Arithmetic Circuit Complexity: NPTEL Course – Nitin Saxena

Soham Chatterjee

Contents

Chapter 1

Introduction

1.1 Turing Machines

Classically, computation i modeled using Turing Machine i.e. a computer program is scene as TM description

Turing Machine (TM): $M = (\Gamma, Q, \delta)$ where

- Γ = Set of alphabets, say 0,1, $\triangleright$ (start), $\square$ (blank)
- Q = Set of states (at least it has start state $= q_s$, final state $= q_f$)

  [whenever we say computation, we mean q_s to q_f finitely many steps are taken and whatever is there in tape is considered as output.]
- δ = transition function

$$\delta : Q \times \underset{\substack{\text{(Assuming 2}\\\text{tapes one for}\\\text{input bit and}\\\text{other for work}\\\text{tape for reading}\\\text{bit at current}\\\text{work tape head)}}}{\Gamma^2} \longrightarrow Q \times \Gamma^2 \times \{\underbrace{\overset{\text{Stay}}{S}, \overset{\text{Left}}{L}, \overset{\text{Right}}{R}}_{\text{head movement}}\}$$

[you can think δ as your C program or computer program]

Since work tape is infinite you don't know how many steps will be taken. TM abstracts every possible device

Definition 1.1.1: Time and Space of TM

- **Time** is the number of steps for a given input x.
- **Space** is the number of worktape-cells used by TM on x

1.2 Complexity Classes

Definition 1.2.1: $Dtime(f(n))$ and $Space(f(n))$

For a function $f : \mathbb{N} \to \mathbb{R}_{>0}$ we can define complexity classes

- $Dtime(\boldsymbol{f}(\boldsymbol{n}))$: { Set of all those problems that can be solved on a TM in time $O(f(n))$ }
- $Space(\boldsymbol{f}(\boldsymbol{n}))$: { Set of all those problems that can be solved on a TM in work space $O(f(n))$ }

This leads to a zoo of complexity classes

Definition 1.2.2: $P,\ PSpace,\ NP,\ \mathbb{L},\ EXP$

- $P \coloneqq \bigcup_{c>0} Dtime(n^c)$
- $PSpace \coloneqq \bigcup_{c>0} Space(n^c)$
- $NP \coloneqq \bigcup_{c>0} \underset{\substack{\downarrow \\ \text{on a non} \\ \text{-deterministic } TM}}{Ntime(n^c)}$
- $\mathbb{L} \coloneqq Space(\log n)$
- $EXP \coloneqq \{$ Problems that can be solved in time 2^{n^c} $\}$

Note:-

$$\mathbb{L} \subseteq P \subseteq NP \subseteq PSpace \subseteq EXP \subseteq EXPSpace \subseteq EEXP \subseteq \cdots$$

There are **randomized versions** (using probabilistic TM)

$$\underset{\substack{\text{zero error} \\ \text{probabilistic} \\ \text{poly-time} \\ \text{(Las-Vegas} \\ \text{Algorithms)}}}{ZPP} \subseteq \underset{\substack{\text{one-sided} \\ \text{error}}}{RP} \subseteq \underset{\substack{\text{both-sided} \\ \text{error} \\ \text{(Bounded error} \\ \text{Probabilistic} \\ \text{poly-time)}}}{BPP} \subseteq \underset{\substack{\text{Probabilistic} \\ \text{poly error}=\frac{1}{2} \\ \text{both sided}}}{PP} \subseteq PSpace$$

Oracle-based complexity classes:

$$\underset{\substack{\| \\ \Sigma_0}}{P} \subseteq \underset{\substack{\| \\ \Sigma_1}}{NP} \subseteq \underset{\substack{\| \\ \Sigma_2}}{NP^{NP}} \subseteq \underset{\substack{\| \\ \Sigma_3}}{NP^{\Sigma_2}} \subseteq \underset{\substack{\| \\ \Sigma_4}}{NP^{\Sigma_3}} \subseteq \cdots \subseteq PH \subseteq PSpace$$

This hierarchy is called Polynomial Hierarchy. Union of all of these is called PH

This course will take a different route to build a zoo of complexity classes

1.3 Arithmetic Circuits

Instead of seeing computation as a sequence of very simple steps (that's what TM does. At each step transition is trivial but in the end something highly non trivial happens.) We'll review it as an algebraic expression

Definition 1.3.1: Arithmetic Circuits

An arithmetic circuit C, over a field $\mathbb{F}[\overline{x}]$, is a rooted DAG as follows

- The **leaves** are the variables $x_1, x_2, \ldots, x_n$ or field constant
- The **root** outputs a polynomial $C(\overline{x}) \in \mathbb{F}[\overline{x}]$ (input)
- The **Internal vertices** are gates that compute $(\times)$ or $(+)$ in $\mathbb{F}[\overline{x}]$
- The **edges** are called wires and they have constant labels to do scalar multiplication.

Theorem 1.3.1

Any polynomial has a depth-2 circuit

Proof. In first layer you have addition and in the bottom layer you have multiplication □

Definition 1.3.2: Size, Depth, Degree

- **Size:** The size of the DAG (# of wires) is the size of the circuit size (c). Sometimes we include the bit size of the constants on the wires
- **Depth:** A Max-path from a leaf to the root determines the depth of the circuit.
- **Degree:** Degree of c is the degree of intermediate polynomials computed in c

Question 1

How many monomials are there in n variable d degree polynomial ?

Solution: $\binom{n+d}{d} \approx \left(\frac{n}{d}\right)^d, \left(\frac{d}{n}\right)^n$

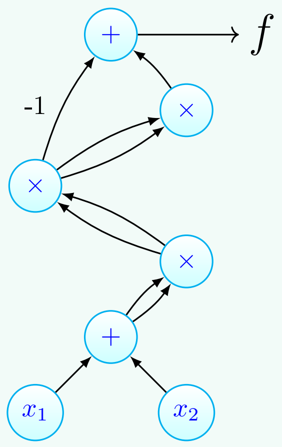
$\square$

Example 1.3.1

$f(x_1, x_2) = (x_1 + x_2)^8 - (x_1 + x_2)^4.$

The circuit size is small because of repeated squaring

Another example for repeated squaring is $(1 + x)^{2^n}$



Question 2: Foundational Question in this area

When a polynomial is not possible to compress by circuit representation and only way is depth-2 (worst possible way)

Definition 1.3.3: Fanin, Fanout, Formula, Family of Circuits

- **Fanin:** Maximum in-degree
- **Fanout:** Maximum out-degree
- **Formula:** A circuit with fanout=1 is called formula

Definition 1.3.4: Family of Circuits

Suppose $\mathcal{F} := \{f_1(x_1, \ldots, x_i) \mid n \geq 1\}$ is a family of polynomials (call it a problem). A family of circuits $\mathcal{C} := \{C_i(x_1, \ldots, x_n) \mid i \geq 1\}$ solves $\mathcal{F}$ $\forall$ i, $C_i = f_i$

In this case, we say that $\mathcal{F}$ can be solved in size bounded by $size(C_n)$ and depth bounded by $depth(C_n)$. This two functions basically tell you the circuit complexity of the set of polynomials $\mathcal{F}$

Note:-

Depth can be thought of as time (In parallel algorithm). Size can be though of as space.
This gives us a new way to measure the complexity of polynomials (or problems)

1.4 Arithmetic Complexity Classes

Arithmetic Complexity Classes were first defined by Valiant (1979). In particular, the arithmetic analogues of P and NP

Definition 1.4.1: VP (Valiant's P)

VP consists of polynomial families say $\{f_n\}_n$ [f_n is a n variate polynomial] that can be solved or computed by Arithmetic Circuits of size-poly(n) and degree-poly(n)

Question 3

Why degree-poly(n) condition?

Solution: You want it to be practically implementable i.e. every aspect of the computation you would ideal want to be efficiently implementable on a turing machine. One necessary condition is degree should not blow p. We don't want circuit at a point to become too large.

The family $\{x^{2^n}\}_n \notin VP$. Though it is computable by $O(n)$ size arithmetic circuits, its degree is not poly(n)

$\square$

We keep some field $\mathbb{F}$ in mind while defining VP.

Note:-

Constants don't contribute to size or degree

An interesting polynomial (family) in VP is the **Determinant**.

$$\underset{n}{\det}(X_{n\times n}) := \sum_{\pi\in Sym(n)} sign(\pi) \overset{\substack{n^{\frac{n}{2}}\approx n!\\ \text{such monomials}\\ \uparrow}}{\underset{\substack{\downarrow\\ \text{Don't have same}\\ \text{row or column}}}{\boxed{\prod_{i=1}^{n} x_{i,\pi(i)}}}}$$

Theorem 1.4.1

Given a specialized X we can compute $\det_n(X)$ in P (nearly quadratic time!).

Proof. Use Gaussian Elimination $\square$

Note:-

This does not give $\{\det_n\}_n \in VP$ because, the above algorithm uses division, if-then-else, permutation etc.

What is the analogue of NP (non-determinism)? Do a large sum

Definition 1.4.2: VNP (Valiant's NP)

Polynomial family $\{f_n\}_n \in VNP$ of

$$f_n(\overline{x}) = \underset{\substack{\overline{w}\in\{0,1\}^{t(n)} \\ \downarrow \\ \text{witness}}}{\sum} \underset{\substack{\downarrow \\ \text{verifier}}}{g(\overline{x},\overline{w})}$$

where

$$t(n) = poly(n), \qquad \{g_n\}_n \in VP$$

So a polynomial family $\{f_n\} \in VNP$ if f_n can be written as a sum over witnesses with verifier evaluated. Replace $\sum$ with $\vee$ (in the boolean world) the you can recover definition of NP right in the boolean case because $g =$ verifier algorithm, poly-time algorithm and it is just going over all possible certificates.

A standard problem in VNP is **Permanent**

$$\text{per}_n(X_{n\times n}) = \sum_{\pi\in Sym(n)} \prod_{i=1}^{n} x_{i,\pi(i)}$$

Question 4

Why Gaussian Elimination would fail on this?

Solution: Adding 2 rows or columns that does not change det but you don't know for per. In general it change.

□

Note:-

So per is the hardest problem, even harder than SAT in a way

Theorem 1.4.2

per $\in VNP$

$$\text{per}(X_{n\times n}) = \sum_{\pi\in Sym(n)} \prod_{i=1}^{n} x_{i,\pi(i)}$$

$\prod_{i=1}^{n} x_{i,\pi(i)}$ is computable in VP as this is a simple multiplication gate.

So we can use this as a verifier. But problem is as you look at different summands for different permutations, from the permutation your verifier has to compute this which is not clear.

$$\begin{tikzcd} \text{per}(X_{n\times n}) = \sum_{\pi\in Sym(n)} \boxed{\prod_{i=1}^{n} x_{i,\pi(i)}} \arrow[r, bend right, "\substack{\text{Think of } g \\ \text{that can compute this,} \\ \text{connect these two}}"'] & f_n = \sum_{\overline{w}\in\{0,1\}^{t(n)}} \boxed{g_{n+t}(\overline{x},\overline{w})} \end{tikzcd}$$

$\overline{w}$ is a just a string. So you have to make your permutation as a string. From that just using polynomial arithmetic it is not clear how to get to this $x_{i,\pi(i)}$

Observation:

(i) It is not direct. Does not follow from the definition

(ii) Look for a more complicated g instead of this monomial.

Proof. Let g be the polynomial that takes $X_{n\times n}$matrix and $\overline{v} \in \{0,1\}^n$ and compute

$$g_{n^2+n}(X,\overline{v}) := \underbrace{\boxed{\prod_{i=1}^{n}\left(\sum_{j=1}^{n} v_j X_{i,i}\right)}}_{\substack{\downarrow \\ \text{Produces} \\ \text{monomials}}} \underbrace{\boxed{\prod_{i=1}^{n}(2v_i - 1)}}_{\substack{\downarrow \\ \text{Sign} = \\ \text{depending on} \\ \text{how many 1's in} \\ \overline{v}\text{, the weight of} \\ \overline{v} \text{ will be sign}}}$$

Claim 1.4.1 Ryser's Formula [Rys63]

$$\sum_{\overline{v}\in\{0,1\}^n} g(X,\overline{v}) = \text{per}_n(X)$$

Proof. Rewrite LHS as

$$\sum_{T\subseteq[n]} \prod_{i=1}^{n}\left(\sum_{j\in T} x_{ij}(-1)^{n-|T|}\right)$$

You are selecting some elements of the row and adding them, that is the $\sum x_{ij}$ for $j \in T$. And then you are going over all the rows $i = 1$ to n with a sign and then you are taking the big sum over all subsets. Since this is straddling over all the rows, it is basically multiplying the variables that appear in distinct rows.

So it is supported on monomials $x_{1,i_1} \cdots x_{n,i_n} = m$, $r := \#$ distinct $\{i_1, i_2, \ldots, i_n\}$. We want to analyze the situation when this monomial does not correspond to a permutation which means $i_1, \ldots, i_n$ are not distinct. We want to show this particular monomial will get ultimately canceled in the big sum. In how many T's will this monomial be produced ? 2^{n-r} many subsets of T. The monomial m can be associated to 2^{n-r} many subsets T with sign $= \sum_{S\subseteq[n-r]} (-1)^{|S|} = 0$ as $r < n$. For $\pi \in sym(n)$, $\prod_{i=1}^{n} x_{i,\pi(i)}$ survives in LHS with sign 1. □

Now $g \in VP$ because it is just a simple product. You compute these inner products by addition gates then you multiply by a single multiplication gate. So it is clearly depth 2 arithmetic circuit.

So g is verifier and $\overline{v}$ is witness. So $\text{per}_n \in VNP$ □

Conjecture 1.4.1 Valiant's Conjecture

$VP \neq VNP$

Then in that case PER is $VNP - complete$. $PER \notin VP$

PER of a boolean (0/1) matrix, that represents a graph, is simply the number of perfect matching. To check if perfect matching exists has an algorithm. But to count them is harder than $NP - hard$ problems. This is equivalent to functional problem of $\#SAT$. (Valiant's $\#P - completeness$ of PER)

Definition 1.4.3: $\#SAT$

The problem of counting version of SAT. Formula satisfiability, boolean formula satisfiability. Given a boolean function how many satisfying assignments are true. For n variables it can be 0 to 2^n. Testing whether it is positive $NP - complete$ problem.

This is called the Valiant's $\#P$ completeness of permanent which is Boolean permanent. So Valiant showed that permanent, the Boolean permanent or computing the permanent of a Boolean matrix which corresponds to a graph. This is equivalent to a $\#SAT$. And $\#SAT$ defines as a class which is called $\#P$.

Theorem 1.4.3 [Bü00]

$VP = VNP \implies P/poly = NP/poly$. [This is much stronger than saying $P = NP$]

From this you can not really deduce whether $P = NP$ or P is different from NP. Because when you talk about P, NP classes you talk about turing machines. You are not talking about Boolean Circuits. $P/poly$ is actually the problems that solve using Boolean Circuits and $NP/poly$ is also defined in the similar way using Boolean Circuits. SO the meaning of efficiency in Boolean Circuits and meaning of non-determinism in Boolean Circuits — these two things are same — this is what the equality is saying.

Definition 1.4.4: NC^3

These are Boolean Circuits where depth is only $\log^3$

Note:-

Bürgisser also shows that $P/poly = NC^3/Poly$. So this exactly models very fast parallel algorithm. So on an input size of n the parallel time complexity is only $\log^3$

So if you show $VP = VNP$ then Bürgisser's proof actually tells you that $NP/poly = P/poly = NC^3$. So non-determinism is the same as efficient parallel algorithms. SO its is saying you can solve SAT in fast parallel time.

When you are working over characteristic zero then Bürgisser's proof requires the assumption of $GRH =$ *Generalized Reimann Hypothesis.*

Chapter 2

Determinant-ABP

Bibliography

[Bü00] Peter Bürgisser. Completeness and reduction in algebraic complexity theory. 7, 01 2000.

[Rys63] Herbert John Ryser. *Combinatorial mathematics*, volume 14 of *The Carus Mathematical Monographs*. American Mathematical Soceity, 1963.