Algebraic Methods In Computer Science 18/11/2014

Lecture 12

Lecturer: Amir Shpilka *Scribe: Barak Gross*

1 Topic to be discussed

- Introduction to algebraic complexity
- Discrete Fourier Transform and the Fast Fourier Transform algorithm by Cooley and Tukey
- Introduction to Arithmetic Circuits
- Arithmetic Circuits with bounded depth
- Lower Bounds on Arithmetic Circuits via Partial Derivatives
- Exam (not included)

2 Algebraic Computation Model

How efficiently can we compute the determinant of a matrix using only $\times, +$ operations?
Apparently we can do this in polynomial time.
How efficiently can we compute the permanent of a matrix using only $\times, +$ operations?
We can do it in $O(2^n)$, whether we can do better is a open question!
In the algebraic world we have two classes *VP* and *VNP* , *VP* is the algebraic analog of *P* and *VNP* is the algebraic analog of *NP*. The question of whether *VNP = VP* is the algebraic analog of the question whether *NP=P*.
Turns out that computing permanent is *VNP-Complete* , meaning that answering the question of whether the permanent can be computed "algebraically" in an efficient way, should be really hard .

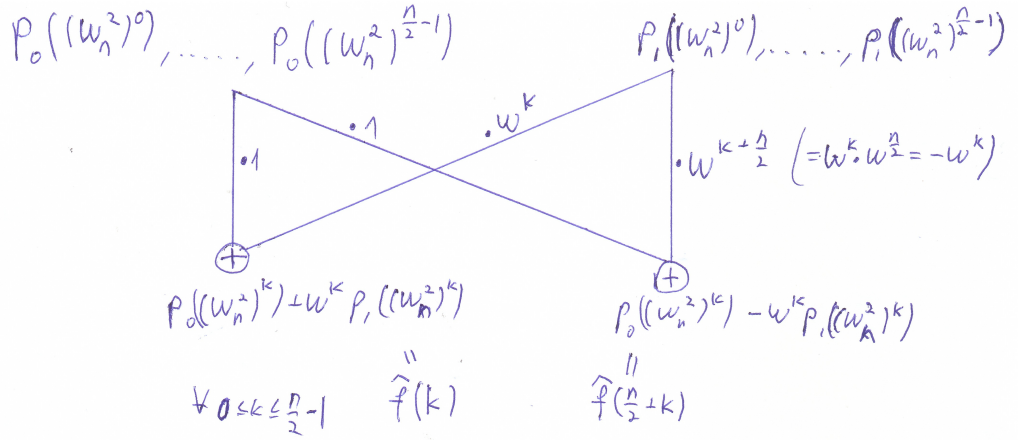
Definition 1 *Projection of Det is a polynomial of the following form:* $p(\vec{x}) = det(A)$ *where* $(A)_{(i,j)} = l_{(i,j)}(\vec{x})$ *while* $l_{(i,j)}$ *is a linear function in n variables.*

Open Problem: Can we present $Perm(X_{n\times n})$ as a projection of $Det(\square_{m\times m})$ where $m \leq n^{O(\log n)}$?

3 Fast Fourier Transform

Until now we have computed fourier transform for functions over $\mathbb{Z}_2^n$, now we want to compute fourier transform for $f : \mathbb{Z}_n \to \mathbb{R}$.
Let w_n be some n-primitive roots (i.e. $\forall 1 \leq i \leq n-1 \ : \ w^n \neq 1$, but $w^n = 1$), then we want to represent f in the following way:
$f(x) = \sum_{t=0}^{n-1} \hat{f}(t) w_n^{tx}$
In the past we had $\chi_\alpha(x) = (-1)^{\langle \alpha, x \rangle} = (-1)^{a_1 x_1} \dots (-1)^{a_n x_n}$ ($w_2 = -1$).
In $\mathbb{Z}_n$ (for some general $n \geq 2$) we have $\chi_t(x) = w_n^{tx}$ and $\langle \chi_t, \chi_s \rangle = \frac{1}{n} \sum_{x \in \mathbb{Z}_n} w_n^{(t-s)x}$
If $t = s$ we get that the inner product is 1 , otherwise it is 0.
Now, we want to know how can we compute $\hat{f}$ efficiently?
DFT: Given $f(0), f(1), \dots, f(n-1)$ we want to compute $\hat{f}(0), \hat{f}(1), \dots, \hat{f}(n-1)$ as fast as we can.
$\hat{f}(k) = \langle f, \chi_k \rangle = \frac{1}{n} \sum_{j=0}^{n-1} f(j) w_n^{-kj}$
This is equivalent to computing Av where $(A)_{kt} = w_n^{kt}$ and $v_k = f(k)$
Denote $p(x) = \frac{1}{n} \sum_{i=0}^{n-1} f(i) x^i$, so if we can compute $f(w_n^0), \dots, f(w_n^{n-1})$ fast , then we have completed our task.
So DFT is actually equivalent to evaluating a polynomial at roots of unity. Trivially we can compute this in quadratic time , but apparently we can compute this in $O(n \log(n))$ using FFT (of course we can't compute in sublinear time).
We will think about p as $p(x) = \sum_{j=0}^{n-1} \frac{f(j)}{n} x^j$, we will write $p(x) = \sum_{i=0}^{n-1} a_i x^i$.
Denote by $p_0 = \sum_{j=0}^{n/2} a_{2j} x^j$, $p_1 = \sum_{j=0}^{n/2} a_{2j+1} x^j$, we can see that $p(x) = p_0(x) + x p_1(x)$.
Meaning that $p(w_n^k) = p_0((w_n^2)^k) + w_n^k p_1((w_n^2)^k)$, now we notice that w_n^2 is $n/2$ primitive root.

Compute DFT of p_0 and p_1 (polynomials of degree at most $\frac{n}{2}$).
Then we can compute the values of the polynomial at the desired points by evaluating p_0 and p_1 at $(w_n^2)^k$ for every $k \leq \frac{n}{2}$, then the value of the fourier coefficient at w^k for $k \leq \frac{n}{2}$ is $p_0((w_n^2)^k) + w_n^k p_1((w_n^2)^k)$ for all other coefficients it equals to $p_0((w_n^2)^k) - w_n^k p_1((w_n^2)^k)$(as depicted at the picture below):



Algorithm 1: FFT Algorithm by Cooley and Tukey 1965

We notice that we can express the running time of our algorithm by the following recurrence function: $T(n) = 2T(\frac{n}{2}) + O(n)$
Using some simple tools we can proof that $T(n) = O(n \log(n))$:

1. Induction
2. We look at a binary tree of depth $O(\log(n))$, such that each leaf has weight $O(n/2^i)$ where i is the level of the node. Level i has 2^{i-1} leaves , meaning each level has weight $O(n)$, from which we deduce that the total weight of the tree is $O(n \log(n))$. So if we model the running time of our algorithm as binary tree , we get that it's "weight" (here running time) is $O(n \log(n))$

Open Question: Can we do better?

Every step of the calculation is of the following form: $\alpha l_i + \beta l_j$, where l_i, l_j are linear functions that we already calculated and α, β are scalar such that $|\alpha| = |\beta| = 1$.

Definition 2 (Linear Algorithm) *An algorithm is called linear if it only uses multiplications by scalars or addition operations.*

Theorem 3 (Morgenstern, 1973) *Every linear algorithm that compute DFT , in which the absolute value of the scalars is bounded by C , takes at least $\Omega_C(n \log n)$ addition operations.*

Proof: At each step we will write the linear function that was calculated at that same step. More accurately we will write its coefficients.
So our input is the identity matrix, at step k the algorithm computes $l_k = \alpha_i l_i + \beta_i l_j$ (l is a row in the matrix), where $i, j < k$ and $|\alpha|, |\beta| \leq C$.
We define the following function $\varphi(k) = \max_{\substack{A \in \text{ matrices we} \\ \text{had until the} \\ \text{k'th step}}} Det(n \times n \text{ Submatrix of } A)$,

$\varphi : \{n, \ldots, m\} \to \mathbb{R}^+$
$\varphi(n) = 1$, $\varphi(m) \geq |Det(A)| = n^{\frac{n}{2}}$
Where $(A)_{kt} = w_n^{-kt}$. The reason for the last equality is that since the rows are perpendicular we have that if we take that matrix and multiply it by its transpose we'll get a diagonal matrix that contains on its diagonal the norm of each and every row in the matrix , squared , and it is not hard to see that it equals to n . Meaning that the determinant of the last matrix , squared, equals to n^n, which proves our weak inequality.

Claim 4 $\varphi(k) \leq 2C\varphi(k-1)$

From the claim we can deduce that $(2C)^{m-n} \geq \varphi(m) \geq n^{\frac{n}{2}}$ which essentially proves our theorem. So lets just prove the claim.
Proof: Proof by induction , the base case is easy (k=n+1) since $\varphi(n+1) \leq 2C = 2C\varphi(n)$. Assuming the induction holds for $k-1$ we will prove it also holds for k . Denote by $i_1, \ldots, i_n \leq k$ the rows from which $\varphi(k)$ is obtained ,then:
1. If $\forall j \leq n_{i_j} \leq k-1$ so $\varphi(k) = \varphi(k-1)$
2. o.w. $\exists s, r < k$ such that $\varphi(k) = det \begin{bmatrix} — & l_{i_1} & — \\ \cdots & \cdots & \cdots \\ — & l_{i_n} & — \\ — & \alpha l_s + \beta l_r & — \end{bmatrix} =$

$$= |\alpha| det \begin{bmatrix} — & l_{i_1} & — \\ \cdots & \cdots & \cdots \\ — & l_{i_n} & — \\ — & l_s & — \end{bmatrix} + |\beta| det \begin{bmatrix} — & l_{i_1} & — \\ \cdots & \cdots & \cdots \\ — & l_{i_n} & — \\ — & l_r & — \end{bmatrix} \leq$$

$\leq (|\alpha| + |\beta|)\varphi(k-1) \leq 2C\varphi(k-1)$
■

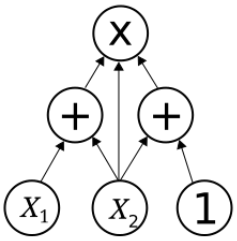
■

4 Arithmetic Circuits

Definition 5 (Circuit) *Circuit is a simple direct acyclic graph.*

Definition 6 (Input) *Vertex with in-degree 0 is called input.*

Inputs can be labeled by a scalar or by x_i for some i (depends on whether it is a variable or a constant). Other vertices are labeled by $\times$ or $+$, for example the following circuit computes the polynomial $(x_1 + x_2)x_2(x_2 + 1)$:



An arithmetic circuit computes a polynomial in the following natural way:
An input gate computes the polynomial it is labeled by.
A sum gate computes the sum of the polynomials computed by its children (a gate u is a child of v if the directed edge (u,v) is in the graph).
A product gate computes the product of the polynomials computed by its children.

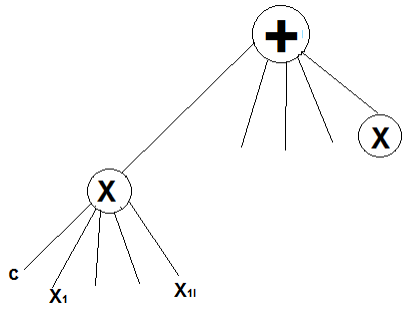
Consider the circuit in the figure, for example: the input gates compute (from right to left) x_1, x_2 and 1, the sum gates compute $x_1 + x_2$ and $x_2 + 1$, and the product gate computes $(x_1 + x_2)x_2(x_2 + 1)$.

Definition 7 (VP) VP *is a set of families of polynomials on n variables with polynomial degree (polynomial in n), and that there exists a constant C , such that each polynomial in that family can be computed by a circuit of size* $O(n^C)$.

VP against *VNP* actually asks whether $PERM_n$ has a polynomial circuit.
fact: If $f(x_1, \ldots, x_n)$ has a circuit of size s , then there exists a matrix of size $s^{\log s} \times^{\log s}$ whose inputs are linear function , and it has determinant that equals to f identically.

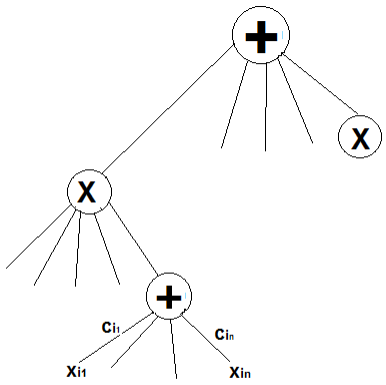
5 Circuits with bounded Depth

Circuit Of Depth 2:



Notice that the family of polynomials that can be computed with arithmetic circuits of depth 2 , are exactly the family of polynomials that can be written as sum of monomials. The complexity of polynomial is the number of monomials.

Circuit Of Depth 3:



Notice that the family of polynomials that can be computed with arithmetic circuits of depth 3 , are exactly the family of polynomials that can be written as sum of products of linear functions

Main Open Question: Prove exponential(in n) lower bound for arithmetic circuit of depth 3 that calculate PERM_n

Theorem 8 (Proved Recently) *If f has a circuit of polynomial size and polynomial degree M then f has arithmetic circuit of depth and size $n^{\tilde{O}(\sqrt{n})}$* [1] [2]

Theorem 9 *Every circuit of depth 3 for $PERM_{n\times n}/DET_n$ in which the degree of each gate is at most n has to be of size $exp(n)$.*

[1] In the reduction the degree of each gate explodes to $\tilde{O}(\sqrt{n})$

[2] Algorithm that runs in $\tilde{O}(f(n))$ means that that there exists a constant C such the algorithm runs in $O(f(n)) \times \log^C(n)$

Note: Such a circuit for the permanent problem is known , but now such circuit is known for the problem of computing determinant.

6 Lower Bounds on Arithmetic Circuits via Partial Derivatives

Denote by $\mu_k(f)$ the dimension of the space spanned by the set of all of f's partial derivatives of order k.

Claim 10 *If f can be computed by a circuit of depth 3 with s multiplication gates , each of degree at most d, then $\mu_k(f) \leq s\binom{d}{k}$*

Claim 11 $\mu_k(DET_n), \mu_k(Perm_n) \geq \binom{n}{k}^2$

Claim 12 *1.* $\mu_k(f+g) \leq \mu_k(f) + \mu_k(g)$

2. $\mu_k(L_1 \cdot \ldots \cdot L_d) \leq \binom{d}{k}$

Proof: The first item is trivial (because of the fact that sum of derivatives is the derivative of the sum).
Every k 'th partial derivative of $L_1, \ldots, L_d$ is linear combination of the polynomials $\prod_{\substack{i \in T \\ |T|=d-k}} L_i$
There are exactly $\binom{d}{k}$ such polynomials.
■

Using the claims we just proved we can easily also prove claim 9.
Proof: [Claim 10] Choose k rows, then k columns , and lets look at the matrix induced by this rows and columns. [3] Let $x_{i_1,j_1}, \ldots, x_{i_k,j_k}$ the elements on the diagonal.
If we derive the determinant function by $x_{i_1,j_1}, \ldots, x_{i_k,j_k}$ then we get $det(X_{[n]\setminus\{i_1,\ldots i_k\},[n]\setminus\{j_1,j_k\}})$
Let's look at the family of polynomials we get by deriving the determinant function according to $x_{i_1,j_1}, \ldots, x_{i_k,j_k}$, then we can find substitutions that shows linearly independence of the polynomials in that family.
One may consider the following substitutions : Examine the diagonal induced by the square submatrix we get by deleting rows $i_1, \ldots, i_k$ and columns $j_1, \ldots, \jmath_k$, for every x that is not on the diagonal substitute $x = 0$ and for x on the diagonal $x = 1$. In similar fashion we can prove this for the permanent.

[3] meaning the matrix contains only elements whose column belongs to the group we choose , and also the same goes for the row of that element

■

Notation:[The Elementary Symmetric Functions] $\sigma_{d,n} = \sum_{\substack{|T|=d \\ T\subseteq[n]}} \prod_{i\in T} x_i$

Claim 13 *There is a depth 3 arithmetic circuit of degree n with n product gates for* $\sigma_{d,n}(x)$

Proof: We write $f(y) = \prod_{i=1}^{n}(y + x_i) = \sum_{e=0}^{n} y^{n-e}\sigma_{e,n}(\vec{x})$.
Using interpolation we can find constant $\alpha_0^d, \ldots, \alpha_{n-1}^d$ such that the $n-d$ coefficient of f is :
$\sum_{i=0}^{n-1} \alpha_i^d f(i) = \sum_{i=0}^{n-1} \alpha_i^d \prod_{j=1}^{n}(1 + x_j)$ ■

Claim 14 *Every arithmetic circuit of depth 3 and degree at most d for* $\sigma_{d,n}$ *is of size* $\Omega(\binom{n}{d/2}/2^d)$

References

[1] Cooley, James W.; Tukey, John W. , *"An algorithm for the machine calculation of complex Fourier series"* ,1965.

[2] Morgenstern, Jacques , *"Note on a lower bound of the linear complexity of the fast Fourier transform"*, J. ACM 20 (2): 305306. ,1973.

[3] Noam N. Noam .; Avi Wigderson, *Lower Bounds For Arithmetic Circuits Via Partial Derivatives*,1996.