Introduction to Complexity Theory*
Lecture 7: Randomized Computations

Notes taken by Erez Waisbard and Gera Weiss

December 24, 1998

Summary: In this lecture we extend the notion of efficient computation by allowing algorithms (Turing machines) to toss coins. We study the classes of languages that arise from various natural definitions of acceptance by such machines. We will focus on polynomial running time machines of the following types:

1. One-sided error machines ($RP, coRP$).
2. Two-sided error machines (BPP).
3. Zero error machines (ZPP)

We will also consider probabilistic machines that uses logarithmic spaces (RL).

1 Probabilistic computations

7.1 ⇒

The basic thought underlying our discussion is the association of efficient computation with probabilistic polynomial time Turing machines. We will consider efficient only algorithms that run in time that is no more than a fixed polynomial in the length of the input.
There are two ways to define randomized computation. One, that we will call *online* is to enter randomized steps, and the second that we will call *offline* is to use an additional randomizing input and evaluate the output on such random input.

In the fictitious model of non-deterministic machines, one accepting computation was enough to include an input in the language accepted by the machine. In the randomized model we will consider the probability of acceptance rather than just asking if the machine has an accepting computation.

> *Then he said, "May the Lord not be angry, but let me speak just once more. What if only ten can be found there?" He answered, "For the sake of ten, I will not destroy it."* [Genesis 18:32].

As God didn't agree to save Sedom for the sake of less then ten peoples, we will not consider an input to be in the accepted language unless it has a noticeable probability to be accepted.

> Oded's Note: *The above illustration is certainly not my initiative. Besides some reservations regarding this specific part of the bible (and more so the interpretations given to it during the centuries), I fear that 10 may not strick the reader as "many" but rather as closer to "existence". In fact, standard interpretations of this passage stress the minimalistic nature of the challenge – barely above unique existence...*

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

$\boxed{7.2} \Rightarrow$ **The online approach:** One way to look at a randomized computation is to allow the Turing machine to make random moves. Formally this can be modeled as letting the machine to choose randomly among the possible moves that arise from a nondeterministic transition table. If the transition table maps one $(< state >, < symbol >)$ pair to two different $(< state >, < move >, < symbol >)$ triples then the machine will choose each transition with equal probabilities.

Syntactically, the online probabilistic Turing machine will look the same as the nondeterministic machine. The difference is at the definition of the accepted language. The criterion of an input to be accepted by a regular nondeterministic machine is that the machine will have at least one accepting computation when it is invoked with this input. In the probabilistic case, we will consider the probability of acceptance. We would be interested in *how many* accepting computation the machine has (or rather what is the probability of such computation). We postulate that the machine choose every step with equal probability, and so get a probability space on possible computations. We look at a computation as a tree, where a node is a configuration and it's children are all the possible configurations that the machine can pass to in a single step. The tree is describing the possible computations of the machine when running on a given input. The output of a probabilistic Turing machine on an input x is not a string but a random variable. Without loss of generality we can consider only binary tree because if the machine has more than two possible steps, it is possible to build another machine that will simulate the given machine with two step transition table. This is possible even if the original machine had steps with probability that has infinite binary expansion. Let say, for example, that the machine has a probability of $\frac{1}{3}$ to get from step A to step B. Then we have a problem when trying to simulate it by unbiased binary coins, because there is the binary expansion of $\frac{1}{3}$ is infinite. But we can still get as close as we want to the original machine, and this is good enough for our purposes.

$\boxed{7.3} \Rightarrow$

The offline approach: Another way to consider nondeterministic machines is, as we did before, to use an additional input as a guess. For NP machines we gave an additional input that was used as a witness. The analogous idea is to view the outcome of the internal coin tosses as an auxiliary input. The machine will receive two inputs, the real input, x, and the guess input, r. Imagine that the machine receives this second input from an external 'coin tossing device' rather than toss coins internally.

Notation: We will use the following notation to discuss various properties of probabilistic machines:

$$Prob_r[M(x,r) = z]$$

Sometimes, we will drop the r and keep it implicitly like in the following notation:

$$Prob[M(x) = z]$$

By this notations we mean the probability that the machine M with real input x and guess input r, distributed uniformly, will give an output z. The probability space is that of all possible r taken with uniform distribution. This statement is more confusing than it seems to be because the machine may use different number of guesses for different inputs. It may also use different number of guesses on the same input, if the computation depends on the outcome of previous guesses.

> Oded's Note: *Actually, the problem is with the latter case. That is, if on each input all computations use the same number of coin tosses (or "guesses"), denoted l, then each*

such computation occurs with probability 2^{-l}. However, in the general case, where the number of coin tosses may depend on the outcome of previous tosses, we may just observe that a halting computation with coin outcome sequence r occurs with probability exactly $2^{-|r|}$.

Oded's Note: *An alternative approach is to modify the randomized machine so that it does use the same number of coin tosses in each computation on the same input.*

2 The classes RP and $coRP$ – One-Sided Error

$\boxed{7.4} \Rightarrow$

The first two classes of languages that arise from probabilistic computations that we consider are the one-sided error (polynomial running time) computable languages. If there exist a machine that can decide the language with good probability in polynomial time it is reasonable to consider the problem as relatively easy. Good probability here means that the machine will be sure only in one case and will give the right answer in the other case but only with good probability (the cases are when $x \in L$ and when $x \notin L$).
From here on, a polynomial probabilistic Turing machine means a probabilistic machine that always (no matter what coin tosses it gets) halts after a polynomial (in the length of the input) number of steps.

Definition 1 (Random Polynomial-time – RP): *The complexity class RP is the class of all languages L for which there exist a probabilistic polynomial-time Turing machine M, such that*

$$x \in L \Rightarrow Prob[M(x) = 1] \geq \frac{1}{2}.$$

$$x \notin L \Rightarrow Prob[M(x) = 1] = 0.$$

Definition 2 (Complementary Random Polynomial-time – $coRP$): *The complexity class $coRP$ is the class of all languages L for which there exist a probabilistic polynomial-time Turing machine M, such that*

$$x \in L \Rightarrow Prob[M(x) = 1] = 1.$$

$$x \notin L \Rightarrow Prob[M(x) = 0] \geq \frac{1}{2}.$$

One can see from the definitions that these two classes complement each other. If you have a machine that decides a language L with good probability (in one of the above senses), you can use the same machine to decide the complementary language in the complementary sense.
That is, an alternative (and equivalent) way to define $coRP$ is:

$$coRP = \{\overline{L} : L \in RP\}$$

7.5 ⇒

Comparing NP to RP: It is instructive to compare the definitions of RP and NP. In both classes we had the offline definition that used an external witness (in NP) or randomization (in RP).

Given an RP machine, M, since the machine run in polynomial-time, the size of the guesses that it can use is bounded by a polynomial in the size of x. For every given integer $n \in \mathsf{N}$ we consider the relation:

$$R_n \stackrel{\text{def}}{=} \left\{(x,r) \in \{0,1\}^n \times \{0,1\}^{p(n)} : M(x,r) = 1\right\}$$

which consists of all accepted inputs of length n and their accepting coin tosses (i.e r).

The same is also applicable for NP machines, which run also in polynomial-time and can only use witnesses that are bounded by a polynomial in the length of the input. So, for NP machine M, we consider the relation:

$$R_n \stackrel{\text{def}}{=} \left\{(x,y) \in \{0,1\}^n \times \{0,1\}^{p(n)} : M(x,y) = 1\right\}$$

which consist of all accepted inputs of length n and their witnesses (i.e y).

In both cases we will use the relation:

$$R = \bigcup_{n=1}^{\infty} R_n$$

which consists of all the accepted inputs and their witness/coin-tosses.

Using this relation we can compare Definition 1 to the definition of NP in the following table:

NP	RP
$x \in L \Rightarrow \exists y,\ (x,y) \in R$	$x \in L \Rightarrow \text{Prob}_r\ [(x,r) \in R] \geq \frac{1}{2}$
$x \notin L \Rightarrow \forall y,\ (x,y) \notin R$	$x \notin L \Rightarrow \forall r,\ (x,r) \notin R$

From this table, it is seems that these two classes are close. The witness in the nondeterministic model is replaced by the coin-tosses and the criteria for acceptance has changed. The difference is that, in the nondeterministic model, one witness was enough for us to say that an input is accepted, and in the probabilistic model we are asking for many coin-tosses. Clearly,

Proposition 2.1 $NP \supseteq RP$

Proof: Let L be an arbitrary language in RP. If $x \in L$ then there exist a Turing machine M and a coin-tosses y such that $M(x,y) = 1$ (more than $\frac{1}{2}$ of the coin-tosses are such). So we can use this y as a witness (considering the same machine as a nondeterministic machine with the coin-tosses as witnesses). If $x \notin L$ then $Prob_r[M(x,r) = 1] = 0$ so there is no witness. ■

Notice that there is a big difference between nondeterministic Turing machines and probabilistic Turing machines. The first is a fictitious concept that is invented to explore the properties of search problems, while the second is a realistic model that describe machines that one can really build.We use the nondeterministic model to describe problems like a search problem with an efficient verification, while the probabilistic model is used as an efficient computation.

It is fair to ask if a computer can toss-coins as an elementary operation. We answer this question positively based on our notion of randomness and the ability of computers to use random-generating instrumentation like reading unstable electric circuits. The question is whether this random operation gives us more power than we had with the regular deterministic machines.

RP is one-sided error: The definition of RP does not ask for the same behavior on inputs that are in the language as it asks for inputs that are in the language.

- If $x \notin L$ then the answer of the machine must be correct no matter what guesses we make. In this case, the probability to get a wrong answer is zero so the answer of the machine is right for every r.
- But, if $x \in L$, the machine is allowed to make mistakes. In this case, we have a non-zero probability that the answer of the machine will be wrong (still this probability is not "too big").

The definition favors one type of mistake while in practice we don't find very good reason to favor it. We will see later that there are different families of languages that do not favor any type of error. We will call these languages two-sided error languages.

It was reasonable to discuss one-sided errors when we where developing NP, because verification is one-sided by nature, but it is less useful for exploring the notion of efficient computation.

$\boxed{7.6} \Rightarrow$

Invariance of the constant and beyond: Recall that for $L \in RP$

$$x \in L \Rightarrow Prob_r[M(x,r) = 1] \geq \frac{1}{2}$$

The constant $\frac{1}{2}$ in the definition of RP is arbitrary. We could choose every constant strictly threshold between zero and one, and get the same complexity class. Our choice of $\frac{1}{2}$ is somewhat appealing because it says that at least half of the witnesses are good.
If you have, for example, a machine that can decide some language L with a greater probability than $\frac{1}{3}$ to say "YES" for an input that is in the language, you can build another machine that will invoke the first machine three times on every input and return the "YES" if one of them answered "YES". Obviously this machine will answer correctly on inputs that are not in the language (because the first machine will always say "NO"), and it will say "YES" on inputs that are in the language with higher probability than before. The original probability of not getting the correct answer when the input is in the language was smaller than $\frac{2}{3}$, when repeating the computation for three time this probability falls down to less than $\left(\frac{2}{3}\right)^3 = \frac{8}{27}$ meaning that we now get the correct answer with probability greater than $\frac{19}{27}$ (which is greater than $\frac{1}{2}$).

So we could use $\frac{1}{3}$ instead of $\frac{1}{2}$ without changing the class of languages. This procedure of amplification can be used to show the same result for every constant, but we will prove further that one can even use thresholds that depend on the length of the input.

We are looking at two probability spaces: one when $x \notin L$ and one when $x \in L$, and defined a random variable (representing the decision of the machine) on each of this spaces. In case $x \notin L$ the latter random variable is identically zero (i.e., "reject"), whereas in case $x \in L$ the random variable may be non-trivial (i.e., is 1 with probability above some given threshold and 0 otherwise).

Moving from one threshold to a higher one amount to the following: In case $x \in L$, the fraction of points in the probability space assigned the value 1 is lower bounded by the first threshold. Our aim is to hit such a point with probability lower bounded by a higher threshold. This is done by merely making repeated independent samples into the space, where the number of the trials is easily determined by the relation between the two thresholds. We stress that in case $x \notin L$ all points in the probability space are identically assigned (the value 0) and so it does not matter how many times we try (we'll always see zeros).

We will show that one can even replace the constant $\frac{1}{2}$ by either $\frac{1}{p(|x|)}$ or $1 - 2^{-p(|x|)}$, where $p(\cdot)$ is any fixed polynomial, and get the same family of languages. We take these two margins, because once we will show the equivalence of these two thresholds, it will follow that every threshold that one might think of in between will do. Consider the following definitions:

Definition 3 ($RP1$): *L is in $RP1$ if there exist a polynomial running-time Turing machine M and a polynomial $p(\cdot)$ such that* $\begin{array}{l} x \in L \Rightarrow Prob_r[M(x,r) = 1] \geq \frac{1}{p(|x|)} \\ x \notin L \Rightarrow Prob_r[M(x,r) = 0] = 1 \end{array}$

Definition 4 ($RP2$): *L is in $RP2$ if there exist a polynomial running-time Turing machine M and a polynomial $p(\cdot)$ such that* $\begin{array}{l} x \in L \Rightarrow Prob_r[M(x,r) = 1] \geq 1 - 2^{-p(|x|)} \\ x \notin L \Rightarrow Prob_r[M(x,r) = 0] = 1 \end{array}$

These definitions seems very far from each other, because in $RP1$ we ask for a probabilistic algorithm (Turing machine) that answer correctly with a very small probability (but not negligible), while in $R2$ we ask for an efficient algorithm (Turing machine) that we can almost ignore the probability of it's mistake. However, these two definition actually define the same class (as we will prove in the next paragraph). This implies that having an algorithm with a noticeable probability of success implies existence of and efficient algorithm with negligible probability of error.

Proposition 2.2 *RP1=RP2*

$\boxed{7.7} \Rightarrow$

Proof:

$RP1 \supseteq RP2$

This direction is trivial because if $|x|$ is big enough then the bound in Definition 3 (i.e $\frac{1}{p(|x|)}$) is smaller than the bound in Definition 4 (i.e $1 - 2^{-p(|x|)}$) so being in $RP2$ implies being in $RP1$ for almost all inputs. The finitely many inputs for which this does not hold can be incorporated in the machine of Definition 3. Thus $RP1 \supseteq RP2$.

$RP1 \subseteq RP2$

We will use a method known as *amplification*:

We will try the weaker machine (of $RP1$) enough times so that the probability of giving a wrong answer will be small enough. Assume that we have a machine M_1 such that

$$\forall x \in L : Prob_r[M_1(x,r) = 1] \geq \frac{1}{p(|x|)}$$

We will define a new machine M_2, up to a function $t(|x|)$ that we will determine later, as follows:

$$M_2(x) \stackrel{\text{def}}{=} \begin{cases} invoke\ M_1(x)\ t(|x|)\ times\ with\ different\ randomly\ selected\ r's \\ if\ some\ of\ these\ invocations\ returned\ 'YES'\ return\ 'YES' \\ else\ return\ 'NO' \end{cases}$$

Let $t = t(|x|)$. Then for $x \in L$

$$Prob[M_2(x) = 0] = (Prob[M_1(x) = 0])^{t(|x|)} \leq \left(1 - \frac{1}{p(|x|)}\right)^{t(|x|)}$$

To find the desired $t(|x|)$ we can solve the equation:

$$\left(1 - \frac{1}{p(|x|)}\right)^{t(|x|)} \leq 2^{-p(|x|)}$$

And obtain

$$t(|x|) \geq p(|x|) \cdot \log_2 \left(1 - \frac{1}{p(|x|)}\right)^{-1} = \frac{p(|x|)^2}{\log_2 e}$$

Where $e \approx 2.7182818...$ is the natural logarithm base.

So by letting $t(|x|) = p(|x|)^2$ in the definition of M_2 we get a machine that run in polynomial time and decides L with probability greater than $1 - 2^{-p(|x|)}$ to give right answer for $x \in L$ (and always correct on $x \notin L$). ■

7.8 ⇒

3 The class BPP – Two-Sided Error

One may argue that RP is too strict because it ask that the machine has to give 100% correct answer for inputs that are not in the language.

We derived the definition of RP from the definition of NP, but NP didn't reflect an actual computational model for search problems but rather a model for verification. One may find that looking at a two-sided error is more appealing as a model for search problem computations.

We want a machine that will recognize the language with high probability, where probability refers to the event "The machine answers correctly on an input x regardless if $x \in L$ or $x \notin L$". This will lead us to two-sided error version of the randomized computation. First recall the notation:

$$\chi_L(x) \stackrel{\text{def}}{=} \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}$$

Definition 5 (Bounded-Probability Polynomial-time – BPP): *The complexity class BPP is the class of all languages L for which there exist a probabilistic polynomial-time Turing machine M, such that*

$$\forall x : Prob[M(x) = \chi_L(x)] \geq \frac{2}{3}.$$

That means that:

$$If\ x \in L\ then\ Prob[M(x) = 1] \geq \frac{2}{3}.$$

$$If\ x \notin L\ then\ Prob[M(x) = 1] < \frac{1}{3}.$$

The phrase "bounded-probability" means that the success probability is bounded away from failure probability.

The BPP machine is a machine that makes mistakes but returns the correct answer most of the time. By running the machine a large number of times and returning the majority of the answers we are guaranteed by the law of large numbers that our mistake will be very small.
The idea behind the BPP class is that M accept by majority with a noticeable gap between the probability to accept inputs that are in language and the probability to accept inputs that are not in the language, and it's running time is bounded by a polynomial.

$\boxed{7.9} \Rightarrow$

Invariance of constant and beyond: The $\frac{2}{3}$ is, again, an arbitrary constant. Replacing the $\frac{2}{3}$ in the definition by any other constant greater than $\frac{1}{2}$ does not change the class defined. If, for example we had a machine, M that recognize some language L with probability $p > \frac{1}{2}$, meaning that $Prob[M(x) = \chi_L(x)] \geq p$, we could easily build a machine that will recognize L with any given probability $q > p$ by invoking this machine sufficiently many times and returning the majority of the answers. This will clearly increase the probability of giving correct answer to the wanted threshold, and run in polynomial time.

In the RP case we had two probability spaces that we could distinguish easily because we had a guarantee that if $x \notin L$ then the probability to get one is zero, hence if you get $M(x) = 1$ for some input x, you could say *for sure* that $x \in L$.
In the BPP case, the amplification is less trivial because we have zeroes and ones in both probability spaces (the probability space is not constant when $x \in L$ nor when $x \notin L$).
The reason that we can apply amplification in the BPP case (despite the above difference) is that invoking the machine many times and counting how many times it returns one gives us an estimation on the fraction of ones in the whole probability space. It is useful to get an estimator for the fraction of the ones in the probability space because when this fraction is greater than $\frac{2}{3}$ we have that $x \in L$, and when this fraction is less than $\frac{1}{3}$ we have that $x \notin L$ (this fraction tells us in which probability space we are in).

If we rewrite the condition in Definition 5 as:

$$If\ x \in L\ then\ Prob[M(x) = 1] \geq \frac{1}{2} + \frac{1}{6}.$$

$$If\ x \notin L\ then\ Prob[M(x) = 1] < \frac{1}{2} - \frac{1}{6}.$$

We could consider the following change of constants:

$$If\ x \in L\ then\ Prob[M(x) = 1] \geq p + \epsilon.$$

$$If\ x \notin L\ then\ Prob[M(x) = 1] < p - \epsilon.$$

for any given $p \in (0,1)$ and $0 < \epsilon < \min\{p, 1-p\}$.

If we had such a machine, we could invoke the machine many times and get increasing probability to have the fraction of ones in our innovations to be in an ϵ neighborhood of the real fraction of ones in the whole space (by the law of large numbers). After some fixed number of iterations (that does not depend on x), we can get that probability to be larger than $\frac{2}{3}$.
This means that if we had such a machine (with p and ϵ instead of $\frac{1}{2}$ and $\frac{1}{6}$), we could build another machine that will invoke it some fixed number of times and will decide the same language with probability greater than $\frac{2}{3}$.

The conclusion is that the $\frac{1}{2} \pm \frac{1}{6}$ is arbitrary in Definition 5, and can be replaced by any $p \pm \epsilon$ such that $p \in (0,1)$ and $0 < \epsilon < \min\{p, 1-p\}$. But we can do more than that and use threshold that depend on the length of the input as we will prove in the following claims:

$\boxed{7.10} \Rightarrow$ **The weakest possible BPP definition:** Using the above framework, we'll show that for every polynomial-time computable threshold, denoted f below, and any "noticeable" margin (represented by 1/poly), we can recover the "standard" threshold (of 1/2) and the "safe" margin of 1/6.

Claim 3.1 *$L \in BPP$ if and only if there exist a polynomial-time computable function $f : \mathsf{N} \mapsto [0,1]$, a positive polynomial $p(\cdot)$ and a probabilistic polynomial-time Turing machine M, such that:*

$$\forall x \in L : Prob[M(x) = 1] \geq f(|x|) + \frac{1}{p(|x|)}$$

$$\forall x \notin L : Prob[M(x) = 1] < f(|x|) - \frac{1}{p(|x|)}$$

Proof:
It is easy to see that by choosing $f(|x|) \equiv \frac{1}{2}$ and $p(|x|) \equiv 6$ we get the original definition of BPP (see Definition 5), hence every BPP language satisfies the above condition.

Assume that we have a probabilistic Turing machine, M, with these bounds on the probability to get 1. Then, for any given input x, we look at the random variable $M(x)$, which is a Bernoulli random variable with unknown parameter $p = Exp[M(x)]$. Using a well known fact that the expectation of a Bernoulli random variable is exactly the probability to get one we get that $p = Prob[M(x) = 1]$.
So by estimating p we can say something about whether $x \in L$ or $x \notin L$. The most natural estimator is to take the mean of n samples of the random variable (i.e the answers of n independent invocations of $M(x)$).
Then we will use the known statistical method of confidence intervals on the parameter p. The confidence interval method gives a bound within which a parameter is expected to lie with a certain probability. Interval estimation of a parameter is often useful in observing the accuracy of an estimator as well as in making statistical inferences about the parameter in question.
In our case we want to know with probability higher than $\frac{2}{3}$ if $p \in \left[0, f(|x|) - \frac{1}{p(|x|)}\right]$ or $p \in \left[f(|x|) + \frac{1}{p(|x|)}, 1\right]$. This is enough because $p \in \left[0, f(|x|) - \frac{1}{p(|x|)}\right] \Rightarrow x \notin L$ and $p \in \left[f(|x|) + \frac{1}{p(|x|)}, 1\right] \Rightarrow x \in L$ (note that $p \in \left(f(|x|) - \frac{1}{p(|x|)}, f(|x|) + \frac{1}{p(|x|)}\right)$ is impossible). So if we can get a bound of size $\frac{1}{p(|x|)}$ within which p is expected to lie within a probability greater than $\frac{2}{3}$, we can decide $L(M)$ with this probability (and hence $L(M) \in BPP$ by Definition 5).
We define the following Turing machine (up to an unknown number n that we will compute later) :

$$M'(x) \stackrel{\text{def}}{=} \begin{cases} \textit{Invoke } M(x)\ n \textit{ times (call the result of the i'th invocation } t_i). \\ \textit{Compute } \hat{p} \leftarrow \frac{1}{n} \cdot \Sigma_{i=1}^{n} t_i \\ \textit{if } \hat{p} > f(|x|) \textit{ say 'YES' else say 'NO'} \end{cases}$$

Note that $\hat{p}$ is exactly the mean of a sample of size n taken from the random variable $M(x)$. This machine do the normal statistical process of estimating a random variable by taking samples and using the mean as an estimator for the expectation. If we will be able to show that with an appropriate n the estimator will not fall too far from the real value with a good probability, it will follow that this machine answers correctly with the same probability.
To resolve n we will use Chernoff's inequality which states that for any set of n independent

Bernoulli variables $\{X_1, X_2, ..., X_n\}$ with the same expectations $p \leq \frac{1}{2}$ and for every $\delta, 0 < \delta \leq p(p-1)$, we have

$$Prob\left[\left|\frac{\Sigma_{i=1}^n X_i}{n} - p\right| > \delta\right] < 2 \cdot e^{-\frac{\delta^2}{2 \cdot p(1-p)} \cdot n} \leq 2 \cdot e^{-\frac{\delta^2}{2 \cdot \frac{1}{4}} \cdot n} = 2 \cdot e^{-2n \cdot \delta^2}$$

So by taking $\delta = \frac{1}{p(|x|)}$ and $n = -\frac{\ln \frac{1}{6}}{2 \cdot \delta^2}$ we get that our Turing machine M' will decide $L(M)$ with probability greater than $\frac{2}{3}$ suggesting that $L(M) \in BPP$. ■

$\boxed{7.11} \Rightarrow$ **The strongest possible BPP definition:** On the other hand, one can reduce the error probability of BPP machines to an exponentially vanishing amount.

Claim 3.2 *For every $L \in BPP$ and every positive polynomial $p(\cdot)$ there exist a probabilistic polynomial-time Turing machine M, such that:*

$$\forall x : Prob[M(x) = \chi_L(x)] \geq 1 - 2^{-p(|x|)}$$

Proof:
If this condition is true for every polynomial, we can choose $p(|x|) \equiv 2$ and get M such that:

$$\begin{aligned} &\forall x : Prob[M(x) = \chi_L(x)] \geq 1 - 2^{-2} = \tfrac{3}{4} \\ &\Rightarrow \forall x : Prob[M(x) = \chi_L(x)] \geq \tfrac{2}{3} \\ &\Rightarrow L \in BPP \end{aligned}$$

Let L be a language in BPP and let M be the machine guaranteed in Definition 5. We can amplify the probability of right answer by invoking M many times and taking the majority of it's answers. Define the following machine (again up to the number n that we will find later):

$$M'(x) \stackrel{\text{def}}{=} \begin{cases} \textit{Invoke } M(x)\ n \textit{ times (call the result of the } i'th \textit{ invocation } t_i). \\ \textit{Compute } \hat{p} \leftarrow \frac{1}{n} \cdot \Sigma_{i=1}^n t_i \\ \textit{if } \hat{p} > \frac{1}{2} \textit{ say } 'YES' \textit{ else say } 'NO' \end{cases}$$

From Definition 5, we get that if we know that $Exp[M(x)]$ is greater than half it follows that $x \in L$ and if we know that $Exp[M(x)]$ is smaller than half it follows that $x \notin L$ (because $Exp[M(x)] = Prob[M(x) = 1]$)
But Definition 5 gives us more. It says that the expectation of $M(x)$ is bounded away from $\frac{1}{2}$ so we can use the confidence interval method.

From Chernoff's inequality we get that

$$Prob\left[|M'(x) - Exp[M(x)]| \leq \frac{1}{6}\right] \geq 1 - 2 \cdot e^{-\frac{n}{18}}$$

But if $|M'(x) - Exp[M(x)]|$ is smaller than $\frac{1}{6}$ we get from Definition 5 that the answer of M' is correct, because it is close enough to the the expectation of $M(x)$ which is guaranteed to be above $\frac{2}{3}$ when $x \in L$ and bellow $\frac{1}{3}$ when $x \notin L$. So we get that:

$$Prob\left[M'(x) = \chi_L(x)\right] \geq 1 - 2 \cdot e^{-\frac{n}{18}}$$

Thus, for every polynomial $p(\cdot)$, we can choose n, such that

$$2^{p(|x|)} \geq 2 \cdot e^{-\frac{n}{18}}$$

and get that:

$$Prob\left[M'(x) = \chi_L(x)\right] \geq 1 - 2^{p(|x|)}$$

So M' satisfies the claimed condition. ∎

Conclusion: We see that a gap of $\frac{1}{p(|x|)}$ and a gap of $1 - 2^{-p(|x|)}$ which look like "weak" and "strong" versions of BPP are the same. As shown above the "weak" version is actually equivalent to the "strong" version, and both are equivalent to the original definition of BPP.

7.12 ⇒

Some comments about BPP:

1. $RP \subseteq BPP$
   It is obvious that one-sided error is a special case of two-sided error.

2. We don't know if $BPP \subseteq NP$. It might be so but we don't get it from the definition like we did in RP.

3. If we define $coBPP \stackrel{\text{def}}{=} \{\overline{L} : L \in BPP\}$ we get, from the symmetry of the definition of BPP, that $coBPP = BPP$.

4 The class PP

7.13 ⇒

The class PP is wider than what we have seen so far. In the BPP case we had a gap between the number of accepting computations and non-accepting computations. This gap enabled us to determine with good probability (using confidence intervals) if $x \in L$ or $x \notin L$.
The gap was wide enough so we could invoke the machine polynomially many times and notice the difference between inputs that are in the language and inputs that are not in the language. The PP class don't put the gap restriction, hence the gap may be very small (even one guess can make a difference).
Running the machine polynomially many times may not help. If we have a machine that answers correctly with probability more than $\frac{1}{2}$, and we want to get another machine that answers correctly with probability greater than $\frac{1}{2} + \epsilon$ (for a given $0 < \epsilon < \frac{1}{2}$) we can't always do it in polynomial time because we might not have the gap that we had in Definition 5.

Definition 6 $PP \stackrel{\text{def}}{=} \left\{ L \subseteq \{0,1\}^* \;\middle|\; \begin{array}{l} \textit{There exist a polynomial time} \\ \textit{Turing machine } M \textit{ s.t} \\ \forall x, Prob[M(x) = \chi_L(x)] > \frac{1}{2} \end{array} \right\}$

Note that it is important that we define $>$ and not $\geq$, since otherwise we can simply "flip a coin" and completely ignore the input (we can decide to say $'YES'$ if we get head and $'NO'$ if we get tail and this will satisfy the definition of the machine) and there is no use for a machine that runs a lot of time and gives no more knowledge than what we already have (assuming one knows how to flip a coin). However the actual definition of PP gives very little as well: The difference between what happens in case $x \in L$ and in case $x \notin L$ is negligible (rather than "noticeable" as in the definition of BPP). We abuse this weak requirement in the proof of Item 3 (below).

From the definition of PP we get a few interesting facts:

1. $PP \subseteq PSPACE$

   7.14 $\Rightarrow$ Let L be a language in PP, let M be the probabilistic Turing machine that exists according to Definition 6. Let $p(\cdot)$ be the polynomial bounding it's running time. We will build a new machine M' that decides L in a polynomial space. Given an input x, the new machine will run M on x using all possible coin tosses with length $p(|x|)$ and decides by majority (i.e if M accepted the majority of it's invocations then M' accepts x, else it rejects x).
   Every invocation of M on x requires a polynomial space. And, because we can use the same space for all invocations, we see that M' uses polynomial space (the fact that we run it exponentially many times does not matter). The answer of M' is correct because M is a PP machine that answers correctly for more than half of the guesses.

2. Small Variants
   We mentioned that, in Definition 6, we can't take $\geq$ instead of of $>$ because this will give us no information. But what about asking for $\geq$ when $x \notin L$ and $>$ when $x \in L$ (or the other way around) ? We will show, in the next claim, that this will not change the class of languages. A language has such a machine if and only if it has a PP machine.

   Consider the following definition:

   Definition 7 $$PP1 \stackrel{\text{def}}{=} \left\{ L \subseteq \{0,1\}^* \;\middle|\; \begin{array}{l} \textit{There exist a polynomial time} \\ \textit{Turing machine } M \textit{ s.t} \\ x \in L \Rightarrow Prob[M(x) = 1] > \frac{1}{2} \\ x \notin L \Rightarrow Prob[M(x) = 0] \geq \frac{1}{2} \end{array} \right\}$$

   The next claim will show that this relaxation will not change the class defined:

   Claim 4.1 $PP1 = PP$

   Proof:
   7.15 $\Rightarrow$ $PP \subseteq PP1$:
   If we have a machine that satisfies Definition 6 it also satisfies Definition 7, so clearly $L \in PP \Rightarrow L \in PP1$.

   $PP \supseteq PP1$:
   Let L be any language in $PP1$. If M is the machine guaranteed by Definition 7, and $p(\cdot)$ is the polynomial bounding it's running time (and thus the number of coins that it uses), we can define another machine M' as follows:

   $$M'\left(x, \left(a_1, a_2, ..., a_{p(|x|)+1}, b_1, b_2, ..., b_{p(|x|)}\right)\right) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \textit{if } a_1 = a_2 = ... = a_{p(|x|)+1} = 0 \textit{ then return } 'NO' \\ \textit{else return } M\left(x, \left(b_1, b_2, ..., b_{p(|x|)}\right)\right) \end{array} \right.$$

   M' chooses one of two moves. One move, which happens with probability $2^{-(p(|x|)+1)}$, will return $'NO'$. The second move, which happens with probability $1 - 2^{-(p(|x|)+1)}$ will invoke

M with independent coin tosses.
This gives us that

$$Prob[M'(x) = 1] = Prob[M(x) = 1] \cdot \left(1 - 2^{-(p(|x|)+1)}\right)$$

and

$$Prob[M'(x) = 0] = Prob[M(x) = 0] \cdot \left(1 - 2^{-(p(|x|)+1)}\right) + 2^{-(p(|x|)+1)}$$

The trick is to shift the answer of M towards the $'NO'$ direction with a very small probability. This shift is smaller than the smallest probability difference that M could have. So if $M(x)$ is biased towards the $'YES'$, our shift will keep the direction of the bias (it will only lower it). But if there is no bias (or bias towards NO), our shift will give us a bias towards the $'NO'$ answer.

If $x \in L$ then $Prob[M(x) = 1] > \frac{1}{2}$, hence $Prob[M(x) = 1] \geq \frac{1}{2} + 2^{-p(|x|)}$ (because the difference is at least one computation which happens with probability $2^{-p(|x|)}$), so:

$$\begin{aligned} Prob[M'(x) = 1] &\geq \left(\frac{1}{2} + 2^{-p(|x|)}\right) \cdot \left(1 - 2^{-(p(|x|)+1)}\right) \\ &= \frac{1}{2} + 2^{-p(|x|)} - 2^{-(p(|x|)+2)} - 2^{-(2p(|x|)+1)} \quad > \quad \frac{1}{2} \end{aligned}$$

If $x \notin L$ then $Prob[M(x) = 0] \geq \frac{1}{2}$, hence

$$\begin{aligned} Prob[M'(x) = 0] &\geq \frac{1}{2} \cdot \left(1 - 2^{-(p(|x|)+1)}\right) + 2^{-(p(|x|)+1)} \\ &= \frac{1}{2} - 2^{-(p(|x|)+2)} + 2^{-(p(|x|)+1)} \quad > \quad \frac{1}{2} \end{aligned}$$

And, as a conclusion, we get that in any case

$$Prob[M'(x) = \chi_L(x)] > \frac{1}{2}$$

So M' satisfies Definition 6, and thus $L \in PP$. ■

3. $NP \subseteq PP$

$\boxed{7.16} \Rightarrow$ Suppose that $L \in NP$ is decided by a nondeterministic machine M with a running-time that is bounded by the polynomial $p(|x|)$. The following machine M' then will decide L by means of Definition 6:

$$M'\left(x, \left(b_1, b_2, ..., b_{p(|x|)+1}\right)\right) \stackrel{\text{def}}{=} \begin{cases} if\ b_1 = 1\ then\ return\ M\left(x, \left(b_2, b_3, ..., b_{p(|x|)+1}\right)\right) \\ else\ return\ 'YES' \end{cases}$$

M' uses it's random coin-tosses as a witness to M with only one toss that it does not pass to M'. This toss is used to choose it's move. One of the two possible moves gets it to the ordinary computation of M with the same input (and the witness is the random input). The

other choice gets it to a computation that always accepts.
Consider a string x.

If M doesn't have an accepting computation then the probability that M' will answer 1 is exactly $\frac{1}{2}$ (it is the probability that the first coin will fall on one). On the other hand, if M has at least one accepting computation then the probability that M' will answer correctly is greater than $\frac{1}{2}$.

So we get that:

$x \in L \Rightarrow \quad Prob[M'(x) = 1] > \frac{1}{2}$
$x \notin L \Rightarrow \quad Prob[M'(x) = 0] \geq \frac{1}{2}$

By Definition 7, we conclude that $L \in PP1$, and by the previous claim ($PP = PP1$), we get that $L \in PP$.

4. $coNP \subseteq PP$
Easily seen from the symmetry in the definition of PP.

5 The class ZPP – Zero error probability.

$\boxed{7.17} \Rightarrow$

RP definition is asymmetric and we can't say whether $RP = coRP$. It would be interesting to examine the properties of $RP \bigcap coRP$ which is clearly symmetric. It seems that problems which are in $RP \bigcap coRP$ can benefit from the accurate result of RP deciding Turing machine (if $x \notin L$) and of $coRP$ deciding Turing machine (if $x \in L$).
Another interesting thing to consider is to let the machine say "I don't know" for some inputs. We will discuss machines that can return this answer but answer correctly otherwise.
We will prove that these two ideas give rise to the same class of languages.

Definition 8 (ZPP): *$L \in ZPP$ if there exist a probabilistic polynomial-time Turing machine M, such that:*

$$\forall x, \qquad Prob[M(x) = \perp] \leq \frac{1}{2}$$
$$\forall x, \quad Prob[M(x) = \chi_L(x) \ or \ M(x) = \perp] = 1$$

Where we denote the unknown answer sign as $\perp$.

Again the value $\frac{1}{2}$ is arbitrary and can be replaced like we did before to be anything between $2^{-p(|x|)}$ to $1 - \frac{1}{p(|x|)}$. If we have a ZPP machine that doesn't know the answer with probability half, we can run it $p(|x|)$ times and get a machine that doesn't know the answer with probability $2^{-p(|x|)}$ because this is the probability that none of our invocation know the answer (the other way is obvious because $2^{-p(|x|)}$ is smaller than $\frac{1}{2}$ for all but final inputs). If we have a machine that know the answer with probability $\frac{1}{p(|x|)}$, we can use it to build a machine that know the answer with probability $\frac{1}{2}$ by invoking it $p(|x|)$ times (the other way is, again, trivial).

Proposition 5.1 $ZPP = RP \bigcap coRP$

Proof: Take $L \in ZPP$. Let M be the machine guaranteed in Definition 8. We will show how to build a new machine M' which decides L according to Definition 1 (this will imply that $ZPP \subseteq RP$).

$$M'(x) \stackrel{\text{def}}{=} \begin{cases} b \leftarrow M(x) \\ if\ b = \perp\ then\ output\ 0 \\ else\ output\ b\ itself \end{cases}$$

By doing so, if $x \notin L$ then by returning 0 when $M(x) = \perp$ we will always answer correctly (because in this case $M(x) \neq \perp \Rightarrow M'(x) = \chi_L(x) \Rightarrow M'(x) = 0$).
If $x \in L$, the probability of getting the right answer with M' is greater than $\frac{1}{2}$ because M will return a definite answer ($M(x) \neq \perp$) with probability greater than $\frac{1}{2}$ and M's definite answers are always correct (it never return a wrong answer because it returns $\perp$ when it is uncertain).
In the same way it can be seen that $ZPP \subseteq coRP$ (the machine that we will build will return 1 when M is uncertain), hence we get that $ZPP \subseteq RP \bigcap coRP$.

Assume now that $L \in RP \bigcap coRP$. Let M_{RP} be the RP machine and M_{coRP} the $coRP$ machine that decides L (according to Definition 1 and Definition 2). We define $M'(x)$ using M_{RP} and M_{coRP} as follows:

$$M'(x) \stackrel{\text{def}}{=} \begin{cases} run\ M_{RP}(x),\ if\ it\ says\ 'YES'\ then\ return\ 1 \\ run\ M_{coRP}(x),\ if\ it\ says\ 'NO'\ then\ return\ 0 \\ otherwise\ return\ \perp \end{cases}$$

If M_{RP} says $'YES'$ then, by Definition 1, we are guaranteed that $x \in L$. Notice that it can happen that $x \in L$ and $M_{RP}(x) = 0$ but not the other way around (There are 1's in the probability space $M(x)$ when $x \in L$, but the probability space $M(x)$ when $x \notin L$ is all zeroes. So if $M(x)$ returns $'YES'$, we know that the first probability space is the case).
In a similar way, if M_{coRP} says $'NO'$ then, by Definition 2, we are guaranteed that $x \notin L$. Thus we never get a wrong answer.

If $x \in L$ then, by Definition 1, we will get a $'YES'$ answer form M_{RP} and hence from M' with probability greater than $\frac{1}{2}$. If $x \notin L$ than, by Definition 2, we will get a $'NO'$ answer form M_{coRP} and hence from M' with probability greater than $\frac{1}{2}$.
So in either cases we can be sure that M' returns a definite (not $\perp$) and correct answer with probability greater than $\frac{1}{2}$.

The conclusion is that M' is indeed a ZPP machine so $RP \bigcap coRP \subseteq ZPP$ and, together with the previous part, we conclude that $RP \bigcap coRP = ZPP$. ■

Summing what we have seen so far we can write the following relations

$$P \subseteq ZPP \subseteq RP \subseteq BPP$$

It is believed that $BPP = P$ so there is no real help that randomized computations can contribute when trying to solve search problems. Also if the belief is true then all the distinctions between the above classes are of no use.

6 Randomized space complexity

$\boxed{7.18} \Rightarrow$

Like we did with NL, we also define randomized space classes. Here also, it is possible to consider both the online and off-line models and we will work with the online model.

6.1 The definition

Definition 9 *For any function $S : \mathsf{N} \to \mathsf{N}$*

$$RSPACE(S) \stackrel{\text{def}}{=} \left\{ L \subseteq \{0,1\}^* \;\middle|\; \begin{array}{l} \textit{There exists a randomized Turing machine } M \\ \textit{s.t. for any input } x \in \{0,1\}^* \\ x \in L \Rightarrow Prob[M(x) = 1] \geq \frac{1}{2} \\ x \notin L \Rightarrow Prob[M(x) = 0] = 0 \\ \textit{and } M \textit{ uses at most } S(|x|) \textit{ space} \\ \textit{and } exp(S(|x|)) \textit{ time.} \end{array} \right\}$$

We are interested in the case where the space is logarithmic. The class which put the logarithmic space restriction is RL.

Definition 10 $RL \stackrel{\text{def}}{=} RSPACE(log)$

The time restriction is very important. Let us see what happens if we don't put the time restriction in Definition 9.

Definition 11 *For any function $S : \mathsf{N} \to \mathsf{N}$*

$$badRSPACE(S) \stackrel{\text{def}}{=} \left\{ L \subseteq \{0,1\}^* \;\middle|\; \begin{array}{l} \textit{There exists a randomized Turing machine } M \\ \textit{s.t. for any input } x \in \{0,1\}^* \\ x \in L \Rightarrow Prob[M(x) = 1] \geq \frac{1}{2} \\ x \notin L \Rightarrow Prob[M(x) = 0] = 0 \\ \textit{and } M \textit{ uses at most } S(|x|) \textit{ space} \\ \text{(no time restrictions!)} \end{array} \right\}$$

Proposition 6.1 badRSPACE(S) = NSPACE(S)

Proof: We start with the easy direction. Let $L \in badRSPACE(S)$. If $x \in L$ then there are many witnesses but one is enough. On the other hand for $x \notin L$ there are no witness.

The other direction is the interesting one. Suppose $L \in NSPACE(S)$. Let M be the Non-deterministic Turing machine which decides L in space $S(|x|)$. Recall that for every $x \in L$ there exists an accepting computation of M on input x which halts within $exp(S(|x|))$ steps (see previous lectures!). Then if $x \in L$ there exist r of length $exp(S(|x|))$, so that $M(x, r) = 1$ (here r denotes the offline non-deterministic guesses used by M). Thus, selecting r uniformly among the strings of length $exp(S(|x|))$, the probability that $M(x, r) = 1$ is at least $2^{-exp(S(|x|))}$. So if we repeatedly invoke $M(x, .)$ on random r's, we can expect that after $2^{exp(S(|x|))}$ tries we will see an accepting computation (assuming all the time that $x \in L$).

> Oded's Note: *Note that the above intuitive suggestion already abuses the fact that badRSPACE has no time bounds. We plan to run in expected time which is* double exponential *in the space bound; whereas the good definition of RSPACE allows only time exponential in the space bound.*

So we want to run M on x and a newly randomly selected r (of length $exp(S(|x|))$) for about $2^{exp(S(|x|))}$ times and accept iff M accepts in one of these tries. A naive implementation is just to do so. But this requires holding a counter capable of counting upto $t \stackrel{\text{def}}{=} 2^{exp(S(|x|))}$, which means using space $exp(S(|x|))$ (which is much more than we are allowed). So we have the basic idea which is good but still have a problem how to count. The solution will be to use a "randomized counter" that will only use $S(|x|)$ space.

The randomized counter is implemented as follows. We "flip" $k = \log_2 t$ coins. If all are heads then we will stop otherwise we go on. The expected number of tries is $2^{-k} = t$, exactly the number of tries we wanted to have. But this randomized counter requires only a real counter capable of counting upto k, and so can be implemented in space $\log_2 k = \log_2 \log_2 t = S(|x|)$. ■

Clearly,

Claim 6.2 $L \subseteq RL \subseteq NL$

6.2 Undirected Graph Connectivity is in RL

7.19 ⇒

In the previous lecture we saw that directed connectivity is NL-Complete. We will now show in brief that undirected connectivity is in RL. The problem is defined as follows.

Input: An undirected graph G and two vertices s and t.

Task: Find if there is a path between s and t in G.

Claim 6.3 *Let n denote the number of vertices in the graph. Then, with probability at least $\frac{1}{2}$, a random walk of length $8n^3$ starting from s visits all vertices in the connected component of s.*

By a random walk, we mean a walk which iteratively selects at random a neighbour of the current vertex and moves to it.

Proof sketch: In the following, we consider the connected component of vertex s, denoted $G' = (V', E')$. For any edge, (u, v) (in E'), we let $T_{u,v}$ be a random variable representing the number of steps taken in a random walk starting at u until v is first encountered. It is easy to see that $\mathrm{E}[T_{u,v}] \leq 2|E'|$. Also, letting $\mathrm{cover}(G')$ be the expected number of steps in a random walk starting at s and ending when the last of the vertices of V' is encountered, and C be any directed cycle which visits all vertices in G', we have

$$\begin{aligned}\mathrm{cover}(G') &\leq \sum_{(u,v)\in C} \mathrm{E}[T_{u,v}] \\ &\leq |C| \cdot 2|E'|\end{aligned}$$

Letting C be a traversal of some spanning tree of G', we conclude that $\mathrm{cover}(G') < 4 \cdot |E'| \cdot |V'|$. Thus, with probability at least $1/2$, a random walk of length $8 \cdot |E'| \cdot |V'|$ starting at s visits all vertices of G'. ■

The algorithm for deciding undirected connectivity is now obvious: Just take a "random walk" of length $8n^3$ starting from vertex s and see if t is encountered. The space requirement is merely a register to hold the current vertex (i.e., $\log n$ space) and a counter to count upto $8n^3$ (again $(\log n)$ space). Furthermore, the use of a counter guarantees that the running time of the algorithm is exponential in its (logarithmic) space bound. The implementation is straightforward

1. Set *counter* $= 0$ and $v = s$. Compute n (the number of vertices in the graph).

2. Uniformly select a neighbour u of v.

3. If $u = t$ then halt and accept, else set $v = u$ and $counter = counter + 1$.

4. If $counter = 8n^3$ then halt and reject, else goto Step (2).

Cleraly, if s is connected to t then, by the above claim, the algorithm accepts with probability at least 1/2. On the other hand, the algorithm alwas rejects if s is not connected to t. Thus, UNdirected graph CONNectivity (UNCONN) is in RL.

Note that the straightforward adaptation of the above algorithm to the directed case (i.e., directed graph connectivity considered in previous lecture) fails: Consider, for example, a directed graph consisting of a directed path $1 \to 2 \to \cdots \to n$ augmented by directed edges going from every vertex $i > 1$ to vertex 1. An algorithm which tries to take a directed random walk starting from vertex 1 is highly unlikely to reach vertex n in poly(n) many steps. Loosely speaking, this is the case since in each step from a vertex $i > 1$, we move towards vertex n with probability 1/2, but otherwise return to vertex 1. The fact that the above algorithm fails should not come as a great surprise, as the directed connectivity problem is NL-complete and so placing it in RL will imply $NL = RL$.

> Oded's Note: *$NL = RL$ is not considred as unlikely as $NP = RP$, but even if $NL = RL$ proving this seems very hard.*

Bibliographic Notes

Probabilistic Turing Machines and corresponding complexity classes (including $\mathcal{BPP}, \mathcal{RP}, \mathcal{ZPP}$ and $\mathcal{PP}$) were first defined by Gill [2]. The proof that NSPACE equals badRSPACE (called RSPACE in [2]), as well as the techinque of a randomized counter is from [2].

The robusteness of the various classes under various thresholds was established above using straightforward amplifications (i.e., running the algorithm several times with independent random choices). Randomness-efficient amplification methods have been extensively studied since the mid 1980's. See Section 3.6 in [3].

The random-walk algorithm for deciding undirected connectivity is due to Aleliunas *et. al.* [1]. Other examples of randomized algorithms can be found in Appendix B.1 of [3]. We specifically recommend the following examples

- Testing primality (B.1.5): This BPP algorithm is different from the famous $coRP$ algorithm for recognizing the set of primes.

- Finding a perfect matching (B.1.2): This algorithm is arguably simpler than known deterministic polynomial-time algorithms.

- Finding minimum cuts in graphs (B.1.7): This algorithm is arguably simpler than known deterministic polynomial-time algorithms.

A much more extensive treatment of randomized algorithm is given in [4].

1. R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th FOCS*, pages 218–223, 1979.

2. J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, Vol. 6(4), pages 675–695, 1977.

3. O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics series (Vol. 17), *Springer*, 1998. Copies have been placed in the faculty's library.

4. R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.