CS 710: Complexity Theory 2/25/2010

Lecture 12: Randomness Continued

Instructor: Dieter van Melkebeek Scribe: Beth Skubak & Nathan Collins

In the last lecture we introduced randomized computation in terms of machines that have access to a source of random bits and that return correct answers more than $\frac{1}{2}$ of the time, and discussed several problems where algorithms involving randomization are useful. Today, we will study more closely the complexity classes related to randomized machines, and show how they are related to previous classes we have studied, including deterministic and non-deterministic classes. We also will show a relationship to non-uniform complexity classes by proving that BPP has polynomial-size circuits. In fact, the conjecture in the community is that BPP = P, though currently, all that is known is that BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$, as we will prove today.

1 Model

We model randomness by equipping a standard Turing machine with a random bit tape, to which the finite control has one-way read access. We assume this tape is filled with random bits from a perfectly uniform distribution before computation begins. (The plausibility of true randomness in this assumption will be discussed in later lectures.)

The machine is given one-way read-only access because the random bit tape is not counted as used space. To reuse a random value it must be stored on some work tape, while using a random value takes a single computational step to read an entry from the random bit tape.

1.1 Time and Space

The complexity classes BPTIME(t) and BPSPACE(s) (bounded-error probabilistic time/space) are analogous to their deterministic counterparts. They represent the classes of problems solvable by a randomized machine with 2-sided error under some time or space bound, with the requirement that:

$$\Pr[\text{error}] \leq \frac{1}{3}.$$

This bound is standard but somewhat arbitrary; recall that any error bounded below and away from $\frac{1}{2}$ can be made small by running the machine multiple times and taking the majority vote. In particular, we will consider the classes BPP (bounded-error probabilistic polynomial time) and BPL (log space).

Complexity classes also exist for 1-sided and 0-sided machines. The classes RTIME(t) and RSPACE(s), and the specific classes R (or RP) and RL, refer to problems solvable by randomized machines with 1-sided error, i.e., which meet the following criteria:

$$\Pr[\text{error} | x \in L] \leq \frac{1}{2}$$
$$\Pr[\text{error} | x \in \bar{L}] = 0.$$

Error in machines in 1-sided machines can be reduced by running the machine a fixed number of times and accepting if a positive result is ever given, since the machine never returns false positives.

The classes ZPTIME(t) and ZPSPACE(s), and specifically the classes ZPP and ZPL, refer to problems solvable by 0-sided randomized machines. These machines must meet these criteria:

$$\Pr[\text{error}] = 0$$
$$\Pr[\text{“Unknown”}] \leq \frac{1}{2}$$

We may similarly reduce error in a 0-sided machine by running it at most a fixed number of times and taking the first (if any) non-”Unknown” answer that is given.

Claim 1. *The class* ZPP *is equivalent to the class* ZPP2 *of problems solvable by randomized machines, also with no chance of error, that run in* expected *polynomial time.*

Proof. A ZPP machine can be run repeatedly until it outputs a definite answer. By the definition of ZPP each run halts in polynomial time, and outputs an answer with probability $\geq \frac{1}{2}$. If N^c is the running time of our ZPP algorithm, then the expected running time of our modified algorithm is:

$$\text{E(run time)} \leq \sum_{i=1}^{\infty}(N^c \cdot i) \cdot (\frac{1}{2})^i \tag{1}$$
$$= N^c \sum_{i=1}^{\infty} i \cdot (\frac{1}{2})^i = 2N^c \tag{2}$$

This new machine has expected running time polynomial and thus satisfies the definition of ZPP2.

Similarly a ZPP2 machine can be run on a clock for $t\cdot$ E(run time), then terminated with output “Unknown”. This modified algorithm outputs “Unknown” with probability at most:

$$\Pr[x \geq t \cdot E(x)] \leq \frac{1}{t} \text{ (Markov's inequality).} \tag{3}$$

To ensure the probability of outputting “Unknown” is at most 1/2, we set $t = 2$. As this new algorithm satisfies the error criterion of ZPP and runs in polynomial time, it is a ZPP algorithm for the language. ☐

Claim 2. ZPP = RP $\cap$ coRP

The proof of this claim is left as an exercise.

1.2 Space Complications

In a deterministic or even non-deterministic setting, any log-space algorithm terminates in polynomial time (if it terminates at all), due to the polynomial bound on the number of possible machine configurations. Obviously a repeated configuration signifies a loop, which occurs if and only if the machine does not halt.

Things become more complicated when randomness is involved. A repeated configuration does not necessarily signify an infinite loop, since a random bit may allow the machine to take a different path rather than repeating. *When discussing space bounds in this setting we assume machines that always halt.* This restriction gives us the nice property that a randomized log-space machine runs in polynomial time (and in general, a space s machine runs in $2^{O(s)}$ time).

Note that machines of this type are distinct from those which halt with probability 1. We define a separate class, $\lfloor \mathrm{ZPL} \rceil$, for that type of machine. An example of the addition power this model has over the one stated above is given by the following claim, which we leave as an exercise. While we do not expect that ZPP $\subseteq$ NP or that BPL $\subseteq$ NL, we know that

Claim 3. $\lfloor \mathrm{ZPL} \rceil = \mathrm{NL}$.

For more on how these definitional issues affect the power of randomness in the log-space setting, see the survey [1].

2 Relation to Deterministic Classes

Relating these randomized classes to the known deterministic classes provides a measure of their computational power. One immediate result is that

$$\mathrm{RTIME}(t) \subseteq \mathrm{NTIME}(t).$$

This follows from the fact that we may view any randomized machine M with 1-sided error as a non-deterministic machine M'. If there exists a sequence of random bits so that M accepts, we say that M' accepts; note that M' runs simultaneously the paths determined by all possible such strings of random bits. We also have the inclusions

$$\begin{aligned}&\mathrm{RP} \subseteq \mathrm{NP}\\ &\mathrm{P} \subseteq \mathrm{ZPP} \subseteq \mathrm{RP} \subseteq \mathrm{BPP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}\end{aligned}$$

All but one inclusion are by definition. The result that BPP $\subseteq$ PSPACE follows from the fact that we can compute the probability of acceptance by a randomized machine, by exhaustively generating all possible strings of results of coin flips, of which there are only polynomially many, and running the machine on each. The space needed for each run is only polynomial, and we need only keep a tally of the acceptance or rejection from each run.

It is an open question whether any of these containments is proper, though at least one must be since P $\subsetneq$ EXP.

Considering space complexity, we have the inclusions

$$\mathrm{L} \subseteq \mathrm{ZPL} \subseteq \mathrm{RL} \subseteq \mathrm{BPL} \subseteq \mathrm{UniformNC}^2 \subseteq \mathrm{DSPACE}(\log^2 N). \tag{4}$$

The result that Uniform $\mathrm{NC}^2 \subseteq \mathrm{DSPACE}(\log^2 N)$ follows by a proof similar to that of $\mathrm{NC}^1 \subseteq \mathrm{L}$ given in the last lecture, in which we evaluate each path through a $\log^2$-depth circuit, and so need only the space necessary to store one such path. The fact that BPL $\subseteq$ Uniform NC^2 takes a bit more work.

Claim 4. *$BPL \subseteq$ Uniform* NC^2

Proof. A BPL computation, can be viewed as a Markov chain of machine configurations. For a log-space machine the size of the set of configurations is polynomial. The Markov chain can be represented as a matrix, with each entry M_{ij} representing the probability of transitioning from state j to state i. (Note that these probabilities are time-independent, as needed for a Markov process.) Exponentiating this matrix N times gives the probability matrix for state transitions taking exactly N steps. This can be used to determine the probability of reaching an accepting configuration. We multiply M^N by the probability vector representing the fact that the machine begins in the first configuration:

$$\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix}^N \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5}$$

This is iterated matrix multiplication, which as discussed last time is in Uniform NC^2. The result is a probability vector, with one entry giving the probability of reaching the accepting state in N steps. □

In fact there is a tighter bound known for the class BPL:

Claim 5. BPL $\subseteq$ DSPACE $(\log^{3/2} N)$

The proof for this claim will be presented in a future lecture, along with evidence for the conjectures BPP = P and BPL = L. If these conjectures hold, it would show that randomness does not truly provide any additional computational power. Even then, however, randomness would still be very useful in practice to reduce computation time for several problems, included those mentioned earlier in this lecture. The speed-up for any problem, however, could only be polynomial. Of course, for now these conjectures remain unproven.

The following facts are also known (proving the second is the final problem on Homework 1).

Proposition 1.

$$\begin{aligned} &\mathrm{BPP}^{\mathrm{BPP}} = \mathrm{BPP} \\ &\mathrm{NP} \subseteq \mathrm{BPP} \implies \mathrm{PH} \subseteq \mathrm{BPP} \\ &\mathrm{NP} \subseteq \mathrm{BPP} \implies \mathrm{NP} = \mathrm{RP} \end{aligned}$$

The results given above relate randomized complexity classes among each other and with deterministic and nondeterministic complexity classes. The following result relates the randomized class BPP to the non-uniform class *P/poly*, showing that in general randomness can be replaced by non-uniformity.

Theorem 1. BPP $\subseteq$ *P/poly and* BPL $\subseteq$ *L/poly*

Proof. Consider a BPP algorithm. We first make the probability of error smaller than the number of inputs of a given length N: $\Pr[\text{error}] < \frac{1}{2^N}$. Recall that running a BPP algorithm with error $1/2 - \delta$ for k times and taking the majority vote results in a BPP algorithm computing the same language but now with error at most $e^{-2k\delta^2}$. We assume original error at most $1/3$, so $\delta \geq 1/6$.

We need to pick k large enough so that $e^{-2k(1/6)^2} < 1/2^N$. Then a large enough $k = \Theta(N)$ suffices, meaning the resulting BPP algorithm still runs in polynomial time.

Because we have reduced the error to less than $1/2^N$, given any distinct setting for the random bit tape, the probability that there exists some input x of length N on which the machine makes an error is less than one. Therefore there exists at least one coin flip sequence for which the machine gives the correct result on all inputs of length N. This sequence of coin flips is the advice given. As the amount of randomness used is some polynomial, this is a polynomial amount of advice.

The proof of the second claim uses the same process and the same advice; note that it may be completed in log space since each run of the original BPL machine takes only log space by definition, and we need only keep a tally of the polynomially many votes to take the majority, which requires only a logarithmic number of bits. □

We stated above that it is conjectured that BPP=P, though we have been unable to prove this. In fact, while $R \subseteq$ NP, we cannot even prove that BPP $\subseteq$ NP. We will show shortly, however, that BPP lies within the polynomial hierarchy.

3 Hierarchy Results for BPP

Generally, whenever we introduce a new computational model, we look for hierarchy results: Does the class of problems we can solve using the model in question depend on the resources we are allowed? Most hierarchy results use a computable enumeration of all machines of the type in question, but the following can be shown.

Exercise 1. *There does not exist a computable enumeration of the randomized machines with error probability bounded away from* $\frac{1}{2}$.

Therefore, typical hierarchy arguments fail for randomized machines. The typical hierarchy arguments can be tailored to prove a hierarchy theorem for a modified model of randomized computation called promise-BPP. See HW 1.2. However, no hierarchy result is known for BPP.

4 BPP and the Poly Time Hierarchy

Although we don't know if BPP = P, or even if BPP $\subseteq$ NP, we do know that BPP $\subseteq$ PH:

Theorem 2. BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$

Proof. Fix M a randomized polytime machine that accepts $L \in$ BPP, and let r be the number of random bits M uses when running on an input x of size $n = |x|$. We will now show that BPP $\subseteq \Sigma_2^p$. Since BPP = *co*BPP it follows that BPP $\subseteq co\Sigma_2^p = \Pi_2^p$, completing the proof.

We need to remove randomness from the computation. For a given input x, the space $\{0,1\}^r$ of r-bit strings gets partitioned into two pieces: $Acc(x)$, the set of random strings on which M accepts x, and $Rej(x)$, the set of strings on which M rejects x. If the error rate ε of M is small, then $Acc(x)$ will be much larger than $Rej(x)$ when $x \in L$, and $Acc(x)$ will be much smaller than $Rej(x)$ when $x \notin L$. See Figure 1. If our random computation has a small error rate then it will be correct on most random bit strings. We'll turn "most" into "all." The idea is that when $x \in L$ a few "shifts" of $Acc(x)$ will cover the whole space $\{0,1\}^r$ of r-bit random sequences, while the same few shifts of $Acc(x)$ will fail to cover the whole space when $x \notin L$.

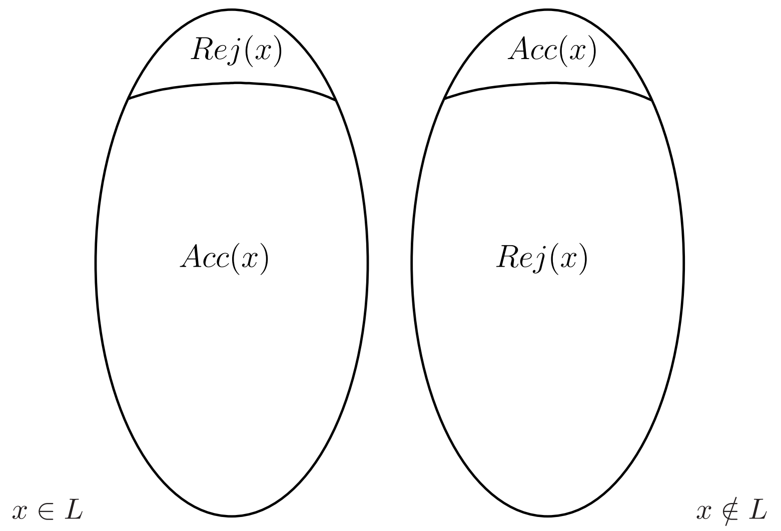

Figure 1: If the error rate ε of M is small, then $Acc(x)$ will be much larger than $Rej(x)$ when $x \in L$, and $Acc(x)$ will be much smaller than $Rej(x)$ when $x \notin L$.

If $S \subseteq \{0,1\}^r$ and $\sigma \in \{0,1\}^r$, then we define $S \oplus \sigma = \{s \oplus \sigma | s \in S\}$, the *shift* of S by σ.[1] Since shifting is invertible (in fact each shift is its own inverse) we see that $|S \oplus \sigma| = |S|$.

We will use shifts to give a Σ_2-predicate using the intuition discussed above. Namely, consider

$$x \in L \iff \exists \sigma_1, \ldots, \sigma_t \forall \rho \in \{0,1\}^r [\rho \in \bigcup_{i=1}^{t} Acc(x) \oplus \sigma_i] \tag{6}$$

where the universal quantifier is over all possible strings ρ of random bits. Now, r is polynomial in n, and so if we can pick t polynomial in n as well, then the above will be a Σ_2^p-predicate, provided that we can verify the membership of ρ in $\bigcup_{i=1}^{t} Acc(x) \oplus \sigma_i$ in time poly in n. This membership check is no problem since we can equivalently check that $\rho \oplus \sigma_i \in Acc(x)$ for some i. We can do this computation in polynomial time since it corresponds to running M on x with the random bit string $\rho \oplus \sigma_i$, for polynomially many σ_i.

We show that we can pick a suitable polynomial t to make (6) true by showing that we can choose the shifts σ_i randomly with a high rate of success. There are two cases to consider:

1. $x \in L$: For ρ fixed, for a given shift σ, the probability that ρ is not in $Acc(x) \oplus \sigma$ is given by

$$\Pr[\rho \notin Acc(x) \oplus \sigma] = \Pr[\rho \oplus \sigma \notin Acc(x)] \leq \varepsilon,$$

since $|Acc(x)|$ is a fixed size at least $1-\varepsilon$. Then for t shifts chosen uniformly and independently, we have

$$\Pr_{\sigma_1,\ldots,\sigma_t}[\rho \notin \bigcup_{i=1}^{t} Acc(x) \oplus \sigma_i] \leq \varepsilon^t.$$

[1]The symbol "$\oplus$" denotes XOR, or, equivalently, addition in $\mathbb{Z}_2^r$

So, by a union bound, the probability that all random bit strings are not covered by the t shifts of $Acc(x)$ is

$$\Pr[\{0,1\}^r \not\subseteq \bigcup_{i=1}^{t} Acc(x) \oplus \sigma_i] \leq |\{0,1\}^r|\varepsilon^t = 2^r\varepsilon^t.$$

Hence, if $2^r\varepsilon^t < 1$, some choice of the shifts σ_i must cover all the random bit strings.

2. $x \notin L$: We need to be sure that for any choice of the shifts σ_i, some bit string ρ makes M reject x. But
$$\mu(\bigcup_{i=1}^{t} Acc(x) \oplus \sigma_i) \leq \sum_{i=1}^{t} \mu(Acc(x) \oplus \sigma_i) = t\varepsilon,$$
since $\mu(Acc(x)) \leq \varepsilon$ when $x \notin L$. Hence in this case we need $t\varepsilon < 1$.

That is, we need to choose t so that both $t\varepsilon$ and $2^r\varepsilon^t$ are less than 1. After a short search, we see that $t = r$ will work, provided that $\varepsilon < \frac{1}{r}$. From the definition of BPP we only know that $\varepsilon < \frac{1}{3}$, but, using the "majority vote" trick introduced during the last lecture, we can in k runs reduce the error to $e^{-2k\delta^2}$, where $\delta = \frac{1}{2} - \varepsilon > 0$. Of course, this increases the number of random bits r that we will need, but since we can exponentially decrease the error with only a linear increase in bits, we see that this is clearly possible to choose such a k. Since k runs will need kr random bits, it suffices to choose a polynomial k large enough so that $e^{-2k\delta^2} < \frac{1}{rk}$. In fact, $k = O(\log r)$ is sufficient, and an examination of the above shows that the time complexity is $O(T^2 \log T)$ if the original run-time was T.

Therefore, our choice of t is sufficient to make (6) a Σ_2^p formula as needed. $\square$

References

[1] Michael Saks. Randomization and Derandomization in Space-Bounded Computation. *Annual Conference on Structure in Complexity Theory*, 1996.