The Polynomial Method in Circuit Complexity Applied to Algorithm Design*

Richard Ryan Williams

Computer Science Department, Stanford University
353 Serra Mall, Stanford, CA 94305, USA
rrw@cs.stanford.edu

Abstract

In circuit complexity, the *polynomial method* is a general approach to proving circuit lower bounds in restricted settings. One shows that functions computed by sufficiently restricted circuits are "correlated" in some way with a low-complexity polynomial, where complexity may be measured by the degree of the polynomial or the number of monomials. Then, results limiting the capabilities of low-complexity polynomials are extended to the restricted circuits.

Old theorems proved by this method have recently found interesting applications to the design of algorithms for basic problems in the theory of computing. This paper surveys some of these applications, and gives a few new ones.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases algorithm design, circuit complexity, polynomial method

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.47

Category Invited Talk

1 Introduction

The polynomial method was developed for proving impossibility results on the capabilities of "low-complexity" circuits. (The usual measures of "low-complexity" for circuits are either low size, or low depth, or both.) The idea of the polynomial method, at a high level, is to show that functions computable by low-complexity circuits can be also be computed (approximately or exactly) by a "low-complexity" polynomial over some algebraic structure. Typically, the complexity of a polynomial is measured by its degree, but the complexity could also be the number of monomials. The survey by Beigel [8] contains many references to papers in which low-complexity circuits are represented via low-complexity polynomials, resulting in lower bounds against those circuits.

While the method was initially conceived to show the limitations of computational devices, the intermediate theorems proved via the method turn out to also be rather useful in the design of algorithms for certain problems – *positive* results about computational devices. Over the last few years, we have found some unexpected applications of the polynomial method to developing more efficient algorithms for several fundamental computational problems. Sometimes it is natural to see how the polynomial method might help; in other cases, it is not at all obvious, and some ingenuity is required. An intuitive outline of the approach is:

1. Find a "hard part" of one's computational problem that can be modeled by low-complexity circuits.

* This work was partially supported by the National Science Foundation Grant CCF 1212372.

34th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014).
Editors: Venkatesh Raman and S. P. Suresh; pp. 47–60

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

2. Apply the polynomial method to convert the low-complexity circuits into an algebraic, polynomial form.
3. Use other algebraic algorithms to efficiently manipulate or evaluate these polynomials, thereby solving the original "hard part" more efficiently.

This article outlines several instances of this approach. Some of the algorithms in this paper are new to the literature; they are included to illustrate the versatility of the polynomial method in algorithm design. Many proofs of the known results are omitted from this article; however, some results stated here are new, and we shall describe their proofs in detail.

1.1 What This Survey is NOT

We make no claims to be the "first" to apply the polynomial method in positive algorithmic ways. There are many theorems in mathematics and theoretical computer science regarding the modeling of efficient functions with polynomials; discussing all of them is neither wise nor possible in this space. Nevertheless, it's important to note that there are more interesting theorems related to the polynomial method, in the hopes that future work will make use of them. To give three examples from different angles:

1. Many theorems from approximation theory (which is effectively the study of pointwise approximating functions via "simple" expressions, such as polynomials over the reals) have seen applications in areas such as communication complexity and quantum computing [31, 7, 1]. We haven't yet personally found algorithmic applications of these polynomials for our problems of interest, but that is probably our own failing, and not that of the polynomials.
2. Another example is the collection of lemmas in the literature informally known as the Schwartz-Zippel-DeMillo-Lipton Lemma [35, 50, 16] concerning the (low) number of zeroes in low-degree polynomials that are not identically zero. These lemmas are already a staple of randomized algorithms [30].
3. The polynomial method has found a large number of applications in *computational learning theory*, such as in algorithms for learning DNFs and low-depth circuits (e.g., [26, 27, 24, 19]) and learning functions with a small number of relevant variables (a.k.a. juntas) [29].

2 The Circuits

We assume the reader is familiar with the usual notion of Boolean circuits as directed acyclic graphs, where n input gates are represented by $2n$ source nodes (the n input bits and their negations), the output gate is represented by a single sink node, and each node (or "gate") is labeled with a boolean function. We shall consider two well-studied restrictions of this general notion. Let $d, m \in \mathbb{N}$.

- An AC circuit of *depth* d is such that the longest path from any source to sink is at most d, and each gate computes either the OR (of its inputs) or the AND (of its inputs).
- An ACC circuit of *depth* d and *modulus* m is such that the longest path from any source to sink is at most d, and each gate computes one of OR, AND, or MOD_m, where $MOD_m(y_1, \ldots, y_t) = 1$ if and only if $\sum_i y_i$ is divisible by m.

More background can be found in the textbooks [5, 42].

3 The Tools

In this survey, we shall focus on just a few polynomial constructions from the literature which have recently been helpful. Again, we have made no attempt to be comprehensive.

Three notions of representation by polynomials will be considered in this article: exact representations, probabilistic representations over finite fields and the integers,

In the following, let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function with 0 representing *false* and 1 representing *true*, and let R be a ring containing 0 and 1. We shall always be evaluating polynomials over the values 0, 1, or -1. Since $x^a \in \{x, -x\}$ for all $a \in \mathbb{N}^+$ and all $x \in \{0, 1, -1\}$, it suffices for us to consider *multilinear* polynomials, of the form

$$p(x_1, \ldots, x_n) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i,$$

where $c_S \in R$ for all S. The *degree* of p is therefore the maximum cardinality of a subset S such that $c_S \neq 0$, and the *sparsity* of a polynomial is the number of S such that $c_S \neq 0$.

3.1 Exact Representations

▶ **Definition 1.** An n-variate polynomial $p(x_1, \ldots, x_n)$ over R *exactly represents* f if for all $(a_1, \ldots, a_n) \in \{0,1\}^n$, $p(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$.

When R is a field, every Boolean function f has a *unique* exact representation as a (multilinear) polynomial p. To give two simple examples, the function AND : $\{0,1\}^2 \to \{0,1\}$ is exactly represented by the polynomial $p(x_1, x_2) = x_1 x_2$, and OR : $\{0,1\}^2 \to \{0,1\}$ is represented by $p(x_1, x_2) = x_1 + x_2 - x_1 x_2$, over any field.

Exact representations are often implicitly used in algorithms, but their influence can be somewhat hidden. For example, the inclusion-exclusion principle from combinatorics can be applied to solve several hard problems more efficiently, e.g., counting the number of Hamiltonian Paths in n-node graphs in $2^n \cdot n^{O(1)}$ time and $n^{O(1)}$ space [23]. This principle is a consequence of the fact that the OR function on n variables can be exactly represented as:

$$\mathrm{OR}(x_1, \ldots, x_n) = 1 - \prod_{i=1}^{n} (1 - x_i) = \sum_{S \subseteq [n], |S| > 0} (-1)^{|S|+1} \prod_{i \in S} x_i.$$

In many situations, it is preferable to think of the Boolean function f with domain $\{-1, 1\}$ and codomain $\{-1, 1\}$ instead, where -1 corresponds to *true* and 1 corresponds to *false*. Then, a monomial $x_1 x_2 \cdots x_n$ represents the *PARITY* of n bits rather than the *AND* of n bits. Studying Boolean functions via this representation is often called the *Fourier analysis* of Boolean functions and is a world unto itself; we recommend O'Donnell's comprehensive textbook on the subject [32].

3.2 Probabilistic Representations

The next representation we consider is a "randomized" notion of polynomial, which is surprisingly powerful.

▶ **Definition 2.** Let $\mathcal{D}$ be a finite distribution of polynomials on n variables over R. The distribution $\mathcal{D}$ is a *probabilistic polynomial over R representing f with error δ* if for all $(a_1, \ldots, a_n) \in \{0,1\}^n$, $\Pr_{p \sim \mathcal{D}}[p(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)] > 1 - \delta$.

The *degree* of $\mathcal{D}$ is the maximum degree over all polynomials in $\mathcal{D}$.

One may also define a probabilistic polynomial as a *single* polynomial with n "input" variables and r "random" variables over a finite domain. Then, the distribution $\mathcal{D}$ in the above definition is obtained by assigning the r variables to uniform random values. However,

it's not hard to see that, for every finite distribution $\mathcal{D}$ of s polynomials of maximum degree d and maximum sparsity m, one can recover a single probabilistic polynomial of degree d (in the input variables) with only $O(\log s)$ random variables and sparsity $O(m \cdot s)$, by simple interpolation (see also Tarui [40]).

Another important fact is that, (essentially) without loss of generality, the distribution $\mathcal{D}$ contains only $O(n)$ polynomials. Given any $\mathcal{D}$ for a function f and a parameter $\varepsilon > 0$, uniformly sample $t = O(n/\varepsilon^2)$ polynomials $p_1, \ldots, p_t \sim \mathcal{D}$, and form the distribution $\mathcal{D}'$ over $\{p_1, \ldots, p_t\}$ (as a multiset). By a standard Chernoff bound and union bound argument, the distribution $\mathcal{D}'$ is also a probabilistic polynomial for f, with essentially the same error (to within $\pm\varepsilon$).

Probabilistic polynomials were first utilized by Razborov [33] and Smolensky [36] in their proofs that the MAJORITY function and MOD3 functions cannot be computed efficiently with ACC circuits of constant depth and modulus 2, respectively. In particular, they showed that every low-depth circuit with modulus 2 has a low-degree probabilistic representation over the field $\mathbb{F}_2$. Here, we cite a strengthened version by Kopparty and Srinivasan:

▶ **Theorem 3** ([36, 25]). *For every* ACC *circuit C of depth d, size s, modulus 2, and n inputs, and $\varepsilon > 0$, there is a probabilistic polynomial $\mathcal{D}_C$ over $\mathbb{F}_2$ representing C with error ε, and degree at most $(4 \log s)^{d-1} \cdot (\log 1/\varepsilon)$, such that a polynomial p can be sampled from $\mathcal{D}_C$ in $n^{O(\log s)^{d-1}(\log 1/\varepsilon)}$ time.*

The basic idea of the proof is to randomly replace each gate in the circuit with very low-degree polynomials over $\mathbb{F}_2$, such that their composition leads to a low-degree polynomial for the entire circuit C. (The proof of Theorem 3 gives a clever way of composing these polynomials so as to keep the degree low, as a function of ε.) How do we construct these very low-degree polynomials? Gates which are MOD2 functions are simply additions over $\mathbb{F}_2$. A gate g which is a NOT of a gate h can be written as $g = 1 + h$ over $\mathbb{F}_2$. Gates which are ANDs can be expressed with NOTs and ORs by DeMorgan's law. Finally, gates which are ORs can be simulated probabilistically by multiplying a few sums of random subsets of the inputs, modulo 2. For example, if the OR of $x_1, \ldots, x_n$ is 1, then

$$\Pr_{r_1,\ldots,r_n \in \{0,1\}} \left[\sum_{i=1}^{n} r_i x_i = 1 \bmod 2 \right] = \frac{1}{2}.$$

On the other hand, if $x_1 = \cdots = x_n = 0$, then no random sum of the x_i's will evaluate to 1. In this way, a MOD2 can simulate an OR; multiplying several copies of such a probabilistic polynomial (carefully) will allow us to reduce the probability of error.

The above ideas can be extended to any finite field; however, the degrees of the probabilistic polynomials obtained may increase as a function of the field characteristic. (In particular, sums of variables will need to be raised to their $(p-1)$th powers, to keep the output Boolean.) It is natural to then ask how probabilistic polynomials over $\mathbb{Z}$ fare in computing AC circuits.

Beigel, Reingold, and Spielman [9] addressed this question, finding an $O(\log^2 n)$-degree probabilistic polynomial for OR. The following improvement is due to Aspnes, Beigel, Furst, and Rudich [6]:

▶ **Theorem 4.** *For all $\varepsilon > 0$, there is a probabilistic polynomial over $\mathbb{Z}$ for the OR of n variables with error ε, and degree $O(\log n \cdot \log 1/\varepsilon)$. Furthermore, for every* AC *circuit C of depth d and size s, there is a probabilistic polynomial for C with error ε having degree $O(\log^d s \cdot \log^d s/\varepsilon)$.*

Let us sketch the proof. To compute the OR of $x_1, \ldots, x_n$, choose progressively smaller random subsets $S_0, \ldots, S_{\log n+1} \subseteq \{1, \ldots, n\}$, where $S_0 = [n]$, and S_i is a uniform random

subset of S_{i-1}. The key claim is that, if the OR of $x_1, \ldots, x_n$ is 1, then with probability at least 1/3, some S_i contains *exactly one* j such that $x_j = 1$. In that case, the polynomial

$$p(x_1, \ldots, x_n) = \prod_{i=0}^{\log n+1} \left(1 - \sum_{j \in S_i} x_j\right)$$

correctly computes the negation of OR (so, $1 - p$ computes OR). To reduce the error to arbitrarily small ε, one can take $O(\log 1/\varepsilon)$ products of independent copies of p.

To get probabilistic polynomials for AC circuits of depth d and size s with error ε, apply this randomized construction of p (and an analogous construction for AND) independently to every gate in the AC circuit, with error parameter set to ε/s. Then, a union bound over all s gates guarantees the result.

3.3 Symmetric Representation

Finally, we consider a polynomial representation of functions which may look somewhat unusual: we try to represent functions by low-complexity polynomials h whose outputs are "filtered" through another function g which gives $\{0, 1\}$ output.

▶ **Definition 5.** Let $h(x_1, \ldots, x_n)$ be a polynomial over R, construed as a function $h : \{0,1\}^n \to R$. Let $g : \mathrm{Im}(h) \to \{0, 1\}$ be arbitrary. We say that (g, h) is a *symmetric representation of* f if for all $(a_1, \ldots, a_n) \in \{0,1\}^n$, $g(h(a_1, \ldots, a_n)) = f(a_1, \ldots, a_n)$.

Why do we call this a "symmetric" representation? Suppose $R = \mathbb{Z}$. If all coefficients of h are in $\{0, 1\}$ and h has s monomials, we have $\mathrm{Im}(h) \subseteq \{0, 1, \ldots, s\}$, and the "filter function" g may then be viewed as a function on s variables which only depends on the number of inputs which are true. That is, we may think of g as a *symmetric* Boolean function. To put it another way, in this situation we can represent $g \circ h$ as a depth-two Boolean circuit with $s + 1$ gates, where the output gate computes a symmetric function and the layer of gates nearest the inputs compute ANDs. (The function h counts up the number of ANDs which output true, and the function g determines the output of the symmetric function.)

Symmetric representations are not as unusual as one might think. The class of *polynomial threshold functions* refer to a particular type of symmetric representation, where the symmetric function is a threshold function (checking whether the sum of all inputs exceeds a fixed value T). Polynomial threshold functions have been studied for a rather long time, especially in the context of neural networks ([28]).

We will use a particularly strong result on symmetric representations of functions computable with ACC circuits, first proved by Beigel and Tarui, building on work of Yao:

▶ **Theorem 6** (Yao [49], Beigel and Tarui [10]). *There is a function $\alpha : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that every Boolean function computable by an* ACC *circuit with size s, depth d, and modulus m has a symmetric representation (g, h) with $\deg(h) \leq (\log s)^{\alpha(d,m)}$.*

That is, every constant-depth and constant-modulus circuit can be symmetrically represented with a polynomial of degree that is polylogarithmic in the circuit size. It is widely believed that this sort of representation should severely restrict the kinds of functions computable with constant depth and modulus. It is believed that the MAJORITY of n bits cannot be computed with polynomial-size ACC circuits of any constant depth or constant modulus.

4 The Applications

Now we discuss how these tools have been recently applied in algorithm design.

4.1 All-Pairs Shortest Paths (APSP)

We first study the dense case of the All-Pairs Shortest Paths problem (APSP) on general weighted graphs.

▶ **Definition 7** (All-Pairs Shortest Paths (APSP)). In *APSP*, the input is a weighted adjacency matrix, and the goal is to output a data structure S encoding all shortest paths between any pair of nodes: when a pair of nodes (s,t) are fed to S, it must reply with the shortest distance from s to t in $\tilde{O}(1)$ time, and an actual shortest path from s to t in $\tilde{O}(\ell)$ time, where ℓ is the number of edges on the path.

The $O(n^3)$ time algorithms for APSP on n-node graphs [17, 43] are in the canon of undergraduate computer science. But these algorithms could be suboptimal, as the input graph can be encoded in $\Theta(n^2 \cdot \log m)$ bits, where $m \in \mathbb{N}$ upper bounds the edge weights. Indeed, in the real RAM model of computation (where additions and comparisons on "real-valued" registers are allowed, and arbitrary bit operations on "word registers" of $O(\log n)$ bits), Fredman [18] showed in 1975 that APSP is solvable in $o(n^3)$ time.

Since then, many papers on the dense case of APSP have been published, steadily decreasing the running time of $O(n^3)$ [37, 38, 51, 20, 39, 21, 13, 14]. All of them obtained only $O(n^3/\log^c n)$ time algorithms for constants $c \leq 2$. A major open problem is whether APSP is solvable in truly subcubic time, i.e., $O(n^{3-\varepsilon})$ time for some fixed $\varepsilon > 0$. A recently developed hardness theory for APSP shows that such an algorithm would have many consequences [34, 41, 3, 2].

Recently, the author gave new algorithms for APSP that run faster than $O(n^3/\log^c n)$ time, for *every* constant c [45]. In fact, the randomized version runs in $n^3/2^{\Omega(\sqrt{\log n})}$ time and the deterministic version runs in $n^3/2^{\Omega((\log n)^\delta)}$ time for some $\delta > 0$; these are asymptotically much better bounds. The algorithms crucially rely on the tools of the previous section: the problem of efficiently computing APSP is reduced to efficiently computing a particular circuit evaluation problem, and it is shown how to evaluate such circuits more efficiently than the obvious approach.

Let us jump directly to the kind of circuit that arises from the proof. Implicit in the APSP paper [45] is the following theorem, which we isolate for the reader's convenience. For $d, n \in \mathbb{N}$, define the Boolean function OR-AND-COMP on $2d^2 \log n$ inputs as follows:

$$\text{OR-AND-COMP}(a_{1,1}, a_{1,2}, \ldots, a_{d,d}, b_{1,1}, b_{1,2}, \ldots, b_{d,d}) := \bigvee_{i=1}^{d} \bigwedge_{j=1}^{d} [a_{i,j} \leq b_{i,j}],$$

where the $a_{i,j}, b_{i,j}$ are construed as $\log n$-bit numbers and $[A \leq B]$ is true if and only if $A \leq B$.

▶ **Theorem 8** (Implicit in [45]). *Let A, B be two sets of n vectors, where each vector is of length d^2 and each vector component has $\log n$ bits. Suppose the function OR-AND-COMP$(a_1, \ldots, a_{d^2}, b_1, \ldots, b_{d^2})$ is computable in $\tilde{O}(n^2)$ time, for all n^2 pairs $(a_1, \ldots, a_{d^2}) \in A$ and $(b_1, \ldots, b_{d^2}) \in B$ simultaneously. Then APSP is solvable in $\tilde{O}(n^3/d)$ time.*

Why is this theorem true? Here's a little intuition. APSP involves comparing the sums of weights on different paths, and determining which sums of weights are minimal among a

collection of sums. The OR-AND-COMP circuit is effectively finding a minimum sum among a particular set of paths of length two. The ability to compute this minimum sum for all n^2 pairs of vectors roughly corresponds to computing APSP in a tripartite graph with n nodes in the first part, d nodes in a middle part, and n nodes in the third part, with first and third part disconnected. Of course this is extremely handwavy, and the reader should consult the paper for more details.

As the circuit OR-AND-COMP has $2d^2 \log n$ inputs, such an evaluation would naively take $O(n^2 d^2 \log n)$ time. Presumably, it is easier to get an $\tilde{O}(n^2)$ time algorithm when d is small. The APSP paper [45] shows that for $d = 2^{c\sqrt{\log n}}$ where $c > 0$ is some constant, the $\tilde{O}(n^2)$ time evaluation required by Theorem 8 is actually possible. Here's a high-level outline of the algorithm.

1. First, computing $[a \leq b]$ for $(\log n)$-bit strings a and b can be done with constant-depth $O((\log n)^2)$-size circuits over AND and OR; that is, OR-AND-COMP is computable with AC circuits of constant depth and polynomial size. So the first idea is to apply Razborov and Smolensky (Theorem 3) (or Beigel-Tarui, Theorem 6) to the AC circuit for OR-AND-COMP, reducing this circuit to a probabilistic polynomial (or a symmetric representation, respectively). Given that this AC circuit has size $O(d^2 \log^2 n)$ on $O(d^2 \log n)$ variables, we find that OR-AND-COMP has a probabilistic polynomial over $\mathbb{F}_2$ with $2^{\mathrm{poly}(\log d, \log\log n)}$ monomials and $< 1/n^3$ error, and there is a symmetric representation of OR-AND-COMP with a similar monomial upper bound.
2. Second, given two sets A, B of n vectors as in the theorem statement, we show how to efficiently evaluate polynomials with at most $n^{.1}$ monomials on all pairs of vectors (one from A and one from B). This step uses a special rectangular matrix multiplication algorithm of Coppersmith [15], and runs in $\tilde{O}(n^2)$ time.
3. Thirdly, we combine 1 and 2. We use part 1 to generate a polynomial representation for OR-AND-COMP with $m = 2^{\mathrm{poly}(\log d, \log\log n)}$ monomials. Choose $d = 2^{(\log n)^\delta}$, so that $\delta > 0$ is small enough to make $m \leq n^{.1}$. Now we can apply part 2 and compute this OR-AND-COMP (with some probability of error) in $\tilde{O}(n^2)$ time. If our polynomials are probabilistic, each evaluation may have some errors. However, if we take $O(\log n)$ independent constructions and evaluations of these probabilistic polynomials for OR-AND-COMP, the MAJORITY values of these $O(\log n)$ evaluations will yield the correct values for OR-AND-COMP on all n^2 pairs of points, with high probability. Finally, applying Theorem 8, we thereby compute APSP in $n^3/2^{\Omega((\log n)^\delta)}$ time.

With a few pages of technical work, the $\delta > 0$ in the algorithm can be tuned down to $1/2$ in the randomized case. The big question is whether we can set $\delta = 1$, and yield a truly-subcubic APSP algorithm. This looks difficult, and not just because it is a thorny circuit evaluation problem.

If we try to use Theorem 8 to get truly-subcubic APSP, we would need a fast algorithm for evaluating OR-AND-COMP with $d \geq n^\delta$ for some $\delta < 1$. However, such an algorithm would also resolve *another* major open problem: we'd be able to solve CNF-SAT in $2^{\delta n}$ time for some $\delta < 1$, contradicting the so-called Strong Exponential Time Hypothesis (SETH) [22, 12]. In the following section, we shall explain why.

4.2 Orthogonal Vectors (OV)

We consider a slightly simpler function than the one needed for solving APSP. Define

$$\text{OR-AND-OR2}_{d_1,d_2}(x_{1,1}, x_{1,2}, \ldots, x_{d_1,d_2}, y_{1,1}, y_{1,2}, \ldots, y_{d_1,d_2}) = \bigvee_{i=1}^{d_1} \bigwedge_{j=1}^{d_2} (x_{i,j} \vee y_{i,j}).$$

That is, OR-AND-OR2$_{d_1,d_2}$ takes $2d_1d_2$ bits of input. Note that we can easily simulate OR-AND-OR2$_{d,d}$ with a call to OR-AND-COMP: $(x_{i,j} \vee y_{i,j}) = 1$ if and only if $[\neg x_{i,j} \leq y_{i,j}]$. Therefore, evaluating OR-AND-OR2$_{d,d}$ is only easier than evaluating OR-AND-COMP. However, quick evaluation of OR-AND-OR2 would yield faster algorithms for other problems than just APSP. Here is the canonical example of such a problem:

▶ **Definition 9** (Orthogonal Vectors (OV)). In *OV*, the input is a set $S \subseteq \{0,1\}^d$, and the goal is to output whether there are vectors $a, b \in S$ such that $\langle a, b\rangle = 0$.

That is, we wish to know if S contains an orthogonal pair of vectors. There are two obvious algorithms: one takes $O(|S|^2 d)$ time, and one takes $O(2^d|S|)$ time. So the interesting case is when we have "high dimensionality", and $d \geq \log n$. It is an open question whether $O(|S|^{2-\varepsilon}2^{o(d)})$ time is possible for some fixed $\varepsilon > 0$. By adding two more dimensions to the vectors, the following version of OV is equivalent to the above:

▶ **Definition 10** (Orthogonal Vectors' (OV')). In *OV'*, the input is two sets $A, B \subseteq \{0,1\}^d$, and the goal is to output whether there are vectors $a \in A$, $b \in B$ such that $\langle a, b\rangle = 0$.

OV captures the difficulty of several problems. Consider the partial match problem from string searching:

▶ **Definition 11** (Batch Partial Match (BPM)). In *BPM*, the input is a database $D \subseteq \{0,1\}^d$, and queries $Q \subseteq \{0,1,\star\}^d$, where $|D| = |Q|$. The goal is to output, for every $q \in Q$, whether or not there is an $x \in D$ such that for all $i = 1, \ldots, d$, $q[i] \neq \star$ implies $q[i] = x[i]$.

That is, we wish to know which queries have a "partial match" in the given database. Recent work with Abboud and Yu [4] proved that BPM is *sub-quadratic equivalent* to OV: roughly speaking, an $|S|^{2-\varepsilon}f(d)$ time algorithm for OV implies an $|Q|^{2-\delta}f(d)$ time algorithm for the BPM, and the converse also holds.

Another string problem related to OV is a generalization of the longest common substring problem to handle wildcard symbols:

▶ **Definition 12** (Longest Common Substring With Don't Cares (LCS*)). In *LCS**, the input is two strings $S, T \in \Sigma^n$ of length n, and the goal is to output the length of the longest string that appears in both S and T as a contiguous substring.

In the same paper with Abboud and Yu, it is proven that LCS* has a faster-than-quadratic time algorithm, given that OV has one. The importance of solving OV in sub-quadratic time is further reinforced by the following connection with exponential-time algorithms for satisfiability.

▶ Conjecture 4.1 (Strong Exponential Time Hypothesis (SETH) [22, 12]). For every $\delta < 1$, there is a $k \geq 3$ such that satisfiability of k-CNF formulas on n variables requires more than $2^{\delta n}$ time.

▶ **Theorem 13** ([44, 48]). *Suppose there is an $\varepsilon > 0$ such that for all $c \geq 1$, OV can be solved in $O(|S|^{2-\varepsilon})$ time on instances with $c \log |S|$ dimensions. Then SETH is false.*

Proof. We prove the contrapositive. Calabro, Impagliazzo, and Paturi [11] show that refuting SETH is equivalent to giving a $\delta < 1$ such that, for all $c \geq 1$, CNF-SAT on instances with n variables and cn clauses can be solved in $O(2^{\delta n})$ time.

We reduce this variant of CNF-SAT to OV. Given a formula F on n variables and cn clauses $C_1, \ldots, C_{cn}$, divide the variables into two sets V_1 and V_2 with at most $n/2 + 1$

variables each. Enumerate all $O(2^{n/2})$ partial assignments to the variables in V_1 and all partial assignments to the variables in V_2. For each such partial assignment A, define a vector v_A with $cn+2$ dimensions as follows. For $i = 1, \ldots, cn$, set $v_A[i] = 0$ iff the clause C_i is satisfied by A. Then, set $v_A[cn+1] = 1$ iff the partial assignment A is on the variables of set V_1, and set $v_A[cn+2] = 1$ iff A is from set V_2. Put all v_A's in the OV instance S.

Suppose F is satisfiable; let A be a satisfying assignment. For $i = 1, 2$, let the partial assignment A_i be the assignment A restricted to variables from V_i. By construction, $v_{A_1}[cn+1] \cdot v_{A_2}[cn+1] = v_{A_1}[cn+2] \cdot v_{A_2}[cn+2] = 0$, and for every clause C_i, at least one of A_1 or A_2 satisfies C_i, so $v_{A_1}[i] \cdot v_{A_2}[i] = 0$. It follows that $\langle v_{A_1}, v_{A_2} \rangle = 0$. Similarly, $\langle v_A, v_{A'} \rangle = 0$ implies that A and A' come from different sets and jointly satisfy F.

Finally, if OV is in $O(|S|^{2-\varepsilon})$ time for $c \log |S|$ dimensional vectors, then we can determine satisfiability of F in $O(2^{n(1-\varepsilon/2)})$ time. ◀

Now we can formally illustrate the importance of evaluating OR-AND-OR2 efficiently:

▶ **Theorem 14** (Implicit in [4]). *Let A, B be two sets of n bit vectors, where each vector has $t = d_1 \cdot d_2$ bits. Suppose OR-AND-OR2$_{d_1,d_2}(a_1, \ldots, a_t, b_1, \ldots, b_t)$ is computable in $\tilde{O}(n^2)$ time, for all n^2 pairs $(a_1, \ldots, a_t) \in A$ and $(b_1, \ldots, b_t) \in B$ simultaneously. Then OV with n vectors in d_2 dimensions can be solved in $\tilde{O}(n^2/d_1)$ time.*

Proof. For convenience, we work with OV' (Definition 10) in which we get two sets of vectors A, B and wish to find $a \in A$ and $b \in B$ that are orthogonal.

Partition both A and B into $\sqrt{d_1}$-size subsets $A_1, \ldots, A_{O(n/\sqrt{d_1})}$ and $B_1, \ldots, B_{O(n/\sqrt{d_1})}$, respectively. The idea is that with a single OR-AND-OR2$_{d_1,d_2}$ computation on $2d_1d_2$ bits, we can check whether the sub-instance (A_i, B_j) contains an orthogonal pair of vectors, for all $i, j = 1, \ldots, \sqrt{d_1}$.

The function OR-AND-OR2$_{d_1,d_2}$ takes the OR over d_1 pairs of vectors of the *complement of the Boolean inner product of d_2-dimensional vectors*. That is, the AND-OR2 parts of the function output 1 if the two relevant d_2-dimensional vectors are orthogonal, and 0 otherwise. By arranging the $\sqrt{d_1}$ vectors of A_i into one d_1d_2-dimensional vector, and doing the same for B_j, we can check whether the $\sqrt{d_1}$-size set A_i and the $\sqrt{d_2}$-size set B_j contain an orthogonal pair with one call to OR-AND-OR2$_{d_1,d_2}$. There are several ways to do this. For example, if the vectors of A_i are $a_1, \ldots, a_{\sqrt{d_1}}$ and the vectors of B_j are $b_1, \ldots, b_{\sqrt{d_1}}$, then we may define the (d_1d_2)-dimensional vectors

$$v_{A_i} := (a_1, \ldots, a_1,\ a_2, \ldots, a_2,\ \ \cdots\ \ , a_{\sqrt{d_1}}, \ldots, a_{\sqrt{d_1}}),$$

$$v_{B_i} := (b_1, b_2, \ldots, b_{\sqrt{d_1}},\ b_1, b_2, \ldots, b_{\sqrt{d_1}},\ \ \cdots\ \ , b_1, b_2, \ldots, b_{\sqrt{d_1}}),$$

where the '…' in the v_{A_i} denote $\sqrt{d_1}$ repetitions of the same vector. Then,

$$\text{OR-AND-OR2}_{d_1,d_2}(v_{A_i}, v_{B_j}) = 0 \iff \text{there is no orthogonal pair in } (A_i, B_j).$$

Constructing the sets of vectors $A' = \{v_{A_1}, \ldots, v_{A_{O(n/\sqrt{d_1})}}\}$ and $B' = \{v_{B_1}, \ldots, v_{B_{O(n/\sqrt{d_1})}}\}$, we conclude that computing OR-AND-OR2$_{d_1,d_2}$ on all pairs of vectors in A' and B' will determine whether A, B has an orthogonal vector. By assumption, this computation can be done in $\tilde{O}((n/\sqrt{d_1})^2) \leq \tilde{O}(n^2/d_1)$ time, which finishes the proof. ◀

▶ **Corollary 15.** *If there is an $\varepsilon > 0$ such that for all $c \geq 1$ the hypothesis of Theorem 14 is true with $d_1 \geq n^\varepsilon$ and $d_2 \geq c \log n$, then SETH is false.*

Proof. Follows from combining Theorem 13 and Theorem 14. ◀

The above relations between OV and other problems show that finding orthogonal pairs of vectors is of importance. Recently, fast evaluation algorithms for OR-AND-OR2 have been developed, tailored to run faster than what's known for OR-AND-COMP (used to solve APSP in the previous section):

▶ **Theorem 16** (Implicit in Abboud, Williams, Yu [4]). *The function OR-AND-OR2$_{s,d}$ can be evaluated on two sets of n vectors in $\tilde{O}(n^2)$ time, provided that*

$$s^2 \cdot \binom{d+1}{3\log s}^2 \leq n^{0.1}.$$

The algorithm of Theorem 16 is obtained by converting OR-AND-OR2 into a probabilistic polynomial over $\mathbb{F}_2$ (via Theorem 3) and carefully counting the monomials that arise in the construction of the polynomial. In particular, each AND is converted into a $3\log s$-degree probabilistic polynomial with error less than $1/s^3$, and the topmost OR on s variables is converted into a product of two random MOD2s. After $O(\log n)$ evaluations of these probabilistic polynomials for OR-AND-OR2, we settle on the correct values for OR-AND-OR2 on all n^2 pairs of points. The fast $\tilde{O}(n^2)$ time evaluation is again done using Coppersmith's fast rectangular matrix multiplication [15].

The inequality of Theorem 16 holds for $s \leq n^{\varepsilon/\log(d/\log n)}$ where $\varepsilon > 0$ is sufficiently small. From this and the above theorems, we derive:

▶ **Corollary 17.**

- *OV on n vectors in d dimensions is in $n^{2-1/O(\log(d/\log n))}$ time.*
- *BPM on n strings of length d each is in $n^{2-1/O(\log(d/\log n))}$ time.*
- *LCS* on two strings of length n is in $n^2/2^{\Omega(\sqrt{\log n})}$ time.*
- *CNF-SAT on n variables and m clauses is solvable in $2^{n(1-1/O(\log(m/n))}$ time.*

For the first three problems, these running times significantly improve upon prior work. The running time stated for CNF-SAT is not new, but it does match (up to constant factors in the big-O) the best known CNF-SAT algorithms, which is fairly surprising given the generality of this approach.

4.3 Counting Solutions to OV and CNF-SAT

Applying probabilistic polynomials over $\mathbb{Z}$ instead of $\mathbb{F}_2$, we can count the number of solutions to an OV instance or a CNF formula. Let us remark that these results have not appeared in print before; while they are not significant extensions of the previous section, they should still give the reader a sense of what else is possible.

Define the function SUM-AND-OR2$_{d_1,d_2} : \{0,1\}^{d_1 \cdot d_2} \to \mathbb{N}$ as:

$$\text{SUM-AND-OR2}_{d_1,d_2}(x_{1,1}, x_{1,2}, \ldots, x_{d_1,d_2}, y_{1,1}, y_{1,2}, \ldots, y_{d_1,d_2}) = \sum_{i=1}^{d_1} \bigwedge_{j=1}^{d_2} (x_{i,j} \vee y_{i,j}).$$

That is, this function outputs the total sum (over the integers) of the true AND-OR2s.

▶ **Theorem 18.** *There is a probabilistic polynomial for SUM-AND-OR2$_{d_1,d_2}$ over $\mathbb{Z}$ with error at most $1/3$ and at most $(d_1)^{O(\log^2 d_2)}$ monomials.*

Proof. Think of the SUM-AND-OR2$_{d_1,d_2}$ as a circuit. Replace each of the d_1 ANDs of fan-in d_2 in this circuit with a probabilistic polynomial over $\mathbb{Z}$ with error set to $\varepsilon = 1/(3d_1)$. By Theorem 4, these polynomials have degree $O(\log d_2 \cdot \log d_1)$, and therefore they have at

most $(d_2)^{O(\log d_2 \cdot \log d_1)}$ monomials, assuming the output of each OR2 gate is a variable in the polynomial. Now, each OR2 can be represented exactly as a sum of three monomials in the original variables, which means we obtain a polynomial with at most $(3d_2)^{O(\log d_2 \cdot \log d_1)} \leq (d_1)^{O(\log^2 d_2)}$ monomials in the original variables. Since each AND had error at most $1/(3d_1)$, their total sum is correct with probability at least $2/3$, by the union bound. ◀

Now, provided that d_1 and d_2 satisfy

$$(d_1)^{O(\log^2 d_2)} \leq n^{0.1},$$

the number of monomials is low, and we can apply the same strategy used in Theorem 14 to solve OV. Since we are taking a SUM instead of an OR, we can now compute the *number* of all orthogonal pairs in a set of d_2-vectors of size $O(\sqrt{d_1})$, in $\tilde{O}(n^2)$ time. The above inequality is certainly achieved when $d_1 \leq n^{1/O(\log^2 d_2)}$. Following the proof of Theorem 14 and computing the number of orthogonal pairs for all $O(n^2/d_1)$ pairs of sets, we obtain:

▶ **Theorem 19.** *The number of orthogonal pairs among n vectors in d dimensions is computable in $n^{2-1/O(\log^2 d)}$ time, with high probability. Consequently, one can count the number of matches in the database on a set of n BPM queries of length d in the same running time, and we can count the number of satisfying assignments to a CNF on n variables and m clauses in $2^{n(1-1/O(\log^2 m))}$ time.*

For counting OV pairs, the above running time is still much faster than $O(n^2/\text{poly}(\log n))$ when $d = \text{poly}(\log n)$. Indeed, it follows that counting the satisfying assignments of a CNF with n variables and $n^{\log n}$ clauses can be done in $2^{n-n/\text{poly}(\log n)}$ time.

5 Conclusion

We have seen several ways in which polynomial tools originally developed in circuit complexity have recently led to many new algorithms. We have not discussed all recent applications of the polynomial method: we've mostly ignored the (more obvious) application of the polynomial method for circuit lower bounds to solving *circuit satisfiability*. For example, the polynomial method tools discussed here also can be used to give faster algorithms for satisfiability of ACC circuits [47], as well as 0-1 linear programming [46] and satisfiability of symmetric Boolean CSPs [4].

We cannot help but point out a discrepancy between the usage of polynomials in circuit complexity and our algorithmic applications thus far. The majority of circuit lower bound results using polynomials focus on minimizing the *degree* of the polynomial representing the low-complexity function. However, for our applications, the *number of monomials*, or the sparsity, is the most important measure for our algorithmic applications. Certainly, a degree-d polynomial in n variables has $n^{O(d)}$ monomials, but this may be an undesirable representation for super-constant d. This survey shows that finding *sparse* polynomial representations for low-complexity functions like OR-AND-OR2 would entail significant algorithmic consequences.

Acknowledgements. I am grateful to Venkatesh Raman for suggesting the topic of this article, and his subsequent patience with me while I was finishing it.

References

1 Scott Aaronson. The polynomial method in quantum and classical computing. In *FOCS*, pages 3–3. IEEE, 2008.

2 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *SODA*, 2015.

3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, 2014.

4 Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *SODA*, page to appear, 2015.

5 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

6 James Aspnes, Richard Beigel, Merrick Furst, and Steven Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994.

7 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001.

8 Richard Beigel. The polynomial method in circuit complexity. In *In Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 82–95. IEEE Computer Society Press, 1995.

9 Richard Beigel, Nick Reingold, and Daniel Spielman. The perceptron strikes back. In *Structure in Complexity Theory Conference*, pages 286–291. IEEE, 1991.

10 Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, pages 350–366, 1994.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *IEEE Conf. Computational Complexity*, pages 252–260, 2006.

12 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

13 Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008. See also WADS'05.

14 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. See also STOC'07.

15 Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982.

16 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.

17 Robert W. Floyd. Algorithm 97. *Comm. ACM*, 5-6:345, 1962.

18 Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):49–60, 1976. See also FOCS'75.

19 Parikshit Gopalan and Rocco A. Servedio. Learning and lower bounds for AC0 with threshold gates. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 588–601. Springer, 2010.

20 Yijie Han. Improved algorithm for all pairs shortest paths. *Information Processing Letters*, 91(5):245–250, 2004.

21 Yijie Han. An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest path. *Algorithmica*, 51(4):428–434, 2008. See also ESA'06.

22 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

23 Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2):49–51, 1982.

24 Adam R. Klivans and Rocco Servedio. Learning DNF in time $2^{O(n^{1/3})}$. In *STOC*, pages 258–265. ACM, 2001.

25 Swastik Kopparty and Srikanth Srinivasan. Certifying polynomials for AC^0(parity) circuits, with applications. In *FSTTCS*, pages 36–47, 2012.

26 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.

27 Yishay Mansour. An $o(n^{\log\log n})$ learning algorithm for dnf under the uniform distribution. *Journal of Computer and System Sciences*, 50(3):543–550, 1995.

28 Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, 1969.

29 Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *Journal of Computer and System Sciences*, 69(3):421–434, 2004.

30 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

31 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994.

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

33 A. A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

34 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *Algorithms–ESA 2004*, pages 580–591. Springer, 2004.

35 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

36 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

37 Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43(4):195–199, 1992. See also WG'91.

38 Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998. See also WG'95.

39 Tadao Takaoka. An $O(n^3 \log\log n/\log n)$ time algorithm for the all-pairs shortest path problem. *Information Processing Letters*, 96(5):155–161, 2005.

40 Jun Tarui. Probabilistic polynomials, AC0 functions and the polynomial-time hierarchy. *Theoretical computer science*, 113(1):167–183, 1993.

41 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654. IEEE, 2010.

42 Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer, 1999.

43 Stephen Warshall. A theorem on Boolean matrices. *J. ACM*, 9:11–12, 1962.

44 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. See also ICALP'04.

45 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673, 2014.

46 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *STOC*, pages 194–202, 2014.

47 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2, 2014.

48 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *SODA*, pages 1867–1877. SIAM, 2014.

49 Andrew Chi-Chih Yao. On ACC and threshold circuits. In *FOCS*, pages 619–627, 1990.

50 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Lecture Notes in Computer Science*, volume 72, pages 216–226. Springer, 1979.

51 Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In *ISAAC 2004*, volume 3341 of *Springer LNCS*, pages 921–932, 2004.