E0 309: Topics in Complexity Theory **Spring 2015**

Lecture 6: Feb 6, 2015

Lecturer: Neeraj Kayal *Scribe: Saravanan K*

6.1 A Small Recap

In the last lecture we have seen two complexity classes $\mathbf{AC}^0$ and $\mathbf{TC}^0$. We then proved some important facts such as,

1. Integer addition $\in \mathbf{AC}^0$
2. Iterated Integer Addition $\notin \mathbf{AC}^0$ but Iterated Integer addition $\in \mathbf{TC}^0$
3. Integer Multiplication $\in \mathbf{TC}^0$
4. Iterated Integer Multiplication $\notin \mathbf{AC}^0$ but Iterated Integer Multiplication $\in \mathbf{TC}^0$

In the complexity classes discussed above ($\mathbf{AC}^0$ and $\mathbf{TC}^0$) we were allowed to use gates with unbounded fanin. We wonder about the efficiency of computation when we restrict gates to bounded fanin. This question leads us to introduce a new class **NC**.

6.2 Class NC (Nick's Class) [1]

$\mathbf{NC}^d$ is the class of all boolean functions computed by boolean circuits of polynomial size and polylogarithmic depth $O(\log^d n)$.

The class **NC** is the union of $\mathbf{NC}^d$ for $d >= 1$ (i.e., $\mathbf{NC} = \cup_{d \geq 1} \mathbf{NC}^d$).

In other words we can say that the set of all problems which can be solved efficiently in parallel belongs to the class **NC** (Because parallel time complexity corresponds to depth of circuits).

Theorem 6.1 *A problem has efficient parallel algorithm (meaning, an algorithm that runs in poly-log time using polynomially many processors) iff it belongs to the class* **NC**.[1]

Proof: Suppose a problem p belongs to the class **NC**. This implies there exists a circuit with polynomial size and polylogarithmic depth that solves the problem p. We can compute all the outputs of nodes of the bottom layer by using multiple processors parallelly at time instant $t = 1$. After computing the outputs of nodes of the first layer we move up to the second layer. At time $t = 2$ all the nodes of the second layer can be computed using multiple processors parallelly. On continuing this way, we observe that the output of the root node can be computed in polylogarithmic time with polynomial number of processors.

Suppose we have a parallel algorithm that solves problem p in polylogarithmic time. From the algorithm we can design a circuit such that the tasks of all parallel computations at a specific time instant t are assigned to multiple nodes at layer t. On careful observation we note that the depth of this circuit is $O(\log^d n)$ (since the algorithm takes time $O(\log^d n)$, for some constant d).

6.3 P-completeness

Linear Programming(LP): Any linear program can be expressed in the inequality form as

$$\begin{aligned} &\text{maximize } c.x \\ &\text{subject to } A.x \leq b \end{aligned}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$ are the input data and $x \in \mathbb{R}^n$ are the output variables.

Linear programming belongs to the class **P**. However till now, no efficient parallel implementation algorithm exist for linear programs. It is still an open question that whether **P** = **NC** or **P** ≠ **NC**. This question motivates the notion of **P**-*completeness*.

P-complete (Definition): A problem p is **P**-*complete* if it is in **P** and every problem in **P** is logspace reducible to it.

Theorem 6.2 *If problem p is* **P**-*complete then $p \in$* **NC** *iff* **P** = **NC**.

Theorem 6.3 *Linear Programming(LP) is* **P**-*Complete. That is, if LP $\in$* **NC** *then* **P** = **NC**. *In other words, linear programming has a fast parallel algorithm if and only if* **P** = **NC**.

6.4 Determinant of Matrices

Let $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nn} \end{bmatrix}_{n \times n}$ where, $A_{ij} \in \mathbb{R}^{n \times n}, \forall\, 1 \leq i, j \leq n$

Now the Determinant of A is given by,

$$Det(A) = \sum_{\sigma} sign(\sigma) \prod_{j=1}^{n} A_{j\sigma(j)}$$

where σ is the set of all permutaions of [n]. The signature $sign(\sigma)$ is $+1$, when the number of swaps in the permutation is even and is -1, otherwise.

6.4.1 Properties of the Determinant

1. The determinant of any triangular matrix is the product of all the diagonal elements. For example if $A_{ut} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\ 0 & A_{22} & A_{23} & \cdots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_{nn} \end{bmatrix}_{n \times n}$
   then, $\text{Det}(A_{ut}) = A_{11}A_{22}A_{33}\cdots A_{nn}$.

2. Multiplication of a scalar to a row corresponds to scalar mulplication of the determinant value. For example if $A_\alpha = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\ \alpha.A_{21} & \alpha.A_{22} & \alpha.A_{23} & \cdots & \alpha.A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nn} \end{bmatrix}_{n\times n}$
   then, $\text{Det}(A_\alpha) = \alpha.Det(A)$.
3. Interchanging any two rows corresponds to negation of the determinant value. For example if $A_i = \begin{bmatrix} A_{21} & A_{22} & A_{23} & \cdots & A_{2n} \\ A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nn} \end{bmatrix}_{n\times n}$
   then $\text{Det}(A_i) = -Det(A)$.
4. Adding any row with another row corresponds to no change in the determinant value. For example if $A_a = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\ A_{21}+A_{11} & A_{22}+A_{12} & A_{23}+A_{13} & \cdots & A_{2n}+A_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nn} \end{bmatrix}_{n\times n}$
   then $\text{Det}(A_a) = Det(A)$

6.4.2 Importance of the Determinant

Efficient computation of the determinant value has various applications in linear algebra. Here we shall see one such example which is Cramer's rule. Let,

$$A_{n\times n}.\mathbf{x} = \mathbf{b}$$

where $A_{n\times n} \in \mathbb{R}^{n\times n}, \mathbf{b} \in \mathbb{R}^n$ are the input data and $\mathbf{x} \in \mathbb{R}^n$ is the output vector with variables $x_1, x_2, \cdots, x_n$. By Cramer's rule we know that,

$$x_i = \frac{Det(A_i)}{Det(A)} \ \forall i \in [n]$$

where A_i is the matrix obtained from A by substituting the i^{th} column of A by the vector $\mathbf{b}$.

Here we need to first compute $Det(A)$. Then the x_i's can be computed using n parallel computations thus forming two levels. Also both the levels involve the computation of the determinant value. Thus if we can find out the determinat value efficiently, then we can also solve the linear equations efficiently. Therefore if computing the determinant belongs to **NC** then the problem of solving linear equations also belongs to **NC**.

6.5 Parallel computation of matrix inverse and determinant

Theorem 6.4 *The problem of computing the determinant value of a matrix is in* **NC***(Csanky's algorithm).*

Theorem 6.5 *The problem of computing the inverse of a matrix is in* **NC***(Csanky's algorithm).*

Before proving the above theorems, let us revise some basic properties of matrices and eigen values. Given a matrix $A \in \mathbb{R}^{n\times n}$, let $\lambda_1, \lambda_2, \lambda_3, \cdots, \lambda_n$ be the eigen values of A. Now,

1. $\text{Det}(A) = \lambda_1\lambda_2\lambda_3\cdots\lambda_n$

2. $\text{Trace}(A) = \sum_{i=1}^{n}(A_{ii}) = \lambda_1 + \lambda_2 + \lambda_3 + \cdots + \lambda_n$

3. The trace of $A^i = (\lambda_1^i + \lambda_2^i + \lambda_3^i + \cdots + \lambda_n^i)$

4. Finding the product of any two matrices is in **NC**.

   Proof: Without loss of generality let us consider only $n \times n$ square matrices. Let $P_{n\times n} = X_{n\times n} \cdot Y_{n\times n}$, where $P_{n\times n}$, $X_{n\times n}$, $Y_{n\times n} \in \mathbb{R}^{n\times n}$. Here $X_{n\times n}, Y_{n\times n}$ are the input matrices and $P_{n\times n}$ is the product matrix. Let P_{ij} be the element in the i^{th} row and j^{th} column of P. By definition of matrix multiplication we observe that $P_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$.

   Computing a P_{ij} involves n field multiplications and $(n-1)$ additions. This can be computed using n number of processors in time $O(\log n)$. This implies computing a P_{ij} is in **NC**.

   We also note that, for all $i, k, j \in [n]$, A_{ik} and B_{ik} are known values (as they are input data). Therefore all the n^2 P_{ij} values can be computed parallelly in time t with total number of processors $= O(n^2 \cdot n)$ $= O(n^3)$. Therefore computing the product of two matrices belongs to the class **NC**.

5. For $1 \le i \le n$, computing A^i is in **NC**.

   Proof: We know that finding A^2 is in **NC**. Using repetitive squaring we can compute any A^i (for example $A^7 = A^4 \cdot A^3 = (A^2)^2 \cdot (A^2)A$). This takes at most $O(\log n)$ number of matrix multiplications. Therefore the problem of finding A^i belongs to the class **NC**.

6. **Cayley Hamilton Theorem**[2] : Let $p(\lambda) = Det(\lambda I - A)$ be the characteristic polynomial of A. By definition the roots of $p(\lambda)$ are the eigen values $\lambda_1, \lambda_2, \cdots, \lambda_n$ of A. As $p(\lambda)$ is a polynomial of degree n, we can write $p(\lambda) = \lambda^n + c_1\lambda^{(n-1)} + \cdots + c_n$. Cayley Hamilton theorem states that every square matrix satisfies its characteristic equation. Therefore we can write,

   $$p(A) = A^n + c_1A^{(n-1)} + \cdots + c_nI = 0 \quad \text{, where } c_n = (-1)^n Det(A)$$

   If A is non-singular then A^{-1} exists and $c_n \neq 0$. By multiplying the above equation by A^{-1} we get,

   $$A^{(n-1)} + c_1A^{(n-2)} + \cdots + c_nA^{-1} = 0 \tag{6.1}$$

   $$\implies A^{-1} = \frac{A^{(n-1)} + c_1A^{(n-2)} + \cdots + c_{n-1}I}{-c_n} \tag{6.2}$$

   If we could somehow compute the coefficients c_i in **NC**, then we could use this formula to evaluate A^{-1}(Because we know that finding A^i belongs to the class **NC**). This evaluation can be computed parallelly with $O(n)$ number of processors. The following gives a way to compute the coefficients in **NC**.

6.5.1 Newton Identities:

For $1 \leq i \leq n$, let $P_i(\lambda_1, \lambda_2, \lambda_3, \cdots, \lambda_n)$ be the sum of the i^{th} power of the eigen values. For example,

$$\begin{aligned} P_1 &= \sum_{j=1}^{n} \lambda_j \\ P_2 &= \sum_{j=1}^{n} \lambda_j^2 \\ P_3 &= \sum_{j=1}^{n} \lambda_j^3 \\ &\vdots \\ P_n &= \sum_{j=1}^{n} \lambda_j^n \end{aligned}$$

For $1 \leq i \leq n$, let $E_i(\lambda_1, \lambda_2, \lambda_3, \cdots, \lambda_n)$ be the sum of all distinct products of i distinct variables. $E_i(\lambda_1, \lambda_2, \cdots, \lambda_n)$ is the i^{th} symmetric polynomial in $\lambda_1, \lambda_2, \cdots, \lambda_n$. For example,

$$\begin{aligned} E_1 &= \sum \lambda_1 \\ E_2 &= \sum_{i<j} \lambda_i \lambda_j \\ E_3 &= \sum_{i<j<k} \lambda_i \lambda_j \lambda_k \\ &\vdots \\ E_n &= \lambda_1 \lambda_2 \cdots \lambda_k \end{aligned}$$

Since finding A^i is in **NC** , all of P_1,P_2,P_3,$\cdots$, P_n can be computed in **NC** (By finding the trace of A^i). Therefore from now on, for $i \in [n]$, the P_i's are treated as known values.

As defined earlier $p(\lambda) = \lambda^n + c_1 x^{(n-1)} + ... + c_n$ is a monic polynomial of degree n, with roots $\lambda_1, \lambda_2, \cdots, \lambda_n$. On careful observation we can note that $c_k = (-1)^k E_k$.

Using the definition of P_i and E_i we can verify that,

$$\begin{aligned} E_1 &= P_1 \\ 2E_2 &= (-)P_2 + P_1 E_1 \\ 3E_3 &= (+)P_3 - P_2 E_1 + P_1 E_2 \\ 4E_4 &= (-)P_4 + P_3 E_1 - P_2 E_2 + P_1 E_3 \\ &\vdots \end{aligned}$$

In general,

$$\begin{aligned} kE_k &= (-1)^{k+1}(P_k - P_{k-1}E_1 + P_{k-2}E_2 + \cdots P_1 E_{k-1}) \\ \implies E_k &= \frac{(-1)^{k+1}(P_k - P_{k-1}E_1 + P_{k-2}E_2 + \cdots P_1 E_{k-1})}{k} \end{aligned}$$

The above relation between polynomials P_i and E_j is called Newton's Identity[3]. In matrix notation we write,

$$P = MC$$

where, $P = \begin{bmatrix} -P_1 \\ -(P_2)/2 \\ \vdots \\ -(P_{n-1})/(n-1) \\ -P_n/n \end{bmatrix}_{n\times 1}$, $M = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -P_1/2 & 1 & 0 & \cdots & 0 \\ P_2/3 & -P_1/3 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (-1)^{n-1}P_{n-1}/n & \cdots & P_2/n & -P_1/n & 1 \end{bmatrix}_{n\times n}$, $C = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix}_{n\times 1}$

Now C can be computed by

$$C = M^{-1} \cdot P \tag{6.3}$$

Now we note that M can be written as the sum of an identity matrix and a nilpotent matrix. That is $M = I + N$ where,

$N = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ -P_1/2 & 0 & 0 & \cdots & 0 \\ P_2/3 & -P_1/3 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (-1)^{n-1}P_{n-1}/n & \cdots & P_2/n & -P_1/n & 0 \end{bmatrix}_{n\times n}$ and $I = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}_{n\times n}$

Hence,

$$\begin{aligned} M^{-1} &= (I+N)^{-1} \\ &= I + N + N^2 + N^3 + \cdots\cdots \\ &= I + N + N^2 + N^3 + \cdots + N^n \quad (\text{Since } N^i = 0,\ \forall i > n) \end{aligned}$$

We know that every N^j for $j \leq n$ can be computed in **NC**. Then we can compute M^{-1} parallelly with polynomially bounded total number of processors. After finding out M^{-1} we can compute the coefficient vector C by the matrix vector multiplication $M^{-1}P$, which belongs to **NC**. Thus we have computed the value of all the coefficients $c_1, c_2, \cdots, c_n$. Now, we can easily compute

1. The determinant of the Matrix $Det(A) = (-1)^n c_n$
2. The Inverse of the Matrix by substituting the values of $c_1, c_2, \cdots, c_n$ in (6.1).

6.6 Parallel computation of rank of matrices

6.6.1 Prerequisites:

Before getting into the algorithm let us look at some basic concepts and lemmas that provides us the necessary tools for proving the algorithm.

Permutation Matrix:
A permutation matrix is a matrix obtained by permuting the rows of the identity matrix in some order (say

σ). Any matrix A when pre-multiplied by a permutation matrix, results in the permutation of rows of A in the same order σ. On post-multiplication, we obtain the permutation of columns of A in the same order σ. For example,

$$\overset{P}{\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}_{4\times4}} \cdot \overset{A}{\begin{bmatrix}a&b&c&d\\e&f&g&h\\i&j&k&l\\m&n&o&p\end{bmatrix}_{4\times4}} = \overset{A'}{\begin{bmatrix}e&f&g&h\\a&b&c&d\\i&j&k&l\\m&n&o&p\end{bmatrix}_{4\times4}}$$

From now on, let us use the notation $\#S$ to denote the cardinality of S, where S is some finite set.

Lemma 6.6 *Schwartz-Zippel lemma : Consider an arbitrary field $\mathbb{F}$. Let $S \subseteq \mathbb{F}$ be a finite subset of cardinality $q = \#S$ and $p \in \mathbb{F}[x_1, x_2, x_3, \cdots, x_n]$ be a multivariate non-zero polynomial of (total) degree at most d. Then*

$$Prob_{\boldsymbol{a} \in_r S^n}(p(\boldsymbol{a}) = 0) \leq \frac{d}{q}$$

where $\boldsymbol{a} = (a_1, a_2, \cdots, a_n)$ is a random vector selected (uniformly at random) from S^n. The proof of this lemma will be covered in the next lecture.

6.6.2 Computation of rank

Now let us see two theorems that provides the efficient parallel computation of rank of matrices.[4]. We apply Schwartz-Zippel lemma and the parallel computation of determinant algorithm (that we have seen earlier) along with several techniques to compute the rank of matrices.

Theorem 6.7 *For any field $\mathbb{F}$, there exists an algorithm in* **NC** *that computes the rank of matrices over $\mathbb{F}$ with error probability < 0.95.* [4]

Proof: Consider an arbitrary field $\mathbb{F}$. Given a matrix $A \in \mathbb{F}^{n\times n}$, we need to find the rank of A. Take $S \subseteq \mathbb{F}$ with cardinality $q = \#S \geq 3n$. If $\#\mathbb{F} < 3n$ then we take $S = \mathbb{F}$.

Here we assume that the input matrix is a square matrix. In case of rectangular matrices, we convert them to square matrices by padding additional entries with 0's.

For a matrix $M \in \mathbb{F}^{nxn}$, let $P_i(M)$ denotes the principal $i \times i$ submatrix of M, for $1 \leq i \leq n$ (i.e., the submatrix formed by the first i rows and first i columns of M). We assume some random element generator from S (with uniform distribution over the values of S).

The algorithm to compute rank(A) is as follows.

1. Choose the matrices $B \in S^{n\times n}$, $C \in S^{n\times n}$ at random.

2. For, $1 \leq i \leq n$, compute $f_i = Det(P_i(BAC))$.
3. Find $s = max\{i : f_i \neq 0 \quad or \quad i = 0\}$
4. return s

Let $r = rank(A)$. By definition, $f_i = 0$, for all $r < i \leq n$, implying $s \leq r$. Hence the output of our algorithm (that is, s) is always less than or equal to the rank of A. Now the remainder of proof is to show that the error probability ($s < r$) is less than or equal to 0.95. That is,

$$Prob(s < rank(A)) \leq 0.95$$

We introduce the polynomial $g_A(X, Y) = Det(P_r(XAY)) \in \mathbb{F}[X_{11}, X_{12}, \cdots\cdots, Y_{nn}]$, where $X_{11}, X_{12}, \cdots, X_{nn}, Y_{11}, Y_{12}, \cdots, Y_{nn}$ are indeterminate entries of matrices B and C respectively. We note that there exists permutation matrices S, T such that $g_A(S, T) = Det(P_r(SAT)) \neq 0$. This implies $g_A(B, C)$ is a non-zero polynomial whose degree is at most $2n$. Based on the size of field $\mathbb{F}$, we analyze two cases.

Case (i) : $q \geq 3n$ (that is, when $\mathbb{F}$ is large ($\#\mathbb{F} \geq 3n$))

Using *lemma* 6.6,

$$Prob(s < r) = Prob(f_r = 0) = Prob_{B,C \in_r S^{n \times n}}(g_A(B, C) = 0) \leq \frac{d}{p} = \frac{2n}{3n} \leq 0.95$$

Case (ii) : $S = \mathbb{F}$ (that is, when $\mathbb{F}$ is small($\#\mathbb{F} < 3n$))

Below lemma (*lemma* 6.8) along with certain techniques are used to prove the analysis (described in [4]). We will not cover the proof here.

Lemma 6.8 *Let $\mathbb{F}$ be any arbitrary field. Consider $S \subseteq \mathbb{F}$ such that $\#S \geq 2$. Let g be a $n \times n$ matrix with entries $g_{ij} \in \mathbb{F}[x_{ij}]$, $\forall i, j \in [n]$, which are univariate polynomials of degree at most d, where x_{ij}'s are indeterminates over $\mathbb{F}$. Also let us define the function $u(t, m) = 1 - \prod_{i=1}^{m}(1 - t^i)$. For a random matrix $A \in_r S^{nxn}$, let $g(A)$ be the matrix with entries $g_{ij}(A_{ij})$, $\forall\ i, j \in [n]$. Then,*

1. *$Prob_{A \in_r S^{nxn}}(Det(g(A)) = 0) \leq u(d/q, n)$*
2. *If $d = 1$, then $Prob_{A \in_r S^{nxn}}(Det(g(A)) = 0) \leq 3/4$*

The proof (described in [4]*) is not covered here.*

Theorem 6.9 *For any field $\mathbb{F}$, there exists an algorithm in* **NC** *that either computes the rank of $n \times n$ matrices over $\mathbb{F}$ or returns failure with failure probability $\leq 2^{-n}$. (Proof is described in* [4]*)*

Therefore with high probability $(1 - 2^{-n})$ we are able to efficiently compute the rank of a matrix over any arbitrary field by an algorithm that runs in poly-log time using polynomially many processors.

6.7 Open Problems:

The following are some of the open problems.

1. It is conjectured that computing the determinant of a matrix does not belong to $\mathbf{TC}^0$. Proving or disproving this argument is still open.

2. Given two sequences $a_1, a_2, \cdots, a_n$ and $b_1, b_2, \cdots, b_n$ of positive integers. Is $A = \sum_i \sqrt{a_i}$ less than, equal to, or greater than $B = \sum_i \sqrt{b_i}$. Finding out whether the above problem is in class $\mathbf{P}$ is still an open problem.[5]

6.8 References

[1] Sanjeev Arora and Boaz Barak, Computational Complexity: A Modern Approach, *Cambridge University Press*, 2009

[2] http://www.cs.berkeley.edu/ demmel/cs267/lecture14.html

[3] http://en.wikipedia.org/wiki/Newton

[4] Allan Borodin, Joachim Von Zur Qathen and John Hopcroft, Fast Parallel Matrix and GCD Computations, *23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*, 1982

[5] http://cstheory.stackexchange.com/questions/79/problems-between-p-and-npc/4