Computational complexity

lecture 10

Probabilistic machines

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

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

As s we can take sequences of length $T(n)$, or infinite sequences, does not matter.

Intuition: a word is in L, if at least half of possible witnesses confirm this (but there are no witnesses for words not in L)

In other words: if a word is not in L, we will certainly reject;
if it is in L, then choosing transitions randomly, we will accept with probability at least 0.5

Amplification

<u>Fact</u> (amplification)

Instead of 0.5, in the definition of **RP** we can take any constant between 0 and 1.

Amplification

At the end of the previous lecture, as a side effect, we have observed the following stronger version of amplification:

<u>Fact</u>

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**.

<u>Proof</u>

- 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:

<u>Fact</u>

Let $L \in$ **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:

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

<u>Proof</u>

- 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?

<u>Solution</u>

- 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**⊆**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**⊆**BPP**⊆**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**⊆**BPP**, then **NP**⊆**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:

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

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:

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

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$\subseteq$**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$\subseteq$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 $b_1,\ldots,b_k$, let $E(b_1,\ldots,b_k)$ be the expected value of the size of a cut in the case when the first k bits are $b_1,\ldots,b_k$. It is clear that:

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

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

  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:
$E(b_1,\ldots,b_n) \geq E(b_1,\ldots,b_{n-1}) \geq \ldots \geq E(b_1) \geq E() = |E|/2$
- Thus at the end we obtain a cut of size $\geq |E|/2$.
- Generally, it is not always possible to quickly compute $E(b_1,\ldots,b_k)$, 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
$E(b_1,\ldots,b_k) = |cut(S,T)| + |X|/2$

Derandomization

2) <u>The method of pairwise-independent variables</u>

- We were assuming that the random bits are all independent. But in the algorithm for MAXCUT it is enough to assume that they are pairwise independent, i.e., that $Pr[b_i=b_j]=1/2$ for all $i \neq j$
- Fact: having $log(n)$ independent random bits, one can produce n pairwise independent bits. Namely, for every nonempty subset of bits we take the XOR of these bits.
- On the other hand, all combinations of $log(n)$ bits can be browsed in polynomial time.