CS 6810 Theory of Computing March 12, 2026

Lecture 14: March 12, 2026

Instructor: Eshan Chattopadhyay *Scribe: Piyali Mittal*

1 Randomized Algorithms and the Class BPP

In this lecture we continue studying randomized algorithms in complexity theory, focusing in particular on the complexity class BPP. We use that BPP captures languages that can be decided by probabilistic polynomial-time algorithms with bounded error.

1.1 Definition of BPP and Error Amplification

Definition 1.1 (BPP)**.** A language L is in BPP if there exists a probabilistic polynomial-time Turing machine M such that for every input x,

$$\Pr_r[M(x,r) = L(x)] \geq \frac{2}{3},$$

where r denotes the random bits used by the machine.

The constant $\frac{2}{3}$ in the definition is not particularly important. The key requirement is that the algorithm's success probability is *bounded away from* $1/2$. The value $\frac{2}{3}$ is simply a convenient constant that is traditionally used. More generally, suppose the algorithm succeeds with probability $\frac{1}{2} + \delta$ for some $\delta > 0$. As long as δ is not too small (for example, $\delta \geq 1/\text{poly}(n)$), we can amplify the success probability by repeating the algorithm independently several times and taking the majority vote of the answers. This process is known as **error amplification**.

If we run the algorithm multiple times and take a majority vote, the error probability decreases exponentially. By repeating the algorithm a polynomial number of times, we can reduce the error probability to something extremely small, such as $\varepsilon = 2^{-\text{poly}(n)}$. Importantly, this amplification only increases the running time polynomially, since the number of repetitions needed is polynomial in the input length.

However, it is crucial that δ is not *too small*. If the advantage over $\frac{1}{2}$ were exponentially small, such as $\frac{1}{2} + 2^{-n}$, then amplification would require exponentially many repetitions, which would no longer be efficient. This phenomenon explains one of the difficulties in the complexity class PP, where the definition only requires the algorithm to succeed with probability strictly greater than $\frac{1}{2}$, but the advantage over $\frac{1}{2}$ could be extremely tiny.

Because error amplification is always available when δ is at least inverse polynomial, throughout the rest of the lecture we will often assume that the error probability has already been reduced to an extremely small value such as $\varepsilon = 2^{-\text{poly}(n)}$. This stronger guarantee simplifies later arguments.

1.2 Strengthening the Error Bound

Using amplification, we may assume that the error probability of our BPP algorithm is extremely small. Concretely, suppose x is an input of length n. After amplification we can assume that

$$\Pr_r[M(x,r) = L(x)] \geq 1 - \varepsilon$$

where $\varepsilon = 2^{-(n+1)}$. In other words, the algorithm fails with probability at most $2^{-(n+1)}$.

Note that the amplification procedure increases the number of random bits used by the algorithm. If the original algorithm used ℓ random bits, the amplified algorithm may use many more random bits because we repeat the algorithm multiple times independently. However, the total number of random bits used remains polynomial in n.

2 Adleman's Theorem: BPP $\subseteq$ P/poly

We now prove a remarkable theorem due to Adleman from the late 1970s.

Theorem 2.1 (Adleman)**.**

$$\text{BPP} \subseteq \text{P/poly}.$$

This theorem states that any randomized polynomial-time algorithm can be simulated by a deterministic polynomial-size circuit family. In other words, if we allow **non-uniform advice**, randomness can be eliminated. Note that it is widely believed that P = BPP, meaning randomness does not add computational power even without advice, but this remains unproven. Adleman's theorem is weaker: it shows that randomness can be removed if we allow non-uniformity.

Let $L \in$ BPP and let M be the corresponding probabilistic polynomial-time Turing machine deciding L. From the error amplification discussion in the previous section, we may assume that the algorithm has extremely small error probability. In particular, for every input x of length n,

$$\Pr_r[M(x,r) = L(x)] \geq 1 - \varepsilon$$

where $\varepsilon = 2^{-(n+1)}$. Here $r \in \{0,1\}^\ell$ denotes the random bits used by the algorithm.

The key idea of the proof is to show that randomness can be replaced by a single carefully chosen string of random bits. Suppose, for the moment, that there existed a particular string $r^* \in \{0,1\}^\ell$ such that $M(x, r^*) = L(x)$ for *every* input x of length n. If such a string existed, then we would not need randomness at all. Instead, we could simply hardcode this string as **advice**. The circuit for inputs of length n would simulate the deterministic computation $M(x, r^*)$ using this fixed string of random bits. Because r^* depends only on the input length and not on the specific input x, it can serve as the advice string for all inputs of that length.

Of course, it is not immediately obvious that such a universal random string exists. To analyze this, we define the set of *bad random strings*. For each input x, define

$$\text{Bad}(x) = \{r \in \{0,1\}^\ell : M(x,r) \neq L(x)\}.$$

This set consists of the random strings on which the algorithm makes a mistake when run on input x. Because the algorithm fails with probability at most ε, the size of this set is bounded by $|\text{Bad}(x)| \leq \varepsilon 2^\ell$. Thus the density of the bad set inside the ambient randomness space $\{0,1\}^\ell$ is at most ε.

Now observe that there are 2^n possible inputs of length n. Consider the union of all bad sets,

$$\bigcup_{x\in\{0,1\}^n} \mathrm{Bad}(x).$$

Using the union bound, we obtain

$$\left|\bigcup_x \mathrm{Bad}(x)\right| \le \sum_x |\mathrm{Bad}(x)| \le 2^n \cdot \varepsilon 2^\ell.$$

Dividing both sides by 2^ℓ gives

$$\sum_x \frac{|\mathrm{Bad}(x)|}{2^\ell} \le 2^n\varepsilon.$$

Recall that $\varepsilon = 2^{-(n+1)}$, so

$$2^n\varepsilon = \frac{1}{2}.$$

Therefore the union of all bad sets occupies at most half of the randomness space. In particular, these bad sets cannot cover the entire space $\{0,1\}^\ell$. This implies that there exists at least one random string

$$r^* \notin \bigcup_x \mathrm{Bad}(x).$$

By definition, this means that r^* is not bad for any input x. Equivalently, the machine produces the correct answer for every input when its random bits are fixed to r^*.

We can now construct the desired circuit family. For inputs of length n, the circuit receives r^* as an advice string and simulates the computation $M(x, r^*)$. Since M runs in polynomial time and the advice string has polynomial length, the resulting circuit has polynomial size.

Thus we conclude that every language in BPP has a polynomial size circuit family, and therefore

$$\mathrm{BPP} \subseteq \mathrm{P/poly}.$$

3 BPP in the Polynomial Hierarchy

We now prove that the class BPP lies within the polynomial hierarchy. This result, due to Gács, Lautemann, and Sipser, provides an important structural characterization of randomized computation and shows that randomized polynomial-time algorithms can be simulated using a small number of alternating quantifiers.

Theorem 3.1 (Gács–Lautemann–Sipser)**.**

$$\mathrm{BPP} \subseteq \Sigma_2 \cap \Pi_2.$$

In this lecture we prove the containment $\mathrm{BPP} \subseteq \Sigma_2$. The containment in Π_2 follows immediately because BPP is closed under complement.

3.1 Setup and Proof Intuition

Let $L \in \text{BPP}$ and let M be the probabilistic polynomial-time Turing machine deciding L. As in the previous section, we assume that the algorithm has extremely small error probability obtained through error amplification. Concretely, for every input x we assume

$$\Pr_r[M(x,r) = L(x)] \geq 1 - \varepsilon$$

where $\varepsilon = 2^{-g}$ for some parameter g that we will choose later. Here $r \in \{0,1\}^\ell$ denotes the random bits used by the algorithm.

To analyze the behavior of the algorithm, it is convenient to examine the set of random strings that cause the machine to output 1. For each input x, define

$$1_x = \{r \in \{0,1\}^\ell : M(x,r) = 1\}.$$

Thus 1_x consists precisely of those random strings for which the algorithm accepts the input x.

The size of this set depends strongly on whether x belongs to the language. If $x \in L$, the correct output is 1, and the algorithm produces this answer with probability at least $1 - \varepsilon$. This means that

$$|1_x| \geq (1 - \varepsilon)2^\ell,$$

so the set 1_x occupies almost the entire randomness space $\{0,1\}^\ell$. On the other hand, if $x \notin L$, the correct answer is 0, and the algorithm outputs 1 only when it makes an error. Since the error probability is at most ε, we obtain $|1_x| \leq \varepsilon 2^\ell$. In this case the set 1_x is extremely small. Therefore deciding whether $x \in L$ can be reformulated as a question about the size of the set 1_x. Either the set occupies almost the entire randomness space or it occupies only a tiny fraction of it. The central idea of the proof is that these two situations behave very differently when we consider *translations* of the set.

3.2 Translates of Sets

To formalize this idea, we view the space $\{0,1\}^\ell$ as a vector space over $\mathbb{F}_2$, where addition corresponds to bitwise XOR. This allows us to shift sets by adding vectors.

Definition 3.1. For a set $S \subseteq \{0,1\}^\ell$ and a vector $y \in \{0,1\}^\ell$, define the translate of S by y as

$$S + y = \{s + y : s \in S\},$$

where addition is performed bitwise modulo 2.

Intuitively, this operation shifts the set S within the ambient space $\{0,1\}^\ell$. The key observation is that translating a set does not change its size. If S contains many elements, then any translate of S also contains the same number of elements.

The behavior of translates differs dramatically depending on whether a set is dense or sparse. If a set occupies almost the entire space, then a small number of its translates will cover the entire space. By contrast, if a set is very small, then even many translates will fail to cover the space.

This distinction will allow us to separate the two cases corresponding to $x \in L$ and $x \notin L$.

3.3 A Covering Lemma for Many Sets

To complete the proof we need a stronger version of the translate argument. In the previous discussion we considered a single dense set. However, in our application we must simultaneously handle many sets, one for each possible input of length n. Since there are 2^n possible inputs, we must ensure that the same collection of translates works for all of them at once. This leads to the following covering lemma.

Lemma 3.1. Let $\{S_i\}_{i=1}^N$ be subsets of $\{0,1\}^\ell$ where $N \leq 2^n$. Suppose each set satisfies

$$|S_i| \geq (1-\varepsilon)2^\ell$$

where $\varepsilon = 2^{-g}$. We will take $k = O(n)$, and assume that $g \geq O(\ell/n)$. Then there exist vectors $y_1, \ldots, y_k \in \{0,1\}^\ell$ such that for every $i \in \{1, \ldots, N\}$,

$$\bigcup_{j=1}^{k} (S_i + y_j) = \{0,1\}^\ell.$$

In other words, a single collection of shifts $y_1, \ldots, y_k$ covers the entire space simultaneously for every set S_i.

Proof. The proof uses the probabilistic method. We select the vectors $y_1, \ldots, y_k$ independently and uniformly at random from $\{0,1\}^\ell$ and show that with positive probability they satisfy the desired property.

Fix a particular index $i \in \{1, \ldots, N\}$ and consider the set S_i. Because $|S_i| \geq (1-\varepsilon)2^\ell$, the complement of S_i contains at most $\varepsilon 2^\ell$ elements. Now fix any string $z \in \{0,1\}^\ell$. The string z fails to be covered by the translates of S_i if $z \notin S_i + y_j$ for all $j = 1, \ldots, k$. Equivalently, $y_j \notin S_i + z$ for all j.

Since $|S_i + z| = |S_i|$, the set $S_i + z$ also has size at least $(1-\varepsilon)2^\ell$. Therefore a randomly chosen y_j lies outside this set with probability at most ε.

Because the vectors $y_1, \ldots, y_k$ are chosen independently, the probability that all of them lie outside $S_i + z$ is at most ε^k. Thus the probability that a fixed pair (i, z) is not covered is at most ε^k.

We now take a union bound over all possible pairs (i, z). There are N choices for i and 2^ℓ choices for z, so the total probability that some pair (i, z) remains uncovered is at most $N \cdot 2^\ell \cdot \varepsilon^k$. Recall that $N \leq 2^n$ and $\varepsilon = 2^{-g}$. Therefore

$$N \cdot 2^\ell \cdot \varepsilon^k \leq 2^n \cdot 2^\ell \cdot 2^{-gk}.$$

By choosing $k = O(n)$ and assuming $g \geq O(\ell/n)$, this quantity becomes strictly less than 1. Hence with positive probability the randomly chosen shifts cover every pair (i, z) simultaneously. Observe that the covering property can be expressed as a Σ_2 predicate: we existentially guess $y_1, \ldots, y_k$, and then for all $z \in \{0,1\}^\ell$ we check in polynomial time whether there exists a $j \in [k]$ such that $z \in S_i + y_j$. While this may initially appear to be a Σ_3 predicate due to the additional existential quantifier over j, this check can be performed efficiently within the universal quantifier, and hence the overall predicate is Σ_2. Therefore such vectors $y_1, \ldots, y_k$ must exist.

3.4 Completing the Proof of the Theorem

We now return to the sets 1_x defined earlier. If $x \in L$, then 1_x is extremely dense. Applying the covering lemma to this set implies that there exist vectors $y_1, \ldots, y_k$ such that

$$\bigcup_{j=1}^{k} (1_x + y_j) = \{0,1\}^\ell.$$

This means that for every string $z \in \{0,1\}^\ell$ there exists an index j such that $z \in 1_x + y_j$. Equivalently, $z + y_j \in 1_x$. By definition of 1_x, this means that the machine accepts when run on random string $z + y_j$, so $M(x, z + y_j) = 1$. Thus if $x \in L$, there exist vectors $y_1, \ldots, y_k$ such that

$$\forall z \in \{0,1\}^\ell,\ \exists j \in [k] : M(x, z + y_j) = 1.$$

Now consider the case where $x \notin L$. In this case the set 1_x is extremely small. Even after taking k translates, the union of these sets remains small and cannot cover the entire space. Therefore there exists some string z such that

$$M(x, z + y_j) = 0 \quad \text{for all } j.$$

Combining these two observations yields the characterization

$$x \in L \iff \exists y_1, \ldots, y_k\ \forall z\ \exists j :\ M(x, z + y_j) = 1.$$

This is a Σ_2 predicate, which shows that $L \in \Sigma_2$.

Since BPP is closed under complement, the same reasoning also places L in Π_2. Therefore

$$\text{BPP} \subseteq \Sigma_2 \cap \Pi_2.$$