6.S897 Algebra and Computation April 18, 2012

Lecture 18 - Arithmetic Complexity

Instructor: Madhu Sudan *Scribe: Omer Paneth*

1 Today - Arithmetic Complexity

1. Models of computation.
2. Basic problems and results.

Most of the material of today's lecture is covered in the survey of [SY10] (see link in the course website).

2 Problems

We will be interested in two types of problems:

1. Computing a function $\phi : \mathbb{F}^n \to \mathbb{F}^m$ of the form $\phi = (\phi_1, \dots, \phi_m)$ where $\phi_i \in \mathbb{F}[x_1, \dots, x_n]$ is a polynomial. (For example, computing the determinant or the permanent of an $n \times n$ matrix).
2. Given $\phi : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^l$ and given $x \in \mathbb{F}^n$, finding $y \in \mathbb{F}^m$ such that $\phi(x, y) = \vec{0}$. (This problem is similar to Hilbert Nullstellensatz).

3 Arithmetic Circuits

Also known as straight-line programs, this is a natural arithmetic model of computation (similar to boolean circuits).

Definition 3.1 (Informal)**.** An *arithmetic circuit* $\mathcal{C}$ over a field $\mathbb{F}$ consists of the following:

Input variables: $x_1, \dots, x_n$.

Gates: A list of gates of the form "$y_i \leftarrow A \diamond B$" where $\diamond \in \{+, -, *, \div\}$ and $A, B \in \{x_1, \dots, x_n, y_1, \dots, y_{i-1}\} \cup \mathbb{F}$. For example:

$$y_1 \leftarrow x_1 + 0$$
$$y_2 \leftarrow y_1 * x_2$$

Output variables: A subset of the variables $\{y_{i_1}, \dots, y_{i_m}\}$.

An arithmetic circuit is a *formula* if every y_i appears in the RHS at most once (every variable y_i serves as input to at most one gate).

We say that an arithmetic circuit $\mathcal{C}$ computes a function ϕ if for every $x = (x_1, \dots, x_n) \in \mathbb{F}^n$ we have that $y_{i_1}, \dots, y_{i_m} = \phi(x)$.

3.1 Complexity Measures

We consider the following complexity measures for an arithmetic circuit $\mathcal{C}$:

$\mathsf{size}(\mathcal{C})$: Number of gates. Similarly, $+\mathsf{size}$ and $*\mathsf{size}$ are the number of addition and the number of multiplication gates, respectively.

$\mathsf{depth}(\mathcal{C})$: Longest chain of dependencies.

$\mathsf{mem}(\mathcal{C})$: The maximal number of variables that are "necessary" at any point:

$$\max_i |\{y_j | j < i \text{ and } y_j \text{ is used after } i\}|$$

$\mathsf{deg}(\mathcal{C})$: Maximal degree of the polynomial that is computed by any gate (clearly we can think of every gate as computing some intermediate polynomial). Note that the degree of the function (as a polynomial) can be smaller than the degree of a circuit computing the function.

For every arithmetic circuit $\mathcal{C}$ we have that $\mathsf{deg}(\mathcal{C}) < 2^{\mathsf{depth}(\mathcal{C})}$, since at every level of the circuit the maximal degree can at most double.

4 Valiant's Classes

In [Val79], Valiant defines the arithmetic complexity classes (known today as) VP and VNP that are analogous to the classes P and NP. A very different analog is presented in [BSS88].

Definition 4.1. A set of functions $\Phi = \left\{\phi_n : \mathbb{F}^{\text{poly}(n)} \to \mathbb{F}\right\}_n$ is in VP iff $\mathsf{deg}(\phi_n) = \text{poly}(n)$ and there exists a polynomial $p(n)$ such that for every n there exists a circuit $\mathcal{C}_n$ that computes ϕ_n and $\mathsf{size}(\mathcal{C}_n) < p(n)$.

Definition 4.2. A set of functions $\Phi = \left\{\phi_n : \mathbb{F}^{\text{poly}(n)} \to \mathbb{F}\right\}_n$ is in VNP iff there exists a function $\Psi \in \mathsf{VP}$, $\Psi = \left\{\psi_n : \mathbb{F}^{\text{poly}(n)} \times \mathbb{F}^{\text{poly}(n)} \to \mathbb{F}\right\}_n$, such that.

$$\phi_n(x) = \sum_{w \in \{0,1\}^{\text{poly}(n)}} \psi_n(x, w)$$

The class VQP is defined similarly to VP, except that we require the existence of a circuit of quasi-polynomial size ($2^{\text{polylog}(n)}$).

Interestingly, every function in VP has an arithmetic circuit of size $O(\log^2(n))$ [VSBR83]. It is believed that the analogous result for boolean circuits is false.

Next, we define the notion of reducibility via a projective reduction.

Definition 4.3. The function ϕ is reducible to the function ψ ($\phi \leq_p \psi$) if there exist $y_1, \ldots, y_m \in \{x_1, \ldots, x_n\} \cup \mathbb{F}$ such that $\phi(x_1, \ldots, x_n) = \psi(y_1, \ldots, y_m)$.

The reduction is polynomial if $m = \text{poly}(n)$ and it is quasi-polynomial if $m = 2^{\text{polylog}(n)}$.

4.1 Complete Problems

Let DET_n and PERM_n be the determinant and permanent of an $n \times n$ matrix, respectively. Clearly $\mathsf{DET} \in \mathsf{VP}$ (follows from Gaussian elimination). To see that $\mathsf{PERM} \in \mathsf{VNP}$, consider Ryser's formula for the permanent.

$$\sum_{S \subseteq [n]} (-1)^{n-|S|} \prod_{i=1}^{n} \sum_{j \in S} x_{i,j}$$

Theorem 4.4 ([Val79])**.** *1.* DET *is* VQP*-complete (w.r.t quasi-polynomial reductions).*

2. If $\mathsf{chr}(\mathbb{F}) \neq 2$*, then* PERM *is* VNP*-complete (w.r.t polynomial reductions).*

We believe (but are unable to prove) that $\mathsf{VP} \neq \mathsf{VNP}$ and that $\mathsf{VQP} \neq \mathsf{VNP}$. Using Theorem 4.4 we can state the above question as follows: for $X \in \mathbb{F}^{n \times n}$, $X = \{x_{i,j}\}_{i,j \in [n]}$ is there a matrix $Y \in \mathbb{F}^{m \times m}$, $m = 2^{\mathrm{polylog}(n)}$ such that every $y_{i',j'} \in \{x_{i,j}\}_{i,j \in [n]} \cup \mathbb{F}$ and $\mathsf{PERM}_n(X) = \mathsf{DET}_n(Y)$?

5 Removing Division Gates

We will show that removing division gates does not affect the power of our model (up to polynomial factors).

Theorem 5.1. *If ϕ is a polynomial in n valuables of degree r that can be commuted by a circuit of size s with gates in $\{+, -, *, \div\}$, then ϕ can be computed with* $\mathrm{poly}(n, r, s)$ *gates in* $\{+, -, *\}$.

We will start by proving the following lemma regarding the computation of $\mathsf{HOM}_i(\phi)$ - the homogeneous degree i part of a polynomial ϕ.

Lemma 5.2 (Homogenization Lemma)**.** *If ψ can be computed by a circuit of degree r and size s, then the function* $(\mathsf{HOM}_0(\psi), \ldots, \mathsf{HOM}_r(\psi))$ *can be computed by a circuit of degree r and size $O(r^2 s)$.*

Proof. We prove by induction on the structure of the circuit computing ϕ. The base case (ϕ is a constant or an input variable) is trivial. If $\phi \leftarrow \phi_1 + \phi_2$ then for every i:

$$\mathsf{HOM}_i(\phi) = \mathsf{HOM}_i(\phi_1) + \mathsf{HOM}_i(\phi_2)$$

If $\phi \leftarrow \phi_1 * \phi_2$ then for every i:

$$\mathsf{HOM}_i(\phi) = \sum_{j=1}^{i} \mathsf{HOM}_j(\phi_1) \cdot \mathsf{HOM}_{i-j}(\phi_2)$$

Let s_1, r_1 and s_2, r_2 be the size and degree of the functions ϕ_1 and ϕ_2, respectively. We have that $s = 1 + s_1 + s_2$ and $r = r_1 + r_2$. By the inductive hypotheses, computing all homogeneous parts of ϕ_1, ϕ_2 requirers size $O(r_1^2 s_1 + r_2^2 s_2)$. In addition, for every pair $i, j \in [r_1] \times [r_2]$, the element $\{\mathsf{HOM}_i(\phi_1) \cdot \mathsf{HOM}_j(\phi_2)\}$ appears in one of the homogeneous parts of ϕ and therefore the total size of the circuit is $O(r_1^2 s_1 + r_2^2 s_2 + r_1 r_2) = O(r^2 s)$. □

Theorem 5.1. Without using division gates we can emulate the computation of the original circuit for ϕ, explicitly computing the numerator and the denominator. That is, using only gates in $\{+, -, *\}$, we can emulate gate of the form:

$$(f, g) = (f_1, g_1) \diamond (f_2, g_2)$$

where $\diamond \in \{+, -, *, \div\}$. After this emulation we get $\phi = f/g$. Next, we show how to eliminate this last division. It must be that $g \not\equiv 0$, that is, $g(\vec{\alpha}) \neq 0$ for some $\vec{\alpha}$. Assuming the field is large enough, we can translate and scale the input such that $g(\vec{0}) = 1$, without significantly increasing the size of the circuit. Now we can write:

$$\phi = \frac{f}{g} = \frac{f}{1-(1-g)} = \sum_{i=0}^{\infty} f(1-g)^i$$

Note that the last inequality only makes sense for every homogeneous part separately. Since $\mathsf{HOM}_0(1-g) = 0$ we get that for $j < i$, $\mathsf{HOM}_j((1-g)^i) = 0$ and therefore:

$$\mathsf{HOM}_k(\phi) = \mathsf{HOM}_k(\sum_{i=0}^{\infty} f(1-g)^i) = \mathsf{HOM}_k(\sum_{i=0}^{k} f(1-g)^i)$$

Since the size of f, g is $\mathrm{poly}(s, r)$ we get that for every $k \leq r$, $\mathsf{HOM}_k(\phi)$ can be computed by a circuit of size $\mathrm{poly}(s, r)$. $\square$

6 Constant Memory Circuits

We will show that any function that can be computed with logarithmic depth can also be computed in polynomial size and constant memory. We get the same qualitative results as Barrington's theorem with a simple proof.

Theorem 6.1 ([BOC92]). *Every function ϕ that is computable by a circuit of depth d, can be computed by a circuit of size 4^d and memory 3.*

Proof. We assume ϕ is computed without division gates (see Theorem 5.1). We will construct a circuit of size 4^d that will only use the variables U, V, W and will compute the function $(U + V \cdot \phi, V, W)$. Construction is recursive. For $d = 0$ the construction is trivial. If $d > 0$, by the inductive hypotheses, for every ψ of depth $d-1$ there exists a circuit $\mathcal{C}_\psi(U, V, W)$ of size 4^{d-1} that computes the function $(U + V \cdot \psi, V, W)$. If $\phi = \phi_1 + \phi_2$ then we construct $\mathcal{C}_\phi$ as follows:

1. $(U + V\phi_1, V, W) \leftarrow \mathcal{C}_{\phi_1}(U, V, W)$.
2. $(U + V(\phi_1 + \phi_2), V, W) \leftarrow \mathcal{C}_{\phi_2}(U + V\phi_1, V, W)$.
3. $(U, V, W) \leftarrow (U + V(\phi_1 + \phi_2), V, W)$.

(The case $\phi = \phi_1 - \phi_2$ is handled similarly). If $\phi = \phi_1 * \phi_2$ then we construct $\mathcal{C}_\phi$ as follows:

1. $(U + V\phi_1, V, W) \leftarrow \mathcal{C}_{\phi_1}(U, V, W)$.

2. $(W + U\phi_2 + V\phi_1\phi_2, U + V\phi_1, V)) \leftarrow \mathcal{C}_{\phi_2}(W, U + V\phi_1, V)$.

3. $(U, V, W + U\phi_2 + V\phi_1\phi_2, , V)) \leftarrow \mathcal{C}_{-\phi_1}(U + V\phi_1, V, W + U\phi_2 + V\phi_1\phi_2)$.

4. $(W + V\phi_1\phi_2, U, V) \leftarrow \mathcal{C}_{-\phi_2}(W + U\phi_2 + V\phi_1\phi_2, U, V)$.

In any case, $\mathsf{size}(\phi) \leq 4 \cdot \max(\mathsf{size}(\phi_1), \mathsf{size}(\phi_2)) \leq 4^d$. □

7 Lower bounds

We cover some lower bounds on the size of arithmetic circuits. Strassen showed that computing the following simple set of polynomials requires a super-linear size circuit.

Theorem 7.1. *Every circuit computing the function $(x_1^r, \ldots, x_n^r)$ must be of size $\Omega(n \log(r))$.*

The existence of a single polynomial that cannot be computed by linear-size circuits follows from the following theorem.

Theorem 7.2 ([BS83])**.** *If the polynomials $\phi_1, \ldots, \phi_n$ can only be jointly computed by a super-linear size circuit, then the following polynomial can only be computed by a super-linear size circuit:*

$$\hat{\phi}(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n} \phi_i(\vec{x}) \cdot y_i$$

Corollary 7.3. *The function $\sum_{i=1}^{n} y_i x_i^r$ requires a super-linear size circuit.*

The proof on Theorem 7.2 uses the following theorem about partial derivatives:

Theorem 7.4 ([BS83])**.** *If the function ϕ can be computed by a circuit of size s, then the following function that jointly computes all the partial derivatives of ϕ:*

$$\left(\frac{\partial \phi}{\partial x_1}, \ldots, \frac{\partial \phi}{\partial x_n}\right)$$

can be computed by a circuit of size s.

References

[BOC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[BS83] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

[BSS88] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation over the real numbers; np completeness, recursive functions and universal machines (extended abstract). In *FOCS*, pages 387–397, 1988.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[VSBR83] Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.