E0309 Topics in Complexity Theory **Spring 2015**

Lecture 7-8:GCD and Bipartite Matching

Lecturer: Neeraj Kayal *Scribe: Abhijat Sharma*

7.1 Greatest Common Divisor

In the last lecture, we proved that computation of the determinant of a $n \times n$ matrix was possible using boolean circuits of size polynomial in n and depth polynomial in $\log n$ (specifically $O(\log^2 n)$). In other words, the determinant is in NC, and so are most of the problems in linear algebra, such as computing the inverse of a matrix, solving a system of linear equations etc. as they can all be reduced to the problem of computing the determinant of a $n \times n$ matrix.

Now, we would like to extend the notion of efficient parallel computation (the class NC) to problems beyond linear algebra. Therefore, let us consider a problem which has been studied by mathematicians for hundreds of years, and is one of the most basic problems related to all number theoretic algorithms, the problem of computing the *greatest common divisor* (GCD) of two integers.
Input: two integers $a, b \in \mathbb{Z}$ $(1 \leq a, b \leq 2^n)$
Output: an integer $g = gcd(a, b)$, such that $g \mid a$ (g divides a), $g \mid b$ and for any other common divisor h of a and b, $h \leq g$.

7.1.1 Euclid's Algorithm (300 BC)

The first algorithm for computing the GCD of two integers was given by the greek mathematician Euclid, as early as 300 BC. It is based on a very simple observation, which is as follows: if r is the remainder you get when you divide a by b,

$$gcd(a, b) = gcd(b, r) \tag{7.1}$$

It is easy to observe from the above equation, that if the remainder $r = 0$, i.e $b \mid a$, then $gcd(a, b) = b$. The proof of equation 7.1 can be shown by similar arguments in both directions:

1. we know that $r \equiv a(\mod b)$ so we can write

   $$a = t.b + r$$

   for some integer t. For any common divisor h of a and b,

   $$h \mid a \text{ and } h \mid b \implies h \mid r$$
   $$\Rightarrow gcd(a, b) \mid r$$

   every common divisor of b and r divides $gcd(b, r)$,

   $$\Rightarrow \gcd(a, b) \mid gcd(b, r)$$

2. looking at the same equation as above: $a = t.b + r$, for any common divisor h of b and r,

$$h \mid b \text{ and } h \mid r \Longrightarrow h \mid a$$
$$\Rightarrow gcd(b, r) \mid a$$

every common divisor of a and b divides $gcd(a, b)$,

$$\Rightarrow \gcd(b, r) \mid gcd(a, b)$$

Combining the above two steps, we prove the basis for Euclid's algorithm, $gcd(a, b) = gcd(b, r)$. When we try to develop an efficient parallel algorithm to implement this algorithm, we observe that it requires at-most n levels of recursion, and therefore the depth of any circuit simulating the Euclid's algorithm would be $O(n)$. If n is the length of the input, we would like to obtain a circuit with depth polynomial in $\log n$, to claim that Integer-GCD is in NC, i.e solvable efiiciently in parallel.

7.1.2 Polynomial GCD

As we have seen in earlier lectures, while trying to analyze the complexity of integer multiplication, it is helpful to look at polynomial multiplication and use the analogy between n-bit integers and n-degree univariate polynomials. Similarly, to analyze Integer-GCD, we look at the problem of computing GCD of two univariate polynomials:
Input: two degree-d polynomials $a(x), b(x) \in \mathbb{Q}[X]$ (coefficients are rational numbers),
Output: the polynomial $g(x) = gcd(a(x), b(x))$, defined similarly as in integers. (gcd of polynomials is well-defined due to unique factorization of polynomials)

Fact: For any two integers a and b, $g = gcd(a, b)$ is the smallest positive integer that can be written as $g = a.x + b.y$ for some $x, y \in \mathbb{Z}$. Similarly, for any two polynomials $a(x)$ and $b(x)$, $g(x) = gcd(a(x), b(x)$ is the smallest degree polynomial, such that $g = a.p + b.q$ for some $p, q \in \mathbb{Q}[X]$.

Theorem 7.1 *Polynomial-GCD can be computed by a family of circuits having size poly(d) and depth poly$(\log d)$, assuming field operations have unit-cost.*

Proof: Let e be degree of the polynomial $g(x)$, $0 \leq e \leq d$. The family of circuits claimed to exist by theorem 7.1 can be described as follows:

- The input is given as $(d + 1)$ coefficients, each of $a(x)$ and $b(x)$. For a given input, there are $(d + 1)$ possible values of e, as mentioned above. So, the circuit has $(d+1)$ branches, one branch for each value of e, and all the branches are computed in parallel.
- Every branch assumes a fixed value of e. Then, it uses the property that $g(x) = a(x).p(x) + b(x).q(x)$ $(degree(p), degree(q) < d)$, substituting $a(x)$ and $b(x)$ that are known to the circuit, and the coefficients of $g(x), p(x), q(x)$ as variables:

$$g(x) = g_0 + g_1.x + ... + g_e.x^e$$
$$p(x) = p_0 + p_1.x + ... + p_{d-1}.x^{d-1}$$
$$q(x) = q_0 + q_1.x + ... + q_{d-1}.x^{d-1}$$

- Substituting the above polynomials transforms $g(x) = a(x).p(x) + b(x).q(x)$ to a system of linear equations in $O(d)$ variables, that are the coefficients of the polynomials $g(x), p(x), q(x)$. We also substitute the leading coefficient, $g_e = 1$ so that we only get a non-trivial solution for g of degree exactly equal to e (if it exists), and the trivial solution ($g_0 = g_1 = ... = g_e = p_0 = ... = p_{d-1} = q_0 = ... = q_{d-1} = 0$) is avoided. As discussed in the previous lecture, solving a system of linear equations in $O(d)$ variables, reduces to computing the determinant of a $O(d) \times O(d)$ matrix, which we know is in NC. ($poly(d)$ size and $poly(\log d)$ depth)
- The circuit solves the above system of linear equations for all branches, and chooses the smallest value of e, for which we obtain a non-trivial solution. Such a solution corresponds to the smallest-degree polynomial $g(x)$ that can be written as $g = a.p+b.q$, and this implies that $g(x)$ is definitely the required output, gcd of polynomials $a(x), b(x)$. Hence, the $(e + 1)$ coefficients of this polynomial are the final outputs of our circuit.

Observe that the above circuit has $(d+1)$ branches, each having $poly(d)$ size and $poly(\log d)$ depth, as solving a system of linear equations is in NC. Therefore, the overall size of circuit is also polynomial in d, and its depth is equal to any of its branches, i.e polynomial in $\log d$, which completes the proof of theorem 7.1. ■

We proved above that computing GCD of two degree-d polynomials can be done efficiently in parallel. However, the same question, when asked with respect to integers, is one of the major open problems currently.

Open Question 7.1 $Integer - GCD \in \mathsf{NC}?$

7.2 Bipartite Matching

Now, we try to analyze one of the popular problems from graph theory, the *Bipartite Matching* problem, which has wide applications in other fields of computer science and mathematics.
Input: A bipartite graph $G = (V_1, V_2, E)$, $|V_1| = |V_2| = n$
Output: YES, if there is a perfect matching in G, NO otherwise.
A perfect matching can be represented as a bijection, $\sigma : V_1 \mapsto V_2$ such that for all $i \in V_1$, $(i, \sigma(i)) \in E$. In simple words, given a bipartite graph with n vertices in both partititons, a perfect matching chooses a set of n out of all the edges, such that all $2n$ vertices are adjacent to exactly one of the edges in the set.

Theorem 7.2 (Edmond's Algorithm (1961)) *Computing a matching in any graph $G = (V, E)$ (not necessarily bipartite) can be done in polynomial time, precisely $O(\sqrt{|V|}.|E|)$ time.* [2]

With the Edmond's algorithm, it was clear that *Matching* can be done efficiently in polynomial time for any graph. However, it is another major open question whether *Matching* can also be solved efficiently in parallel.

Open Question 7.2 $Matching \in \mathsf{NC}?$

Still, as some respite, we will prove the following theorem for finding a perfect bipartite matching:

Theorem 7.3 $Bipartite - Matching \in Randomized - \mathsf{NC}.$

Before we start describing a randomized parallel algorithm to prove the above theorem, we first introduce two basic inter-related problems related to matrices, that also form the basic steps of other problems in computer science and graph theory such as bipartite matching, the *Determinant* and the *Permanent*.

7.2.1 Permanent and Determinant

Consider any given bipartite graph $G = (V_1, V_2, E)$, with $V_1 = \{u_1, u_2, ..., u_n\}$ and $V_2 = \{v_1, v_2, ..., v_n\}$, it can be represented as a boolean $n \times n$ matrix X with entries x_{ij} $(1 \leq i, j \leq n)$ as follows:

$$x_{ij} = \begin{cases} 1 & \text{if the edge } (u_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Observe that every perfect matching in the graph G is a bijection $\sigma : V_1 \mapsto V_2$, so the following polynomial, known as the *permanent* of the $n \times n$ matrix X, is exactly equal to the number of perfect matchings in G:

$$PERM(X) = \sum_{\sigma:[n]\mapsto[n]} x_{1\sigma(1)}.x_{2\sigma(2)}.....x_{n\sigma(n)} \tag{7.2}$$

Every monomial in the above polynomial corresponds to one possible matching, and it would equal to 1 iff every variable in the monomial is 1, which would be true only when σ corresponds to a perfect matching in G.

Now, note that the above described polynomial, the *permanent* of a $n \times n$ matrix, is very similar in syntax, to the *determinant* of the matrix. Given a $n \times n$ matrix X, the determinant can be written as the polynomial:

$$DET(X) = \sum_{\sigma:[n]\mapsto[n]} sgn(\sigma).x_{1\sigma(1)}.x_{2\sigma(2)}.....x_{n\sigma(n)} \tag{7.3}$$

where $sgn(\sigma)$ refers to the sign of the permutation σ, which depends on the number of swaps required to get σ from the identity permutation ($\sigma(i) = i$ for $i = 1, 2, ..., n$). A formal way to compute the sign of a given permutation is as follows:

1. given a permutation $\sigma : [n] \mapsto [n]$, try to find cycles of the form $(a_1, a_2, ..., a_k)$ such that $\sigma(a_1) = a_2, \sigma(a_2) = a_3, ..., \sigma(a_k) = a_1$. Note that such cycles will definitely exist because σ is a bijection.
2. proceeding as above, break every permutation into disjoint cycles covering all numbers from 1 to n. Every permutation has a unique representation as a union of such cycles.
3. Then, $sgn(\sigma) = (-1)^{\text{no. of cycles of even length}}$. It can be easily verified that the identity permutation has sign $+1$ because all its cycles are of length 1. This definition can also be directly compared to the number of swaps required to obtain the premutation σ, and you will see that the sign is positive if even number of swaps are required, and negative otherwise.

Although, syntactically the two polynomials *permanent* and *determinant* are very similar syntactically except for the *sgn* factor, when we consider the complexity of computing these polynomials, they differ widely. While we have seen that $DET(X)$ can be computed by a NC circuit, $O(n^3)$ size and $O(\log^2 n)$ depth, there is no known polynomial-time algorithm, neither sequential nor parallel, for computing $PERM(X)$. In fact, it is a widely believed conjecture that $PERM(X)$ cannot be computed in polynomial time, (in fact not even in sub-exponential time). We will now see how the problem of bipartite matching is related to the two polynomials $DET(X)$ and $PERM(X)$.

7.2.2 Back to Bipartite Matching

Claim 7.4 *If the permanent can be computed by a circuit of size s, then bipartite matching can be solved by a circuit of size s.*

Proof: As we stated earlier, it can be inferred from equation 7.2 that the value of the polynomial $PERM(X)$, evaluated at the appropriate values for x_{ij}s corresponding to any given bipartite graph G, is exactly equal to the number of perfect matchings in G. In other words,

$$G \in Bipartite - Matching \iff PERM(X) > 0 \tag{7.4}$$

Hence, if the permanent of a $n \times n$ matrix can be computed by a family of circuits having size s, we can use the same family of circuits to solve the *bipartite matching* problem, and just check if $PERM(X) > 0$ in the end. ■

Unfortunately, the best known circuit to compute $PERM(X)$ is of size roughly 2^n given by Ryser's formula[1].

But, as we have efficient poly-sized circuits to compute the determinant in parallel, we would like an equation similar to equation 7.4, with the permanent replaced by the determinant. Observe that the determinant consists of the same monomials as the permanent, but along with an additional coefficient, $sgn(\sigma)$. Therefore, just like the permanent, every monomial in the determinant also corresponds to a perfect matching, but unlike the permanent, the value of $DET(X)$ does not exactly refer to the number of perfect matchings in graph G. Some permutations σ corresponding to perfect matchings might have $sgn(\sigma) = -1$, and would cancel out other monomials corresponding to perfect matchings but with sign $+1$. For example, consider the graph $G = (V_1, V_2, E)$ defined as follows: $V_1 = \{u_1, u_2, u_3\}, V_2 = \{v_1.v_2, v_3\}, E = \{(u_1, v_1), (u_1, v_2), (u_2, v_1), (u_2, v_2), (u_3, v_3)\}$. The matrix X for this graph would be:

$$X = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here, we have two perfect matchings, $M_1 = \{(u_1, v_1), (u_2, v_2), (u_3, v_3)\}, M_2 = \{(u_1, v_2), (u_2, v_1), (u_3, v_3)\}$, but the permutations corresponding to these 2 matchings, $\sigma_{M_1} = (1, 2, 3)$ and $\sigma_{M_2} = (2, 1, 3)$ have opposite signs, which makes $DET(X) = 0$. Hence, if the matrix X is described this way, it is difficult to find a relation between $DET(X)$ and perfect matchings in G.

Hence, we define X in a slightly different way, related to the *Tutte's Matrix*[3]: [1]

$$X_{ij} = \begin{cases} x_{ij} & \text{if the edge } (u_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

So, $DET(X)$ will be a polynomial in the formal variables, x_{ij} $(1 \leq i, j \leq n)$ i.e at-most n^2 variables. Now, we can claim something similar to equation 7.4 for the determinant,

Claim 7.5 *Determinant of X will be a zero polynomial if and only if G has no perfect matching. In other words,*

$$G \in Bipartite - Matching \iff DET(X) \neq 0$$

Proof: Recall that the determinant of the $n \times n$ matrix X, can be written as in equation 7.3:

$$DET(X) = \sum_{\sigma:[n]\mapsto[n]} sgn(\sigma).x_{1\sigma(1)}.x_{2\sigma(2)}.....x_{n\sigma(n)}$$

[1]For any general n-vertex graph, the Tutte matrix T is a $n \times n$ matrix defined as follows:

$$T_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in E \text{ and } i < j \\ -x_{ji} & \text{if } (i,j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

We know that each permutation σ corresponds to a possible perfect matching $\{(u_1, v_{\sigma(1)}), (u_2, v_{\sigma(2)}), ..., (n, v_{\sigma(n)})\}$ in G. Note that if this perfect matching does not exist in G (i.e. some edge $(u_i, v_{\sigma(i)}) \notin E$), then the term corresponding to σ in the summation is 0. Hence, equation 7.3 can be rewritten as:

$$DET(X) = \sum_{\sigma \in P} sgn(\sigma).x_{1\sigma(1)}.x_{2\sigma(2)}.....x_{n\sigma(n)}$$

where P is the set of perfect matchings in G. It is clear to see that the above polynomial is zero if $P = \emptyset$ i.e G has no perfect matching. This proves one side of the claim.

For the other side, if G has a perfect matching, there is a $\sigma \in P$ and the monomial corresponding to σ, $\sum_{i=1}^{n} x_{i\sigma(i)} \neq 0$. 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, $DET(X) \neq 0$. ■

From the above claim, the problem of checking whether a given bipartite graph G has a perfect matching, reduces to the problem of just checking if a given polynomial in n^2 variables, $DET(X)$ is zero or not. This problem is a case of the *Polynomial Identity Testing (PIT)* problem.

But sadly, finding an efficient algorithm to solve PIT is an open problem.

Open Question 7.3 *Polynomial Identity Testing is in* P*?*

But PIT, like Bipartite Matching can be solved by an efficient randomized algorithm. The randomized algorithms for both the problems are based on a very important result which is now discussed.

7.2.3 Schwartz-Zippel Lemma and Polynomial Identity Testing

Theorem 7.6 (Schwartz-Zippel Lemma) *Let $p \in \mathbb{F}[y_1, y_2, ..., y_m]$ be a non-zero polynomial of (total) degree d, and $S \subset \mathbb{F}$ be a finite subset of the underlying field $\mathbb{F}$, then*

$$Pr_{\underline{a} \in S}[p(\underline{a}) = 0] \leq \frac{d}{|S|} \tag{7.5}$$

where $\underline{a} = (a_1, a_2, ..., a_m)$ are m field-elements selected uniformly at random from the set S.

Before we prove the above lemma, we first describe how it helps to prove theorem 7.3, i.e $Bipartite - Matching \in Randomized - \mathsf{NC}$.

1. To apply the Schwartz-Zippel lemma to solve Bipartite Matching, we take the set S to be the set of integers $\{1, 2, ..., n^2\}$.
2. We have the polynomial $DET(X)$ in n^2 variables x_{ij} $(1 \leq i, j \leq n)$. We choose the values of each variable x_{ij} uniformly at random from S, and evaluate $DET(X)$ for these values. Let $\underline{(a)}$ denote the randomly chosen values, then we call the evaluation $DET(\underline{a})$.
3. If $DET(\underline{a}) \neq 0$ (i.e $DET(X)$ evaluated at the randomly chosen x_{ij} is non-zero), then $DET(X)$ is definitely a non-zero polynomial, and therefore we output $G \in Bipartite - Matching$.
4. Otherwise, if $DET(X)$ is evaluated and it is equal to zero, we output that G has no perfect matching. But, in this case it is possible that the polynomial $DET(X)$ is not identically zero, in which case our output would be incorrect, and that *failure probability* is given by equation 7.5 ($degree(DET(X)) = n$):

$$\begin{aligned} Pr_{\underline{a} \in S}[DET(\underline{a}) = 0] &\leq \frac{n}{n^2} \\ &= 1/n \end{aligned}$$

5. We can also repeat the algorithm by choosing another random set of x_{ij}'s and output NO only if $DET(X)$ is evaluated to zero in every iteration, thus reducing the overall failure probability.

This proves that Bipartite Matching is in Randomized-NC, as $DET(X)$ can be computed in NC. This can also be extended to Matching in general (non-bipartite) graphs using Tutte's Matrix. Now, we proceed to prove the basis of above analysis, the Schwartz-Zippel lemma.

Proof:(Of Theorem 7.6) We prove the lemma by induction on m. For the base case $m = 1$, the polynomial is $p(y_1) = p_d y_1^d + ... + p_0$. It is known that any univariate polynomial of degree d, over a field, can have at-most d roots irrespective of the set S. So the base case is proved.

Now, we assume that the result holds upto m variables and prove that it also holds for $(m+1)$ variables. We can consider p to be a polynomial in the new variable y_{m+1} by writing it as follows:

$$p(y_1, y_2, ..., y_{m+1}) = \sum_{i=0}^{d} q_i(y_1, y_2, ..., y_m).y_{m+1}^i$$
$$\Rightarrow p = q_d.y_{m+1}^d + ... + q_1.y_{m+1} + q_0$$

where q_is, coefficients of y_{m+1}^i, are polynomials of degree at-most $(d-i)$, i.e q_d is a constant. Let e be the degree of polynomial p with respect to the variable in y_{m+1}. Then, q_e is the first non-zero polynomial (of degree $d-e$ starting from the left, and $q_i = 0$ for all $e < i \leq d$.

$$p(y_1, y_2, ..., y_{m+1}) = q_e(y_1, ..., y_m).y_{m+1}^e + ... + q_1.y_{m+1} + q_0$$

Now, let A be the event that $p(y_1, ..., y_{m+1}) = 0$, and B be the event that $q_e(y_1, ..., y_m) = 0$. Then we can write:

$$Pr[A] = Pr[A \cap B] + Pr[A \cap \overline{B}]$$
$$= Pr[B].Pr[A/B] + Pr[\overline{B}].Pr[A \mid \overline{B}]$$

$\overline{B}$ denotes the complement event of B. As, $Pr[A \mid B], Pr[\overline{B}] \leq 1$, so

$$Pr[A] \leq Pr[B] + Pr[A \mid \overline{B}]$$

By induction hypothesis, $Pr[B] = Pr[q_e(y_1, ..., y_m) = 0] \leq (d-e)/|S|$ and similarly, $Pr[A \mid \overline{B}]$ is the $Pr[p(y_1, ..., y_{m+1}) = 0]$ given that $q_e(y_1, ..., y_m) \neq 0$ i.e $degree(p) = e$ so $Pr[A \mid \overline{B}] \leq e/|S|$

$$Pr[A] \leq \frac{d-e}{|S|} + \frac{e}{|S|}$$
$$= d/|S|$$

■

Apart from the randomized algorithm for bipartite-matching discussed above, the Schwartz-Zippel Lemma has several other applications. Most of the applications involve the *Polynomial Identity Testing* problem mentioned above. Let us define the problem more formally:

Definition 7.7 (Polynomial Identity Testing) *Given an arithmetic circuit of size s computing the polynomial $C(y_1, ..., y_m)$, output YES if the polynomial C is identically zero, i.e when all the monomials computed by the circuit cancel out to give the zero polynomial.*

As mentioned earlier, solving PIT in deterministic $poly(s)$ time is an open problem. The trivial algorithm has exponential running time, which independently checks whether each coefficient of C is zero or not.

Polynomial Identity Testing and the randomized algorithm for it (based on Schwartz-Zippel Lemma), form a handy tool in a lot of other problems in computer science and mathematics. Especially, if we have to prove a specific property for a given input, we can sometimes represent the task as a polynomial in some set of variables, and claim that the property is true iff the polynomial is identically zero. For example, consider the *Centroid Property* in Euclidean Geometry. For any triangle ABC with vertices $A = (a_1, a_2), B = (b_1, b_2), C = (c_1, c_2)$, if we draw the medians i.e line joining one vertex to the mid-point of the opposite side, we claim that the three medians are always concurrent. We first write the equations of the three medians in the variables $(a_1, a_2, b_1, b_2, c_1, c_2)$, then express the concurrency condition of these medians as a polynomial $p(a_1, a_2, b_1, b_2, c_1, c_2)$ being identically zero. We generate random triangles by selecting random values for the 6 variables, and solve the PIT problem for polynomial p using randomization, and output the correct answer with high probability. Similar techniques can be thought of to solve other related problems as well.

7.3 References

[1] Amir Shpilka and Amir Yehudayaoff, Arithmetic Circuits: A survey of recent results and open questions, 2010.

[2] Jack Edmonds, Paths, trees, and flowers. ,Canad. J. Math, 1965.

[2] W.T Tutte, The factorization of Linear Graphs. ,J. London Mathematical Society, 1947.