Computational complexity

lecture 10

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a machine M with a source of random bits, working in polynomial time, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Examples of randomized algorithms

An example of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: “is n composite?” is in **RP**.]

Examples of randomized algorithms

An example of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: "is n composite?" is in **RP**.]

History:

- Clearly primality$\in$**coNP**: a nontrivial divisor is a witness, but it is difficult to find it.
- For years it was not known how to check that a number is prime (even before the era of computers, this was an interesting problem)

Examples of randomized algorithms

An example of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: "is n composite?" is in **RP**.]

History:

- Clearly primality$\in$**coNP**: a nontrivial divisor is a witness, but it is difficult to find it.
- For years it was not known how to check that a number is prime (even before the era of computers, this was an interesting problem)
- Pratt 1975, primality$\in$**NP** (i.e., $\in$**NP**$\cap$**coNP**) – certificate for primality that can be checked in polynomial time
- probabilistic tests discovered, showing primality$\in$**coRP** (Solovay-Strassen test 1977, Miller-Rabin test 1976-1980)

Examples of randomized algorithms

An example of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: "is n composite?" is in **RP**.]

History:

- Clearly primality∈**coNP**: a nontrivial divisor is a witness, but it is difficult to find it.
- For years it was not known how to check that a number is prime (even before the era of computers, this was an interesting problem)
- Pratt 1975, primality∈**NP** (i.e., ∈**NP**∩**coNP**) – certificate for primality that can be checked in polynomial time
- probabilistic tests discovered, showing primality∈**coRP** (Solovay-Strassen test 1977, Miller-Rabin test 1976-1980)
- Adleman-Pomerance-Rumely 1983: determin. alg., $|n|^{O(\ln \ln |n|)}$
- Adleman-Huang 1992: primality∈**RP**∩**coRP**
- Agrawal-Kayal-Saxena 2002: primality∈**P**, best known time: $O(|n|^6)$
- in practice, probabilistic tests are used (the determ. alg. is too slow)

Examples of randomized algorithms

An example of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

Miller-Rabin test:

- let $n-1=2^s \cdot d$
- choose randomly $a \in \{1,...,n-1\}$
- If $a^d \not\equiv 1 \pmod n$ and $a^{2^r d} \not\equiv -1 \pmod n$ for all $r \in \{0,...,s-1\}$, say "composite", otherwise say "prime"

Examples of randomized algorithms

An example of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

Miller-Rabin test:

- let $n-1=2^s \cdot d$
- choose randomly $a \in \{1,...,n-1\}$
- If $a^d \not\equiv 1 \pmod n$ and $a^{2^r d} \not\equiv -1 \pmod n$ for all $r \in \{0,...,s-1\}$, say "composite", otherwise say "prime"

➔ For prime numbers, the algorithm always says "prime"

➔ For composite numbers, the algorithm says "composite" with probability $\geq 3/4$ (probability of error $\leq 1/4$)

➔ We skip a proof

Examples of randomized algorithms

An example of an algorithm in **RP**: check that a polynomial (given in an implicit form) is nonzero?

Formally, we are given an arithmetic circuit, where we have gates +, *, –, and input gates corresponding to variables – notice that such a circuit always encodes a polynomial with integer coefficients. We ask whether there is a valuation of variables for which the result of the circuit is nonzero.
The problem can be considered over rational (real) numbers, or over some finite field.

Examples of randomized algorithms

An example of an algorithm in **RP**: check that a polynomial (given in an implicit form) is nonzero?

Formally, we are given an arithmetic circuit, where we have gates +, *, –, and input gates corresponding to variables – notice that such a circuit always encodes a polynomial with integer coefficients. We ask whether there is a valuation of variables for which the result of the circuit is nonzero.
The problem can be considered over rational (real) numbers, or over some finite field.

In both cases, we do not know whether the problem is in **P**.

An idea for a probabilistic algorithm:
take a random valuation of variables, and check the result

Does it make sense?
Yes, if every nonzero polynomial is nonzero for a majority of valuations of variables.

Examples of randomized algorithms ($\star$)

Lemma (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,...,r_k \in S$ randomly.

Then $Pr[p(r_1,...,r_k)=0] \leq d / |S|$

Examples of randomized algorithms ($\star$)

Lemma (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,...,r_k \in S$ randomly.

Then $Pr[p(r_1,...,r_k)=0] \leq d / |S|$

Proof: induction over k

$k=1 \rightarrow$ Bezout's theorem: a polynomial of degree d has $\leq d$ zeroes

Examples of randomized algorithms ($\star$)

Lemma (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,...,r_k \in S$ randomly.

Then $Pr[p(r_1,...,r_k)=0] \leq d / |S|$

Proof: induction over k

$k=1 \rightarrow$ Bezout's theorem: a polynomial of degree d has $\leq d$ zeroes

Induction step: write p as a polynomial of the variable x_1:

$$p(x_1,...,x_k)=\sum_{i=0}^{d} x_1^i \cdot p_i(x_2,...,x_k)$$

Take the greatest i, for which p_i is nonzero (exists, since $p \neq 0$).

The degree of p_i is $\leq d$-i. From the induction assumption:

$Pr[p_i(r_2,...,r_k)=0] \leq (d-i) / |S|$

Examples of randomized algorithms (⋆)

Lemma (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,...,r_k \in S$ randomly.

Then $Pr[p(r_1,...,r_k)=0] \le d / |S|$

Proof: induction over k

$k=1 \rightarrow$ Bezout's theorem: a polynomial of degree d has $\le d$ zeroes

Induction step: write p as a polynomial of the variable x_1:

$$p(x_1,...,x_k)=\sum_{i=0}^{d} x_1^i \cdot p_i(x_2,...,x_k)$$

Take the greatest i, for which p_i is nonzero (exists, since $p \neq 0$).

The degree of p_i is $\le d$-i. From the induction assumption:

$Pr[p_i(r_2,...,r_k)=0] \le (d-i) / |S|$

If $p_i(r_2,...,r_k) \neq 0$, then $p(x_1,r_2,...,r_k)$ is of degree i , so

$Pr[p(r_1,...,r_k)=0 \mid p_i(r_2,...,r_k) \neq 0] \le i / |S|$

This is enough, since $Pr[A] \le Pr[B] + Pr[A|B^c]$ for arbitrary events A,B

Examples of randomized algorithms ($\star$)

The Schwartz-Zippel lemma shows that the question whether a polynomial is nonzero is in **RP**, when we consider it over a large finite field.

- We see that a circuit with n gates defines a polynomial of total degree at most 2^n. If there are k variables, it is enough to pick k random numbers from $0,\ldots,10\cdot 2^n$ (it requires $O(kn)$ bits), compute the value of the polynomial (i.e., simulate the circuit), and accept if the result is nonzero. We are wrong with probability ≤ 0.1.

Examples of randomized algorithms ($\star$)

The Schwartz-Zippel lemma shows that the question whether a polynomial is nonzero is in **RP**, when we consider it over a large finite field.

- We see that a circuit with n gates defines a polynomial of total degree at most 2^n. If there are k variables, it is enough to pick k random numbers from $0,\ldots,10\cdot 2^n$ (it requires $O(kn)$ bits), compute the value of the polynomial (i.e., simulate the circuit), and accept if the result is nonzero. We are wrong with probability ≤ 0.1.

Over the field of rationals (or if the considered finite field is too large) there is an additional problem: how to evaluate the circuit in polynomial time? Even if the final result is 0, intermediate results can be very long.

- Solution: pick a random number m from $2,\ldots,2^{2n}$, and compute everything modulo m

  Why does this work well?

Examples of randomized algorithms ($\star$)

- We take random m from $2,\ldots,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,\ldots,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, we will prove that with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)

Examples of randomized algorithms ($\star$)

- We take random m from $2,\ldots,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,\ldots,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, we will prove that with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)
- The number $\pi(N)$ of prime numbers smaller than N satisfies:

$$\lim_{N\to\infty} \frac{\pi(N)}{N/ln(N)}=1$$

- Thus m is prime with probability at least $1/(5n)$ (for large enough n)

Examples of randomized algorithms ($\star$)

- We take random m from $2,\ldots,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,\ldots,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, we will prove that with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)
- The number $\pi(N)$ of prime numbers smaller than N satisfies:

$$\lim_{N\to\infty} \frac{\pi(N)}{N/\ln(N)}=1$$

- Thus m is prime with probability at least $1/(5n)$ (for large enough n)
- We have $Y\leq(10\cdot 2^n)^{2^n}$
- The number of prime divisors of Y is at most logarithmic, namely $\leq 5n2^n$
- A randomly chosen m is among these divisors with probability $\leq 5n2^n/2^{2n}<1/(10n)$
- Thus m is prime and is NOT a divisor of Y with probability $>1/(10n)$

Examples of randomized algorithms ($\star$)

- We take random m from $2,\ldots,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,\ldots,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, then with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)
- This is still not enough – our algorithm fails with prob. $\leq 1-1/(10n)$
- Let us repeat the whole algorithm n times. This is enough, since

$$\lim_{n\to\infty} (1-1/n)^n = 1/e$$

- For large n this is <0.5
- We have finitely many „small” n, where the error is a constant; we can decrease it using the standard amplification

Amplification

As a side effect, we obtain the following stronger version of amplification:

Fact

Suppose that a language L is recognized by a machine M with a source of random bits, working in polynomial time, and such that for some polynomial $p(n)$:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 1/p(n)$ (error probability almost 1)
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Then $L \in$ **RP**.

Proof

- We execute the machine $p(n)$ times. This is enough, since

  $\lim_{n \to \infty} (1-1/p(n))^{p(n)} = 1/e$

- For large n this is < 0.5
- We have finitely many „small” n, we can deal with them somehow

Amplification

We can go even further:

Fact

Let $L \in \mathbf{RP}$. Then, for every polynomial $q(n)$ there is a machine M with a source of random bits, working in polynomial time, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 1 - 1/2^{q(n)}$ (error probability exponentially small)
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Proof

- We take a machine that makes a mistake with probability $< 1/2$, and we run it $q(n)$ times

Examples of randomized algorithms

Perfect matching in a bipartite graph:

input: bipartite graph $G=(V_1,V_2,E)$, where $|V_1|=|V_2|$

question: is there a perfect matching in G?

- several deterministic algorithms are known for detecting if a perfect matching exists
- we present here a randomized algorithm

Examples of randomized algorithms

Perfect matching in a bipartite graph:
input: bipartite graph $G=(V_1,V_2,E)$, where $|V_1|=|V_2|=n$

question: is there a perfect matching in G?

Solution

- Consider the $n \times n$ matrix X whose entry X_{ij} is a variable x_{ij} if $(i,j) \in E$, and 0 otherwise.
- Recall that the determinant $det(X)$ is

$$det(X)=\sum_{\sigma \in S_n}(-1)^{sign(\sigma)}\prod_{i=1}^{n}X_{i,\sigma(i)}$$

- Every permutation in S_n is a potential perfect matching.
- A perfect matching exists iff the determinant is nonzero.

Examples of randomized algorithms

Perfect matching in a bipartite graph:
input: bipartite graph $G=(V_1,V_2,E)$, where $|V_1|=|V_2|=n$

question: is there a perfect matching in G?

Solution

- Consider the $n \times n$ matrix X whose entry X_{ij} is a variable x_{ij} if $(i,j) \in E$, and 0 otherwise.
- Recall that the determinant $det(X)$ is

$$det(X)=\sum_{\sigma \in S_n}(-1)^{sign(\sigma)}\prod_{i=1}^{n}X_{i,\sigma(i)}$$

- Every permutation in S_n is a potential perfect matching.
- A perfect matching exists iff the determinant is nonzero.
- The determinant itself, as a polynomial, is large.
- But for specific numbers substituted for the variables, we can compute in quickly (as fast as matrix multiplication).
- Randomized algorithm: substitute something for the variables, and check that the determinant is nonzero.
- Advantage: the algorithm parallelizes (matrix_determinant $\in$ **NC**)

Class PP

Class **PP** (probabilistic polynomial): like **RP**, but:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow Pr_s[(w,s) \in L_M] < 0.5$

Intuition: acceptance by voting of witnesses

Class PP

Class **PP** (probabilistic polynomial): like **RP**, but:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow Pr_s[(w,s) \in L_M] < 0.5$

Intuition: acceptance by voting of witnesses

Advantages of this class:

- errors allowed on both sides ⇒ closed under complement
- a „syntactic” class ⇒ has a complete problem MAJSAT:
  is a given boolean formula satisfied by at least half of possible valuations?

Class PP

Class **PP** (probabilistic polynomial): like **RP**, but:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow Pr_s[(w,s) \in L_M] < 0.5$

Intuition: acceptance by voting of witnesses

Advantages of this class:

- errors allowed on both sides ⇒ closed under complement
- a „syntactic” class ⇒ has a complete problem MAJSAT: is a given boolean formula satisfied by at least half of possible valuations?

Disadvantages of this class – it is too large:

- in practice: maybe M gives probabilities close to 0.5, so running it (even many times) does not tell us too much about the result
- **NP**$\subseteq$**PP** – tutorials

Class BPP

Class **BPP** (bounded probabilistic polynomial): errors allowed on both sides, but only small errors:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 3/4$
- $w \notin L \Rightarrow Pr_s[(w,s) \in L_M] \leq 1/4$

In other words: the probability of an error is $\leq 1/4$, both sides

Class BPP

Class **BPP** (bounded probabilistic polynomial): errors allowed on both sides, but only small errors:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 3/4$
- $w \notin L \Rightarrow Pr_s[(w,s) \in L_M] \leq 1/4$

In other words: the probability of an error is $\leq 1/4$, both sides

Remarks:

- easy fact: **RP**$\subseteq$**BPP**$\subseteq$**PP**
- we are not aware of any problem, which is in **BPP**, and about which we do not know whether it is in **RP** or in **coRP**
- **BPP** is a good candidate for the class of these problems, which can be quickly solved "in practice"
- open problem: how is **BPP** related to **NP**?
- tutorials: if **NP**$\subseteq$**BPP**, then **NP**$\subseteq$**RP** (i.e., **NP**=**RP**)
- conjecture (open problem): **BPP**=**P** ("randomization doesn't add anything")

Class BPP

Amplification for **BPP**:
instead of the error $1/4$, one can take any number $p \in (0,1/2)$

Proof:

Let the original error probability be $p<1/2$.
We run the algorithm $2m+1$ times, and we take the decision of majority. The probability of error decreases to:

$$\sum_{i=0}^{m} \underbrace{\binom{2m+1}{i}(1-p)^i p^{2m+1-i}}_{\text{precisely } i \text{ correct answers}} \leq \sum_{i=0}^{m} \binom{2m+1}{i}(1-p)^m p^{m+1} = 2^{2m}(1-p)^m p^{m+1} \leq (4p(1-p))^m$$

Class BPP

Amplification for **BPP**:
instead of the error $1/4$, one can take any number $p \in (0,1/2)$

Proof:

Let the original error probability be $p<1/2$.
We run the algorithm $2m+1$ times, and we take the decision of majority. The probability of error decreases to:

$$\sum_{i=0}^{m} \underbrace{\binom{2m+1}{i}(1-p)^i p^{2m+1-i}}_{\text{precisely } i \text{ correct answers}} \leq \sum_{i=0}^{m} \binom{2m+1}{i}(1-p)^m p^{m+1} = 2^{2m}(1-p)^m p^{m+1} \leq (4p(1-p))^m$$

Remark:

As for **RP**, we can prove a stronger version: we can start from an algorithm with error probability $1-1/p(n)$ (very large), and obtain an algorithm with error probability $1/2^{q(n)}$, for polynomials $p(n)$, $q(n)$.
It is enough to take as m appropriate polynomial.

Class ZPP

Two types of randomized algorithms:

- **Monte Carlo**: the algorithm is always fast, usually the answer is correct – **RP**, **BPP**, **PP**
- **Las Vegas**: the answer is always correct, usually the algorithm is fast – **ZPP**

Class **ZPP** (zero-error probabilistic polynomial time): problems that can be solved in expected polynomial time.

Class ZPP

Two types of randomized algorithms:

- **Monte Carlo**: the algorithm is always fast, usually the answer is correct – **RP**, **BPP**, **PP**
- **Las Vegas**: the answer is always correct, usually the algorithm is fast – **ZPP**

Class **ZPP** (zero-error probabilistic polynomial time): problems that can be solved in expected polynomial time.

How do we compute the expected running time?

- The machine has access to an infinite tape with random bits
- Every bit is chosen independently (0 or 1 with probability 0.5)
- I.e., a computation that halts after reading k random bits has probability 2^{-k}
- The probability of looping forever is required to be 0

Class ZPP

Two types of randomized algorithms:

- **Monte Carlo**: the algorithm is always fast, usually the answer is correct – **RP**, **BPP**, **PP**
- **Las Vegas**: the answer is always correct, usually the algorithm is fast – **ZPP**

Class **ZPP** (zero-error probabilistic polynomial time): problems that can be solved in expected polynomial time.

How do we compute the expected running time?

- The machine has access to an infinite tape with random bits
- Every bit is chosen independently (0 or 1 with probability 0.5)
- I.e., a computation that halts after reading k random bits has probability 2^{-k}
- The probability of looping forever is required to be 0

Tutorials: **ZPP**=**RP**∩**coRP** (i.e, Las Vegas algorithms can be changed to Monte Carlo algorithms)

Non-uniform derandomization

Theorem (Adleman 1978):
BPP⊆P/poly

Remark: this theorem says that for every language in **BPP** there is a sequence of circuits of polynomial size, which recognizes it.
If this sequence would be uniform, the language would be in **P** (it would be possible to derandomize every language from **BPP**).
This is an open problem, though, so the sequence of circuits obtained in our proof should be “strange”, i.e., difficult to compute.

Non-uniform derandomization

Theorem (Adleman 1978):
BPP⊆P/poly

Proof:

- Suppose that M recognizes L with error probability $\leq 1/4$.
- On input of length n we repeat the computation $2(3n)+1$ times; the error decreases to $\leq (4p(1-p))^{3n} = (3/4)^{3n}$ (running time is still polynomial, number of random bits polynomial)
- The probability that a random sequence of bits gives an incorrect answer for a fixed input of length n is $\leq (3/4)^{3n}$, thus the probability that a random sequence of bits gives an incorrect answer for at least one input of length n is $\leq 2^n (3/4)^{3n} = (27/32)^n < 1$
- Thus there exists a sequence of bits, which gives a correct answer for every input of length n – we take this sequence of bits as the advice

Derandomization

Generally, we only know that **BPP**$\subseteq$**PSPACE**, but some algorithms can be derandomized, and there are some techniques for this.

Consider the example: approximation of MAX-CUT – for an undirected graph $G=(V,E)$ compute a subset $S\subseteq V$ such that

$$cut(S)=\{\{u,v\}\in E \mid u\in S,\ v\notin S\}$$

is largest possible.

The decision problem (is $cut(S)\geq k$ some S?) is **NP**-complete.

Derandomization

Generally, we only know that **BPP**$\subseteq$**PSPACE**, but some algorithms can be derandomized, and there are some techniques for this.

Consider the example: approximation of MAX-CUT – for an undirected graph $G=(V,E)$ compute a subset $S\subseteq V$ such that

$$cut(S)=\{\{u,v\}\in E \mid u\in S,\ v\notin S\}$$

is largest possible.

The decision problem (is $cut(S)\geq k$ some S?) is **NP**-complete.

There is a simple randomized algorithm, which computes S so that the expected value of $cut(S)$ is $\geq |E|/2$: for every node, take it to S with probability $1/2$:

- Every edge is in cut with probability $1/2$ (because the choices are independent), thus by linearity of the expected value, the expected size of cut is $|E|/2$.

Derandomization

There is a simple randomized algorithm, which computes S so that the expected value of $cut(S)$ is $\geq |E|/2$: for every node, take it to S with probability $1/2$:

- Every edge is in cut with probability $1/2$ (because the choices are independent), thus by linearity of the expected value, the expected size of cut is $|E|/2$.

And how can be bound (from below) the probability that the resulting cut has size at least $|E|/2$?

- The worst case is when rarely (in k cases, for some k) the algorithm returns a cut of size $|E|$, and often (in $2^{|V|}$-k cases) it returns a cut of size $|E|/2$-1. We have an inequality:
  $k|E|+(2^{|V|}-k)(|E|/2-1)\geq 2^{|V|}|E|/2$
  This gives: $k(|E|/2+1)\geq 2^{|V|} \Rightarrow k/2^{|V|}\geq 2/(|E|+2)$
- Thus the probability that the resulting cut has size $\geq |E|/2$ is $\geq 1/|E|$ (assuming $|E|\geq 2$). By repeating the algorithm a linear number of times, the probability can be changed to a constant, since $\lim_{n\to\infty}(1-1/n)^n = 1/e$

Derandomization

Thus: we have a randomized algorithm, giving with probability $\geq 1/2$ a cut of size $\geq |E|/2$.
How can we derandomize it, i.e., give a deterministic algorithm computing S for which $cut(S) \geq |E|/2$?

We will show two concepts:

1) The method of conditional expected values

- In order to derandomize the algorithm, we should be able to find a "good" witness – in our case such that $cut(S) \geq |E|/2$
- For a fixed sequence of guesses $b_1,...,b_k$, let $E(b_1,...,b_k)$ be the expected value of the size of a cut in the case when the first k bits are $b_1,...,b_k$. It is clear that:

  $E(b_1,...,b_k) = E(b_1,...,b_k,0)/2 + E(b_1,...,b_k,1)/2$

  so either $E(b_1,...,b_k,0)$ or $E(b_1,...,b_k,1)$ is $\geq E(b_1,...,b_k)$
- Assume that we can deterministically compute $E(b_1,...,b_k)$.

  In such a situation, we can proceed "greedly": we choose this b_{k+1} which gives larger expected size of a cut.

Derandomization

1) The method of conditional expected values

- Then we have that:
  $E(b_1,...,b_n) \geq E(b_1,...,b_{n-1}) \geq ... \geq E(b_1) \geq E() = |E|/2$
- Thus at the end we obtain a cut of size $\geq |E|/2$.
- Generally, it is not always possible to quickly compute $E(b_1,...,b_k)$, but for MAXCUT we can do it: if we have chosen nodes from S, and we have discarded nodes from T, and X is the set of those edges in which at least one end is neither in S nor in T, then $E(b_1,...,b_k) = |cut(S,T)| + |X|/2$