4
The Polynomial Method

Version: 158

In this chapter we will introduce an algorithm design technique called "polynomial method", and we will use it to design the fastest known algorithm for OV. We assume some familiarity with randomized algorithms. Recall that OV is the following problem.

Problem 4.1.

Orthogonal Vectors (OV)

Given: *Sets $A, B \subseteq \{0,1\}^d$ of size n*

Determine: *Is there an orthogonal pair $a \in A, b \in B$?*

Conjecture: *No $\mathcal{O}(n^{2-\varepsilon}\text{poly}(d))$-time algorithm exists.*

The naive algorithm for this problem runs in time $\mathcal{O}(n^2 d)$. In small dimension, we can make use of the fact that there are at most 2^d different vectors, leading to an algorithm that runs in time $\mathcal{O}(nd \cdot 2^d)$. This is subquadratic time for $d \leq (1-\varepsilon)\log n$.[1] So what about dimension $d = 100 \log n$? As it turns out, in this regime OV still is in strongly subquadratic time, as shown by our main result of this chapter.

[1] By "log" we always denote the logarithm base 2. One can use additional simple tricks to push this bound to $d \leq (2-\varepsilon)\log n$ and even slightly beyond.

Theorem 4.2. *(Polynomial Method for OV[2]) Let $c = c(n)$ with $c \geq 2$ and $c = n^{o(1/\log\log n)}$. There is a randomized algorithm solving OV in dimension $d = c \cdot \log n$ in time*

$$n^{2-\frac{1}{\mathcal{O}(\log c)}}.$$

[2] A. Abboud, R. Williams, and H. Yu. More applications of the polynomial method to algorithm design. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 218–230, 2015

Let us discuss the implications of this theorem.

- For any constant c, OV is in time $\mathcal{O}(n^{2-\varepsilon_c})$ for some constant $\varepsilon_c > 0$.
- The above theorem does not falsify OVH, since for any $c = \omega(1)$ the running time is $n^{2-o(1)}$, which is not strongly subquadratic. However, it yields non-trivial lower order improvements over quadratic time, e.g., for $c = \log n$ (thus, $d = \log^2 n$) the time bound is

$$n^{2-1/\mathcal{O}(\log\log n)} = n^2/2^{\Omega(\log n/\log\log n)}.$$

- Combining the above theorem with the reduction from k-SAT to OV from Lecture 2 recovers one of the best algorithms for k-SAT. What running time do we get for k-SAT? ($\rightarrow$ **exc.**)

Notation We will use a parameter s to be chosen later. For simplicity we assume that s divides n and we set $g := \frac{n}{s}$.

We denote by $\mathcal{OV} = \mathcal{OV}_{s,d}$ a function $\mathcal{OV}\colon \{0,1\}^{2sd} \to \{0,1\}$ that encodes the orthogonal vectors problem as follows. We will interchangeably interpret the input of $\mathcal{OV}$ as $2sd$ bits or as two sets of s vectors of length d. Thus, the function $\mathcal{OV}$ is given sets A, B of vectors in $\{0,1\}^d$, with $|A| = |B| = s$, and returns $\mathcal{OV}(A,B) = 1$ if there is an orthogonal pair $a \in A, b \in B$, and $\mathcal{OV}(A,B) = 0$ otherwise.

Algorithm Outline Our plan is to solve OV as follows on input A, B.

1. Split A, B into groups $A_1, \ldots, A_g, B_1, \ldots, B_g$ of size s each.
2. Compute a **suitable representation** of the function $\mathcal{OV} = \mathcal{OV}_{s,d}$.
3. Evaluate $\mathcal{OV}(A_i, B_j)$ for **all pairs** $1 \le i, j \le g$.
4. Output 'yes' if $\mathcal{OV}(A_i, B_j) = 1$ for some i, j, and 'no' otherwise.

The two non-trivial steps of this outline are steps 2 and 3 (note that naively step 3 takes time $O(n^2 d)$). These steps correspond to the two ingredients of the polynomial method. For step 2, we will convert the function $\mathcal{OV}$ into a *multivariate polynomial with few monomials* (or, more precisely, a distribution from which we will sample a random multivariate polynomial with few monomials). For step 3, we will show an efficient evaluation algorithm for multivariate polynomials with few monomials.

We start with the latter, i.e., by describing the basics of multivariate polynomials and showing how to evaluate them efficiently.

4.1 Multivariate Polynomial Evaluation

Basics of Multivariate Polynomials

Let $\mathbb{F}$ be a field (e.g., the real numbers $\mathbb{R}$). A **multivariate polynomial** over $\mathbb{F}$ is a function $p\colon \mathbb{F}^k \to \mathbb{F}$ of the form

$$p(X_1, \ldots, X_k) = \sum_{(i_1,\ldots,i_k) \in M} \alpha_{i_1,\ldots,i_k} \cdot X_1^{i_1} \cdots X_k^{i_k},$$

where M is a finite subset of $\mathbb{N}^k$. Each summand $X_1^{i_1} \cdots X_k^{i_k}$ is called a **monomial** and the corresponding factor $\alpha_{i_1,\ldots,i_k} \in \mathbb{F}$ is its **coefficient**.

In this lecture, we will always consider the field $\mathbb{F} = \mathbb{F}_2$.

Observation 4.3. *Over $\mathbb{F}_2$ we can assume without loss of generality that $M \subseteq \{0,1\}^k$. In particular, any polynomial with k variables has at most 2^k monomials.*

Multivariate means that the function p has multiple variables $X_1, \ldots, X_k$, in contrast to a *univariate* polynomial $p(X)$ which has only one variable.

We denote $\mathbb{N} = \{0, 1, 2, \ldots\}$.

Recall that $\mathbb{F}_2$ is the set $\{0,1\}$ with the operations "$\oplus$" (addition modulo 2) and "$\cdot$" (multiplication), defined by:

$$\begin{array}{ll} 0 \oplus 0 = 0 & 0 \cdot 0 = 0 \\ 0 \oplus 1 = 1 & 0 \cdot 1 = 0 \\ 1 \oplus 0 = 1 & 1 \cdot 0 = 0 \\ 1 \oplus 1 = 0 & 1 \cdot 1 = 1 \end{array}$$

Proof. Follows from $x^2 = x$ for any $x \in \mathbb{F}_2$. □

Example 4.4. *An example for a multivariate polynomial is* $p(X_1, X_2, X_3) = 2X_1X_2X_3 + 3X_1^2X_3 + X_2 - 1$. *Reducing its coefficients modulo 2, we obtain a polynomial over* $\mathbb{F}_2$, *namely* $X_1^2X_3 \oplus X_2 \oplus 1$. *This is the same function as* $X_1X_3 \oplus X_2 \oplus 1$ *over* $\mathbb{F}_2$, *as* $X_1^2 = X_1$.

Polynomial Evaluation (on Cartesian Products)

Let us write m for the number of monomials of polynomial p. For given values $x_1, \ldots, x_k \in \mathbb{F}$ we can evaluate the polynomial p at the point $(x_1, \ldots, x_k)$ in time $\mathcal{O}(km)$. In particular, we can evaluate p at n given points in time $\mathcal{O}(kmn)$.

This running time can be improved in case the given points form a **Cartesian product**. More precisely, suppose we are given "left half-points" $A_1, \ldots, A_g$ and "right half-points" $B_1, \ldots, B_g$ and our task is to evaluate p on all combinations (A_i, B_j). In particular, the output size is g^2. The following tool asserts that this polynomial evaluation problem can be solved in time **near-linear in the output size**, assuming that m is not too large.

Lemma 4.5. *Let* $A_1, \ldots, A_g, B_1, \ldots, B_g \in \mathbb{F}^k$, *let* $p\colon \mathbb{F}^{2k} \to \mathbb{F}$ *be a polynomial with* m *monomials, and suppose that* $m, k = \mathcal{O}(g^{0.1})$. *Then we can compute* $p(A_i, B_j)$ *for all pairs* $1 \le i, j \le g$ *in total time* $\mathcal{O}(g^2 \log^2 g)$.

We will use this lemma in order to implement step 3 of the algorithm overview given earlier.

To prove this lemma we will make use of *rectangular matrix multiplication*, which we introduce next.

(Rectangular) Matrix Multiplication

In **matrix multiplication** we are given an $n_1 \times n_2$-matrix A and an $n_2 \times n_3$-matrix B, both with entries in $\mathbb{F}$. The task is to compute the $n_1 \times n_3$-matrix $C = A \cdot B$ (again with entries in $\mathbb{F}$) given by

$$C[i,j] := \sum_{k=1}^{n_2} A[i,k] \cdot B[k,j] \qquad \text{for any } 1 \le i \le n_1, 1 \le j \le n_3.$$

A well-studied case is $n_1 = n_2 = n_3 = n$, where it is known that the problem can be solved in time $\mathcal{O}(n^\omega)$ with:

- $\omega \le 3$: naive evaluation,
- $\omega \le 2.81$: Strassen[3],
- ...
- $\omega \le 2.376$: Coppersmith and Winograd[4],
- $\omega \le 2.3736898$: Vassilevska Williams[5], Stothers[6],
- $\omega \le 2.3728639$: Le Gall[7],

[3] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal on Symbolic Computation*, 9(3):251–280, 1990

[5] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Annual ACM Symposium on Theory of Computing Conference (STOC'12)*, pages 887–898, 2012

[6] A. J. Stothers. On the complexity of matrix multiplication. 2010. PhD Thesis

[7] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Symposium on Symbolic and Algebraic Computation (ISSAC'14)*

Rectangular matrix multiplication refers to the case $n_1 = n_3 = n$ and $n_2 = n^\alpha$ for some small constant $\alpha > 0$. In this case, the input size is proportional to $n^{1+\alpha}$ (since A, B have $n^{1+\alpha}$ entries) and the output size is n^2. Surprisingly, rectangular matrix multiplication can be solved in **near-linear time** in the output size!

Theorem 4.6. *(Rectangular Matrix Multiplication[8]) Multiplication of an $n \times n^\alpha$-matrix with an $n^\alpha \times n$-matrix is in time $\mathcal{O}(n^2 \log^2 n)$ for any $\alpha \leq 0.172$.*

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

We will use this theorem as a black box, since its proof is well beyond the scope of this course. We refer to Bläser[9] for an introduction to fast matrix multiplication.

[9] M. Bläser. Fast matrix multiplication. *Theory of Computing, Graduate Surveys*, 5:1–60, 2013

Polynomial Evaluation on Cartesian Products, Continued

Equipped with rectangular matrix multiplication, let us design a fast algorithm for polynomial evaluation over a Cartesian product.

Proof of Lemma 4.5. We can write any polynomial p on $2k$ variables consisting of m monomials in the form

$$p(X_1, \ldots, X_k, Y_1, \ldots, Y_k) = \sum_{r=1}^{m} \alpha_r \cdot X_1^{s_{r,1}} \cdots X_k^{s_{r,k}} \cdot Y_1^{t_{r,1}} \cdots Y_k^{t_{r,k}}.$$

Denote by $A_{i,1}, \ldots, A_{i,k}$ the entries of the input vector $A_i \in \mathbb{F}^k$, similarly let $B_{j,1}, \ldots, B_{j,k}$ be the entries of B_j.

We start by computing the values

$$\hat{A}[i,r] := \alpha_r \cdot A_{i,1}^{s_{r,1}} \cdots A_{i,k}^{s_{r,k}}$$
$$\hat{B}[r,j] := B_{j,1}^{t_{r,1}} \cdots B_{j,k}^{t_{r,k}},$$

for $1 \leq i, j \leq g$ and $1 \leq r \leq m$. Note that $\hat{A}$ is a $g \times m$-matrix and $\hat{B}$ is an $m \times g$-matrix, so the product $C := \hat{A} \cdot \hat{B}$ is well-defined. We obtain

$$C[i,j] = \sum_{r=1}^{m} \hat{A}[i,r] \cdot \hat{B}[r,j] \stackrel{!}{=} p(A_i, B_j).$$

The notation $\stackrel{!}{=}$ indicates a step that needs more thought than usual to verify.

The matrices $\hat{A}$ and $\hat{B}$ can be constructed in time $\mathcal{O}(gmk)$. By the assumption $m, k = \mathcal{O}(g^{0.1})$, this takes time $\mathcal{O}(g^2)$. Using rectangular matrix multiplication (Theorem 4.6), matrix C can be computed in time $\mathcal{O}(g^2 \log^2 g)$. □

4.2 *Conversion to Multivariate Polynomial*

In order to use the tools developed in the last section, the obvious plan now is to convert the function $\mathcal{OV}\colon \{0,1\}^{sd} \to \{0,1\}$ to a multivariate polynomial with few monomials.

We start by expressing $\mathcal{OV}$ as a **Boolean circuit** (or more precisely Boolean formula). That is, we formally define the behaviour of $\mathcal{OV}$ using the operations $\wedge$ (and), $\vee$ (or), and $\neg$ (negation). Writing the input to $\mathcal{OV}$ as $A = \{a_1, \ldots, a_s\}, B = \{b_1, \ldots, b_s\} \subset \{0,1\}^d$, observe that we can express $\mathcal{OV}$ as follows:

$$\mathcal{OV}(A,B) := \bigvee_{1 \le i,j \le s} \mathrm{Orth}(a_i, b_j)$$
$$\mathrm{Orth}(a,b) := \bigwedge_{\ell=1}^{d} (\neg a[\ell]) \vee (\neg b[\ell]).$$

Our plan is to transform this Boolean circuit into a polynomial over $\mathbb{F}_2$. That is, instead of using the logical operations $\wedge, \vee, \neg$ we now want to use the operations $\oplus, \cdot$ over $\mathbb{F}_2$.

Lemma 4.7. *For converting between logical and $\mathbb{F}_2$-operations, we may use the following rewriting rules:*

$$\begin{aligned} \neg x &= 1 \oplus x \\ x_1 \wedge x_2 \wedge \ldots \wedge x_n &= x_1 \cdot x_2 \cdots x_n \\ x_1 \vee x_2 \vee \ldots \vee x_n &= 1 \oplus \big((1 \oplus x_1) \cdot (1 \oplus x_2) \cdots (1 \oplus x_n)\big). \end{aligned}$$

Proof. The first two of these rewriting rules can be easily checked from the definitions of the involved operations. For the last rule, we use the DeMorgan rule to express

$$x_1 \vee \ldots \vee x_n = \neg\big((\neg x_1) \wedge \ldots \wedge (\neg x_n)\big),$$

and then apply the first two rewriting rules. □

Applying these rules to our logical formulation of $\mathcal{OV}$ would yield the following expression:

$$\mathcal{OV}(A,B) = 1 \oplus \prod_{1 \le i,j \le s} \big(1 \oplus \mathrm{Orth}(a_i, b_j)\big)$$
$$\mathrm{Orth}(a,b) = \prod_{\ell=1}^{d} \big(1 \oplus a[\ell] \cdot b[\ell]\big).$$

In order to express $\mathcal{OV}$ as a sum of monomials, we need to expand the products in the above expression. However, already the expanded form of $\mathrm{Orth}(a,b)$ consists of 2^d monomials, which is too large to even construct. This issue also propagates to $\mathcal{OV}$. In fact, it can be shown that any function $f\colon \{0,1\}^k \to \{0,1\}$ has a **unique** representation as a polynomial over $\mathbb{F}_2$ (in the form of Observation 4.3), and therefore this is an inherent issue: $\mathcal{OV}$ has no representation as a polynomial over $\mathbb{F}_2$ with less than 2^d monomials!

Probabilistic Polynomials

A typical workaround when exact representations fail is to **relax** the goal and ask for representations that may **err** on some inputs, but are correct almost everywhere. This notion is particularly useful when it is combined with **randomness**, in such a way that the errors occur at random places. More precisely, we want that on each fixed input the representation is correct with good probability.

This leads to the notion of a *probabilistic polynomial*.

Definition 4.8. *We denote by r a sequence of random bits $r_1, r_2, \ldots, r_L$. A **probabilistic polynomial** is a polynomial $p_r = p_r(X_1, \ldots, X_k)$ whose coefficients depend on r. A probabilistic polynomial yields a **distribution** over polynomials. To sample from this distribution, we sample the random bits $r_1, r_2, \ldots, r_L$ and then evaluate the coefficients of p_r.*

Example 4.9. *Consider the probabilistic polynomial*

$$p_r(X_1, X_2) = r_1 X_1 X_2 \oplus (1 - r_1) X_1 \oplus X_2 \oplus r_1.$$

Here r_1 denotes a random bit, i.e., $\Pr[r_1 = 1] = \Pr[r_1 = 0] = \frac{1}{2}$. This expression describes the following distribution over polynomials:

$$p_r(X_1, X_2) = \begin{cases} X_1 X_2 \oplus X_2 \oplus 1, & \text{with probability } \frac{1}{2} \\ X_1 \oplus X_2, & \text{with probability } \frac{1}{2} \end{cases}$$

Definition 4.10. *Let $\delta \geq 0$. For a function $f\colon \{0,1\}^k \to \{0,1\}$ and a probabilistic polynomial p_r over $\mathbb{F}_2$ we write*

$$f \approx_\delta p_r$$

if for any $(x_1, \ldots, x_k) \in \{0,1\}$ we have

$$\Pr_r\left[f(x_1, \ldots, x_k) \neq p_r(x_1, \ldots, x_k)\right] \leq \delta.$$

Here the notation "$\Pr_r$" highlights that the only randomness that the event depends upon is the sequence r of random bits.

Fortunately, we can indeed represent $\mathcal{OV}$ as a probabilistic polynomial with few monomials, as shown by our main result of this section:

Lemma 4.11. *Let $12 \leq s \leq 2^{d/6}$. There is a probabilistic polynomial $p_{\mathcal{OV}}$ with $p_{\mathcal{OV}} \approx_{1/3} \mathcal{OV}$, such that $p_{\mathcal{OV}}$ has $\mathcal{O}(\overline{m})$ monomials and we can sample a polynomial from the distribution $p_{\mathcal{OV}}$ in time $\mathcal{O}(\overline{m}^3)$, where*

$$\overline{m} := s^6 \binom{d}{3 \log s}^2.$$

In the remainder of this section, we will prove this lemma. Instead of representing an expression $\bigwedge_{\ell=1}^{n} x_\ell$ by the product $\prod_{\ell=1}^{n} x_\ell$ as in Lemma 4.7, we want to represent it by a polynomial of smaller degree. The following construction achieves this goal.

Lemma 4.12. *(Razborov[10]-Smolensky[11]) Define for any $t \in \mathbb{N}$,*

$$\mathrm{RS}_t(x_1, \ldots, x_n) := \prod_{k=1}^{t} \left(1 \oplus \bigoplus_{\ell=1}^{n} r_{k,\ell} \cdot (1 \oplus x_\ell) \right).$$

where the $r_{k,\ell}$ are (independent, uniform) random bits. Then we have

$$\bigwedge_{\ell=1}^{n} x_\ell \quad \approx_{2^{-t}} \quad \mathrm{RS}_t(x_1, \ldots, x_n).$$

Proof. Since $r_{k,\ell} \cdot (1 \oplus x_\ell)$ is equal to $r_{k,\ell}$ if $x_\ell = 0$, and equal to 0 otherwise, we can write

$$\mathrm{RS}_t(x_1, \ldots, x_n) = \prod_{k=1}^{t} \left(1 \oplus \bigoplus_{\ell \in Z} r_{k,\ell} \right),$$

where $Z := \{1 \leq \ell \leq n \mid x_\ell = 0\}$.

Case $x_1 = \ldots = x_n = 1$: In this case we have $Z = \emptyset$ and thus $\mathrm{RS}_t(x_1, \ldots, x_n) = \prod_{k=1}^{t} (1 \oplus 0) = 1 = \bigwedge_{\ell=1}^{n} x_\ell$. Since this is independent of the random bits $r_{k,\ell}$, it holds with probability 1.

Case $x_\ell = 0$ for some $1 \leq \ell \leq n$: Then $\bigwedge_{\ell=1}^{n} x_\ell = 0$ and thus

$$\begin{aligned} \Pr\left[\mathrm{RS}_t(x_1, \ldots, x_n) \neq \bigwedge_{\ell=1}^{n} x_\ell \right] &= \Pr\left[\forall k \in \{1, \ldots, t\} \colon \bigoplus_{\ell \in Z} r_{k,\ell} = 0 \right] \\ &= \Pr\left[r_1, \ldots, r_{|Z|} \text{ has even number of 1's} \right]^t \\ &= \frac{1}{2^t}. \end{aligned}$$

In both cases, we have $\Pr\left[\mathrm{RS}_t(x_1, \ldots, x_n) \neq \bigwedge_{\ell=1}^{n} x_\ell\right] \leq 2^{-t}$. □

Using Razborov-Smolensky, we construct a probabilistic polynomial for $\mathcal{OV}$ by following the same steps as before.

Lemma 4.13. *On input $A = \{a_1, \ldots, a_s\}, B = \{b_1, \ldots, b_s\} \subset \{0,1\}^d$, define the polynomials*

$$\begin{aligned} p_{i,j,\ell} &:= 1 \oplus a_i[\ell] \cdot b_j[\ell] \\ p_{i,j} &:= \mathrm{RS}_{3 \log s}(p_{i,j,1}, \ldots, p_{i,j,d}) \\ p_{\mathcal{OV}} &:= 1 \oplus \mathrm{RS}_2(1 \oplus p_{1,1}, 1 \oplus p_{1,2}, \ldots, 1 \oplus p_{s,s}) \end{aligned}$$

Then if $s \geq 12$ we have

$$\begin{aligned} p_{i,j,\ell} &= (\neg a_i[\ell]) \vee (\neg b_j[\ell]) \\ p_{i,j} &\approx_{s^{-3}} \mathrm{Orth}(a_i, b_j) \\ p_{\mathcal{OV}} &\approx_{1/3} \mathcal{OV}(A, B) \end{aligned}$$

[10] 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

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

In this notation we suppress the variables that these polynomials depend upon. To be more precise, we should write $p_{i,j,\ell}(a_i[\ell], b_j[\ell])$, $p_{i,j}(a_i, b_j)$, and $p_{\mathcal{OV}}(A, B)$.

Proof. For $p_{i,j,\ell}$ we simply used the rewriting rules of Lemma 4.7.

Correctness of $p_{i,j}$ follows directly from using Razborov-Smolensky on the $\wedge$-operation of $\mathrm{Orth}(a,b) = \bigwedge_{\ell=1}^{d}(\neg a[\ell]) \vee (\neg b[\ell])$.

By correctness of $p_{i,j}$ and the Union Bound, we have

$$\Pr\left[\exists i,j\colon p_{i,j} \neq \mathrm{Orth}(a_i,b_j)\right] \leq \sum_{1\leq i,j\leq s} \Pr\left[p_{i,j} \neq \mathrm{Orth}(a_i,b_j)\right] \leq \frac{s^2}{s^3} = \frac{1}{s}.$$

Moreover, using DeMorgan and Razborov-Smolensky on the $\vee$ operation of $\mathcal{OV}(A,B) = \bigvee_{1\leq i,j\leq s} \mathrm{Orth}(a_i,b_j)$ yields

$$\Pr\left[1 \oplus \mathrm{RS}_2\left(1 \oplus \mathrm{Orth}(a_1,b_1),\ldots,1 \oplus \mathrm{Orth}(a_s,b_s)\right) \neq \mathcal{OV}(A,B)\right] \leq \frac{1}{4}.$$

Combining these two bounds with the Union Bound yields

$$\Pr\left[p_{\mathcal{OV}} \neq \mathcal{OV}(A,B)\right] \leq \frac{1}{s} + \frac{1}{4} \leq \frac{1}{3},$$

for $s \geq 12$. □

To finally obtain a probabilistic polynomial for $\mathcal{OV}$ we simply expand our expression for $p_{\mathcal{OV}}$. It remains to show that this probabilistic polynomial $p_{\mathcal{OV}}$ satisfies the properties promised in Lemma 4.11, i.e., we need to show that $p_{\mathcal{OV}}$ has few monomials, and that we can efficiently sample from the distribution described by $p_{\mathcal{OV}}$.

Number of monomials

Let us bound the number of monomials of $p_{\mathcal{OV}}$. Observe that

$$p_{i,j} = \prod_{k=1}^{3\log s}\left(1 \oplus \bigoplus_{\ell=1}^{d} r_{k,\ell}\cdot a_i[\ell]\cdot b_j[\ell]\right).$$

Thus, any monomial of $p_{i,j}$ corresponds to a subset of $\{1,\ldots,d\}$ of size at most $3\log s$. We can therefore bound the number of monomials of $p_{i,j}$ by

$$\sum_{q=0}^{3\log s}\binom{d}{q} \leq 3\log s\cdot\binom{d}{3\log s} = \mathcal{O}\left(s\binom{d}{3\log s}\right),$$

where we used the assumption $s \leq 2^{d/6}$ to argue that $\binom{d}{3\log s}$ is the largest among the binomial coefficients $\binom{d}{k}$.

Moreover, observe that

$$p_{\mathcal{OV}} = 1 \oplus \prod_{k=1}^{2}\left(1 \oplus \bigoplus_{1\leq i,j\leq d} r_{k,i,j}\cdot p_{i,j}\right).$$

Since each $p_{i,j}$ has $\mathcal{O}\left(s\binom{d}{3\log s}\right)$ monomials, the sum $\bigoplus_{1\leq i,j\leq d} r_{k,i,j}\cdot p_{i,j}$ has $\mathcal{O}\left(s^3\binom{d}{3\log s}\right)$ monomials, and the product over $k \in \{1,2\}$ has $\mathcal{O}\left(s^6\binom{d}{3\log s}^2\right)$ monomials.

Sampling a polynomial

In order to sample from the distribution $p_{\mathcal{OV}}$, we first sample all involved random bits. We then compute each polynomial $p_{i,j}$ by expanding its definition (as given in Lemma 4.13). Finally, we compute $p_{\mathcal{OV}}$ by expanding its definition.

In this procedure, we have to perform $\mathcal{O}(s^2 \log s) = \mathcal{O}(\overline{m})$ polynomial multiplications, each on at most $\overline{m}$ monomials. Since multiplying two polynomials with $\overline{m}$ monomials can be done in time $\mathcal{O}(\overline{m}^2)$, we obtain total time $\mathcal{O}(\overline{m}^3)$. This finishes the proof of Lemma 4.11.

4.3 Final Algorithm

We combine our probabilistic polynomial for $\mathcal{OV}$ (Lemma 4.11) with the efficient evaluation of multivariate polynomials (Lemma 4.5) and standard boosting of the success probability to obtain our final algorithm.

Final Algorithm On input A, B we do the following.

0. Split A, B into groups $A_1, \ldots, A_g, B_1, \ldots, B_g$ of size s each.
1. For $t = 1, \ldots, 100 \ln n$ do:
   2. Sample polynomial p_t from the probabilistic polynomial for $\mathcal{OV}$
   3. Evaluate $p_t(A_i, B_j)$ for all pairs $1 \leq i, j \leq g$
4. Let $o_{i,j}$ be the majority vote among $p_1(A_i, B_j), \ldots, p_{100 \ln n}(A_i, B_j)$
5. Output 'yes' if $o_{i,j} = 1$ for some i, j, and 'no' otherwise

By "ln" we denote the natural logarithm, i.e., base e.

Claim 4.14 (Correctness)**.** *The algorithm outputs the correct answer with probability at least* $1 - \frac{1}{n}$.

Proof. Note that for any t the value $p_t(A_i, B_j)$ differs from $\mathcal{OV}(A_i, B_j)$ with probability at most $\frac{1}{3}$ (by Lemma 4.11). Also, the algorithm only errs if for some i, j the majority among $p_1(A_i, B_j), \ldots, p_{100 \ln n}(A_i, B_j)$ differs from $\mathcal{OV}(A_i, B_j)$.

We show that the number of samples $h := 100 \ln n$ is chosen sufficiently large for a high overall success probability. Note that by boosting (Lemma A.11), for fixed i, j the output $o_{i,j}$ is correct with probability at least $1 - \exp(-h/20) = 1 - \exp(-5 \ln n) = 1 - 1/n^5$. By a union bound over all $1 \leq i, j \leq g$, we obtain success probability at least $1 - g^2/n^5 \geq 1 - 1/n^3$. □

Claim 4.15 (Applicability of Multivariate Polynomial Evaluation)**.** *There is a constant* $\varepsilon > 0$ *(independent of n and d) such that* $s := n^{\varepsilon / \log c}$ *satisfies* $\overline{m} = s^6 \binom{d}{3 \log s}^2 \leq g^{0.1}$.

Before proving this claim, we use it to analyze the running time of our algorithm. In order to use our fast polynomial evaluation (Lemma 4.5) in step 3, we need to check that the number m of monomials and the number k of variables of each polynomial p_t satisfy $m, k = \mathcal{O}(g^{0.1})$. For m, this follows from the upper bound $\mathcal{O}(\overline{m})$ (by Lemma 4.11) together with the above claim. For k, we observe that $k = 2s^2 d$ is smaller than $\overline{m}$. Therefore, fast polynomial evaluation is applicable and the running time for step 3 is $\mathcal{O}(g^2 \log^2 g) = \mathcal{O}(g^2 \log^2 n)$, since $g = \frac{n}{s} \leq n$. Over $O(\log n)$ repetitions, this yields total time $\mathcal{O}(g^2 \log^3 n)$.

The running time of step 2 is (by Lemma 4.11) $\mathcal{O}(\overline{m}^3) = \mathcal{O}(g^{0.3})$. The remaining steps can be performed in time $\mathcal{O}(g^2 \log n)$. This is dominated by the time for step 3.

Hence, the total running time of our algorithm is $\mathcal{O}(g^2 \log^3 n) = \mathcal{O}(n^2 s^{-2} \log^3 n)$. Plugging in $s := n^{\varepsilon/\log c}$ yields time

$$\mathcal{O}\left(n^{2-\varepsilon/\log c} \log^3 n\right).$$

Note that $n^{2-\varepsilon/\log c} = n^{2-\Omega(1/\log c)} = n^{2-1/\mathcal{O}(\log c)}$.

Finally, we argue that for $c = n^{o(1/\log\log n)}$ the $\log^3 n$-factor can be ignored. Note that $s = n^{\varepsilon/\log c} = 2^{\varepsilon \log n/\log c} \geq 2^{\omega(\log\log n)} = (\log n)^{\omega(1)}$ using the presumed bound on c. Thus, $s \gg \log^3 n$, and we can bound $s^{-2} \log^3 n = \mathcal{O}(s^{-1})$. Using this bound yields the same asymptotic savings as before, but without the $\log^3 n$-factor.

It remains to prove Claim 4.15 to finish the analysis of the final algorithm.

Proof of Claim 4.15. We use the fact $\binom{n}{k} \leq (\frac{en}{k})^k$ to bound

$$\overline{m} = s^6 \binom{d}{3\log s}^2 \leq s^6 \left(\frac{ed}{3\log s}\right)^{6\log s} \leq s^6 \left(\frac{d}{\log s}\right)^{6\log s}. \tag{4.1}$$

Since $s = n^{\varepsilon/\log c}$ we have $\log s = \varepsilon \log n/\log c$. Using $d = c \log n$, we bound

$$\frac{d}{\log s} = \frac{c \log c \log n}{\varepsilon \log n} \leq \frac{c^2}{\varepsilon}.$$

Together with (4.1) this yields

$$\begin{aligned}
\log \overline{m} &\leq 6 \log s + 6 \log s \cdot (2 \log c + \log(1/\varepsilon)) \\
&= \frac{6\varepsilon \log n}{\log c} \cdot (1 + 2 \log c + \log(1/\varepsilon)) \\
&= \left(\frac{6\varepsilon}{\log c} + 12\varepsilon + \frac{6\varepsilon \log(1/\varepsilon)}{\log c}\right) \log n.
\end{aligned}$$

Using $c \geq 2$, and thus $\log c \geq 1$, we further bound

$$\log \overline{m} \leq (18\varepsilon + 6\varepsilon \log(1/\varepsilon)) \log n.$$

Observe that for $\varepsilon \to 0$ this factor tends to 0. Therefore, for sufficiently small $\varepsilon > 0$ we obtain

$$\log \overline{m} \leq 0.05 \log n,$$

which is equivalent to $\overline{m} \leq n^{0.05}$. Since $s = n^{\varepsilon / \log c} \leq n^{0.5}$ for sufficiently small ε, we can further bound $\overline{m} \leq (\frac{n}{s})^{0.1} = g^{0.1}$. □

Bibliography

[1] A. Abboud, R. Williams, and H. Yu. More applications of the polynomial method to algorithm design. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 218–230, 2015.

[2] M. Bläser. Fast matrix multiplication. *Theory of Computing, Graduate Surveys*, 5:1–60, 2013.

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

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal on Symbolic Computation*, 9(3):251–280, 1990.

[5] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Symposium on Symbolic and Algebraic Computation (ISSAC'14)*, pages 296–303, 2014.

[6] 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.

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

[8] A. J. Stothers. On the complexity of matrix multiplication. 2010. PhD Thesis.

[9] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969.

[10] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Annual ACM Symposium on Theory of Computing Conference (STOC'12)*, pages 887–898, 2012.