm that acts in a measurable way. Then the set of n-variate polynomials over $\mathbb{R}$ for which A correctly solves the polynomial summation problem is measurable.*

Proof. Suppose p is an n-variate polynomial over $\mathbb{R}$ that A is correct on. Then we have

1. $p(q) = a_1$.
2. $p(f_1(q, a_1)) = a_2$.
3. $p(f_2(q, a_1, f_1(q, a_1), a_2)) = a_3$.

   ...

k+1. $\sum_{x\in\{0,1\}^n} p(x) = f_k(q, a_1, \dots, a_k)$.

Think of the polynomial p as a vector c of 3^n coefficients. Each query q_i gives rise to the vector m_{q_i} (of values of all 3^n monomials at the point q_i) so that $p(q_i) = \langle m_{q_i}, c\rangle$. Clearly, this inner product is a measurable function of c and q_i. Since a composition of finitely many measurable functions is a measurable function, we conclude that in each step i, $1 \leqslant i \leqslant k$, algorithm A computes some measurable function of the input p.

Also note that $\sum_{x\in\{0,1\}^n} p(x)$ can be written as $\langle s, c\rangle$ for some vector s. So the $(k+1)$st equation can be re-written as $\langle s, c\rangle = f_A(c)$, for some measurable function f_A. So every coefficient vector c of a polynomial on which A is correct must satisfy the equation $g(c) = 0$ for the measurable function g defined as $g(c) = f_A(c) - \langle s, c\rangle$. Thus, the set of polynomials c on which A is correct must be of the form $g^{-1}(\{0\})$, which is a measurable set by definition. □

Now we can state the following analogue of Theorem 12 for the case of $\mathbb{F} = \mathbb{R}$ when we allow only algorithms acting in a measurable way.

Theorem 17. *Suppose there is a possibly infinite list of algorithms $A_1, \dots, A_i, \dots$ acting in a measurable way such that the following holds. For every polynomial $p \in \mathbb{R}[x_1, \dots, x_n]$ of degree at most 2 in each variable, there is an algorithm A_i on the list that given oracle access to p, queries p (possibly adaptively) at most k times and outputs $\sum_{\bar{b}\in\{0,1\}^n} p(\bar{b})$. Then $k \geqslant 2^n$.*

The proof of this theorem is similar to that of Theorem 12. For the proof, we will need a probability distribution over all polynomials in $\mathbb{R}[x_1, \dots, x_n]$ of degree at most 2 in each variable. Each such polynomial can be viewed as a vector of $N = 3^n$ coefficients.

It will be convenient for us to use a multidimensional Gaussian (or normal) distribution. For a vector $m \in \mathbb{R}^d$ and a positive definite matrix $\Sigma \in \mathbb{R}^{d\times d}$, we say that a random vector $X = (X_1, \dots, X_d)$ is distributed according to a multivariate normal distribution $\mathcal{N}(m, \Sigma)$ if its density function is

$$f_X(x_1, \dots, x_d) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-m)^T\Sigma^{-1}(x-m)}, \tag{3}$$

where $x = (x_1, \dots, x_d)$, $|\Sigma|$ is the determinant of Σ, and v^T denotes the transpose of a column vector v. Here m is the expected value of X, and Σ is the covariance matrix of the components X_i.

For us, the important basic property of a multivariate normal distribution is that it remains normal (possibly with different expectation and covariance matrix) under linear transformations and under conditioning. More precisely, we have the following fact (which can be found in most textbooks treating multivariate normal distributions).

Fact 18. *Let $X = (X_1, \dots, X_d)$ be a random vector distributed according to a multivariate normal distribution.*

1. *For an $r \times d$ nonzero matrix B, the random vector $Y = BX$ is distributed according to a normal distribution.*

2. *For $1 < t \leqslant d$ and a vector $b \in \mathbb{R}^{d-t+1}$, the random vector $(X_1, \ldots, X_{t-1})$ conditioned on $(X_t, \ldots, X_d) = b$ is distributed according to a normal distribution.*

Now we can give the proof of Theorem 17.

Proof of Theorem 17. Consider the normal distribution (Gaussian measure) μ on $\mathbb{R}^{3^n}$ (the space of polynomials), with expectation 0 and the identity covariance matrix (i.e. $m = 0$ and $\Sigma = I$ in Equation (3)). For each algorithm A_i, let P_i be the set of polynomials for which A_i outputs the correct answer. By Lemma 16, each P_i is measurable.

We will argue that $\mu(P_i) = 0$ for every i. Suppose this is not the case. Assume without loss of generality that $\mu(P_1) = \epsilon > 0$. We shall focus on A_1 from now on.

For simplicity of notation, denote $A = A_1$ and $P = P_1$. Let q_1 be the first query asked by A. Define V_α to be the set of polynomials p such that $p(q_1) = \alpha$; observe that V_α is an affine subspace of the space $\mathbb{R}^{3n}$ of co-dimension 1. Define $P_\alpha = P \cap V_\alpha$. As μ is normal, we have from Fact 18 that $\mu|_{V_\alpha}$ is also normal (possibly with a different expectation and covariance matrix). We shall abuse notation and denote by $\mu(P_\alpha)$ the measure $\mu|_{V_\alpha}(P_\alpha)$. Since

$$\mu(P) = \int_\alpha \mu(P_\alpha) d\mu(\alpha),$$

there must exist some $\alpha_1 \in \mathbb{R}$ such that $\mu(P_{\alpha_1}) \geqslant \epsilon$. We shall restrict our attention to P_{α_1}.

In a similar fashion, we consider query q_2 asked by A given the answer α_1 to q_1, and so on. After k steps, we get a set $P_{\alpha_1,\ldots,\alpha_k}$ of measure at least ϵ (inside a co-dimension k subspace $V_{\alpha_1,\ldots,\alpha_k}$ of $\mathbb{R}^{3n}$) on which the output of A is fixed to some value $\alpha^* \in \mathbb{R}$.

Suppose that $k < 2^n$. Then, as in the proof of Theorem 12, we conclude that the vectors $m_{q_1}, \ldots, m_{q_k}, s$ are linearly independent over $\mathbb{R}$. Recall that $m_{q_i} = (m_1(q_i), \ldots, m_{3^n}(q_i))$ for all 3^n monomials m_j, and s is a vector such that for a polynomial p with the coefficient vector c, $\langle s, c\rangle = \sum_{x \in \{0,1\}^n} p(x)$. As the answer of A is fixed on $P_{\alpha_1,\ldots,\alpha_k}$ to the value α^*, it must be the case that this set is contained in the set of solutions c of the following system of linear equations:

$$\begin{aligned} \langle m_{q_j}, c\rangle &= \alpha_j, \quad 1 \leqslant j \leqslant k \\ \langle s, c\rangle &= \alpha^* \end{aligned}$$

The first k equations mean that $P_{\alpha_1,\ldots,\alpha_k} \subseteq V_{\alpha_1,\ldots,\alpha_k}$. By Fact 18, the vector of coefficients $c \in P_{\alpha_1,\ldots,\alpha_k}$ is distributed according to the normal distribution $\mu|_{V_{\alpha_1,\ldots,\alpha_k}}$, and the random variable $\langle s, c\rangle$ is also distributed according to some normal distribution μ'. It follows that the measure of vectors $c \in P_{\alpha_1,\ldots,\alpha_k}$ such that $\langle s, c\rangle = \alpha^*$ is $\mu'(\{\alpha^*\}) = 0$. But we argued earlier that the measure of this set of polynomials must be at least $\epsilon > 0$. A contradiction.

Thus, the measure of every set P_i is zero, and so the measure of the countable union of all P_is is also zero. This means that there is a polynomial whose sum over the Boolean cube cannot be computed by any of the algorithms on our list. □

6 Open questions

Most of lower bounds in this paper were obtained by arguing the existence of a problematic polynomial on which the query algorithm makes a mistake. However, such a polynomial will likely have large circuit complexity. On the other hand, polynomials arising from Boolean formulas via arithmetization have low arithmetic circuit complexity. Thus the main open question is to determine the query complexity of polynomial summation for low-degree polynomials p that are computable by *small* arithmetic formulas.

It is also an open question whether the result of Theorem 17 continues to hold if algorithms are not restricted to act in a measurable way (although any reasonable model of computation is "measurable").

Acknowledgments We thank Dima Grigoriev and Russell Impagliazzo for helpful discussions.

References

[BF92] L. Babai and P. Frankl. *Linear Algebra Methods in Combinatorics with Applications to Geometry and Computer Science, Preliminary Version 2.* Department of Computer Science, The University of Chicago, 1992.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[Juk01] S. Jukna. *Extremal Combinatorics.* Texts in Theoretical Computer Science. Springer Verlag, 2001.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the Association for Computing Machinery*, 39(4):859–868, 1992.

[Raz04] R. Raz. Multi-linear formulas for Permanent and Determinant are of super-polynomial size. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 633–641, 2004.

[Rud87] W. Rudin. *Real and complex analysis, Third edition.* McGraw-Hill Inc., 1987.

[Sha92] A. Shamir. IP=PSPACE. *Journal of the Association for Computing Machinery*, 39(4):869–877, 1992.

[Str88] G. Strang. *Linear Algebra and Its Applications, Third Edition.* Brooks/Cole, 1988.

A An omitted proof

Theorem 19. *Let $\phi(x_1, \ldots, x_n)$ be a cnf where each variable appears in at most 3 clauses. Then there is a formula $\psi(y_1, \ldots, y_{3n})$ that has the same number of satisfying assignments as ϕ, and there is a polynomial $p(y_1, \ldots, y_{3n})$ of degree at most 2 in each variable such that p and ψ agree on the Boolean cube $\{0,1\}^{3n}$. Moreover, the polynomial p can be computed by an arithmetic circuit of size polynomial in the size of ϕ.*

Proof. Assume without loss of generality that each variable x_i of ϕ occurs in exactly three clauses. Replace the three occurrences of x_i by three new variable x_i^1, x_i^2, x_i^3. Apply this replacement procedure to each variable x_i of ϕ. Denote the resulting cnf by ϕ'. Add to ϕ' the formula ϕ'' that expresses the condition that each triple of variables x_i^1, x_i^2, x_i^3, for $1 \leqslant i \leqslant n$, are mutually equivalent, i.e., $x_i^1 \leftrightarrow x_i^2 \leftrightarrow x_i^3$. The required formula ψ will be the conjunction $\phi' \wedge \phi''$.

To construct p, we first apply the standard arithmetization procedure (described in the Introduction) to the formula ϕ'. This gives us a multilinear polynomial p' since no variable of ϕ' appears in more than one clause. For each $1 \leqslant i \leqslant n$, define the multilinear polynomial $p_i(x_i^1, x_i^2, x_i^3)$ so that it is equal to 1 on all binary triples (b_1, b_2, b_3) where $b_1 = b_2 = b_3$, and is equal to 0 on all other binary triples. Such a polynomial is easily obtained as a multilinear extension of the underlying Boolean function. Since each p_i depends on only 3 variables, it will be computable by an arithmetic formula of constant size. Finally, define $p = p' \cdot \prod_{i=1}^{n} p_i$. It is easy to see that p is the product of two multilinear polynomials p' and $\prod_{i=1}^{n} p_i$, and so p has degree at most 2 in each variable. On the other hand, by construction, p agrees with ψ over the Boolean cube. $\square$

ECCC ISSN 1433-8092

http://eccc.hpi-web.de/