15-859M: Randomized Algorithms Anupam Gupta
Lec 10: Polynomial Identity Testing and Parallel Matchings February 9, 2011

1 Matrix multiplication checking

The *matrix multiplication checking* problem is to verify the process of matrix multiplication: Given three $n \times n$ matrices A, B, and C, is it the case that $AB = C$? The fastest known deterministic algorithm is to actually multiply A and B and compare the result to C—this takes $O(n^\omega)$ time, where ω is the exponent of matrix multiplication, and currently $\omega = 2.376$ due to an algorithm of Coppersmith and Winograd. Note that an easy lower bound on the running time of any randomized algorithm for matrix multiplication verification is $\Omega(n^2)$ since the input has to at least be read (see Lecture 4 for more details on this). We will now give a randomized algorithm (in co-RP) which takes only $O(n^2)$ time.

Let us introduce some notation for the rest of the course: $x \in_R X$ means "choose x uniformly at random from the set X". Our checking problem algorithm is as follows:

- Pick a vector $x \in_R \{0,1\}^n$.
- Compare ABx with Cx. This takes $O(n^2)$ time, since we can first compute $y = Bx$, and then compute $Ay = ABx$. Each matrix-vector product takes only $O(n^2)$ time.
- If $ABx = Cx$, then output *Yes*, otherwise output *No*.

(We're imagining working over the reals; if the matrices are over the field $\mathbb{F}$, the computations should also carried out over the field $\mathbb{F}$. The proofs remain unchanged.) Now if $AB = C$, our algorithm always outputs the correct answer. If $AB \neq C$, the algorithm may output the wrong answer. We now bound the probability of such an error.

First, we need a simple lemma:

Lemma 1 *Given n-digit strings $a, b \in \mathbb{R}^n$ and $x \in_R \{0,1\}^n$,* $\Pr[a \cdot x = b \cdot x] \leq \frac{1}{2}$.

Proof: Suppose $a_i \neq b_i$. Let $\alpha = \sum_{j \neq i} a_j x_j$ and $\beta = \sum_{j \neq i} b_j x_j$. We can write $a \cdot x = \alpha + a_i x_i$ and $b \cdot x = \beta + b_i x_i$. This gives us

$$a \cdot x - b \cdot x = (\alpha - \beta) + (a_i - b_i)x_i.$$

We can invoke the Principle of Deferred Decisions (see Section 3.5 of M&R) to assume that we've first picked all the values x_j for $j \neq i$. Then we can write

$$\Pr[a \cdot x - b \cdot x = 0] = \Pr\left[x_i = \frac{\alpha - \beta}{b_i - a_i}\right] \leq \frac{1}{2},$$

where we use the fact that $(\alpha - \beta)/(b_i - a_i)$ can only be either 0 or 1 (or neither), and a randomly chosen x_j will take that value with probability at most half. $\square$

Theorem 2 (Freivalds) *If $AB \neq C$, our algorithm fails with probability at most $\frac{1}{2}$.*

PROOF: If $AB \neq C$, then there is at least one row in AB, say $(AB)_i$, that differs from the corresponding row C_i in C. Apply Lemma 1 with $a = (AB)_i$ and $b = C_i$. The probability that $a \cdot x = b \cdot x$ is at most $1/2$. For the algorithm to output *Yes*, we must have $a \cdot x = b \cdot x$. Therefore, the probability of failure for the algorithm is at most $1/2$. $\square$

2 Polynomial identity checking

In the *polynomial identity checking* problem, we are given two multi-variate polynomials $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ each with degree d; again we are computing in some field $\mathbb{F}$. We may not be given the polynomials explicity, so we may not be able to read the polynomials in poly-time — we just have "black-box" access for evaluating a polynomial. Given these two polynomials, the problems is to determine if the polynomials are equal: i.e., if $f = g$, or equivalently, $f - g = 0$. Letting $Q = f - g$, it suffices to check if a given polynomial is identically zero. There is no known poly-time algorithm for this problem. But we will now show that it is in co-RP.

First consider the univariate case. We can pick $d + 1$ distinct values at random from $\mathbb{F}$. If $Q(x) = 0$ for all $d + 1$ values for x, then $Q = 0$. This follows from the basic and super-useful fact, that for any field $\mathbb{F}$, a polynomial of degree at most d over that field can have at most d roots.

This approach does not directly apply to the multivariate case; in fact, the polynomial over two variables $f(x, y) = xy - 3$ over the reals has an infinite number of roots. Over the finite field $\mathbb{F}_q$, the degree-d polynomial over n variables

$$Q(x_1, x_2, \ldots, x_n) = (x_1 - 1)(x_1 - 2) \cdots (x_1 - d)$$

has dq^{n-1} roots (when $d \leq q = |\mathbb{F}|$).

However, things still work out. Roughly speaking, we can handle the multivariate case by fixing $n-1$ variables and applying the result from the univariate case. Consider the following algorithm, which assumes we have some subset $S \subset \mathbb{F}$ with $|S| \geq 2d$.

- Pick $r_1, \ldots, r_n \in_R S$.
- Evaluate $Q(r_1, \ldots, r_n)$.
- If 0, return $Q = 0$.

Theorem 3 (Schwartz (1980), Zippel (1979)) *If, in the above algorithm, the polynomial $Q \neq 0$, we have*

$$\Pr[Q(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Proof: By induction on n. The base case is the univariate case described above. With $Q \neq 0$, we want to compute $\Pr[Q(r_1, \ldots, r_n) = 0]$. Let k be the largest power of x_1. We can rewrite

$$Q(x_1, \ldots, x_n) = x_1^k A(x_2, \ldots, x_n) + B(x_1, \ldots, x_n)$$

for some polynomials A and B. Now we consider two events. Let $\mathcal{E}_1$ be the event that $Q(r_1, \cdots, r_n)$ evaluates to 0, and $\mathcal{E}_2$ be the event that $A(r_2, \cdots, r_n)$ evaluates to 0.

We can rewrite the probability that $Q(r_2, \cdots, r_n)$ is 0 as:

$$\begin{aligned}
\Pr[Q(r) = 0] = \Pr[\mathcal{E}_1] &= \Pr[\mathcal{E}_1 \mid \mathcal{E}_2] \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2] \Pr[\neg\mathcal{E}_2] \\
&\leq \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2]
\end{aligned}$$

Let us first bound the probability of $\mathcal{E}_2$, or the probability that $A(r_2, \cdots, r_n) = 0$. The polynomial A has degree $d - k$ and fewer varaibles, so we can use the inductive hypothesis to obtain

$$\Pr[\mathcal{E}_2] = \Pr[A(r_2, \ldots, r_n) = 0] \leq \frac{d-k}{|S|}.$$

Similarly, given $\neg\mathcal{E}_2$ (or $A(r_2, \cdots, r_n) \neq 0$), the univariate polynomial $Q(x_1, r_2, \ldots, r_n)$ has degree k. Therefore, again by inductive hypothesis,

$$\Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2] = \Pr[Q(x_1, r_2, \ldots, r_n) = 0 \mid A(r_2, \ldots, r_n) \neq 0] \leq \frac{k}{|S|}.$$

We can substitute into the expression above to get

$$\begin{aligned}
\Pr[Q(r) = 0] &\leq \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2] \\
&\leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}
\end{aligned}$$

This completes the inductive step. $\square$

Polynomial identity testing is a powerful tool, both in algorithms and in complexity theory. We will use it to find matchings in parallel, but it arises all over the place. Also, as mentioned above, there is no poly-time deterministic algorithm currently known for this problem. A result of Impagliazzo and Kabanets (2003) shows that proving that the polynomial identity checking problem is in P would imply that either $NEXP$ cannot have poly-size non-uniform circuits, or *Permanent* cannot have poly-size non-uniform circuits. Since we are far from proving such strong lower bounds, the Impagliazzo-Kabanets result suggest that deterministic algorithms for polynomial identity checking may require us to develop significantly new techniques.

For more on the polynomial identity checking problem, see Section 7.2 in M&R. Dick Lipton's blog has an interesting post on the history of the theorem, as well as comparisons of the results of Schwartz, Zippel, and DeMillo-Lipton. One of the comments points out that the case when $S = \mathbb{F}_q$ was known at least as far back as Ore in 1922; his proof appears as Theorem 6.13 in the book *Finite Fields* by Lidl and Niederreiter; a different proof by Dana Moshkowitz appears here.

3 Perfect matchings in bipartite graphs

We will look at a simple sequential algorithm to determine whether a perfect matching exists in a given bipartite graph or not. The algorithm is based on polynomial identity testing from the previous section.

A bipartite graph $G = (U, V, E)$ is specified by two disjoint sets U and V of vertices, and a set E of edges between them. A *perfect matching* is a subset of the edge set E such that every vertex has exactly one edge incident on it. Since we are interested in perfect matchings in the graph G, we shall assume that $|U| = |V| = n$. Let $U = \{u_1, u_2, \cdots, u_n\}$ and $V = \{v_1, v_2, \cdots, v_n\}$. The algorithm we study today has no error if G does not have a perfect matching (no instance), and errs with probability at most $\frac{1}{2}$ if G does have a perfect matching (yes instance). This is unlike the algorithms we saw in the previous lecture, which erred on no instances.

Definition 4 *The Tutte matrix of bipartite graph $G = (U, V, E)$ is an $n \times n$ matrix M with the entry at row i and column j,*

$$M_{i,j} = \begin{cases} 0 & if (u_i, v_j) \notin E \\ x_{i,j} & if (u_i, v_j) \in E \end{cases}$$

(Apparently, Tutte came up a matrix for general graphs, and this one for bipartite graphs is due to Jack Edmonds, but we'll stick with calling it the Tutte matrix.)

The determinant of the Tutte matrix is useful in testing whether a graph has a perfect matching or not, as the following lemma shows. Note that we do not think of this determinant as taking on some numeric value, but purely as a function of the variables $x_{i,j}$.

Lemma 5 $\det(M) \neq 0 \iff$ *There exists a perfect matching in G*

Proof: We have the following expression for the determinant :

$$\det(M) = \sum_{\pi \in S_n} (-1)^{sgn(\pi)} \prod_{i=1}^{n} M_{i,\pi(i)}$$

where S_n is the set of all permutations on $[n]$, and $sgn(\pi)$ is the sign of the permutation π. There is a one to one correspondence between a permutation $\pi \in S_n$ and a (possible) perfect matching $\{(u_1, v_{\pi(1)}), (u_2, v_{\pi(2)}), \cdots, (u_n, v_{\pi(n)})\}$ in G. Note that if this perfect matching does not exist in G (*i.e.* some edge $(u_i, v_{\pi(i)}) \notin E$) then the term corresponding to π in the summation is 0. So we have

$$\det(M) = \sum_{\pi \in P} (-1)^{sgn(\pi)} \prod_{i=1}^{n} x_{i,\pi(i)}$$

where P is the set of perfect matchings in G. This is clearly zero if $P = \emptyset$, *i.e.*, if G has no perfect matching. If G has a perfect matching, there is a $\pi \in P$ and the term corresponding to π is $\prod_{i=1}^{n} x_{i,\pi(i)} \neq 0$. Additionally, there is no other term in the summation that contains the same set of variables. Therefore, this term is not cancelled by any other term. So in this case, $\det(M) \neq 0$. $\square$

This lemma gives us an easy way to test a bipartite graph for a perfect matching — we use the polynomial identity testing algorithm of the previous lecture on the Tutte matrix of G. We accept if the determinant is not identically 0, and reject otherwise. Note that $\det(M)$ has degree at most n. So we can test its identity on the field Z_p, where p is a prime number larger than $2n$. From the analysis of the polynomial testing algorithm, we have the following :

- G has no perfect matching $\implies \Pr[accept] = 0$.
- G has a perfect matching $\implies \Pr[accept] \geq \frac{1}{2}$.

The above algorithm shows that Perfect Matching for bipartite graphs is in RP. (The non-bipartite case may appear as a homework exercise.) Also, this algorithm for checking the existence of a perfect matching can be easily converted to one that actually computes a perfect matching as follows:

1. Pick $(u_i, v_j) \in E$.
2. Check if $G \backslash \{u_i, v_j\}$ has a perfect matching.
3. If "Yes", output (u_i, v_j) to be in the matching and recurse on $G \backslash \{u_i, v_j\}$, the graph obtained after the removal of vertices u_i and v_j.
4. If "No", recurse on $G - (u_i, v_j)$, the graph obtained after removing the edge (u_i, v_j).

Note that this algorithm seems inherently sequential; it's not clear how to speed up its running time considerably by using multiple processors. We'll consider the parallel algorithm in the next section.

Some citations: the idea of using polynomial identity testing to test for matchings is due to Lovász. The above algorithm to find the matching runs in time mn^ω, where n^ω is the time to multiply two $n \times n$-matrices. (It is also the time to compute determinants, and matrix inverses.) Rabin and Vazirani showed how to compute perfect matchings in general graphs in time $O(n^{\omega+1})$, where n. Recent work of Mucha and Sankowski, and Harvey show how to use these ideas (along with many other cool ones) to find perfect matchings in general graphs in time n^ω.

4 A parallel algorithm for finding perfect matchings

However, we can give a slightly different algorithm (for a seemingly harder problem), that indeed runs efficiently on parallel processors. The model here is that there are polynomially many processors run in parallel, and we want to solve the problem in poly-logarithmic depth using polynomial work. We will use the fact that there exist efficient parallel algorithms for computing the determinant of a matrix to obtain our parallel algorithm for finding perfect matchings.

We could try the following "strawman" parallel algorithm:

> Use a processor for every edge (u_i, v_j) that tests (in parallel) if edge (u_i, v_j) is in some perfect matching or not. For each edge (u_i, v_j) that lies in some perfect matching, the processor outputs the edge, else it outputs nothing.

We are immediately faced with the problem that there may be several perfect matchings in the graph, and the resulting output is not a matching. The algorithm may in fact return all the edges in the graph. It will only work if there is a *unique* perfect matching.

So instead of testing whether an edge (u_i, v_j) is in some perfect matching or not, we want to test whether an edge (u_i, v_j) is in a specific perfect matching or not. The way we do this is to put random weights on the edges of the graph and test for the minimum weight perfect matching. Surprisingly, one can prove that the minimum weight perfect matching is unique with a good probability, even when the weights are chosen from a set of integers from a relatively small range.

Lemma 6 *Let $S = \{e_1, \cdots, e_m\}$ and $S_1, \cdots S_k \subseteq S$. For every element e_i there is a weight w_i picked u.a.r. from $\{0, 1, \cdots, 2m-1\}$. The weight of subset S_j is $w(S_j) = \sum_{e_i \in S_j} w_i$. Then*

$$\Pr[\text{ minimum weight set among } S_1, \cdots, S_k \text{ is unique }] \geq \frac{1}{2}.$$

Proof: We will estimate the probability that the minimum weight set is *not* unique. Let us define an element e_i to be *tied* if

$$\min_{S_j | e_i \in S_j} w(S_j) = \min_{S_j | e_i \notin S_j} w(S_j)$$

It is easy to see that there exists a tied element if and only if the minimum weight subset is not unique. Below we bound the probability that a fixed element e_i is tied. The result will then follow using a union bound.

We use the principle of deferred decisions. Let us fix the weights $w_1, \cdots, w_m$ of all the elements except w_i. We want to bound $\Pr_{w_i}[e_i \text{ is tied } \mid w_1, \cdots, w_{i-1}, w_{i+1}, w_m]$. Let

$$W^- = \min_{S_j | e_i \notin S_j} w(S_j) \qquad \text{and} \qquad W^+ = \min_{S_j | e_i \in S_j} w(S_j)$$

with w_i assigned the value 0. It is easy to see that e_i is tied iff $W^- = W^+ + w_i$. So,

$$\begin{aligned} & Pr_{w_i}[e_i \text{ is tied } \mid w_1, \cdots, w_{i-1}, w_{i+1}, w_m] \\ = \; & Pr_{w_i}[w_i = W^- - W^+ \mid w_1, \cdots, w_{i-1}, w_{i+1}, w_m] \\ \leq \; & \frac{1}{2m}. \end{aligned}$$

The last inequality is because there is at most on value for w_i for which $W^- = W^+ + w_i$. This holds irrespective of the particular values of the other $w_{i'}$s. So $\Pr[e_i \text{ is tied }] \leq \frac{1}{2m}$, and

$$\Pr[\exists \text{ a tied element }] \leq \sum_{i=1}^{m} \Pr[e_i \text{ is tied}] \leq \frac{1}{2}.$$

Thus $\Pr[\text{ minimum weight set is unique }] \geq \frac{1}{2}$. $\Box$

Now we can look at the parallel algorithm for finding a perfect matching. For each edge (u_i, v_j), we pick a random weight $w_{i,j}$, from $[2m-1]$, where $m = |E|$ is the number of edges in G. Let the sets S_j denote all the perfect matchings in G. Then the Isolation Lemma implies that there is a unique minimum weight perfect matching with at least a half probability. We assign the value $x_{i,j} = 2^{w_{i,j}}$ to the variables in the Tutte matrix M. Let D denote the resulting matrix. We use the determinant of D to determine the weight of the min-weight perfect matching, if it is unique, as suggested by the following lemma.

Lemma 7 *Let W_0 be the weight of the minimum weight perfect matching in G. Then,*

- *G has no perfect matching* $\implies \det(D) = 0$.
- *G has a unique min-weight perfect matching* $\implies \det(D) \neq 0$ *and the largest power of 2 dividing* $\det(D)$ *is W_0.*
- *G has more than one min-weight perfect matching* $\implies$ *either* $\det(D) = 0$ *or the largest power of 2 dividing* $\det(D)$ *is at least W_0.*

Proof: If G has no perfect matching, it is clear from lemma 5 that $\det(D) = 0$.

Now consider that case when G has a unique min-weight perfect matching. From the expression of the determinant, we have

$$\det(D) = \sum_{\pi \in P} (-1)^{sgn(\pi)} \prod_{i=1}^{n} 2^{w_{i,\pi(i)}} = \sum_{\pi \in P} (-1)^{sgn(\pi)} 2^{\sum_{i=1}^{n} w_{i,\pi(i)}} = \sum_{\pi \in P} (-1)^{sgn(\pi)} 2^{w(\pi)}$$

where $w(\pi)$ is the weight of the perfect matching corresponding to π and P is the set of all perfect matchings in G. Since there is exactly one perfect matching of weight W_0 and other perfect matchings have weight at least $W_0 + 1$, this evaluates to an expression of the form $\pm 2^{W_0} \pm 2^{W_0+1} \cdots \pm$ other powers of 2 larger than W_0. Clearly, this is non-zero, and the largest power of 2 dividing this is W_0.

Now consider the case when G has more than one min-weight perfect matchings. In this case, if the determinant is non-zero, every term in the sumation is a power of 2, at least 2^{W_0}. So 2^{W_0} divides $\det(D)$. $\square$

We refer to the submatrix of D obtained by removing the i-th row and j-th column by $D_{i,j}$. Note that this is a matrix corresponding to the bipartite graph $G \backslash \{u_i, v_j\}$. The parallel algorithm would run as follows.

1. Pick random weights $w_{i,j}$ for the edges of G. (In the following steps, we assume that we've isolated the min-weight perfect matching.)
2. Compute the weight W_0 of the min-weight perfect matching from $\det(D)$ (using the parallel algorithm for computing the determinant): this is just the highest power of 2 that divides $\det(D)$.
3. If $\det(D) = 0$, output "no perfect matching".
4. For each edge $(u_i, v_j) \in E$ do, in parallel,:
   (a) Evaluate $\det(D_{i,j})$.
   (b) If $\det(D_{i,j}) = 0$, output nothing.
   (c) Else, find the largest power of 2, $W_{i,j}$, dividing $\det(D_{i,j})$.
   (d) If $W_{i,j} + w_{i,j} = W_0$, output (u_i, v_j).
   (e) Else, output nothing.

It is clear that, if G has no perfect matching, this algorithm returns the correct answer. Now suppose G has a unique minimum weight perfect matching, we claim Lemma 7 ensures that precisely all the edges in the unique min-weight perfect matching are output. To see this, consider an edge (u_i, v_j) not in the unique min weight perfect matching. From the lemma, $\det(D_{i,j})$ is either zero (so the edge will not be output), or $W_{i,j}$ is at least as large as the min-weight perfect matching in $G \backslash \{u_i, v_j\}$. Since the min-weight perfect matching is unique and does not contain edge (u_i, v_j), this implies $w_{i,j} + W_{i,j}$ will be strictly larger than W_0, and this edge will not be output in this case either. Finally, if an edge (u_i, v_j) is in the unique min-weight perfect matching, removing this edge from the matching gives us the unique min-weight perfect matching in $G \backslash \{u_i, v_j\}$. So, in this case $W_{i,j} = W_0 - w_{i,j}$ and the edge is output.

Thus, if G has a perfect matching, this algorithm will isolate one with probability at least $\frac{1}{2}$, and will output it—hence we get an RNC algorithm that succeeds with probability at least 1/2 on “Yes” instances, and never makes mistakes on “No” instances.

Finally, some more citations. This algorithm is due to Mulmuley, Vazirani and Vazirani; the first RNC algorithm for matchings had been given earlier by Karp, Upfal, and Wigderson. It is an open question whether we can find perfect matchings *deterministically* in parallel using poly-logarithmic depth and polynomial work, even for bipartite graphs.