Arithmetic Circuit Complexity: NPTEL Course – Nitin Saxena

Soham Chatterjee

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

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

- $P := \bigcup_{c>0} Dtime(n^c)$
- $PSpace := \bigcup_{c>0} Space(n^c)$
- $NP := \bigcup_{c>0} \underset{\substack{\text{on a non} \\ \text{-deterministic } TM}}{\underset{\downarrow}{Ntime(n^c)}}$
- $\mathbb{L} := Space(\log n)$
- $EXP :=$ { 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:

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

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

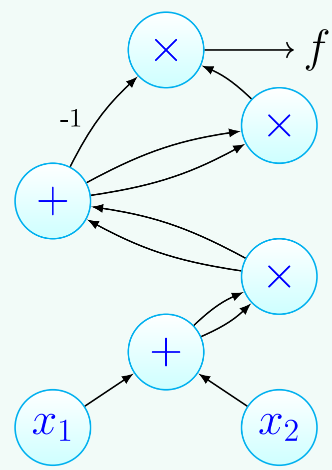
$\square$

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