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

Lecture 2: Jan 24, 2015

Lecturer: Neeraj Kayal *Scribe: Saravanan K*

2.1 A Small Recap

In the last lecture, it was shown that multiplication of two $(n-1)$-degree univariate polynomials $f(x)$ and $g(x)$ requires at least $(2n-1)$ multiplications over the base ring, assuming addition and scalar multiplication operations are completely free. The following ideas were discussed to derive the above bound.

Given two $(2n-1)$-degree polynomials as evaluations, we can compute the product polynomial by pointwise product of the evaluations. This product polynomial is in evaluations form. We can convert it into coefficient vector form using **Interpolation** technique.

Evaluation: Suppose the input polynomials are represented as coefficient vectors. We can process the coefficient vectors into $(2n-1)$ evaluation values. Let $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_{2n-1}$ be the evaluation points. Using these points we construct a Vandermonde matrix A such that $Ax = y$, where x is the coefficient vector and y is the list of evaluation values. Clearly we can compute y as a linear combination of the coordinates of x.

Interpolation: Using the evaluation points $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_{2n-1}$ we construct a Vandermonde matrix A such that $Ax = y$, where A is the Vandermonde matrix, x is the coefficient vector and y is the list of evaluation values. Since A is invertible and independent of input, we can precompute A and A^{-1}. Hence we can compute the coefficient vector by $x = A^{-1}y$. Clearly x is a linear combination of the coordinates of y.

Under the assumption that additions and scalar multiplications are free, computing a linear combination takes linear time and thus x or y can be computed in linear time.

2.2 Polynomial Multiplication (Continuation)

Now we will discuss the complexity of polynomial multiplication while the addition and scalar multiplication operations are not assumed to be free. Thus, the linear combination cannot be considered to take linear time. We need to find the complexity of computing **evaluation** and **interpolation**.

Note: Proving a tight lower bound for this problem is still an open question.

However the trivial lower bound is given by $Complexity(PM) = \Omega(n)$. This is because reading the inputs itself will take time $\Omega(n)$.
In the following discussion we will show that $Complexity(PM) = O(n \log n)$

Let the product polynomial be

$$h(x) = a_0 + (a_1)x + (a_2)x^2 + \text{.....} + (a_{2n-2})x^{2n-2}$$

and the evaluations of $h(x)$ at $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_{2n-1}$ be $h(\alpha_1), h(\alpha_2), \text{.....}, h(\alpha_{2n-1})$

Now,

$$\begin{aligned}
h(\alpha_1) &= a_0 + (a_1)\alpha_1 + (a_2)\alpha_1^2 + \text{.....} + (a_{2n-2})\alpha_1^{2n-2} \\
h(\alpha_2) &= a_0 + (a_1)\alpha_2 + (a_2)\alpha_2^2 + \text{.....} + (a_{2n-2})\alpha_1^{2n-2} \\
&\vdots \\
h(\alpha_{2n-1}) &= a_0 + (a_1)\alpha_{2n-1} + (a_2)\alpha_{2n-1}^2 + \text{.....} + (a_{2n-2})\alpha_{2n-1}^{2n-2}
\end{aligned}$$

Therefore we get

$$y = Ax$$

where, $y = \begin{bmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_m) \end{bmatrix}_{m\times 1}$, $A = \begin{bmatrix} \alpha_1^0 & \alpha_1^1 & \alpha_1^2 & \cdots & \alpha_1^{m-1} \\ \alpha_2^0 & \alpha_2^1 & \alpha_2^2 & \cdots & \alpha_2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_m^0 & \alpha_m^1 & \alpha_m^2 & \cdots & \alpha_m^{m-1} \end{bmatrix}_{m\times m}$, $x = \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{bmatrix}_{m\times 1}$ and $m = 2n-1$

The above illustration gives us the motivation to find out the complexity of Matrix Vector Multiplication.

2.3 Matrix Vector Multiplication (MVM)

Input: Fixed Matrix A $\in \mathbb{F}^{m\times m}$, Input Vector $x \in \mathbb{F}^m$, where $\mathbb{F} = \mathbb{F}_p$

Output: $y = Ax \in \mathbb{F}^m$

Observation: It is trivial to see that $MVM = O(m^2)$ (Since naive multiplication requires time $O(m^2)$, because for each row of A we require m scalar multiplications and m additions. Therefore m rows takes time $O(m^2)$). Also it has been observed that for most matrices A, with overwhelming probability,

$$MVM_A = \Omega(m^2)$$

The above complexity can be proved using the concept of SLP(Straight Line Programs). Let $x = (a_0, \cdots, a_{m-1}) \in \mathbb{F}^m$. Consider the following SLP with s computations

$$
\begin{aligned}
l_1 &= a_0 \\
l_2 &= a_1 \\
&\vdots \\
l_m &= a_{m-1} \\
l_{m+1} &= \alpha_1 l_{r_1} + \beta_1 l_{t_1} \\
&\vdots \\
l_{m+i} &= \alpha_i l_{r_i} + \beta_i l_{t_i} \\
&\vdots \\
l_s &= \alpha_{s-m} l_{r_{(s-m)}} + \beta_{s-m} l_{t_{(s-m)}}
\end{aligned}
$$

where α_i, β_i are field constants, $\forall i \in [s-m]$
and $r_i, t_i < m+i$, $\forall i \in [s-m]$.

Without loss of generality, higher degree terms are ignored (in $l_{m+i}, i \in [s-m]$) because the output contains only single degree terms.

Now,

$$\text{Number of possible matrices} = p^{m^2}$$

$$\text{Number of programs of length s} = \prod_{i \le s} \left(p^2(i-1)^2\right) \le p^{2s} s^{2s} \quad (\text{say } p > m^2)$$

Therefore,

$$\Pr[\text{Ax can be computed by a program of length } s = m^2/4] \le \frac{(p)^{2m^2/4} * (m^2/4)^{2m^2/4}}{p^{m^2}} = \frac{(m^2/4)^{m^2/2}}{(p)^{m^2/2}} = (m^2/4p)^{m^2/2}$$

$$= \text{extremely small even when } p = m^2$$

The above probability is taken over the random choice of a program of size s.

Therefore even when the size of the program is $s = m^2/4$, the probability of computing Ax is very low. Thus for most matrices A, with overwhelming probability $MVM_A = \Omega(m^2)$.

Open Problem: Is it possible to find a specific $A \in \mathbb{F}^{m \times m}$ such that $MVM_A = \Omega(m^{1+\epsilon})$ for some $\epsilon > 0$? But till now we don't know of any explicit A that provides the above bound.

2.4 Fast Fourier Transform

Since A is a fixed Vandermonde matrix, naturally the question arises that can we choose $(\underline{\alpha})$ [$\underline{\alpha}$ denotes that α is a vector] such that A($\underline{\alpha}$)x has low complexity ?

Fortunately the answer turns out to be yes. We can choose ($\underline{\alpha}$) such that the complexity of finding Ax can be improved to $O(m \log m)$.

Technique(Divide and Conquer): Without loss of generality assume $m = 2^k$(if m is not a power of 2 then increase m to its nearest power of 2 and fill the vectors with zero for the newest entries), let $\omega_m = e^{2\pi i/m}$ (m^{th} primitive root of unity).
Let,

$$(\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_m) = (\omega_m^0 = 1, \omega_m, \omega_m^2, \omega_m^3, \cdots, \omega_m^{m-1})$$

We have,

$$h(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1}$$

Now for $0 \leq i \leq m-1$, the i^{th} component of y is given by

$$(Ax)_i = h(\alpha_{i+1}) = h(\omega_m^i) = a_0 + a_1(\omega_m)^i + a_2(\omega_m)^{2i} + \cdots + a_{m-1}(\omega_m)^{i(m-1)}$$

$$\implies (Ax)_i = (A_0^i) + (A_1^i)\omega_m^i$$

$$\text{where, } A_0^i = (a_0 + a_2(\omega_m^2)^i + a_4(\omega_m^2)^{2i} \cdots + a_{m-2}(\omega_m^2)^{i(m-2)/2}$$
$$\textit{and} \quad A_1^i = (a_1 + a_3(\omega_m^2)^i + a_5(\omega_m^2)^{2i} \cdots + a_{m-1}(\omega_m^2)^{i(m-2)/2}$$

Let, $\omega_{m/2} = \omega_m^2$. As $\omega_{m/2}$ is the $(m/2)^{th}$ primitive root of unity, we can recursively compute A_0^i and A_1^i individually.

Let $T(m)$ = Time taken to compute $A(\omega_m)x$
On continuing the above recursive process we can observe that

$$\begin{aligned} T(m) &= 2T(\frac{m}{2}) + O(m) \\ &= O(m \log m) \end{aligned}$$

$\therefore$ **Time taken for evaluation** $= O(m \log m)$

Such a computation of $A(\omega_m).x$, where ω_m is a primitive m^{th} root of unity is known as Fast Fourier Transform.

2.4.1 Inversion of Vandermonde matrix:

The above Vandermonde matrix is of the form $\overset{A}{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_m & \omega_m^2 & \cdots & \omega_m^{m-1} \\ 1 & \omega_m^2 & \omega_m^4 & \cdots & \omega_m^{2(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_m^{m-1} & \omega_m^{2(m-1)} & \cdots & \omega_m^{(m-1)(m-1)} \end{bmatrix}}_{m \times m}$

Summation Lemma: For any integer $n \geq 1$, and k ($k \neq 0$ and k not divisible by n),

$$\sum_{j=0}^{n-1} (\omega_n)^k)^j = 0$$

Proof:

$$\sum_{j=0}^{n-1} (\omega_n)^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{(1)^k - 1}{\omega_n^k - 1} = 0,$$

as ω_n is a primitive n^{th} root of unity and k is not divisible by n, $\omega_n^k - 1 \neq 0$.

Theorem: The $(i,j)^{th}$ entry of A^{-1} is $\frac{\omega_m^{-ji}}{m}$, for $0 \leq i, j < m$
Proof:

$$[A^{-1}A]_{ij} = \sum_{k=0}^{m-1} (\frac{\omega_m^{-ki}}{m})(\omega_m^{kj}) = \sum_{k=0}^{m-1} (\frac{\omega_m^{k(j-i)}}{m})$$

The above summation equals to 1 iff $i = j$.
Also by the summation lemma, the above summation equals 0 if $i \neq j$.
Thus $A^{-1}A = I_{m \times m}$

2.4.2 Interpolation using inversion:

Therefore we can interpolate the coefficients, given the list of evaluations of the polynomial using another Fast Fourier transform as ω^{-1} is also primitive m^{th} root of unity. This takes $O(m \log m)$ operations.

Thus we are able to multiply any two n-degree polynomials in time $O(n \log n)$ with inputs either in coeffient form or in evaluations form. Therefore

$$Complexity(PM) = O(n \log n) \text{ operations.}$$

2.5 Credits

[1] Discussion with Abhijat Sharma

[2] My T. Thai @ UF, www.cise.ufl.edu/class/cot5405sp11/slides/ch30.pdf