COMS 6998: Algebraic Techniques in TCS (Fall'21) Oct 12, 2021

Lecture 5: OV and Closest Pair problems, Prob Poly for MAJ

Instructor: *Josh Alman* Scribe notes by: *Dean Hirsch, Elena Gribelyuk*

Disclaimer: This draft may be incomplete or have errors. Consult the course webpage for the most up-to-date version.

1 Useful Bounds

Last lecture we used a Chernoff bound for the direction that's the reverse of the usual one used. We mention here the bounds we'll be using for reference.

Two-sided Chernoff bound. Suppose $x_1, ..., x_n$ are independent Bernoulli(p) random variables, meaning $x_i = 1$ with probability p, and $x_i = 0$ with probability $1 - p$. Then:

$$e^{-9\delta^2 pn} \leq \Pr\left[\left|\sum_{i=1}^{n} x_i - np\right| > \delta pn\right] \leq e^{-\delta^2 pn/3}.$$

A bound on binomial coefficients. For any $0 < k < n$ the following inequalities hold:

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k.$$

In particular, it follows that $\binom{n}{k} = \left(\Theta\left(\frac{n}{k}\right)\right)^k$.

2 Orthogonal Vectors Problem

Definition 1 (Orthogonal Vectors Problem)**.** *Given $x_1, ..., x_n, y_1, ..., y_n \in \{0,1\}^d$, determine whether there exist i, j such that $\langle x_i, y_j \rangle = 0$. Note that the dot product is taken over $\mathbb{Z}$.*

Note that asking whether $\langle x_i, y_j \rangle = 0$ is the same as asking if in every coordinate, at least one of x_i and y_j is zero.

We focus on the case where $d = c \log n$ for some constant c. Our goal is to show that we can solve the problem in time $n^{2-\frac{1}{O(\log c)}}$. Compare this with the naive algorithm, that iterates over all n^2 pairs of vectors and computes their dot product in time $O(d)$ each. This algorithm requires $O(n^2 d) = O(cn^2 \log n)$ time.

We start by trying the matrix multiplication technique. We form the matrix $X \in \{0,1\}^{n \times d}$ whose rows are the x_i's, and a corresponding matrix $Y \in \{0,1\}^{n \times d}$ whose rows are the y_j's. Compute the $n \times n$ matrix XY^T. This computes all inner products by the definition of matrix multiplication. If the dimensions of X and Y were $n \times n$, this would produce an $O(n^{2.373})$-time algorithm, but we're aiming for a even smaller time bound. To this end, we'll rely on the fact that the matrices are rectangular, and in this case there are better known algorithm for the matrix multiplication.

Theorem 2 (Fast rectangular matrix multiplication [Cop82])**.** *There is an algorithm to multiply an $n \times n^{0.17}$ matrix with an $n^{0.17} \times n$ matrix in $O(n^2 \log^2 n)$ operations.*

This is close to as quickly as we can hope, as the output is of size n^2, and therefore n^2 is an obvious lower bound for the time required. We note that there's also the following theorem, which proves somewhat better bounds. However, we will not need this result.

Theorem 3 (Improved fast rectangular matrix multiplication, [GU18])**.** *$n \times n^{0.3}$ by $n^{0.3} \times n$ matrix multiplication can be done in $n^{2+o(1)}$ operations.*

The general idea: The trick we'll need is to reduce the problem to matrix multiplication of smaller size. Specifically, let $s = n^a$ (where we'll later see that $a = 1/O(\log c)$ suffices), and group the inputs into sets $X_1, ..., X_{n/s}, Y_1, ..., Y_{n/s}$ of s points each. Our goal is to construct, for each X_i, a vector $v_i \in \mathbb{F}_2^{n^{0.16}}$, and a corresponding vector $u_j \in \mathbb{F}_2^{n^{0.16}}$ for each Y_j, such that with probability at least $\frac{1}{3}$, $\langle v_i, u_j \rangle = 1$ (over $\mathbb{F}_2$) if and only if there exists an orthogonal pair of vectors (over $\mathbb{Z}$) in the group $X_i \times Y_j$.

Suppose we managed to do that. Then we could form the matrix $V \in \mathbb{F}_2^{(n/s) \times n^{0.16}}$ whose rows are the v_i's, and the matrix $U \in \mathbb{F}_2^{(n/s) \times n^{0.16}}$ whose rows are the u_j's, and compute UV^T which computes all these dot products, in time $O\left(\frac{n^2}{s^2} \log^2 \frac{n}{s}\right) = O(n^{2-2a} \cdot \log^2 n) = O\left(n^{2-\frac{1}{O(\log c)}}\right)$, as promised.

2.1 Constructing the v_i, u_j

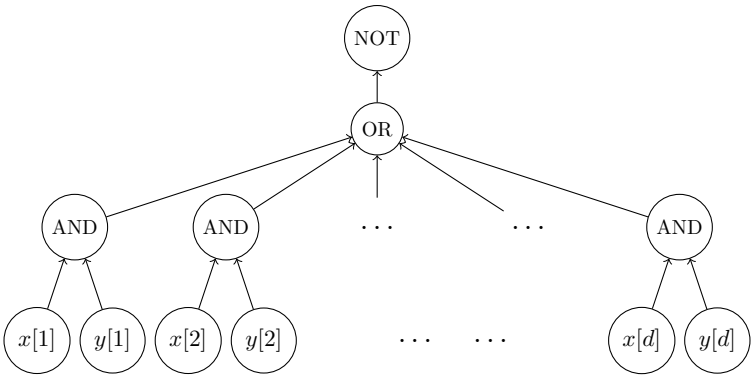
Our plan is to construct, for each x, a vector $x' \in \mathbb{F}_2^{n^{0.16}}$ (and similarly construct a y' for each y), such that $\langle x', y' \rangle_{\mathbb{F}_2} = 1$ if and only if $\langle x, y \rangle_{\mathbb{Z}} = 0$, with error probability at most $\frac{1}{3s^2}$. We can then let $v_i = \sum_{x \in X_i} x'$ and $u_j = \sum_{y \in Y_j} y'$, and get:

$$\langle v_i, u_j \rangle = \langle \sum_{x \in X_i} x', \sum_{y \in Y_j} y' \rangle = \sum_{x \in X_i} \sum_{y \in Y_j} \langle x', y' \rangle.$$

This last sum will count, with error at most $\frac{1}{3}$ (by the union bound on the s^2 events, each of which has its error bounded by $\frac{1}{3s^2}$), the *parity* of the number of different $(x, y) \in X_i \times Y_j$ with $\langle x, y \rangle_{\mathbb{Z}} = 0$. To adjust this to check whether or not we have *any* orthogonal pair, we can use the method of random subsampling we've seen in previous problems: we drop each vector with probability $\frac{1}{2}$, and in the case that there is any orthogonal pair, there must be at least some probability that the number of remaining orthogonal pairs is odd. We henceforth assume that if any orthogonal pair exist among $X_i \times Y_j$, then there are an odd number of orthogonal pairs.

2.2 Constructing correspondence $x \in \{0,1\}^d \to x' \in \mathbb{F}_2^{n^{0.016}}$

Define the following AC^0 circuit representing a function $F : \{0,1\}^{2d} \to \{0,1\}$ (using x, y to denote length-d vectors):



Claim 4. *$F(x, y)$ outputs 0 if and only if $x, y \in \{0,1\}^d$ are orthogonal over $\mathbb{Z}$.*

We can now use a result we've seen in Lecture 4 on the probabilistic degree of OR: Let p be a probabilistic polynomial for OR of error ε and degree $\lceil \log \frac{1}{\varepsilon} \rceil$. Then we can replace the OR in the above expression for F, to get the polynomial

$$q(x, y) = 1 - p(x[1]y[1], x[2]y[2], ..., x[d]y[d]).$$

Observation 5. *$q(x, y)$ defines a probabilistic polynomial for F, with error $\leq \varepsilon$ and degree* $\deg(q) = 2\deg(p) = 2\lceil \log \frac{1}{\varepsilon} \rceil$.

For our purposes we'll take $\varepsilon = \frac{1}{3s^2}$.

To form the corresponding x', y', we do the following: expand the polynomial $q(x, y)$ into monomials. Then group separately the parts corresponding to entries of x and those of y. For example, if the polynomial was $q(x, y) = x[1]y[1]x[3]y[3] + x[7]y[7] + ...$, we would take $x' = (x[1]x[3], x[7], ...)$ and $y' = (y[1]y[3], y[7], ...)$.

What is left is to show that q has at most $n^{0.16}$ monomials. The time for constructing x', y' can then be seen to also be bounded by $O(n^{0.16})$.

2.3 Bounding the length of x'

Observation 6. *q has degree $2\lceil \log \frac{1}{\varepsilon} \rceil = 2\lceil \log(3s^2) \rceil = 2\lceil \log(3n^{2a}) \rceil$. For large enough n this is $\leq 3\log(n^{2a}) = 6a \log n$. Further, q has $2d = 2c \log n$ inputs.*

Since we are free to assume that the polynomial is multilinear (as for every input variable $z \in \{0,1\}$, we have $z = z^2 = z^3 = ...$), we can bound the number of monomials by the number of subsets of size at most $\deg(q)$ of the $2d$ inputs:

$$\#\text{monomials} \leq \sum_{i=0}^{6a \log n} \binom{2c \log n}{i} \leq (6a \log n) \binom{2c \log n}{6a \log n},$$

where we used the fact that, assuming $a < \frac{c}{3}$, the largest of the binomial coefficients in the sum is the last one. Recall the general bound $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$ from earlier. Plugging this in, we get:

$$\#\text{monomials} \leq (6a \log n) \left(\frac{e \cdot 2c \log n}{6a \log n}\right)^{6a \log n} = (6a \log n) \left(\frac{ec}{3a}\right)^{6a \log n} = n^{O\left(a \log \frac{c}{a}\right)}.$$

If we pick $a = \frac{0.001}{\log c}$, then this will be at most $n^{0.16}$. This concludes the algorithm.

3 Closest Pair Problem

Next, we apply a similar technique to solve the closest pair problem, defined as follows:

Definition 7 (Closest Pair Problem)**.** *Given input vectors $x_1, ..., x_n, y_1, ..., y_n \in \{0,1\}^d$, and $t \in \mathbb{Z}$, determine whether or not there exist i, j such that $HammingDist(x_i, y_j) \leq t$.*

$HammingDist(x, y)$ equals the number of coordinates i where $x[i] \neq y[i]$.

Note that after we determine if such a pair exists in $x_1, ..., x_n, y_1, ..., y_n$, finding the precise pair $(x_i, y_j$ such that $HammingDist(x_i, y_j) \leq t$ is easy: we simply apply some version of binary search. Thus, it suffices for us to come up with an efficient problem for the decision problem.

In particular, we may be interested in this problem due to its applications to nearest-neighbor search. Our goal is to show that we can solve the problem in time $n^{2-\frac{1}{O(c \log^2 c)}}$. Also, note that the naive algorithm – which iterates over all n^2 pairs of vectors and computes their Hamming distance in time $O(d)$ each, takes a total of $O(n^2 d)$ time. Our algorithm will provide an improved runtime for the closest pair problem.

Observation 8. *We cannot hope for a much better running time than this, if we assume the Strong Exponential Time Hypothesis (SETH), from which it follows that for all $\varepsilon > 0$ there exsits some c, such that the closest pair problem with $d = c \log n$ cannot be solved in time $O\left(n^{2-\varepsilon}\right)$.*

3.1 Algorithm

Just as what we did in our solution of the orthogonal vectors problem, we will proceed by taking the following steps:

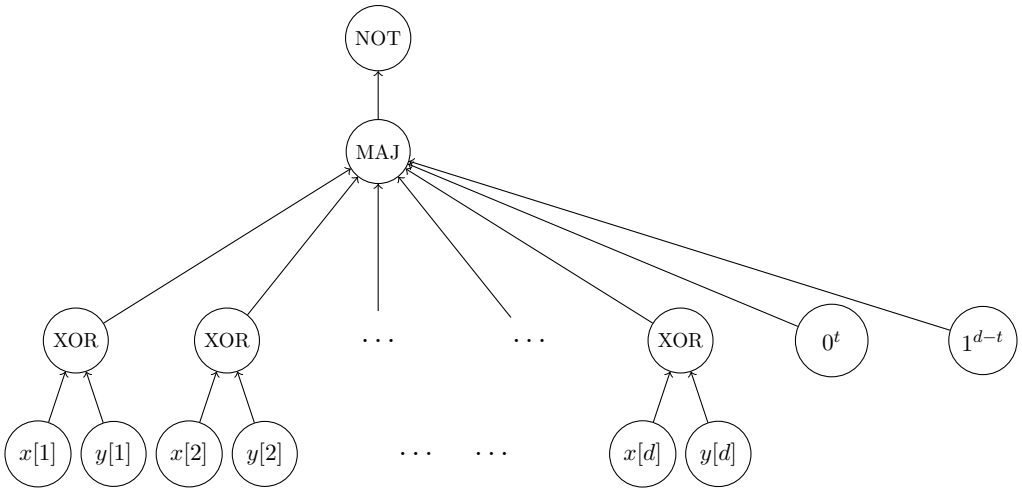
1. Design a circuit that tests whether the Hamming Distance between any two inputs x_i, y_j is at most t. Note that this circuit will require a MAJ gate, as we will show below.

2. Since we don't yet have a probabilistic polynomial for MAJ, we will need to define one as part of our construction.

3. Design a probabilistic polynomial for the overall circuit.

4. Proceed exactly as we did in solving the orthogonal vectors problem (divide up polynomial into vectors, group vectors, perform matrix multiplication to check if any entry is 1).

First, let us design an appropriate circuit for testing the Hamming Distance between any two vectors $x, y \in \{0,1\}^d$.



In this circuit 0^t denotes a vector of t zeroes, and 1^{d-t} denotes a vector of $d-t$ ones, so the MAJ gate takes $m = 2d$ inputs.

Note that this circuit is no longer AC^0. Last time, we only proved a lower bound for the degree lower bound for MAJ. Now, we will need to explicitly construct a probabilistic polynomial for MAJ. Once we have accomplished this step, our algorithm will be to simply replace the MAJ gate with the polynomial we construct, and proceed exactly as we did in solving the orthogonal vectors problem above.

3.2 Probabilistic polynomial for MAJ

At this point, we make the following generalized claim:

Lemma 9. *The threshold function $F : \{0,1\}^m \to \{0,1\}$ where*

$$F = \begin{cases} 1 & \text{if } \sum_i x_i \leq t, \\ 0 & \text{otherwise,} \end{cases}$$

has a probabilistic polynomial with error ϵ and degree $O(\sqrt{m \log \frac{1}{\epsilon}})$ over $\mathbb{R}$.

Note that the majority function simply uses F with $t = m/2$, and in the above circuit the majority function has $m = 2d$ inputs.

Next, we will state two crucial ingredients for our probabilistic polynomial. To give some high-level motivation for our next steps, it's important to note that our algorithm will be recursive in nature; in particular, we will take a polynomial for computing a threshold function on a smaller number of randomly-sampled inputs, and use it to construct a polynomial for computing a threshold function on a larger number of inputs. So, we will define a probabilistic polynomial such that if the input is very far from the threshold, the polynomial is very likely to output the correct answer, and if the input is close to the threshold, there might be a larger possibility of error.

Claim 10. *For $x \in \{0,1\}^m$, let $\tilde{x} \in \{0,1\}^{\frac{m}{k}}$ be a random sample of entries of x. Then, $Pr[|x| - k \cdot |\tilde{x}| \notin [-a, a]) \leq \frac{\epsilon}{3}$ if constants c and k are chosen to be large enough, $a = c\sqrt{m \log \frac{1}{\epsilon}}$, where we denote $|x| = \sum_i x_i$.*

This statement is proved using Chernoff bound. Observe that this statement suggests that after randomly sampling $\frac{m}{k}$ entries of any given x vector, the difference between the number of 1's in x and the number of 1's in $\tilde{x}$ must be between $-a$ and a with high probability. At this point, we will introduce the second ingredient in our construction: we will define a deterministic polynomial which will correctly compute on inputs x where $|x| \in [t-2a, t+2a]$.

Claim 11. *There exists a polynomial $A : \{0,1\}^m \to \mathbb{R}$ such that*

$$A(x) = \begin{cases} 1 & \text{if } |x| \in [t-2a, t], \\ 0 & \text{if } |x| \in (t, t+2a], \end{cases}$$

and the degree of A is $\leq 4a+1$.

Note that we do not require anything when x is not in $[t-2a, t+2a]$.

How will we construct such a polynomial A? We use polynomial interpolation: there is a polynomial $q : \mathbb{R} \to \mathbb{R}$ with $\deg(q) \leq 4a+1$ such that $q(z) = 1$ when $z \in \{t-2a, ..., t\}$ and $q(z) = 0$ when $z \in \{t+1, ..., t+2a\}$. Then, we let

$$A(x) = q\left(\sum_{i=1}^{n} x[i]\right), \quad \deg(A) \leq 4a+1.$$

Finally, we are ready to formally state our recursive construction for the probabilistic polynomial of the majority function, proceeding as follows:

1. Recursively draw polynomials from two probabilistic polynomials

   $$R \in \mathcal{P}_1, S \in \mathcal{P}_2,$$

   where $\mathcal{P}_1$ is the probabilistic polynomial which tests $|\tilde{x}| \leq \frac{t+a}{k}$, and $\mathcal{P}_2$ is the probabilistic polynomial which tests $|\tilde{x}| \leq \frac{t-a}{k}$. Both $\mathcal{P}_1$ and $\mathcal{P}_2$ take $\frac{m}{k}$ inputs, and both of them have error $\epsilon/3$.

2. Our probabilistic polynomial is defined as

   $$P(x) = A(x)R(\tilde{x})(1 - S(\tilde{x})) + S(\tilde{x}).$$

It suffices for us to show the correctness of our probabilistic polynomial, i.e., that P correctly computes the threshold function F with error ϵ, and bound the degree of P.

Proof of Lemma 9. **Correctness.** Assume R, S output the correct value for testing $|\tilde{x}| \leq \frac{t+a}{k}$ and $|\tilde{x}| \leq \frac{t-a}{k}$ respectively. This assumption holds with probability $\geq 1 - \frac{2\epsilon}{3}$. Consider four cases:

1. If $|\tilde{x}| > t+2a$, then by Claim 10 $|\tilde{x}| > \frac{t+a}{k}$ with probability $\geq 1 - \frac{\epsilon}{3}$, so we get that $R(\tilde{x}) = S(\tilde{x}) = 0$ and $P(x) = 0$.

2. If $|x| \in (t, t+2a]$, then $A(x) = 0$ and $|\tilde{x}| > \frac{t-a}{k}$, so $S(\tilde{x}) = 0$, and thus $P(x) = 0$.

3. If $|x| \in [t-2a, t]$, then $A(x) = 1$ and $|\tilde{x}| < \frac{t+a}{k}$, so $R(\tilde{x}) = 1$, and so $P(x) = 1$.

4. If $|x| < t-2a$, then $S(\tilde{x}) = 1$, so $P(x) = 1$.

Using Union bound, the total error is bounded by $\frac{2\epsilon}{3} + \frac{\epsilon}{3} = \epsilon$.

Degree. As for the degree, denote by $D(m, \varepsilon)$ the degree of the resulting polynomial. By the recursive definition, we have

$$D(m, \varepsilon) \leq 2D\left(\frac{m}{k}, \frac{\varepsilon}{3}\right) + 4a + 1$$

We will prove by induction that $D(m, \varepsilon) \leq b\sqrt{m \log \frac{1}{\varepsilon}}$ for a large enough constant b. The requirement by the induction step, using the recursive definition (recall $a = c\sqrt{m \log \frac{1}{\varepsilon}}$):

$$D(m, \varepsilon) \leq 2b\sqrt{\frac{m}{k} \log \frac{3}{\varepsilon}} + 4c\sqrt{m \log \frac{1}{\varepsilon}} \leq \left(\frac{2b}{\sqrt{k}} + 4c\right)\sqrt{m \log \frac{3}{\varepsilon}}.$$

So we require

$$b > \frac{2b}{\sqrt{k}} + 4c + \log 3,$$

which will be satisfied by a large enough b, as promised, as long as ε is bounded away from 1. It is noted that any $\varepsilon \geq \frac{1}{2}$ can be reached trivially by randomly choosing between the two constant polynomials $\{0, 1\}$, and we therefore do not lose anything by assuming $\varepsilon < \frac{1}{2}$. □

References

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

[GU18] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.