1 Last Time

In the previous lecture, we discussed how to solve the Orthogonal Vectors problem via a randomized reduction to a polynomial evaluation [AWY15]. The general strategy (of recasting the problem you'd like to solve into a problem about polynomials, then solving the polynomial problem using fast known algorithms) is often called the "polynomial method".[1]

Recall that $OV_{s,d}$ denotes the orthogonal vectors problem on s vectors from $\{0,1\}^d$. Last time, the main theorem we proved was:

Theorem 1.1 (OV Conversion Theorem) *For every s and d, there is a distribution $\mathcal{D}$ of polynomials over $\mathbb{F}_2$, where each polynomial has sd variables and at most $M(s,d) := poly(s) \cdot \binom{2d}{O(\log s)}$ monomials, such that for all inputs $v_1, \ldots, v_s \in \{0,1\}^d$ to the $OV_{s,d}$ problem,*

$$\Pr_{p \sim \mathcal{D}}[OV_{s,d}(v_1, \ldots, v_s) = p(v_1, \ldots, v_s) \bmod 2] \geq 3/4.$$

Moreover, we can construct a random p from the distribution $\mathcal{D}$ in $poly(M(s,d))$ time.

This distribution $\mathcal{D}$ is often called a *probabilistic polynomial*, and in the above theorem we are constructing a probabilistic polynomial for $OV_{s,d}$. We used this probabilistic polynomial, along with a self-reduction for OV, to design the following algorithm (with a parameter s).

(1) Divide the n vectors into at most $(n/s) + 1$ groups of at most s vectors each.

(2) Draw $t = 60\log(n)$ probabilistic polynomials $P_1, \ldots, P_t$ for $OV_{2s,d}$.
For all pairs of groups, evaluate $P_1, \ldots, P_t$ on the union of the two groups, and output the majority value of $P_1, \ldots, P_t$ for each pair.

(3) If any pair of groups has a majority value of 1, return "yes" else return "no".

If we can implement step (2) in $\tilde{O}(n^2/s^2)$ time, the entire algorithm can be run in $\tilde{O}(n^2/s^2)$ time. Two factors affect the running time:

1. The running time required to construct the probabilistic polynomials $P_1, \ldots, P_t$. Last time we showed that, for $d = c\log(n)$, if we set $s \leq n^{\delta/\log(c)}$ for a sufficiently small $\delta > 0$, then these probabilistic polynomials can be constructed in subquadratic time.

2. The running time required to evaluate one of the polynomials P_i on all $O(n^2/s^2)$ pairs of groups. Modulo a Batch Evaluation Lemma (stated below), we can do this evaluation in $\tilde{O}(n^2/s^2)$ time.

Lemma 1.1 (Batch Evaluation Lemma) *Given any prime power q, any $A, B \subseteq \{0,1\}^m$ with $|A| = |B| = N$, and given any $\mathbb{F}_q$-polynomial P with $2m$ variables and $N^{0.1}$ monomials, P can be evaluated on all N^2 points in $A \times B$ in $\tilde{O}(N^{1.1}m + N^2)$ time.*

[1] But note there is at least one other "polynomial method" in combinatorics which "solves" different problems.

Observe that the obvious algorithm for batch-evaluating a polynomial with $N^{0.1}$ monomials would take at least $N^2 \cdot N^{0.1} = N^{2.1}$ time: on each point $(x, y) \in (A \times B)$, it takes at least $N^{0.1}$ time to evaluate $P(x, y)$ (because that's how many monomials P has). The above lemma shows that for $m \leq N^{0.9}$, we can essentially evaluate P on all N^2 points in $A \times B$ in *nearly optimal time*!

In our particular case, we will set $N = n/s$, $m = sd$, $q = 2$. A and B will be a set of n/s vectors, where each vector of length sd encodes an entire group of s vectors. (Thus a pair $(x, y) \in A \times B$ exactly represents a pair of groups.) In particular, if we set $s = n^{\delta/\log(c)}$ for $d = c\log(n)$ then the number of monomials in each P_i will be at most $(n/s)^{0.1} = N^{0.1}$, the Batch Evaluation Lemma will apply, and we will be done, getting an $\tilde{O}(n^{2-2\delta/\log(c)})$ time algorithm.

Let's now turn to proving the Batch Evaluation Lemma. It's a very versatile trick; we show how to reduce the polynomial evaluation problem to a matrix multiplication problem where essentially optimal algorithms are known.

Proof. (Of the Batch Evaluation Lemma) Let P have variables $x_1, \ldots, x_m, y_1, \ldots, y_m$. Let $A = \{a_1, \ldots, a_N\}$ and $B = \{b_1, \ldots, b_N\}$. We will reduce the evaluation problem to the multiplication of an $N \times N^{0.1}$ matrix M_A with an $N^{0.1} \times N$ matrix M_B. In particular, the rows of M_A will be indexed by the a_i's, and the columns of M_B will be indexed by the b_i's.

Let the monomials of P be $m_1, \ldots, m_{N^{0.1}}$ where each m_k has the form

$$m_k = c_k \cdot X_k \cdot Y_k,$$

such that

- $c_k \in \mathbb{F}_q$ (the coefficient of the monomial),
- X_k is the product of all x_i's in m_k, and equals 1 if no x_i variables appear in m_k, and
- Y_k is the product of all y_i's in m_k, and equals 1 if no y_i variables appear in m_k.

Then for all $i, j = 1, \ldots, N$, and all $k = 1, \ldots, N^{0.1}$, we define

$$M_A[i, k] := c_k X_k(a_i), M_B[k, j] := Y_k(b_j)$$

(where $X_k(a_i)$ and $Y_k(b_j)$ denote the evaluation of X_k on the assignment a_i and the evaluation of Y_k on b_j, respectively). Note it takes $\tilde{O}(N^{1.1} \cdot m)$ time to prepare M_A and M_B: both matrices have $N^{1.1}$ entries, where each entry is a product of at most m variables. Observe that

$$(M_A \cdot M_B)[i, j] := \sum_k c_k \cdot X_k(a_i) \cdot Y_k(b_j) = \sum_k m_k(a_i, b_j) = P(a_i, b_j).$$

Therefore, computing $M_A \cdot M_B$ amounts to evaluating P on all inputs in $A \times B$. Finally, computing $M_A \cdot M_B$ can actually be done in $\tilde{O}(N^2)$ time(!). The algorithm is due to Coppersmith [Cop82]. □

1.1 Open Problem: Can We Improve This Algorithm? Can We Show a Barrier?

The algorithm for OV has the following very intriguing corollary:

Corollary 1.1 *Suppose there is a probabilistic polynomial for $OV_{s,d}$ that has only poly(s, d) monomials. Then $OV_{n,d}$ can be solved in $n^{2-\varepsilon} \cdot$ poly(d) time for some $\varepsilon > 0$, and SETH is false.*

Exercise: Convince yourself that the corollary holds!

Unfortunately, in unpublished work, we have shown that every probabilistic polynomial over $\mathbb{F}_p$ for $OV_{s,c\log n}$ **requires** at least $n^{\Omega(\log c)}$ monomials (for any prime p). So there is at least one sense in which we cannot improve the OV algorithm. (Let Ryan know if you'd like to read a copy of the paper!)

However, there is a potential loophole in our lower bound: our proof requires that 0 corresponds to "false" and 1 corresponds to "true" (or vice-versa). **We do not know such a lower bound for other representations**, such as polynomial representations where -1 represents true and 1 represents false (which actually comes up: this kind of representation is central to the area of Fourier analysis of Boolean functions).

1.2 More Problems Equivalent to OV

Recall the Orthogonal Vectors Conjecture (or Hypothesis) is:

> For every $\varepsilon > 0$ there is a $c \geq 1$ such that $OV_{n,c\log(n)}$ cannot be solved in $n^{2-\varepsilon}$ time.

Besides the Partial Match and Subset Query problems mentioned in the previous lecture, other interesting problems are known to be equivalent to OV. These are surprising equivalences: some are problems that look obviously harder than OV, and some involve approximations which seemed easier than OV. For instance, the following statements are all equivalent to the OV Conjecture [CW19]:

- **(Min-IP/Max-IP)** For every $\varepsilon > 0$ there is a $c \geq 1$ such that finding a red-blue pair of vectors with minimum (respectively, maximum) inner product, among n red vectors and n blue vectors in $\{0,1\}^{c\log(n)}$, cannot be solved in $n^{2-\varepsilon}$ time.
- **(Exact-IP)** For every $\varepsilon > 0$ there is a $c \geq 1$ such that finding a red-blue pair of vectors with inner product *equal* to a given target integer, among n red vectors and n blue vectors in $\{0,1\}^{c\log(n)}$, cannot be solved in $n^{2-\varepsilon}$ time.
- **(Apx-Min-IP/Apx-Max-IP)** For every fixed $\kappa > 1$ (think of κ as huge, like $\kappa = 1000$) and $\varepsilon > 0$, there is a $c \geq 1$ such that finding a red-blue pair of vectors that is a κ-approximation to the minimum (resp. maximum) inner product (among n red and blue vectors in $\{0,1\}^{c\log(n)}$) cannot be solved in $n^{2-\varepsilon}$ time.
- **(Approximate Bichromatic Closest Pair)** For every $\omega(\log n) \leq d(n) \leq n^{o(1)}$, and every $\varepsilon > 0$, there is a $\delta > 0$ such that a $(1+\delta)$-approximation to the closest red-blue pair (among n red and blue points in $\mathbb{R}^{d(n)}$) cannot be found in $n^{2-\varepsilon}$ time.
- **(Approximate Furthest Pair)** For every $\omega(\log n) \leq d(n) \leq n^{o(1)}$, and every $\varepsilon > 0$, there is a $\delta > 0$ such that a $(1+\delta)$-approximation to the furthest pair among n points in $\mathbb{R}^{d(n)}$ cannot be found in $n^{2-\varepsilon}$ time.

Note that all of the above problems have trivial $n^2 \cdot \text{poly}(d)$ time algorithms.

The above equivalence has many implications. For example, faster algorithms for finding constant-factor approximations to the maximum inner product imply faster algorithms for *exact* solutions to the maximum inner product, in the $O(\log n)$-dimensional setting. And both are equivalent to finding faster algorithms for OV! This underscores the fundamental nature of OV in solving problems on pairs of points.

2 Algorithm for All-Pairs Shortest Paths in Dense Graphs

We now turn to another application of the polynomial method: solving the All-Pairs Shortest Paths problem, from [Wil14]. Here, we will just give the main ideas, and won't go into all the details. First, let's recall the definition.

All-Pairs Shortest Paths (APSP): Given an n-node edge-weighted (directed) graph with positive real weights, compute for all pairs of nodes i, j the shortest distance between i and j.

(There are other ways of framing the APSP problem, but the above is the most common one.) On n-node graphs, the Floyd-Warshall algorithm (from undergrad algorithms) gives an $O(n^3)$ time algorithm assuming additions and comparisons of weights are constant-time operations.[2] The problem can also be solved in $\tilde{O}(mn + n^2)$ time. A major open problem is whether APSP has an $O(n^{2.99})$ time algorithm. Later in the semester, we will see a host of problems that are surprisingly *equivalent* to APSP.

2.1 Funny Matrix Multiplication

It has been known for decades that algorithms for APSP are innately related to computing a "funny" matrix multiplication. Formally, the operation is known as min-plus matrix multiplication, or *tropical* matrix multiplication.[3] Given two matrices $A, B \in \mathbb{R}^{n \times n}$, we define another $n \times n$ matrix

$$(A \star B)[i, j] := \min_k (A[i, k] + B[k, j]).$$

Note, we are substituting $\min$ in place of addition, and we are substituting $+$ in place of multiplication. (To say another buzzword, we are working over the "min-plus semiring" or the "tropical semiring".)

Theorem 2.1 *Let $T(n) \leq O(n^3)$. APSP on n-node graphs is in $O(T(n))$ time if and only if min-plus matrix multiplication on $n \times n$ matrices is in $O(T(n))$ time.*

One direction of the above theorem is pretty straightforward, so we'll pose it as an exercise:

Exercise: Convince yourself that, if APSP on n-node graphs is in $O(T(n))$ time, then min-plus matrix multiplication on $n \times n$ matrices is in $O(T(n))$ time. If you need a hint, there are some in the next lecture notes. (The other direction is harder, and will come up later.)

For the other direction, it is not too hard to show that APSP can be solved in $O(T(n) \log n)$ time, given that min-plus matrix multiplication in $O(T(n))$ time. The idea that, given the weighted adjacency matrix A of a graph, we can do a repeated squaring of A to determine the minimum distance between each node. More details in the next lecture!

2.2 Reduce "Funny" to "Normal" Matrix Multiplication

Now we turn to the problem of computing min-plus matrix multiplication faster. A **dream theorem** for us would be to show:

Theorem 2.2 (DREAM THEOREM) *If matrix multiplication over a field is in $O(n^c)$ time (on the Word RAM) then min-plus matrix multiplication is in $O(n^c)$ time (on the Real RAM).*

[2] More formally, we are studying algorithms for APSP in the *Real RAM* model. This model is an extension of the *Word RAM* model. In the Word RAM, information is stored in "word registers" that hold $\log(n)$ bits, and you can do all the normal word operations on them (XOR two words into a third word, AND two words into a third word, etc) in constant time. In the Real RAM, there are also "real-valued registers", each of which can hold an arbitrary real number. We are allowed to add two real-registers and put the result in another real register in $O(1)$ time, and one can compare if a real A is at most a real B, and put the (Boolean) outcome of that comparison into a word register, in $O(1)$ time. (The Real RAM is often also called the "addition-comparison model".) We will assume that the input to APSP is given to us in n^2 real-valued registers. All this is really not strictly necessary for understanding what follows, so I put it in a footnote. But if you really care about the details of the algorithm, it's important!

[3] There is an entire "tropical mathematics" where all your favorite objects over $\mathbb{R}$ are studied but addition is replaced by the minimum operation and multiplication is replaced by the sum operation.

This would be amazing, because we know (from decades of work, some of which has been done by one of your instructors) that matrix multiplication over fields can be computed in $O(n^c)$ time for $c < 2.373$. So the DREAM THEOREM above would immediately imply that APSP can be solved in $O(n^{2.373})$ time, as well.

However, the min-plus semiring (the mathematical structure that we're working over in min-plus matrix multiplication) is simply not as structured as a field, or even a ring, and the known fast matrix multiplication algorithms (for fields or for rings) exploit the extra structure: namely, all of them exploit the fact that every element r in a ring has an *additive inverse*, an element $-r$ such that $r+(-r)=0$. The min-plus semiring has no such property: there is no nice "inverse" of the minimum operation! For this reason, it seems strictly harder to multiply matrices over the min-plus semiring.

Nevertheless, we *can* prove the following theorem, which reduces min-plus matrix multiplication to *rectangular* matrix multiplication:

Theorem 2.3 (REAL THEOREM) *If matrix multiplication of $n \times n^{0.1}$ and $n^{0.1} \times n$ matrices over a field is in $\tilde{O}(n^2)$ time (on the Word RAM), then min-plus matrix multiplication is in $O(n^3/2^{\alpha\sqrt{\log n}})$ time, for some constant $\alpha > 0$.*

The nice part about the REAL THEOREM is that its hypothesis is actually true (as we saw and used earlier, in the OV algorithm). Unfortunately, the consequence is not quite as nice, because there is some considerable "blow-up" happening in the translation from min-plus matrix product to field matrix product. The big idea here is to translate the "min-plus inner product" operation (which is what you compute in each entry of a min-plus matrix multiplication) into "normal" inner product operations, and we can do that by **thinking of the min-plus inner product operation as a logical expression that we can randomly reduce to a polynomial over $\mathbb{F}_2$**, just as in the OV algorithm!

In particular, the REAL THEOREM follows from two other theorems:

Theorem 2.4 (From Rectangular Field Multiplication to Rectangular Min-Plus Multiplication) *If there is a $\delta > 0$ such that multiplication of an $n \times n^\delta$ matrix and an $n^{0.1} \times n$ matrix over $\mathbb{F}_2$ can be computed in $\tilde{O}(n^2)$ time, then there is an $\alpha > 0$ such that min-plus multiplication of an $n \times 2^{\alpha\sqrt{\log n}}$ matrix and $2^{\alpha\sqrt{\log n}} \times n$ matrix can be computed in randomized $\tilde{O}(n^2)$ time with high probability.*

Theorem 2.5 (From Rectangular Min-Plus Multiplication to Square Min-Plus Multiplication) *For every d, if min-plus multiplication of an $n \times d$ matrix and a $d \times n$ matrix can be computed in $T(n,d)$ time, then min-plus multiplication of two $n \times n$ matrices can be computed in $O(nT(n,d)/d)$ time.*

> **Exercise:** Verify that these two theorems, if true, really do imply that APSP can be solved in $O(n^3/2^{\alpha\sqrt{\log n}})$ time for some constant $\alpha > 0$.

In what remains, we sketch the proofs of these two theorems.

2.3 Proof of Theorem 2.5

We'll start with the proof of Theorem 2.5 because it's much easier to prove and is more well-known.

Given $n \times n$ matrices A and B, we split A into n/d separate $n \times d$ matrices $A_1, \ldots, A_{n/d}$, so that

$$A = [A_1 \cdots A_{n/d}].$$

Similarly, we split B into n/d matrices which are $d \times n$, so that

$$B = \begin{bmatrix} B_1 \\ \vdots \\ B_{n/d} \end{bmatrix}$$

Then for all $i, j = 1, \ldots, n$, observe that

$$(A \star B)[i,j] = \min_{\ell=1,\ldots,n/d} (A_\ell \star B_\ell)[i,j].$$

So, computing $A \star B$ can be done with n/d calls to min-plus multiplication of an $n \times d$ matrix A_ℓ and an $d \times n$ matrix B_ℓ, followed by $O(n^2 \cdot n/d)$ additions and comparisons.

2.4 Sketch of Theorem 2.4

Let us sketch the idea of Theorem 2.4. The theorem follows if we can construct a probabilistic polynomial for computing the min-plus inner product of two vectors of length d with high probability, where each polynomial in the resulting distribution has at most $d^{O(\log d)}$ monomials. In such a case, if we set $d = 2^{\alpha\sqrt{\log n}}$ for a sufficiently small $\alpha > 0$, we will obtain a probabilistic polynomial with at most $n^{0.1}$ monomials. Given such a polynomial P, we can compute the min-plus product of an $n \times 2^{\alpha\sqrt{\log n}}$ matrix A and $2^{\alpha\sqrt{\log n}} \times n$ matrix B by evaluating P on all possible pairs of rows from A and columns from B, in $\tilde{O}(n^2)$ time (using the Batch Evaluation Lemma).

How can we get a probabilistic polynomial (over $\mathbb{F}_2$) for min-plus inner product? Given two vectors $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ over $\mathbb{R}$, we want to compute

$$(x \star y) := \min_k (x_k + y_k).$$

We actually want to do this using matrix multiplication *over the field of two elements*, so we have to actually reduce the problem of computing $x \star y$ (which involves real numbers) to a problem on *bit operations*.

Our first step is to move to computing a slightly different operation:

$$(x \circ y) := \arg\min_k (x_k + y_k).$$

That is, $x \circ y$ computes an integer $k \in \{1, \ldots, d\}$ such that $x_k + y_k$ is minimized.[4] (Note that $(x \star y)$ can be easily determined from knowing $(x \circ y)$.) By definition, $(x \circ y) = k$ if and only if for every $\ell = 1, \ldots, d$, $x_k + y_k \leq x_\ell + y_\ell$. We will use the deep fact that this is true if and only if $\ell = 1, \ldots, d$, $x_k - x_\ell \leq y_\ell - y_k$. (This even has a name: it's called "Fredman's trick.")

To turn this into a Boolean-valued problem (which has only one bit of output), we think of k as a $\log(d)$-bit string $k_1 \cdots k_{\log(d)}$, pick an index $i = 1, \ldots, \log(d)$, and consider the logical expression:

$$MINSUM_i = \bigvee_{k' \in [d] : k'_i = k_i} \bigwedge_{\ell \in [d]} [x_{k'} - x_\ell \leq y_\ell - y_{k'}],$$

where $[C]$ is 1 if condition C is true and 0 if C is false. Then, the expression $MINSUM_i$ outputs the i-th bit of k such that $(x \circ y) = k$.

Finally, we can randomly reduce $MINSUM_i$ to vectors of length $d^{O(\log d)}$, so that computing an inner product over $\mathbb{F}_2$ of these two vectors yields the value of $MINSUM_i$ with high probability. The idea is that we can replace each $x_{k'} - x_\ell$ and $y_\ell - y_k$ for all $\ell, k = 1, \ldots, d$ with small integers, because we only need to know the relative order of these quantities to compute the $[x_{k'} - x_\ell \leq y_\ell - y_{k'}]$ predicate. By sorting all these pairwise differences using additions and comparisons, we can get rid of the real numbers altogether and replace them with small integers. (Over the entire matrices A and B, this can be done in $\tilde{O}(n \cdot d^2)$ time.) From there, $MINSUM_i$ is computing a large OR of at most d ANDs of d such comparisons. Using a similar randomized reduction as with OV (applying the XOR Tricks from before to the OR and to the ANDs), we can obtain a probabilistic polynomial of $d^{O(\log d)}$ monomials. All the lovely details are in [Wil14].

[4] It is certainly possible in principle that multiple values of k minimize $x_k + y_k$. We will assume that the entries of our matrices are perturbed so that this does not happen, and so $(x \circ y)$ is always well-defined.

3 Derandomization

Both of the OV and APSP algorithms can be made deterministic. I will add notes about how to do that if students are interested. The paper showing how to do it is [CW16].

References

[AWY15] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015.

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

[CW16] Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255. SIAM, 2016.

[CW19] Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40. SIAM, 2019.

[Wil14] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. Preliminary version in STOC 2014.