CS760 Topics in Computational Complexity **2024-25 Sem I**

Lecture 13: 13-09-2024

Scribe: Ananth Krishna Kidambi, Soham Joshi *Lecturer: Rohit Gurjar*

1 Circuit lower bounds

From the previous lectures, we know that,

$$\mathrm{P} \subseteq \mathrm{P/Poly}$$

We also recall the Karp-Lipton theorem,

Theorem 13.1. *(Karp-Lipton) NP $\subseteq$ P/Poly $\implies$ PH $= \Sigma_2^p$*

It is not believed that the polynomial hierarchy collapses to Σ_2^p. Since P $\subseteq$ P/Poly, showing NP $\not\subseteq$ P/Poly implies P $\neq$ NP. So, one can try to show NP $\not\subseteq$ P/Poly by obtaining a language in NP which does not have a polynomial sized boolean circuit, i.e. we try to show a lower bound on the circuit size. However, the best known lower bound for an NP language is around $5n$.

2 Turing Machines with Advice

Definition 13.2. *A language L is in DTIME $(T(n)/a(n))$ if there exists a Turing machine M running in time $T(n)$ and a sequence of strings (A_n), $|A_n| = O(a(n))$ such that:*

$$x \in L \iff M(x, A_{|x|}) = 1$$

In the above definition, $A_{|x|}$ is the advice given to the turing machine for input x.

We can define P/Poly as a special case of TMs with advice, where the Turing machine runs in poly-time and the advice also has polynomial length ($T(n) = a(n) = \mathrm{poly}(n)$). Note that this definition is equivalent to the previous definition of P/Poly (defined using boolean circuits). The backward direction is true since we can provide an encoding of the boolean circuit as advice to a TM which evaluates it in time polynomial in the size of the circuit, which in turn is polynomial in the size of the input. For the forward direction, we can directly encode the TM and the advice into a boolean circuit, which has a polynomial-size increase.

3 Uniform Circuits

Definition 13.3. *A family of circuits is called logspace-uniform if, given 1^n, we can compute C_n in log-space.*

Theorem 13.4. *$L \in P$ iff there is a logspace-uniform family of circuits accepting L.*

The only if direction is easy to show - construct a Turing machine that can construct the Boolean circuit C_n. Since this can be done in log-space, and $L \subseteq P$, this runs in polynomial time. Also, this circuit can be evaluated in polynomial time, and hence, the TM runs in poly-time.

The proof of the other direction is more involved. You need to argue the computation of a polytime Turing machine can be represented by a circuit computable in logspace. Refer to [AB09] for the proof sketch.

4 Bounded depth circuits

Why should we believe that it is possible to show exponential circuit size lower bounds? The following is an existential argument, that shows almost all Boolean functions require exponential size circuits. The challenge is to show an explicit function that requires large size circuits.

Lemma 13.5. *There is a Boolean function $f : \{0,1\}^n \to \{0,1\}$ which requires $\Omega(\frac{2^n}{n})$ sized circuits.*

Proof. We will proceed via a counting argument. The number of Boolean functions is 2^{2^n}. The number of circuits with size s can be bounded by number of DAGs with s nodes (and hence $O(s)$ edges). If we represent the circuit by adjacency list, number of edges is $O(s)$ and vertex needs $\log s$ bits for representation. Hence, number of circuits of size s is given by $2^{O(s \log s)}$. Now, let $s = c \cdot \frac{2^n}{n}$, then number of circuits is bounded above by 2^{2^n}. □

Given a Boolean function, we can encode it trivially using a DNF. We can take the disjunction of all the configurations of $(x_1, \cdots, x_n)$ which output 1. For example $(x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3)$. The number of possible inputs is 2^n, and number of gates per clause of DNF is n. Hence, total size of the circuit is $O(n \cdot 2^n)$.

Remarkably, there is better upper bound, showing that above lower bound for representing Boolean functions given in lemma 13.5 is actually tight.

Lemma 13.6. *For every Boolean function there is a circuit of size $O(\frac{2^n}{n})$.*

The proof for this lemma was given by Lupanov in 1958. A proof can be found in [Nit19].

This raises further questions such as which Boolean functions require large circuits. Namely do there exist languages in NP or EXP which require exponential sized circuits?

Theorem 13.7. *(Karp-Lipton) NP $\subseteq$ P/Poly $\implies$ PH $= \Sigma_2^p$*

Theorem 13.8. *(Meyer) EXP $\subseteq$ P/Poly $\implies$ EXP $= \Sigma_2^p$*

The above theorems show that if all languages in NP or EXP require only polynomially sized circuits then PH, EXP classes collapse to Σ_2^p which is not believed to be true.

Since there is not much progress in showing lower bounds for general circuits, we can analyze circuits with limitation in compute power. Namely, we will analyze circuits with bounded depth.

4.1 AC, NC classes

The analysis of circuits was a field on interest in 80s because it was a model of parallel computation, with the depth of the circuit a measure of running time of the parallel computer in some sense. We define bounded depth classes of Boolean circuits as follows,

- NC^i : Class of problems which have Boolean circuits of polynomial size and $O(\log^i n)$ depth with a fan-in of atmost 2 for all gates.
- AC^i : Class of problems which have Boolean circuits of polynomial size and $O(\log^i n)$ depth and arbitrary fan-in

From the above definitions, it is clear that AC^i is at least as expressive as class NC^i. Moreover, any gate with fan-in of atmost m can be converted into a circuit with a fan-in of 2 and depth $\log m$ using a binary tree-like construction of gates. Hence, we get that

$$\mathrm{NC}^0 \subseteq \mathrm{AC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{AC}^1 \subseteq \cdots$$

Further, we can define complexity classes,

$$\mathrm{NC} := \bigcup_i \mathrm{NC}^i$$

$$\mathrm{AC} := \bigcup_i \mathrm{AC}^i$$

We know that parity is not in AC^0 (discussion ahead), however if given the power of xor gates, parity will have depth 1 circuit. This motivates us to define the class

- $\mathrm{AC}^i[k]$: Class of problems which have Boolean circuits with additional mod k gate of polynomial size and $O(\log^i n)$ depth, arbitrary fan-in.

So, parity is in $\mathrm{AC}^0[2]$. These classes can be further combined to get a class with any mod gate given by,

$$\mathrm{ACC}^i := \bigcup_j \mathrm{AC}^i[j]$$

With respect to the expressive power of these classes, Ryan Williams gave an important result,

Theorem 13.9. *([Wil11]) NEXP $\not\subseteq$ ACC^0.*

4.2 Problems in AC, NC classes

- *Addition* $\in AC^0$: This can be shown by directly constructing a circuit of constant depth for adding 2 numbers. Note that to find the ith bit of the sum, we only need the ith bits of the two input numbers and the carry bit of the sum of the last $i-1$ bits of those numbers. So if we can find the carry in constant depth, we are done.

  Let the two numbers be $x = x_n x_{n-1} \dots x_1$ and $y = y_n y_{n-1} \dots y_1$, where x_i, y_i are the bits of x, y. Note that the carry bit at the ith position is 1 iff

  $$\exists_{i>j\geq 0} : \left(\forall_{i>k>j} x_k \vee y_k\right) \wedge x_j \wedge y_j$$

  Since this formula can be expressed using a Boolean circuit of depth 4, we can find the carry bit in constant depth, and hence add with a Boolean circuit of constant depth.

- *Parity* $\notin AC^0$: Refer to [Nic18] for the proof. Will prove in later lectures.

- *Parity* $\in \mathrm{NC}^1$: Since parity is simply the *xor* of all bits, we can compute it in a binary tree fashion of depth $\log n$.

- *Multiplication* $\in AC^0$: This can be done by multiplying each bit of the second number with the first number and adding them in a binary tree fashion (having $\log n$ depth). Since addition is in AC^0, this circuit has depth $O(\log n)$.

  We can rule out an AC^0 circuit for multiplication as follows.

- *Multiplication* $\in AC^0 \implies$ *Parity* $\in AC^0$: Let there be a constant depth circuit for multiplication of two numbers. Then given bits $x_1, x_2, \cdots, x_n$, consider the product of $x_1 0^{\log n} x_2 \cdots 0^{\log n} x_n$ and $10^{\log n}1 \cdots 0^{\log n}1$. Using the high-school multiplication method, this will give us the addition of $x_1, x_2, \cdots, x_n$ in the column corresponding to x_1 in the input. Moreover, addition of n bits in a column needs a carry over in atmost $\log n$ columns, hence the column corresponding to x_1 does not get a carry bit from any other column. Hence the output bit corresponding to the column of x_1 will give the parity of $x_1, x_2, \cdots, x_n$.

We will state some more results without proof.

- Multiplication $\in \mathrm{NC}^1$
- Matrix Multiplication $\in \mathrm{NC}^1$
- System of Linear Equations $\in \mathrm{NC}^2$
- Determinant, Rank $\in \mathrm{NC}^2$
- $\mathrm{NC}^1 \subseteq L \subseteq NL \subseteq AC^1 \subseteq NC^2$: The proof for NL $\subseteq \mathrm{AC}^1$ is given in [CK02].

An important open question is whether bipartite matching is in NC. This is one of most important problems in context of the question whether NC = P. If true, it will mean that any problem with efficient algorithm can be parallelized with a significant speed-up.

5 P-complete

Definition 13.10. *A problem Q is P-complete if Q is in P and every problem in P logspace-reduces to Q.*

5.1 Examples of P complete problems

1. *Circuit evaluation*: Given a circuit C (in a graph form) and an input x for it, the problem asks for finding $C(x)$. P-completeness directly follows from 13.4. This is because the theorem says that for any polynomial time TM and a given input length, we can compute a circuit for it in logspace. Note that even though the size of the circuit may not be logarithmic in the input size, we can compute the ith bit of the circuit in logspace (and we can compute the size of the circuit in logspace).

2. *Min cost flow*

3. *Linear programming*

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[CK02] Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. 01 2002.

[Nic18] Nicholas Shiftan. Parity not in AC^0, 2018. `https://people.seas.harvard.edu/~salil/cs221/fall02/scribenotes/nov18.pdf`.

[Nit19] Nitin Saurabh. Boolean function complexity, 2019. `https://nitinsau.github.io/teaching/BFC19-mpii/lecture2.pdf`.

[Wil11] Ryan Williams. Non-uniform acc circuit lower bounds. In *2011 IEEE 26th Annual Conference on Computational Complexity*, pages 115–125, 2011.