Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky

Timothy M. Chan* Ryan Williams†

Abstract

We show how to solve all-pairs shortest paths on n nodes in *deterministic* $n^3/2^{\Omega(\sqrt{\log n})}$ time, and how to count the pairs of orthogonal vectors among n 0-1 vectors in $d = c\log n$ dimensions in *deterministic* $n^{2-1/O(\log c)}$ time. These running times essentially match the best known randomized algorithms of (Williams, STOC'14) and (Abboud, Williams, and Yu, SODA 2015) respectively, and the ability to count was open even for randomized algorithms. By reductions, these two results yield faster deterministic algorithms for many other problems. Our techniques can also be used to deterministically count k-SAT assignments on n variable formulas in $2^{n-n/O(k)}$ time, roughly matching the best known running times for detecting satisfiability and resolving an open problem of Santhanam (2013).

A key to our constructions is an efficient way to deterministically simulate certain probabilistic polynomials critical to the algorithms of prior work, carefully applying small-biased sets and modulus-amplifying polynomials.

1 Introduction

We investigate a recently introduced method for randomized algorithm design that was inspired by lower bound techniques in low-depth circuit complexity. The *polynomial method* in circuit complexity is a general strategy for proving circuit lower bounds, by (a) modeling low-complexity functions *approximately* with low-degree polynomials, then (b) proving that certain simple functions do not have low-degree polynomial approximations. A generation of papers (for instance [Raz87, Smo87, Yao90, BT94, BRS91, ABFR94, NS94, PS94, Bei95, KS12]) have proved circuit complexity lower bounds in this generic way.

In most cases, the approximations computed by these polynomials are a *worst-case* notion: for every possible input, a random choice of a low-degree polynomial (from some efficiently computable distribution) correctly computes the desired function, with high probability. This notion of *probabilistic polynomial* has been recently applied to algorithm design in the following sense:

- Start with an algorithm whose worst-case running time is constrained by a subprocedure that is executed many times on different inputs. (For example, if one is searching for a pair of strings with some property P, a redundant subprocedure might be the property test for P. Another example would be a matrix multiplication algorithm which computes many inner products on different pairs of vectors.)
- Next, give a way to model this subprocedure with a low-complexity circuit.
- Finally, randomly convert that low-complexity circuit into a sparse polynomial (via the polynomial method), and use the algebraic structure of the polynomial to speed up the repeated execution of the subprocedure. (Multiple random conversions may be needed, to increase the probability of success.)

This approach has led to faster randomized algorithms for solving several fundamental problems: all-pairs shortest paths (and therefore min-plus convolution, minimum weight triangle, minimum cycle, second shortest paths, replacement paths, etc.) [Wil14b], finding a pair of disjoint vectors in a collection (and therefore computing partial match queries in batch, computing the longest common substring with wildcards, evaluating DNFs on many chosen assignments, etc.) [Wil14a, AWY15], and (recently) computing Hamming distance queries in batch as well [AW15].

A fundamental characteristic of the polynomial method is its reliance on *probability* and *approximation*: for the method to work, it is necessary that the polynomial representations are random/approximate. For example, representing the OR function on n bits exactly requires $\Omega(2^n)$ monomials and degree $\Omega(n)$ over any fixed field, but probabilistic and approximate representations can be much more succinct over finite fields, requiring only $O(\text{poly}(n))$ monomials and degree $O(1)$ to achieve less than 1% probability of error [Raz87, Smo87]. Such a sparse representation is simply impossible to achieve deterministically/exactly.

Therefore it is somewhat surprising that, in this paper, we strongly *derandomize* almost all of the algorithms of prior work: the running times of our deterministic algorithms

*Cheriton School of Computer Science, University of Waterloo, tmchan@uwaterloo.ca. Supported by an NSERC grant.

†Computer Science Department, Stanford University, rrw@cs.stanford.edu. Supported in part by a David Morgenthaler II Faculty Fellowship, a Sloan Fellowship, and NSF CCF-1212372. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

essentially match those of the randomized ones. To do this, we show how the applications of probabilistic polynomials in the previous algorithms can be avoided, by deterministically and efficiently constructing algebraic counterparts to these polynomials. In some cases (for example, in APSP) we rely on the structure of the underlying problem in order to achieve the derandomization.

1.1 Deterministic All-Pairs Shortest Paths

In the all-pairs shortest paths (APSP) problem, we are given an n-node graph with arbitrary integer edge weights, and wish to compute a representation of the shortest paths between all pairs of nodes in the graph. It is well-known (see for example Seidel [Sei95]) that such a representation can be represented with $O(n^2 \log n)$ bits, in the form of a *successor matrix*. (Alternatively, we may wish to compute the shortest distances between every pair of nodes; our algorithm can solve both problems.) As is typical, we work over the *real RAM* model which has two kinds of registers: "real registers" holding elements of $\mathbb{R}$, and "word registers" holding $(\log n)$-bit words, where additions and comparisons of two real registers take unit time and arbitrary operations on two word registers can be done in unit time.

Decades of work on the dense case of APSP (such as [Flo62, War62, Fre75, Dob90, Tak91, Han04, Tak04, Zwi04, Cha05, Han06, Cha07, HT12]) yielded only $\log^{2-o(1)} n$ factor improvements over the well-known $O(n^3)$-time solution, until recently, Williams [Wil14b] showed that APSP can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time with a Monte Carlo algorithm. The paper [Wil14b] also proved that there is some $\delta > 0$ such that APSP can be solved in $O(n^3/2^{(\log n)^\delta})$ time deterministically, using a deterministic polynomial transformation due to Beigel and Tarui [BT94]. However, the value of δ was left undetermined; straightforward calculation indicates that $\delta \leq 1/4$. In this paper, we design a deterministic algorithm that runs in roughly the *same* time as the randomized algorithm:

THEOREM 1.1. *APSP on n-node weighted graphs can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time* deterministically.

1.2 Deterministic Orthogonal Vectors and Applications

We can also derandomize algorithms for another category of problems that have been successfully attacked with the polynomial method.

In the BATCH PARTIAL MATCH problem, we are given a database D of n points in $\{0,1\}^d$ and a set Q of n queries in $\{0,1,\star\}^d$; we wish to find for all $q \in Q$ a point $x \in D$ that matches all the non-star characters in q (or, report that no x exists). Equivalently, in BATCH BOOLEAN ORTHOGONAL VECTORS, the database $\mathscr{D}$ is a set of *vectors* from $\{0,1\}^d$ and Q is a set of vectors in $\{0,1\}^d$; we wish to determine for all $q \in Q$ if there is a $v \in \mathscr{D}$ such that $\langle q, v\rangle = 0$. Yet another equivalent problem is BATCH SUBSET QUERIES: given a database $\mathscr{D}$ of n subsets of $[d]$ and n queries Q which are subsets of $[d]$, determine for all $S \in Q$ if there is a $T \in \mathscr{D}$ such that $S \subseteq T$. These problems capture extremely basic yet difficult aspects of search, and have a long history (see [Riv74] and Section 6.5 of [Knu73]).

In the LONGEST COMMON SUBSTRING WITH WILDCARDS problem, we are given two strings $S, T \in (\Sigma \cup \{\star\})^n$; we wish to find the longest string over Σ that matches both a contiguous substring of S and a contiguous substring of T in the non-star characters.

In the DNF BATCHED EVALUATION problem, we are given a DNF formula F with d variables and t terms, and v different assignments; we wish to evaluate F on all v assignments.

It was shown by Abboud, Williams, and Yu [AWY15] that the difficulties in solving all the above problems stem from an apparently much simpler problem:

DEFINITION 1.1. (BOOLEAN ORTHOGONAL VECTORS) *Given n vectors in $\{0,1\}^{c\log n}$, are there distinct $u, v \in \{0,1\}^{c\log n}$ such that $\langle u, v\rangle = 0$ (over the integers)?*

The problem can be easily solved in $O(cn^2 \log n)$ time, or in $O(n^{c+1})$ time; it cannot be solved in $n^{1.999}$ time for all constants $c \geq 1$, unless the Strong Exponential Time Hypothesis is false [Wil04, WY14]. Abboud, Williams and Yu [AWY15] showed that BOOLEAN ORTHOGONAL VECTORS can in fact be solved in $n^{2-1/O(\log c)}$ time with a Monte Carlo algorithm, using the probabilistic polynomial method. This algorithm was used to derive faster algorithms for all aforementioned problems. We derandomize their algorithm, and in fact derive a stronger result: we can *count* the number of solutions in essentially the same running time.

THEOREM 1.2. *Given n vectors in $\{0,1\}^{c\log n}$ for any $c \leq 2^{o(\log n/\log\log n)}$, the number of distinct $u, v \in \{0,1\}^{c\log n}$ such that $\langle u, v\rangle = 0$ can be* counted *in $n^{2-1/O(\log c)}$ time* deterministically.

Applying the reductions of Abboud, Williams, and Yu [AWY15], we immediately obtain:

COROLLARY 1.1. *The following problems can be solved* deterministically*:*

- BATCH BOOLEAN ORTHOGONAL VECTORS, BATCH PARTIAL MATCH, *and* BATCH SUBSET QUERIES *in $n^{2-1/O(\log c)}$ time for $d = c\log n$ with $c \leq 2^{o(\log n/\log\log n)}$.*
- LONGEST COMMON SUBSTRING WITH WILDCARDS *in $n^2/2^{\Omega(\sqrt{\log n})}$ time.*

- DNF BATCHED EVALUATION *in* $v \cdot t^{1-1/O(\log c)}$ *time for* $t \le v$ *and* $d = c \log t$ *with* $c \le 2^{o(\log t/\log\log t)}$.

The fact that we can *count* solutions (and not just detect them) permits some new applications. For example:

COROLLARY 1.2. *The number of satisfying assignments to a CNF formula with cn clauses and n variables can be computed in* $2^{n-n/O(\log c)}$ *time* deterministically *for any* $c \le 2^{o(n/\log n)}$.

THEOREM 1.3. *Let* $k \ge 3$ *be a constant. The number of satisfying assignments to a k-CNF formula with n variables can be computed in* $2^{n-n/O(k)}$ *time* deterministically *for any constant k.*

The theorem resolves an open problem posed by Rahul Santhanam [HPSW13], who was motivated by the question of whether algorithms for counting k-SAT solutions could be made as competitive as algorithms for k-SAT. Impagliazzo, Matthews, and Paturi [IMP12] gave Las Vegas randomized algorithms for counting CNF-SAT with n variables and cn clauses in $2^{n-n/O(\log c)}$ time, and for counting k-SAT in $2^{n-n/O(k)}$ time. The best general deterministic k-SAT algorithms we could find in the literature are Dubois [Dub91] and Zhang [Zha96] who count solutions to k-SAT instances in $2^{n-n/O(2^k)}$ time. This is much worse than our bound (the running time converges to 2^n much faster).

1.3 Counting High-Dimensional Dominances in Subquadratic Time

Given a set of red vectors and a set of blue vectors in $\mathbb{R}^d$, a RED-BLUE DOMINATING PAIR is a pair of red and blue vectors (r,b) such that $r[i] < b[i]$ for all coordinates i.

Impagliazzo *et al.* [ILPS14] gave an $n^{2-1/\mathrm{poly}(c)}$ time algorithm for detecting or counting red-blue dominating pairs for n vectors in $\mathbb{R}^{c\log n}$; the runtime was subsequently refined to $n^{2-1/O(c\log^2 c)}$ by Chan [Cha15]. For dimensions greater than $\mathrm{poly}(\log n)$ this yields no improvement over exhaustively trying all pairs. On the other end of the dimensionality spectrum, an elegant algorithm of Matoušek [Mat91] can be used to enumerate *all* dominating pairs in $d \le n$ dimensions in $O(n^{(3+\omega)/2}) \le O(n^{2.69})$ time. For very high dimensions, this method improves on $O(n^2 d)$ time substantially. It is possible to make Matoušek's time bound sensitive to d by using rectangular matrix multiplication, but the method inherently requires $\Omega(n^2)$ time even for the detection or counting problem.

We can extend our method for counting orthogonal pairs of vectors to counting dominating pairs in high dimensions, noticeably faster than enumerating them:

THEOREM 1.4. *For all* $d \le 2^{c\sqrt{\log n}}$ *for a sufficiently small constant c, we can deterministically count the number of* RED-BLUE DOMINATING PAIRS *for n vectors in* $\mathbb{R}^d$ *in* $n^2/2^{\Omega(\sqrt{\log n})}$ *time.*

2 The Main Polynomial Construction: SUM of ORs

The key to our new algorithms is a succinct deterministic algebraic expression for the following function, which is repeatedly computed (with different parameters) in straightforward algorithms for all of the problems mentioned in this paper. Let d_1, d_2 be positive integers. Define the function $\text{SUM-OR}_{d_1,d_2} : \{0,1\}^{d_1 \cdot d_2} \to \{0,\ldots,d_1\}$ as

$$\text{SUM-OR}_{d_1,d_2}(x_{1,1},\ldots,x_{1,d_2},\ldots\ldots,x_{d_1,1},\ldots,x_{d_1,d_2}) := \sum_{i=1}^{d_1}\left(\bigvee_{j=1}^{d_2} x_{i,j}\right).$$

That is, $\text{SUM-OR}_{d_1,d_2}$ computes d_1 ORs on d_2 disjoint inputs, and returns the number of ORs that are true.

The following theorem states that we can deterministically produce a "somewhat short" multilinear polynomial simulating $\text{SUM-OR}_{d_1,d_2}$.

THEOREM 2.1. *Let* d_1, d_2 *be positive integers such that* $10\log(d_1 \cdot d_2) < d_2$. *There are integers* $\ell = 5\log(d_1 \cdot d_2)$, $M \le \mathrm{poly}(d_1, d_2)$, *and a polynomial* P_{d_1,d_2} *in* $d_1 \cdot d_2$ *variables over* $\mathbb{Z}$ *with* $m \le \binom{d_2}{2\ell} \cdot \mathrm{poly}(d_1 \cdot d_2)$ *monomials, such that for all* $\vec{x} \in \{0,1\}^{d_1 d_2}$, *SUM-OR*$_{d_1,d_2}(\vec{x})$ *equals the nearest integer to* $(P_{d_1,d_2}(\vec{x}) \bmod 2^\ell)/M$. *Furthermore,* P_{d_1,d_2} *can be constructed in* $\mathrm{poly}(d_1) \cdot \binom{d_2+1}{\ell}^2$ *time.*

For a concrete example of the theorem (which will also be useful in the case of APSP), let $d := d_1 = d_2$. Then Theorem 2.1 says:

COROLLARY 2.1. (OF THEOREM 2.1) *In* $2^{O(\log^2 d)}$ *time, we can build a polynomial* $P_{d,d}$ *over* $\mathbb{Z}$ *simulating SUM-OR*$_{d,d}$ *(in the above sense) with only* $2^{O(\log^2 d)}$ *monomials.*

In comparison, a deterministic polynomial *exactly* representing $\text{SUM-OR}_{d,d}$ over $\mathbb{Z}$ requires $\Omega(2^d)$ monomials (indeed, the OR function requires this many). The construction of Theorem 2.1 uses two mathematical ingredients, which we now introduce and motivate.

Small-bias sets. The Razborov–Smolensky [Raz87, Smo87] probabilistic polynomial for OR boils down to the following observation: for any non-zero vector $v \in \{0,1\}^n$, if $r \in \{0,1\}^n$ is a uniform random vector,

$$\Pr_r[\langle v, r\rangle = 0 \text{ over } \mathbb{F}_2] = 1/2.$$

This "random XOR" fact is frequently utilized in complexity theory (cf. Arora and Barak [AB09]). We would like to substitute the uniform random choice of r with deterministic choices. Trying all possible vectors r isn't efficient, however, since the sample space of possible vectors has size 2^n. Our first observation is that the sample space can be reduced considerably while achieving a similar property, using ε-*biased sets*:

DEFINITION 2.1. (NAOR AND NAOR [NN93]) *Let $\varepsilon \in (0,1/2)$. A set $S \subseteq \{0,1\}^n$ of n-dimensional vectors is* ε-biased *if for all non-zero $v \in \{0,1\}^n$,*

$$\Pr_{w \in S}[\langle v, w\rangle = 0 \text{ over } \mathbb{F}_2] \in (1/2-\varepsilon, 1/2+\varepsilon).$$

By the above observation, the set $S = \{0,1\}^n$ is 0-biased. An ε-biased set of small cardinality can be used to "simulate" the behavior of a uniform random XOR. Several deterministic constructions of ε-biased sets are known; for concreteness, let us cite a particular one with good dependence on n and ε:

THEOREM 2.2. (ALON *et al.* [AGHP92]) *For every positive integer n and $\varepsilon \in (0,1/2)$, there is an ε-biased set $S_{n,\varepsilon} \subseteq \{0,1\}^n$ of cardinality $\tilde{O}(n^2/\varepsilon^2)$, constructible in $\mathrm{poly}(n/\varepsilon)$ time.*

Modulus-amplifying polynomials. Small-bias sets let us substitute a completely random vector in a mod-2 inner product with an enumeration over a polynomial-sized set of vectors. Our second ingredient aids this enumeration: it is a special univariate polynomial $F_\ell(x)$ of degree $2\ell-1$ (for a parameter ℓ) that, given an integer a which is odd, $F_\ell(a)$ equals 1 mod 2^ℓ, else $F_\ell(a)$ equals 0 mod 2^ℓ. This polynomial F_ℓ lets us tally up a collection of "modulo 2" sums over $\mathbb{Z}$, by computing $F_\ell(A_1)+\cdots+F_\ell(A_k)$ where the A_i are various sums to be computed modulo 2. Such polynomials are called *modulus-amplifying* [Yao90], and were critical in both the proof of Toda's theorem [Tod91] (that the polynomial hierarchy is contained in $\mathsf{P}^{\#\mathsf{P}}$) and a famous depth-reduction theorem for ACC^0 circuits [BT94].

THEOREM 2.3. (BEIGEL AND TARUI [BT94]) *For every positive integer ℓ, the degree $2\ell-1$ polynomial*

$$F_\ell(y) = 1-(1-y)^\ell \sum_{j=0}^{\ell-1} \binom{\ell+j-1}{j} y^j \tag{2.1}$$

has the property that for all $y \in \mathbb{Z}$,

- *if $y \bmod 2 = 1$ then $F_\ell(y) \bmod 2^\ell = 1$, and*
- *if $y \bmod 2 = 0$ then $F_\ell(y) \bmod 2^\ell = 0$.*

In what follows, we will also use the fact that

$$F_\ell(0) = 1-(1-0)^\ell \sum_{j=0}^{\ell-1} \binom{\ell+j-1}{j} 0^j = 0. \tag{2.2}$$

The SUM-OR polynomial. We are now prepared to prove Theorem 2.1.

Proof of Theorem 2.1. Let $S \subseteq \{0,1\}^{d_2}$ be a $1/(4d_1)$-biased set; Theorem 2.2 guarantees that S can be constructed in $\mathrm{poly}(d_1 \cdot d_2)$ time and that $|S| \le \tilde{O}((d_1 \cdot d_2)^2)$. Set $M := |S|/2$ and $\ell := 5\log(d_1 \cdot d_2)$. Define

$$P_{d_1,d_2}(x_{1,1},\ldots,x_{1,d_2},\ldots\ldots,x_{d_1,1},\ldots,x_{d_1,d_2}) := \sum_{i=1}^{d_1} \sum_{v \in S} F_\ell\left(\sum_{j=1}^{d_2} v[j]\cdot x_{i,j}\right),$$

where $F_\ell(x)$ is the modulus-amplifying polynomial of (2.1) guaranteed by Theorem 2.3.

Let us first describe how to construct P_{d_1,d_2} efficiently. Let $G_i = F_\ell\left(\sum_{j=1}^{d_2} v[j]\cdot x_{i,j}\right)$. Each G_i has d_2 variables and degree $2\ell-1$. Notice that without loss of generality, we can make each G_i a multilinear polynomial of the same degree, since $x_i^2 = x_i$ over $\{0,1\}$. Therefore the total number of monomials in each G_i can be upper-bounded by

$$m_i = \sum_{j=0}^{2\ell-1} \binom{d_2}{j} \le O\left(\binom{d_2}{2\ell-1}\right),$$

where the above inequality holds when $2\ell-1 < d_2/2$.

Producing a multilinear version of each G_i can be done in $\tilde{O}\left(\binom{d_2+1}{\ell}^2\right)$ time, as follows. First, expand $s_j = \left(\sum_{j=1}^{d_2} v[j]\cdot x_{i,j}\right)^j$ for $j = 0,\ldots,\ell$ and $s_{\ell+1} = \left(1-\sum_{j=1}^{d_2} v[j]\cdot x_{i,j}\right)^\ell$ into a multilinear sum of products, each of which can be produced in $\tilde{O}\left(\binom{d_2+1}{\ell}\right)$ time. Then, multiply $s_{\ell+1}$ and s_j for $j = 0,\ldots,\ell-1$, and each of these products can be done in $\tilde{O}\left(\binom{d_2+1}{\ell}^2\right)$ time. (Computing the coefficients $\binom{\ell+j-1}{j}$ can easily be done in $\mathrm{poly}(\ell) \le \mathrm{poly}(\log(d_1\cdot d_2))$ time.) Finally, since P_{d_1,d_2} is just a sum of $O(d_1^3\cdot d_2^2)$ G_i's, the total number of monomials in P_{d_1,d_2} is $O\left(d_1^3 d_2^2 \cdot \binom{d_2}{2\ell-1}\right)$.

Now we argue for the correctness of P_{d_1,d_2}. For all $i = 1,\ldots,d_1$,

- If $\bigvee_{j=1}^{d_2} x_{i,j}$ is true, then the vector $(x_{i,1},\ldots,x_{i,d_2})$ is non-zero. Since S is ε-biased,

$$\Pr_{v \in S}\left[\sum_{j=1}^{d_2} v[j]\cdot x_{i,j} = 1 \text{ over } \mathbb{F}_2\right] \in (1/2-\varepsilon, 1/2+\varepsilon).$$

Since F_ℓ is modulus-amplifying, it follows that

$$Y_i := \sum_{v \in S} F_\ell\left(\sum_{j=1}^{d_2} v[j]\cdot x_{i,j}\right)$$

is congruent to an integer in the interval $((1/2-\varepsilon)|S|, (1/2+\varepsilon)|S|)$, modulo 2^ℓ.

- On the other hand, if $\bigvee_{j=1}^{d_2} x_{i,j}$ is false, then since $F_\ell(0) = 0$ (by (2.2)) we have

$$Y_i = \sum_{v \in S} F_\ell \left(\sum_{j=1}^{d_2} v[j] \cdot x_{i,j} \right) = 0.$$

For notational simplicity, let $K = \text{SUM-OR}_{d_1,d_2}(\vec{x})$. It follows from the above that, for all $\vec{x} \in \{0,1\}^{d_1 \cdot d_2}$, $P_{d_1,d_2}(\vec{x}) = \sum_{i=1}^{d_1} Y_i$ modulo 2^ℓ is congruent to an integer in the interval

$$((1/2-\varepsilon)|S| \cdot K, (1/2+\varepsilon)|S| \cdot K).$$

Since $\varepsilon = 1/(4d_1)$ and $0 \le \text{SUM-OR}_{d_1,d_2}(\vec{x}) \le d_1$, the above interval is contained in the interval

$$((1/2)|S|(K-1/2), (1/2)|S|(K+1/2)).$$

Observe that $2^\ell = (d_1 \cdot d_2)^5 > d_1 \cdot |S|$. It follows that

$$\frac{(P_{d_1,d_2}(\vec{x}) \bmod 2^\ell)}{M} \in (K-1/2, K+1/2).$$

(Note that the statement is trivially true in the special case when $K = \text{SUM-OR}_{d_1,d_2}(\vec{x}) = 0$, i.e., $\vec{x} = 0$.) □

3 A Deterministic APSP Algorithm

Our first application of the SUM-OR polynomial of Theorem 2.1 is to solve all-pairs shortest paths deterministically. We will use the polynomial simulating SUM-OR to evaluate an extension of the SUM-OR function on many pairs of points efficiently. The SUM-OR evaluation algorithm is then used to efficiently compute min-plus matrix multiplication, which is sufficient (and necessary) for solving APSP.

This approach is somewhat different from Williams' randomized APSP algorithm [Wil14b], which worked with expressions more complicated than SUM-OR, and reduced those expressions to polynomials over $\mathbb{F}_2$. The fact that APSP can be efficiently reduced to SUM-OR evaluation is interesting in its own right.

Given vectors $\vec{x}, \vec{y} \in \{0,1\}^m$, let $\vec{x} * \vec{y} \in \{0,1\}^m$ denote their element-wise product.

COROLLARY 3.1. *Given sets $X, Y \subseteq \{0,1\}^{d^2}$ with $|X| = |Y| = n$, SUM-OR$_{d,d}(\vec{x} * \vec{y})$ can be computed for every $\vec{x} \in X$ and $\vec{y} \in Y$ in $n^2 \cdot \text{poly}(\log n)$ deterministic time when $d \le 2^{c\sqrt{\log n}}$, for some fixed constant $c > 0$.*

Since this proof follows the pattern of prior work fairly closely, we give only a sketch.

Proof. (Sketch) Let $d \le 2^{O(\sqrt{\log n})}$, and $X, Y \subseteq \{0,1\}^{d^2}$, with $|X| = |Y| = n$. By Theorem 2.1, it suffices to evaluate the polynomial $P_{d,d}(\vec{x} * \vec{y})$ with $m \le \binom{d}{O(\log d)} \cdot \text{poly}(d) \le 2^{O(\log^2 d)}$ monomials over the ring $\mathbb{Z}/(2^\ell \mathbb{Z})$, on all $\vec{x} \in X$ and $\vec{y} \in Y$, where $\ell = O(\log d)$.

Williams [Wil14b] proves that, given any polynomial $P(\vec{x}, \vec{y})$ in $2d^2$ variables and $m \le n^{0.1}$ monomials over a field $\mathbb{F}$, we can evaluate P on all $\vec{x} \in X$ and $\vec{y} \in Y$ in $n^2 \cdot \text{poly}(\log n)$ arithmetic operations, by reducing the problem to evaluating P on an $n \times m$ and $m \times n$ rectangular matrix multiplication over $\mathbb{F}$. There is a minor complication in applying this result, because we need to evaluate $P_{d,d}(\vec{x} * \vec{y})$ over $\mathbb{Z}/(2^\ell \mathbb{Z})$, which is not a field. However, by inspection there is a fixed $c_0 \ge 1$ such that, for all $\vec{x}, \vec{y} \in \{0,1\}^{d^2}$, $P_{d,d}(\vec{x} * \vec{y}) \in [-d^{c_0 \log d}, d^{c_0 \log d}]$. Therefore, without loss of generality it suffices to compute the value of $P_{d,d}$ on all $(\vec{x}, \vec{y})$ pairs over a prime field $\mathbb{F}_p$, where $p \in [2^{c_0 \log^2 d}, 2^{1+c_0 \log^2 d}]$. The rest of the computations (the remainders modulo 2^ℓ, and divisions by M) can then be computed over $\mathbb{Z}$ in $\text{poly}(\log d)$ time, for each $(\vec{x}, \vec{y})$ pair.

Hence the SUM-OR evaluation problem can be reduced to the multiplication of an $n \times 2^{O(\log^2 d)}$ and an $2^{O(\log^2 d)} \times n$ matrix over $\mathbb{F}_p$, where each entry requires $O(\log^2 d)$ bits. For $c > 0$ sufficiently small, setting $d := 2^{c\sqrt{\log n}}$ implies that $m \le n^{0.1}$. By known results on rectangular matrix multiplication [Cop82] over fields, we can multiply an $n \times n^{0.1}$ and $n^{0.1} \times n$ matrix over $\mathbb{F}_p$ in $n^2 \cdot \text{poly}(\log n)$ time. □

Our APSP algorithm reduces APSP to the above SUM-OR evaluation problem of Corollary 3.1.

Reminder of Theorem 1.1 *APSP on n-node weighted graphs can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time* deterministically.

Proof. Let $A, B \in \mathbb{R}^{n \times n}$. Recall that the *min-plus matrix multiplication of A and B* is the $n \times n$ matrix

$$(A \star B)[i,j] := \min_{k=1,\ldots,n} (A[i,k] + B[k,j]).$$

First, it is well-known ([Mun71, FM71, AHU74]) that a $T(n)$-time algorithm for min-plus matrix multiplication implies an $O(T(n))$-time algorithm for APSP. Furthermore, it is also easy to see ([Fre75]) that if the min-plus matrix product of $n \times d$ and $d \times n$ matrices (with $d \le n$) is computable in $T'(n)$ time, then the min-plus matrix product of $n \times n$ matrices can be computed in $O\left(\frac{n}{d} \cdot T'(n)\right)$ time, by simply partitioning the given $n \times n$ matrices into n/d matrix products of dimensions $n \times d$ and $d \times n$.

We show how to efficiently reduce the min-plus product of an $n \times d$ matrix A and a $d \times n$ matrix B to the SUM-OR evaluation problem of Corollary 3.1. Let $c > 0$ be

the constant guaranteed in Corollary 3.1; our reduction will yield an $n^2 \cdot \text{poly}(\log n)$ time algorithm for min-plus matrix product of $n \times 2^{(c/4)\sqrt{\log n}}$ and $2^{(c/4)\sqrt{\log n}} \times n$ matrices; by the previous paragraph, it will follow that APSP is in $O(n^3/2^{(c/4)\sqrt{\log n}})$ time.

Set $d := 2^{(c/4)\sqrt{\log n}}$. Fix an index $q \in [\log d]$.[1] It suffices to describe how to compute, for every pair $i, j \in [n]$, the q-th bit of the index $k_{i,j} \in [d]$ that minimizes $A[i,k] + B[k,j]$. (From this, we can compute all $\log d$ bits of $k_{i,j}$ for all i, j, and recover the min-plus matrix product.) We call this q-th bit c^q_{ij} in the following.

Our reduction to SUM-OR evaluation requires some precomputation. Let K_q be the set of all indices in $[d]$ having their q-th bit equal to 1. For $k, k' \in [d]$, let $L_{k,k'}$ be the sorted list of $2n$ elements containing $A[i,k] - A[i,k']$ for all $i \in [n]$ and $B[k', j] - B[k, j]$ for all $j \in [n]$. These lists are computable in $O(d^2 \cdot n \log n)$ time. Let $r^{(i)}_{k,k'}$ be the rank of $A[i,k] - A[i,k']$ in $L_{k,k'}$, and let $s^{(j)}_{k,k'}$ be the rank of $B[k', j] - B[k, j]$ in $L_{k,k'}$.

By Fredman's trick [Fre75] that $A[i,k] + B[k,j] > A[i,k'] + B[k',j]$ implies $A[i,k] - A[i,k'] > B[k',j] - B[k,j]$, we have

$$(\neg c^q_{ij}) = \bigwedge_{k \in K_q} \bigvee_{k' \in [d]} [r^{(i)}_{k,k'} > s^{(j)}_{k,k'}],$$

where $[P]$ outputs 1 if property P is true, and 0 otherwise. Hence it suffices to compute the above AND of ORs of comparisons, over all $i, j \in [n]$.

We will use a SUM-OR evaluation to compute some of the $c^q_{i,j}$'s, and use a counting argument to directly compute the rest. Let $D := 2^{(3c/4)\sqrt{\log n}}$, and observe that $d \cdot D = 2^{c\sqrt{\log n}}$. The key idea is to compare rank values with multiples of n/D, which partition $[2n]$ into $O(D)$ "buckets"—this idea is inspired by Matoušek's dominance algorithm [Mat91], which will play a role again later in Section 6 (the observation that APSP is related to solving multiple dominance problems was noted, e.g., in [Cha05]). More precisely, for each $\ell \in [2 \cdot D]$, define the Boolean values

$$x^{(i)}_{k,k',\ell} = [r^{(i)}_{k,k'} > \ell n/D] \text{ and } y^{(j)}_{k,k',\ell} = [s^{(j)}_{k,k'} \leq \ell n/D].$$

Observe that all $x^{(i)}_{k,k',\ell}$ and $y^{(j)}_{k,k',\ell}$ can be computed in $O(n \cdot D \cdot d^2)$ time. We then compute for all i, j

$$e^q_{i,j} = \sum_{k \in K_q} \left(\bigvee_{k' \in [d], \ell \in [D]} (x^{(i)}_{k,k',\ell} \wedge y^{(j)}_{k,k',\ell}) \right).$$

Since $d \cdot D = 2^{c\sqrt{\log n}}$, the ORs of $e^q_{i,j}$ are over $2^{c\sqrt{\log n}}$ variables. The outer sum is over $|K_q| \leq d \leq 2^{(c/5)\sqrt{\log n}}$ ORs. Therefore $e^q_{i,j}$ can be computed in $n^2 \cdot \text{poly}(\log n)$ time for all $i, j \in [n]$, by applying Corollary 3.1.

Let $(i, j) \in [n]^2$. Call the pair (i, j) *bad* if $r^{(i)}_{k,k'}$ and $s^{(j)}_{k,k'}$ both lie in a common interval $((\ell - 1)n/D, \ell n/D]$ for some $k, k' \in [d], \ell \in [2 \cdot D]$. If (i, j) is not bad, then note that $(\neg c^q_{i,j}) = 1 \iff e^q_{i,j} = |K_q|$, so our computation of $e^q_{i,j}$ decides the bit $c^q_{i,j}$ for all not-bad pairs (i, j).

Now we consider the bad pairs. Each bad pair (i, j) can be specified by choosing k, k' (specifying $L_{k,k'}$), then choosing $\ell \in [2 \cdot D]$ and two numbers in the interval $((\ell - 1)n/D, \ell n/D]$. Hence the number of bad pairs is at most $O(d^2 \cdot D \cdot (n/D)^2) \leq O(d^2 n^2/D)$. Enumerating over all such choices, we can compute $c^q_{i,j}$ for each bad pair (i, j) by directly computing $\min_k(A[i,k] + B[k,j])$, in $O(d)$ time per pair. Hence the bad pairs can be enumerated in $O(d^3 n^2/D)$ time.

Therefore for $d = 2^{(c/4)\sqrt{\log n}}$, the bad pairs can be enumerated in $O(n^2)$ time by our choice of D. It follows that we can compute the min-plus product of the $n \times d$ matrix A and $d \times n$ matrix B in $n^2 \cdot \text{poly}(\log n)$ deterministic time. From the above discussion, it follows that APSP can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ deterministic time. □

4 A Deterministic Orthogonal Vectors Algorithm

We now turn to derandomizing the algorithms of Abboud, Williams, and Yu [AWY15]. An algorithm for BOOLEAN ORTHOGONAL VECTORS can be directly derived from the following theorem:

THEOREM 4.1. *For every positive $c \leq 2^{o(\log n/\log\log n)}$, there is a constant $k \geq 1$ such that for $s = n^{1/(k \log c)}$ the following holds. Let $S = \{u_1, \ldots, u_s\}$ and $T = \{v_1, \ldots, v_s\}$ be two sets of $(c \log n)$-dimensional Boolean vectors. The number of $u_i \in S$ and $v_j \in T$ such that $\langle u, v \rangle = 0$ can be computed by a multilinear polynomial $Q_s(u_1, \ldots, u_s, v_1, \ldots, v_s)$ in $s^2 \cdot c \log n$ variables and $O(n^{0.1})$ monomials, over $\mathbb{Z}$. Furthermore, the polynomial Q_s can be constructed in time $O(n^{0.2+1/\Omega(\log c)})$.*

Proof. Let $S = \{u_1, \ldots, u_s\}$ and $T = \{v_1, \ldots, v_s\}$. We wish to sum, over all s^2 pairs of vectors u_i and v_j from S and T, the value of $\bigvee_k (u_i[k] \cdot v_j[k])$; this will exactly count the number of non-orthogonal pairs (u_i, v_j). The function we wish to compute is precisely

$$\text{SUM-OR}_{s^2, c\log n}(u_1 * v_1, u_1 * v_2, \ldots, u_1 * v_s, \ldots \\ \ldots, u_s * v_1, u_s * v_2, \ldots, u_s * v_s),$$

where $u * v \in \{0,1\}^{c \log n}$ is the component-wise product of u and v. By Theorem 2.1, this function is computable with a polynomial $P_{s^2, c\log n}$ with a number of monomials equal to

$$m \leq \binom{2c \log n}{O(\log s + \log\log n)} \cdot \text{poly}(s^2 \cdot c \log n),$$

[1] $[m]$ denotes $\{1, \ldots, m\}$.

and $P_{s^2,c\log n}$ can be constructed in $\mathrm{poly}(s)\cdot\binom{2c\log n+1}{O(\log s+\log\log n)}^2$ time.

Setting $s=n^{1/(k\log c)}$ for a fixed and sufficiently large constant k, the number of monomials is at most

$$\mathrm{poly}(n^{1/(k\log c)},\log n)\cdot\binom{2c\log n}{10((\log n)/(k\log c)+\log\log n)} \le n^{O(1)/(k\log c)}\cdot\left(\frac{2ec\log n}{\frac{10\log n}{k\log c}}\right)^{\frac{10\log n}{k\log c}},$$

due to $c\le 2^{o(\log n/\log\log n)}$, and the inequality $\binom{n}{K}\le(en/K)^K$. Finally, the above quantity is less than

$$\begin{aligned} &n^{O(1)/(k\log c)}\cdot(ck\log c)^{10(\log n)/(k\log c)}\\ &\le n^{O(1)/(k\log c)}\cdot n^{10\log(ck\log c)/(k\log c)}\\ &\le n^{O(1)/(k\log c)}\cdot n^{10/k+(\log k)/(k\log c)+(\log\log c)/(k\log c)}\\ &\le k\cdot n^{0.1}, \end{aligned}$$

for sufficiently large constant k. □

Armed with Theorem 4.1, the orthogonal pairs counting algorithm follows from previous arguments.

Reminder of Theorem 1.2 *Given n vectors in $\{0,1\}^{c\log n}$ for any $c\le 2^{o(\log n/\log\log n)}$, the number of distinct $u,v\in\{0,1\}^{c\log n}$ such that $\langle u,v\rangle=0$ can be* counted *in $n^{2-1/O(\log c)}$ time* deterministically.

Proof. (Sketch) Analogous to [AWY15]. We partition the set of n vectors into $O(n/s)$ groups of at most s vectors each. Theorem 4.1 constructs a polynomial Q_s in $O(n^{0.1})$ monomials such that, for any pair of groups, we can count the number of orthogonal pairs among the pair of groups with a single evaluation to Q_s. Therefore the counting problem reduces to evaluation of Q_s on all $O(n^2/s^2)$ pairs of groups. As in the proof of Corollary 3.1, this evaluation of Q_s can be computed using a multiplication of an $n/s\times n^{0.1}$ and $n^{0.1}\times n/s$ matrix, where $s=n^{1/O(\log c)}$.

By inspection, the value of Q_s over all Boolean inputs is always an integer in the range $[d^{-c_0(\log s+\log d)},d^{c_0(\log s+\log d)}]$, for a sufficiently large constant $c_0>0$. Therefore, our matrix multiplication can be performed over a field of characteristic greater than $d^{c_0(\log s+\log d)}$, analogously to the proof of Corollary 3.1. By known results [Cop82], the matrix multiplication takes $(n^2/s^2)\cdot\mathrm{poly}(\log d,\log n)\le n^{2-1/O(\log c)}$ time, and recovering the number of orthogonal pairs from the resulting matrix entries can be done efficiently. □

5 Counting SAT Assignments Deterministically

The ability to *count* the number of orthogonal pairs yields some new applications, such as faster deterministic algorithms for counting satisfying assignments to CNF formulas.

Reminder of Corollary 1.2 *The number of satisfying assignments to a CNF formula with cn clauses and n variables can be computed in $2^{n-n/O(\log c)}$ time* deterministically *for $c\le 2^{o(n/\log n)}$.*

Proof. (Sketch) A simple reduction of Williams [Wil04] takes a CNF F with cn clauses and n variables and produces an orthogonal vectors instance with $O(2^{n/2})$ vectors in $2+cn$ dimensions, such that the number of orthogonal pairs equals the number of satisfying assignments to F. The result then follows from the orthogonal vectors algorithm (Theorem 1.2). □

We cite the next consequence as a "theorem" rather than a "corollary", as it does not immediately follow from the counting algorithm for orthogonal vectors.

Reminder of Theorem 1.3 *The number of satisfying assignments to a k-CNF formula on n variables can be computed in $2^{n-n/O(k)}$ time* deterministically *for any constant k.*

Proof. Given a k-CNF formula F on n variables and m clauses, let $\{C_1,\ldots,C_m\}$ be its set of clauses, and let $\#(F)$ denote its number of satisfying assignments. Note that $m\le O(n^k)$, without loss of generality.

Instead of reducing to the orthogonal vectors problem (in which the clause width k is lost in the reduction) we work directly with F. In particular, we consider the following expression which counts the number of falsifying assignments $(2^n-\#(F))$:

$$\sum_{a_1,\ldots,a_n\in\{0,1\}}\left(\bigvee_{i=1}^{m}p_i(a_1,\ldots,a_n)\right), \tag{5.3}$$

where the $\vee$ is an OR outputting 0 or 1, and p_i is a function such that

$$\begin{aligned} &p(a_1,\ldots,a_n)=0 \text{ if } (a_1,\ldots,a_n) \text{ satisfies } C_i\\ &p(a_1,\ldots,a_n)=1 \text{ if } (a_1,\ldots,a_n) \text{ does not satisfy } C_i. \end{aligned}$$

Since C_i has at most k literals, each p_i can easily be defined as a polynomial of degree at most k. For example, if $C_i=\{x_1,\neg x_2,x_3\}$ then $p_i(x_1,\ldots,x_n)=(1-x_1)x_2(1-x_3)$.

Let $\delta\in(0,1)$ be a parameter, and define the $(1-\delta)n$-variate expression

$$P(x_1,\ldots,x_{n(1-\delta)}):=\sum_{a_1,\ldots,a_{\delta n}\in\{0,1\}}\bigvee_{i=1}^{m}p_i(a_1,\ldots,a_n). \tag{5.4}$$

Then it is clear that

$$\begin{aligned} &\sum_{b_1,\ldots,b_{n-\delta n}\in\{0,1\}}P(b_1,\ldots,b_{n(1-\delta)})\\ &=\sum_{a_1,\ldots,a_n\in\{0,1\}}\bigvee_{i=1}^{m}p_i(a_1,\ldots,a_n). \end{aligned}$$

So to compute $\#(F)$, it suffices to evaluate $P(x_1,\ldots,x_{n(1-\delta)})$ on all $2^{n(1-\delta)}$ Boolean assignments.

Set $\ell := 5(\delta n + \log m)$, and $\varepsilon := 1/(4\cdot 2^{\delta n})$. Let $S_{m,\varepsilon} \subseteq \{0,1\}^m$ be an ε-biased set of size $\tilde{O}(m^2/\varepsilon^2)$ (from Theorem 2.2), and let $F_\ell(x)$ be a modulus-amplifying polynomial of degree ℓ (from Theorem 2.3). By Theorem 2.1, computing the expression $P(x_1,\ldots,x_{n(1-\delta)})$ of (5.4) on a Boolean assignment is equivalent to computing the polynomial

(5.5)
$$\sum_{a_1,..,a_{\delta n}\in\{0,1\}} \sum_{\vec{r}\in S_{m,\varepsilon}} F_\ell\left(\sum_{i=1}^m r_i \cdot p_i(x_1,\ldots,x_{n(1-\delta)},a_1,\ldots,a_{\delta n})\right).$$

For every vector $\vec{r}=(r_1,\ldots,r_m)\in S_{m,\varepsilon}$, each term $F_\ell(\sum_{i=1}^m r_i \cdot p_i(x_1,\ldots,x_{n(1-\delta)},a_1,\ldots,a_{\delta n}))$ can be written as a multilinear polynomial in $n(1-\delta)$ variables, of degree at most $(2\ell-1)\cdot k$. Therefore we can expand the expression 5.5 into a sum of monomials, in time no more than $O^\star\left(2^{\delta n}\cdot|S_{m,\varepsilon}|\cdot\sum_{i=0}^{(2\ell-1)\cdot k}\binom{n(1-\delta)}{i}\right)$.[2] When

(5.6)
$$(2\ell-1)\cdot k < n(1-\delta)/2,$$

the time for converting (5.5) into a sum of monomials is at most

$$O^\star\left(2^{\delta n}\cdot|S_{m,\varepsilon}|\cdot\binom{n}{(2\ell-1)\cdot k}\right).$$

This time bound can be achieved by "slowly" producing (5.5): inside of each F_ℓ, we compute the powers of sums by multiplying one factor at a time (no repeated squaring).

Since it is now a sum of monomials, we can now evaluate the polynomial of (5.5) on all $2^{n-\delta n}$ Boolean assignments to its variables in only $O^\star(2^{n(1-\delta)})$ time, using a simple divide-and-conquer strategy (cf. [Wil11]) or Yates' dynamic programming algorithm (cf. [BHK09], Section 2.2). Therefore we can count the number of satisfying assignments to F in time

$$O^\star\left(2^{n(1-\delta)}+2^{\delta n}\cdot|S_{m,\varepsilon}|\cdot\binom{n(1-\delta)}{(2\ell-1)\cdot k}\right)$$
$$\le O^\star\left(2^{n(1-\delta)}+m^2 2^{3\delta n}\cdot\binom{n(1-\delta)}{10(\delta n+\log m)\cdot k}\right).$$

Supposing $\delta := 1/(ck)$ for some sufficiently large constant $c>1$, inequality 5.6 holds, and the running time is upper bounded by $O^\star(2^{n(1-1/(ck))})$. In particular, the running time for constructing the $(n-\delta n)$-variate polynomial (modulo $\mathrm{poly}(n,m)$ factors) is at most

$$2^{2n/(ck)}\cdot\binom{n(1-1/(ck))}{10n/c+10k\log m}.$$

[2]The $O^\star$ notation omits $\mathrm{poly}(n)$ factors from the running time.

Letting $c=30$, the above is at most

$$2^{n/(15k)}\cdot\binom{n(1-1/(30k))}{n/3+10k\log m} < 2^{n/(15k)+.919n},$$

which is less than $2^{n(1-1/(30k))}$ for all $k\ge 2$. (Note we have made little attempt to optimize c.) □

6 A Dominance Counting Algorithm

In the last section, we show how to count dominances in dimensions up to $2^{c\sqrt{\log n}}$ in subquadratic time. In particular, we give a subquadratic-time reduction from RED-BLUE DOMINATING PAIR with "short" n red and blue vectors in $\mathbb{R}^d$ to BOOLEAN ORTHOGONAL VECTORS with n vectors that are "somewhat short":

THEOREM 6.1. *If counting* BOOLEAN ORTHOGONAL VECTORS *with n vectors in d dimensions is in $T(n,d)$ time, then for any given $s\le n$, counting* RED-BLUE DOMINATING PAIRS *with n red and blue vectors in $\mathbb{R}^d$ is in $O(n^2d^2/s+T(n,2+ds))$ time. In particular, the reduction maps a* RED-BLUE DOMINATING PAIR *instance to a* BOOLEAN ORTHOGONAL VECTORS *instance with n Boolean vectors in $2+ds$ dimensions.*

Proof. For each $i\in[d]$, sort the $2n$ numbers in coordinate i from all red and blue vectors in $O(n\log n)$ time. Replace these numbers with their ranks in sorted order, using the ordering red $<$ blue to break ties. Let this ordered list of coordinates be L_i. Partition L_i into s contiguous "buckets" with $O(n/s)$ elements in each bucket. We count two disjoint kinds of dominating pairs:

1. First we count dominating pairs of vectors (r,b) with a coordinate i such that $r[i]$ and $b[i]$ are in the same bucket of L_i. These pairs can be counted in $O(n\cdot d\cdot(n/s)\cdot d) = O(n^2d^2/s)$ time, by enumerating all n red vectors r', all d coordinates i', and all blue vectors b' in the bucket defined by r' and i', then computing directly if b' dominates r'. Note we can store the list of all pairs found to avoid overcounting.

2. Second, we count dominating pairs (r,b) such that for all coordinates i, $r[i]$ and $b[i]$ appear in different buckets. Here we can follow a similar strategy as in an algorithm by Matoušek [Mat91] (who reduced a similar case to matrix multiplication). For those (r,b) pairs, we can replace their coordinates in $[2n]$ with the indices of the buckets they appear in, which are numbers in $[s]$. Then we can reduce to counting orthogonal pairs of Boolean vectors, but in dimension $2+d\cdot s$ instead: for each coordinate $i\in[d]$ of each red vector r, replace $r[i]$ with the standard basis vector $e_j\in\{0,1\}^s$, where j is the index of the bucket of L_i that contains $r[i]$. For each blue vector b and coordinate i, replace $b[i]$ with the 0-1 vector $\sum_{k\ge j'} e_k$ of length s, where j' is the index of the bucket of L_i that contains $b[i]$. Finally, append

two more coordinates onto each vector: append $1,0$ to the red vectors, and $0,1$ to the blue vectors.

We claim that the number of orthogonal pairs in this instance of $2+d\cdot s$ dimensions equals the number of dominating pairs of this second kind; this follows because $\langle e_j, \sum_{k\geq j'} e_k\rangle = 0$ (over the integers) if and only if $j < j'$ — this corresponds to the case where $r[i]$ is in a strictly smaller bucket than $b[i]$. □

Setting s to balance the factors of n^2d^2/s and $T(n,2+ds)$, we obtain a new time bound for counting dominances in high dimensions:

Reminder of Theorem 1.4 *For all $d \leq 2^{c\sqrt{\log n}}$ for a sufficiently small constant c, we can deterministically count the number of* RED-BLUE DOMINATING PAIRS *for n vectors in $\mathbb{R}^d$ in $n^2/2^{\Omega(\sqrt{\log n})}$ time.*

Proof. Applying the algorithm for counting orthogonal vectors of Theorem 1.2 to the reduction of Theorem 6.1, we obtain an algorithm for counting dominances that runs in time

$$O\left(n^2d^2/s + n^{2-1/O(\log(ds/(\log n))}\right).$$

Setting $s := 2^{3c\sqrt{\log n}}$ for $d \leq 2^{c\sqrt{\log n}}$, we obtain the claimed bound. □

7 Conclusion

We have shown that randomness is not essential to the recent algorithmic applications of the polynomial method, despite the fact that randomness and/or approximation are necessary in *circuit lower bound proofs* via the polynomial method. Let us highlight a few interesting further directions.

- In our construction of the SUM-OR polynomial (Theorem 2.1), we composed the output of the OR function with a univariate polynomial F_ℓ of degree $2\ell-1$. This led to a polynomial with about $\binom{d^2}{2\ell}$ monomials. Is there a *sparser* polynomial representation for SUM-OR? Such a polynomial would yield even faster algorithms, since in our applications we only care about the number of monomials in the polynomial.
- We have derandomized all algorithms of [Wil14b, AWY15, Wil14a]. In recent work [AW15], new probabilistic polynomials yield a subquadratic time algorithm for a nearest neighbor problem: for any constant c, given n red and blue Boolean vectors in $c\log n$ dimensions, the closest red-blue pair in the Hamming metric can be computed in $n^{2-1/O(c\log^2 c)}$ randomized time. However, the new polynomial construction is very different from Razborov-Smolensky, and we do not know how to derandomize it. Computing Hamming closest pairs in deterministic subquadratic time looks like a challenging open problem.
- Our algorithm for counting dominances in high dimensions also has applications to solving NP-hard problems: by a reduction of Impagliazzo, Paturi, and Scheider [IPS13] (Corollary 4.2), it follows that we can also count solutions to arbitrary 0-1 integer linear programs of up to $2^{o(\sqrt{n})}$ linear inequalities in $2^{n-\Omega(\sqrt{n})}$ time. Consider the following generalization: allow real-valued "weights" w_i on the coordinates of u's and v's, and for a threshold value $t \in \mathbb{R}$, count the number of pairs (u,v) such that

$$\left(\sum_k w_k \cdot [u[k] \leq v[k]]\right) > t.$$

  The ability to count "weighted thresholds of dominances" in subquadratic time would yield a faster algorithm for counting satisfying assignments to depth-two threshold circuits, which would (likely) imply depth-two circuit lower bounds [Wil14c].

Acknowledgments. We thank Rahul Santhanam, Huacheng Yu, and the anonymous referees for helpful comments and discussions.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[ABFR94] James Aspnes, Richard Beigel, Merrick Furst, and Steven Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Series in Computer Science and Information Processing, 1974.

[AW15] Josh Alman and Ryan Williams. Probabilistic polynomials and Hamming nearest neighbors. In *FOCS*, page to appear, 2015.

[AWY15] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *SODA*, pages 218–230, 2015.

[Bei95] Richard Beigel. The polynomial method in circuit complexity. In *Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 82–95. IEEE Computer Society Press, 1995.

[BHK09] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[BRS91] Richard Beigel, Nick Reingold, and Daniel Spielman. The perceptron strikes back. In *Structure in Complexity Theory Conference*, pages 286–291. IEEE, 1991.

[BT94] Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, pages 350–366, 1994.

[Cha15] Timothy M. Chan. Speeding up the Four Russians algorithm by about one more logarithmic factor. In *SODA*, pages 212–217, 2015.

[Cha05] 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.

[Cha07] 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.

[Cop82] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982.

[Dob90] Wlodzimierz Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *International Journal of Computer Mathematics*, 32(1-2):49–60, 1990.

[Dub91] Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science*, 81(1):49–64, 1991.

[Flo62] Robert W. Floyd. Algorithm 97. *Comm. ACM*, 5-6:345, 1962.

[FM71] Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *IEEE Symposium on Switching and Automata Theory*, pages 129–131, 1971.

[Fre75] 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.

[Han04] Yijie Han. Improved algorithm for all pairs shortest paths. *IPL*, 91(5):245–250, 2004.

[Han06] 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.

[HPSW13] Thore Husfeldt, Ramamohan Paturi, Gregory B. Sorkin, and Ryan Williams. Exponential Algorithms: Algorithms and Complexity Beyond Polynomial Time (Dagstuhl Seminar 13331). *Dagstuhl Reports*, 3(8):40–72, 2013.

[HT12] Yijie Han and Tadao Takaoka. An $O(n^3\log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In *SWAT*, volume 7357 of *Springer LNCS*, pages 131–141, 2012.

[ILPS14] Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *CoRR*, abs/1401.5512, 2014.

[IMP12] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In *SODA*, pages 961–972, 2012.

[IPS13] Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *FOCS*, pages 479–488, 2013.

[Knu73] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.

[KS12] Swastik Kopparty and Srikanth Srinivasan. Certifying polynomials for AC^0(parity) circuits, with applications. In *FSTTCS*, pages 36–47, 2012.

[Mat91] Jirí Matoušek. Computing dominances in E^n. *Inf. Process. Lett.*, 38(5):277–278, 1991.

[Mun71] Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1:56–58, 1971.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[NS94] Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994.

[PS94] Ramamohan Paturi and Michael E. Saks. Approximating threshold circuits by rational functions. *Inf. Comput.*, 112(2):257–272, 1994.

[Raz87] Alexander 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.

[Riv74] Ronald Linn Rivest. *Analysis of Associative Retrieval Algorithms.* PhD thesis, 1974.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

[Tak04] Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *Computing and Combinatorics*, volume 3106 of *Springer LNCS*, pages 278–289. 2004.

[Tak91] Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *IPL*, 43(4):195–199, 1992. See also WG'91.

[Tod91] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[War62] Stephen Warshall. A theorem on Boolean matrices. *J. ACM*, 9:11–12, 1962.

[Wil11] Ryan Williams. A casual tour around a circuit complexity bound. *SIGACT News*, 42(3):54–76, 2011.

[Wil14a] R. Ryan Williams. The polynomial method in circuit complexity applied to algorithm design (Invited Talk). In *FSTTCS*, pages 47–60, 2014.

[Wil14b] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673, 2014.

[Wil14c] Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *STOC*, pages 194–202, 2014.

[Wil04] 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.

[WY14] Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. pages 1867–1877. SIAM, 2014.

[Yao90] Andrew Chi-Chih Yao. On ACC and threshold circuits. In *FOCS*, pages 619–627, 1990.

[Zha96] Wenhui Zhang. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science*, 155(1):277–288, 1996.

[Zwi04] 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.