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

Lecture 21: 22-10-2024

Scribe: Priyanshu Singh *Lecturer: Rohit Gurjar*

1 Pseudorandom Generators and Circuit Lower Bounds

One possible way to show that BPP = P is by demonstrating the existence of a $2^{l/900}$-PRG (pseudorandom generator). In other words, we need to establish certain circuit lower bounds to prove that BPP = P. There may be other methods to show this, but PRGs provide a particularly elegant approach.

1.1 Theorem: Existence of $O(\log n)$-bit Seed PRG Implies Circuit Lower Bounds

Theorem: If a pseudorandom generator (PRG) with a seed length of $O(\log n)$ exists, then there exists a function $f \in E$ (the class of problems solvable in exponential time) that cannot be computed by any circuit of size n^k, for some constant k.

1.2 Proof

Step 1: PRG Construction and Seed Length

Assume there exists a pseudorandom generator G that takes a seed of length $O(\log n)$ and stretches it into a longer sequence that is indistinguishable from true randomness by any polynomial-sized circuit of size n^k.

Step 2: Function $f \in E$

We aim to construct a function $f \in E$ that has a circuit lower bound, meaning no polynomial-sized circuit can compute f on all inputs.

Assume towards a contradiction that such a function f can be computed by circuits of size n^k.

Step 3: Contradiction from Circuit Efficiency

If a function $f \in E$ can be computed by circuits of size n^k, then we could simulate the outputs of the PRG G using these circuits, effectively distinguishing the pseudorandom output from true random bits. This would contradict the assumption that G is a PRG that fools all polynomial-sized circuits.

Step 4: Conclusion

Hence, the existence of an efficient $O(\log n)$-bit seed PRG implies that there must be functions in E that cannot be computed by polynomial-sized circuits, establishing a circuit lower bound.

This completes the proof.

2 PRG Against Restricted Computation Models

2.1 Problem

Question 1: Can you design a pseudorandom generator $G : \{0,1\}^{\log n} \to \{0,1\}^n$ such that for all subsets $S \subseteq \{1, 2, \ldots, n\}$,

$$\Pr_{x \leftarrow U_n}\left[\bigoplus_{i \in S} x_i = 1\right] \approx \Pr_{y \leftarrow G(U_l)}\left[\bigoplus_{i \in S} y_i = 1\right],$$

where U_n is the uniform distribution over $\{0,1\}^n$, U_l is the uniform distribution over $\{0,1\}^{\log n}$, and $\oplus$ denotes the XOR operation?

This question has been solved using the concept of ϵ-biased spaces.

2.2 Solution: ϵ-Biased Spaces

An ϵ-biased space is a set of binary strings such that for any subset $S \subseteq \{1, 2, \ldots, n\}$, the XOR of the elements in S behaves like the XOR over truly random bits, with a bias of at most ϵ.

Thus, we can construct a pseudorandom generator $G : \{0,1\}^{\log n} \to \{0,1\}^n$ such that:

$$\left| \Pr_{y \leftarrow G(U_l)} \left[\bigoplus_{i \in S} y_i = 1 \right] - \Pr_{x \leftarrow U_n} \left[\bigoplus_{i \in S} x_i = 1 \right] \right| \leq \epsilon,$$

where ϵ is a small bias that can be made arbitrarily small.

2.3 How the PRG Works

The pseudorandom generator G takes a seed of length $O(\log n)$ and outputs n bits. The construction ensures that the XOR over any subset of the output bits behaves similarly to the XOR over truly random bits, with a bias controlled by ϵ.

The construction of ϵ-biased spaces uses algebraic techniques and finite fields to efficiently produce these biased spaces, allowing the PRG to fool XOR-based computations on subsets of bits.

2.4 Question 2: Replacing XOR with DNF and SAT Approximation

In Question 1, we considered the problem of fooling XOR functions over subsets of bits. Now, let's replace XOR with *Disjunctive Normal Form* (DNF) formulas. In this case, the goal is to construct a pseudorandom generator that can fool any DNF formula.

A DNF formula consists of a conjunction (AND) of disjunctions (ORs) of literals. The challenge here is to generate pseudorandom bits that fool these DNF formulas so that the PRG-generated bits are indistinguishable from random bits when evaluated by any DNF formula.

Furthermore, the approximate number of solutions to the *Satisfiability* (SAT) problem can be reduced to this problem of fooling DNF formulas. In this case, we seek to construct a PRG with seed length $O(\log^2 n)$.

2.5 Question 3: Constant Depth Circuits Fooling

Fooling Constant Depth Circuits: Constant-depth circuits are Boolean circuits where the depth (i.e., the number of layers of gates) is fixed, regardless of the size of the input. These circuits (often denoted as AC^0) are highly structured, consisting of AND, OR, and NOT gates.

The goal is to construct a PRG that can fool these constant-depth circuits. This means that the output of the pseudorandom generator should be indistinguishable from true randomness when processed by such circuits.

For a PRG to fool constant-depth circuits, it needs to generate pseudorandom outputs that pass all tests that a constant-depth circuit can apply. Techniques like the Nisan-Wigderson PRG or ϵ-biased generators are often used to accomplish this.

2.6 Question 4: RL =? L (Randomized Logspace)

Open Problem: Is $RL = L$, where RL stands for *Randomized Logspace* and L stands for *Deterministic Logspace*?

This is an open problem in computational complexity. The problem asks whether randomized logspace algorithms (algorithms that use logarithmic space and randomness) can be simulated by deterministic logspace algorithms.

2.7 Current State of the Problem

Currently, it is known that there exist pseudorandom generators (PRGs) with seed length $O(\log^2 n)$ that can fool any computation in L (deterministic logspace).

These PRGs have the following properties:

- **Time Complexity:** $O(n \log n)$, meaning the PRG can be computed in time proportional to $n \log n$.
- **Space Complexity:** $O(\log^n)$, meaning the PRG uses logarithmic space with respect to the input size n.

The existence of such PRGs provides evidence that RL may indeed equal L, but a full resolution of the question remains open.

3 K-Round Interactive Proof

In a k-round interactive protocol, we have two functions f and g, both of which are polynomial-time computable. These functions govern the sequence of messages exchanged between the prover and the verifier in each round of the protocol. The messages are computed as follows:

- The first message is computed by g, depending only on the input x:
$$a_1 = g(x).$$
- The second message is computed by f, which takes as input the previous message a_1 and the input x:
$$a_2 = f(a_1, x).$$
- The third message is computed by g, which takes as input the previous two messages a_2, a_1, and the input x:
$$a_3 = g(a_2, a_1, x).$$
- This process alternates between f and g for k rounds. If k is odd, the last function applied is g; if k is even, the last function applied is f.

After k rounds, the final output of the protocol is computed as:

$$\text{Out}_{f,g}^{(k)}(x) = g(a_k, a_{k-1}, \ldots, a_1, x),$$

where a_k is the message computed in the k-th round.

The exact structure of the sequence depends on whether k is even or odd, as the functions alternate between f and g during the interactive proof.

4 IP[K] and Interactive Proof Systems

4.1 Definition of IP[K]

A language L is said to be in the class $IP[K]$ if there exists a probabilistic polynomial-time verifier V such that:

- **Completeness:** If $x \in L$, there exists a prover P such that the probability that the verifier accepts is at least $\frac{2}{3}$:
$$\Pr[\text{Out}_{f,g}^{(k)}(x) = 1] \geq \frac{2}{3},$$
where k is the number of rounds in the interactive protocol.
- **Soundness:** If $x \notin L$, for every prover P, the probability that the verifier accepts is at most $\frac{1}{3}$:
$$\Pr[\text{Out}_{f,g}^{(k)}(x) = 1] \leq \frac{1}{3}.$$

Here, the number of rounds k is fixed, and the functions f and g are used to generate the interactions between the prover and the verifier, as discussed in the previous sections.

4.2 Definition of IP

The complexity class IP (Interactive Polynomial Time) is defined as the union of all classes $IP[n^c]$, where $c \geq 1$. This means that the number of rounds in the interaction can grow as a polynomial function of the input size n.

Formally, we define:

$$IP = \bigcup_{c \geq 1} IP[n^c],$$

where n is the size of the input. This implies that IP includes interactive proofs with a number of rounds that can be any polynomial function of the input size.

4.3 Notes on NP, IP, and DIP

Note:

- **NP is a subset of IP:** The class NP (nondeterministic polynomial time) is a subset of IP (interactive polynomial time). In NP, a verifier can be thought of as interacting with a prover in one round, where the prover provides a proof and the verifier checks it deterministically.
- **DIP = NP:** If the verifier in an interactive proof is deterministic, the class is called DIP (Deterministic Interactive Proof). In this case, $DIP = NP$. This is because in NP, the verifier only needs to verify the correctness of a proof deterministically in polynomial time without probabilistic checks or randomness.
- **Randomized Prover Still Yields IP:** If the prover in the interactive proof is randomized (instead of being deterministic or computationally unbounded), the interactive protocol remains within the class IP. This is because the class IP allows both the verifier and the prover to use randomness.
- **Completeness Error Can Be Made Zero:** In an interactive proof system, the completeness error (i.e., the probability that a correct proof is incorrectly rejected) can be reduced to zero. This can be achieved by repeating the protocol or amplifying it so that the verifier always accepts valid proofs.

5 CoNP vs IP and Arithmetization of CNF

5.1 IP is a Subset of PSPACE

The class IP (Interactive Polynomial time) is a subset of $PSPACE$ (Polynomial Space). This is because any interaction in IP using a probabilistic polynomial-time verifier can be simulated deterministically using polynomial space.

5.2 CoNP vs IP

One of the significant results is that $UNSAT$ (which is $CoNP$-complete) belongs to IP. This shows that $CoNP \subseteq IP$, which means that interactive proof systems are more powerful than previously thought.

5.3 Arithmetization of a CNF

Lets take an example of arithmetization of a CNF formula:

$$\Phi(x_1, x_2, x_3, x_4) = (x_1 \vee \neg x_2) \wedge (x_3 \vee \neg x_4 \vee x_2).$$

We arithmetize this formula as follows:

$$P_\Phi(y_1, y_2, y_3, y_4) = [1 - (1 - y_1)(y_2)][1 - (1 - y_3)(y_4)(1 - y_2)].$$

For all $b \in \{0, 1\}^n$, we have:

$$\phi(b) = P_\Phi(b).$$

5.4 Counting Satisfying Assignments

Let K be the number of assignments that satisfy Φ:

$$K = \sum_{(b_1,b_2,\ldots,b_n)\in\{0,1\}^n} P_\Phi(b_1, b_2, \ldots, b_n).$$

We choose a prime $p \geq 2^{2n}$ and do all computations modulo p.

5.5 Interactive Proof for K

The prover sends K and a polynomial $S(y_1)$, where:

$$S(y_1) = \sum_{(b_2,\ldots,b_n)\in\{0,1\}^{n-1}} P_\Phi(y_1, b_2, \ldots, b_n).$$

The verifier performs the following checks:

- Verify that $K = S(0) + S(1)$.
- Check that the degree of $S(y_1)$ is equal to the degree of y_1 in P_Φ.
- Send a random $\alpha \in \{0, \ldots, p-1\}$ and ask the prover to compute $S(\alpha)$.

The verifier recursively checks the validity of the provers claims.

5.6 Soundness and Completeness

Completeness: The completeness error can be made zero in this protocol if the prover is honest.

Soundness: The soundness error is $\frac{k \cdot d}{p}$, where k is the number of rounds and d is the degree of P_Φ, ensuring a small probability of the verifier being fooled by a dishonest prover.