CoCoNuT - Complexity - Lecture 5

N.P. Smart

Dept of Computer Science
University of Bristol,
Merchant Venturers Building

September 29, 2014

Outline

Randomised Computation

The Class BPP

The Class RP

The Class ZPP

De-Randomization

Randomised Computation

Allow a TM to make **random** choices of the next step

Example. *(Polynomial identities)*

- Given: $Q(x_1, \ldots, x_n)$, a polynomial in n variables.
- Decide: Is Q identically zero?

Fact. Let $Q(x_1, \ldots, x_n)$ have degree $\leq d$ in every variable and Q not identically zero.

Then for any set S of values, with $|S| \geq 2 \cdot n \cdot d$, the number of tuples $(a_1, \ldots, a_n) \in S^n$ s.t. $Q(a_1, \ldots, a_n) = 0$, is at most $\frac{1}{2}|S|^n$.

Randomised Computation

Allow a TM to make **random** choices of the next step

Example. *(Polynomial identities)*

- Given: $Q(x_1, \ldots, x_n)$, a polynomial in n variables.
- Decide: Is Q identically zero?

Fact. Let $Q(x_1, \ldots, x_n)$ have degree $\leq d$ in every variable and Q not identically zero.

Then for any set S of values, with $|S| \geq 2 \cdot n \cdot d$, the number of tuples $(a_1, \ldots, a_n) \in S^n$ s.t. $Q(a_1, \ldots, a_n) = 0$, is at most $\frac{1}{2}|S|^n$.

Randomised Computation

Allow a TM to make **random** choices of the next step

Example. *(Polynomial identities)*

- Given: $Q(x_1, \ldots, x_n)$, a polynomial in n variables.
- Decide: Is Q identically zero?

Fact. Let $Q(x_1, \ldots, x_n)$ have degree $\leq d$ in every variable and Q not identically zero.

Then for any set S of values, with $|S| \geq 2 \cdot n \cdot d$, the number of tuples $(a_1, \ldots, a_n) \in S^n$ s.t. $Q(a_1, \ldots, a_n) = 0$, is at most $\frac{1}{2}|S|^n$.

Poly Identities

Checking $Q(x_1, \ldots, x_n)$ is identically zero:

Algoithm R :

On input Q

1. Choose S with $|S| > 2 \cdot n \cdot d$.
2. Choose $(a_1, \ldots, a_n)$ at random from $S \times \ldots \times S$
3. If $Q(a_1, \ldots, a_n) \neq 0$ output *"reject"*

   Otherwise output *"accept"*.

If Q **is zero** then R outputs *accept* with probability 1

If Q **is not zero** then R outputs *accept* with probability $\leq 1/2$

Amplification. Repeat Steps 2 and 3 k times

...

If Q **is not zero** then R outputs *accept* with probability $\leq 1/(2^k)$

Poly Identities

Checking $Q(x_1, \ldots, x_n)$ is identically zero:

Algoithm R :

On input Q

1. Choose S with $|S| > 2 \cdot n \cdot d$.
2. Choose $(a_1, \ldots, a_n)$ at random from $S \times \ldots \times S$
3. If $Q(a_1, \ldots, a_n) \neq 0$ output *"reject"*

   Otherwise output *"accept"*.

If Q **is zero** then R outputs *accept* with probability 1

If Q **is not zero** then R outputs *accept* with probability $\leq 1/2$

Amplification. Repeat Steps 2 and 3 k times

...

If Q **is not zero** then R outputs *accept* with probability $\leq 1/(2^k)$

Poly Identities

Checking $Q(x_1, \ldots, x_n)$ is identically zero:

Algoithm R :

On input Q

1. Choose S with $|S| > 2 \cdot n \cdot d$.
2. Choose $(a_1, \ldots, a_n)$ at random from $S \times \ldots \times S$
3. If $Q(a_1, \ldots, a_n) \neq 0$ output *"reject"*

   Otherwise output *"accept"*.

If Q **is zero** then R outputs *accept* with probability 1

If Q **is not zero** then R outputs *accept* with probability $\leq 1/2$

Amplification. Repeat Steps 2 and 3 k times

...

If Q **is not zero** then R outputs *accept* with probability $\leq 1/(2^k)$

Probabilistic TMs

Defn: PTM

A **probabilistic Turing machine** (PTM) is a NTM in which each non-deterministic step ("coin flip") has *two* legal moves.

- For every computation branch b, let k be the number of coin flips on b. Then

$$\Pr[b] := 1/2^k$$

- Accepting probability:

$$\Pr[M \text{ accepts } w] := \sum_{b \text{ is an accepting branch}} \Pr[b]$$

- Rejecting probability: $1 - \Pr[M \text{ accepts } w]$

PTMs are real devices, NTMs are not

Probabilistic TMs

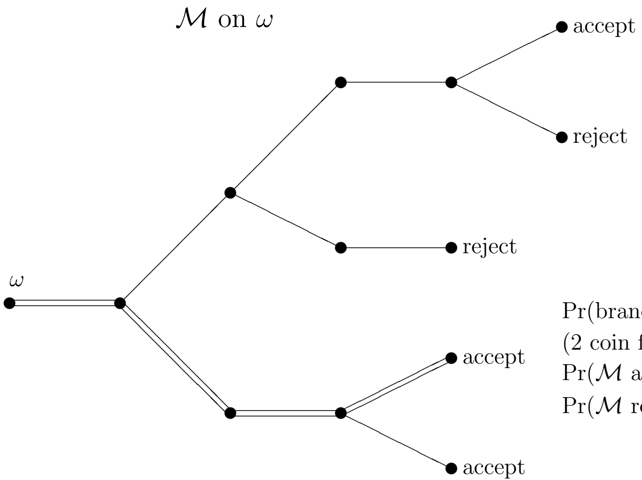
Defn: PTM

A **probabilistic Turing machine** (PTM) is a NTM in which each non-deterministic step ("coin flip") has *two* legal moves.

- For every computation branch b, let k be the number of coin flips on b. Then
$$\Pr[b] := 1/2^k$$
- Accepting probability:
$$\Pr[M \text{ accepts } w] := \sum_{b \text{ is an accepting branch}} \Pr[b]$$
- Rejecting probability: $1 - \Pr[M \text{ accepts } w]$

PTMs are real devices, NTMs are not

Probabilistic TMs



Pr(branch ═══) $= \frac{1}{4}$
(2 coin flips on this branch)
$\Pr(\mathcal{M} \text{ acc. } \omega) = \frac{1}{8} + \frac{1}{4} + \frac{1}{4} = \frac{5}{8}$
$\Pr(\mathcal{M} \text{ rej. } \omega) = \frac{1}{8} + \frac{1}{4} = \frac{3}{8}$

Probabilistic TMs

An alternative definition of PTMs is

- A DTM with two input tapes.
- One input tape is the normal input.
- The second input tape is a read-only sequence of random bits.
- The second tape is called the “random tape”.

We are interested in such PTMs which

- Either run in poly-time (result may be wrong though).
- Or run in expected poly-time (result will be correct though).

As before the question is whether the randomness resource gives us some extra power.

The class BPP (Atlantic City Algorithms)

PTM's Error Probability

A PTM M recognises L with **error probability** ε if

- $w \in L \Rightarrow \Pr[M \text{ accepts } w] \geq 1 - \varepsilon$; and
- $w \notin L \Rightarrow \Pr[M \text{ accepts } w] \leq \varepsilon$

BPP

BPP (bounded-error probabilistic polynomial time) is the class of languages that are recognised by a polynomial-time PTM with error probability 1/3.

The class BPP (Atlantic City Algorithms)

PTM's Error Probability

A PTM M recognises L with **error probability** ε if

- $w \in L \Rightarrow \Pr[M \text{ accepts } w] \geq 1 - \varepsilon$; and
- $w \notin L \Rightarrow \Pr[M \text{ accepts } w] \leq \varepsilon$

BPP

BPP (bounded-error probabilistic polynomial time) is the class of languages that are recognised by a polynomial-time PTM with error probability 1/3.

The class BPP (Atlantic City Algorithms)

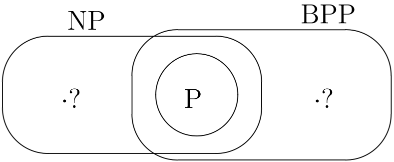
PTM's Error Probability

A PTM M recognises L with **error probability** ε if

- $w \in L \Rightarrow \Pr[M \text{ accepts } w] \geq 1 - \varepsilon$; and
- $w \notin L \Rightarrow \Pr[M \text{ accepts } w] \leq \varepsilon$

BPP

BPP (bounded-error probabilistic polynomial time) is the class of languages that are recognised by a polynomial-time PTM with error probability 1/3.



The class BPP (Atlantic City Algorithms)

Amplification Lemma

Let $0 < \varepsilon_1 \leq \varepsilon_2 < 1/2$.
If L can be recognised by a poly-time PTM with error probability ε_2 then it can be recognised by a poly-time PTM with error prob. ε_1.

Proof is by repeation and applying the Chernoff bound

- The Chernoff bound allows us to estimate the tail of a sum of Bernoulli trials.
- Basically the sum of a bunch of Bernoulli trials concentrates around the mean.
- So the tails vanish to almost zero.

The class BPP (Atlantic City Algorithms)

Amplification Lemma

Let $0 < \varepsilon_1 \leq \varepsilon_2 < 1/2$.
If *L* can be recognised by a poly-time PTM with error probability ε_2 then it can be recognised by a poly-time PTM with error prob. ε_1.

Proof is by repeation and applying the Chernoff bound

- The Chernoff bound allows us to estimate the tail of a sum of Bernoulli trials.
- Basically the sum of a bunch of Bernoulli trials concentrates around the mean.
- So the tails vanish to almost zero.

More BPP

The class of BPP algorithms are sometimes called “Two-Sided Monte Carlo Algorithms”

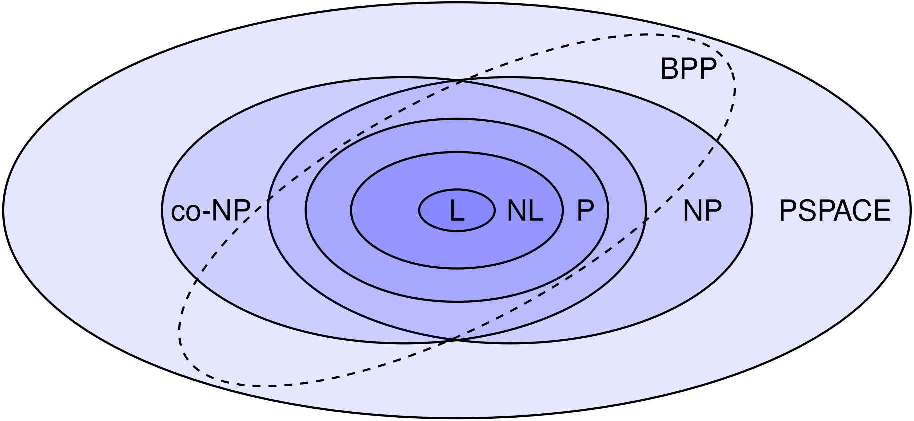
We often equate BPP with the class of “feasible computations”

- Compare to P which is the class of easy computations.
- Conjecture: P $=$ BPP.

It is not known whether BPP $\subset$ NP, or NP $\subset$ BPP

- NP $\subset$ BPP is considered unlikely, as that means there are practical solutions to problems in NP.

The Classes So Far



BPP and POLY-IDENTITIES

The earlier *POLY-IDENTITIES* $\in$ BPP.

If Q is zero then our algorithm outputs “accept”.

But if Q is non-zero then our algorithm can output “accept”

In fact *POLY-IDENTITIES* $\in$ co-RP

- See next slides for RP
- The class co-RP follows easily

The class RP (Monte-Carlo Algorithms)

The Class RP

RP (randomised polynomial time) is the class of languages for which there is a poly-time PTM with

- $w \in L \Rightarrow \Pr[M \text{ accepts } w] \geq 1/2;$ and
- $w \notin L \Rightarrow \Pr[M \text{ rejects } w] = 1$

If M accepts, we know $w \in L$ *("one-sided error")*

Example. The Fermat primality test is given p s.t.

- p is prime $\Rightarrow \Pr[M \text{ accepts } p] = 1$
- p is composite $\Rightarrow \Pr[M \text{ accepts } p] \leq 1/2^k$

Thus *COMPOSITES* $\in$ RP

The class RP (Monte-Carlo Algorithms)

The Class RP

RP (randomised polynomial time) is the class of languages for which there is a poly-time PTM with

- $w \in L \;\Rightarrow\; \Pr[M \text{ accepts } w] \geq 1/2;$ and
- $w \notin L \;\Rightarrow\; \Pr[M \text{ rejects } w] = 1$

If M accepts, we know $w \in L$ (*"one-sided error"*)

Example. The Fermat primality test is given p s.t.

- p is prime $\Rightarrow\; \Pr[M \text{ accepts } p] = 1$
- p is composite $\Rightarrow\; \Pr[M \text{ accepts } p] \leq 1/2^k$

Thus $COMPOSITES \in \mathrm{RP}$

The class RP (Monte-Carlo Algorithms)

The Class RP

RP (randomised polynomial time) is the class of languages for which there is a poly-time PTM with

- $w \in L \;\Rightarrow\; \Pr[M \text{ accepts } w] \geq 1/2;$ and
- $w \notin L \;\Rightarrow\; \Pr[M \text{ rejects } w] = 1$

If M accepts, we know $w \in L$ *("one-sided error")*

Example. The Fermat primality test is given p s.t.

- p is prime $\Rightarrow \Pr[M \text{ accepts } p] = 1$
- p is composite $\Rightarrow \Pr[M \text{ accepts } p] \leq 1/2^k$

Thus $\mathit{COMPOSITES} \in \mathrm{RP}$

More RP

Algorithms for RP problems sometimes called “One-Sided” Monte-Carlo algorithms.

Clearly

- $P \subset RP \subset BPP$
- $P \subset co\text{-}RP \subset BPP$.

Unknown whether

- $RP = co\text{-}RP$.
- $RP \subset (NP \cap co\text{-}NP)$.

The class ZPP (Las-Vegas Algorithms)

Think of randomized QuickSort

- Where we choose the pivot value at random.

This has the following properties

- Run time is expected to be $O(n \cdot \log n)$
- Worst case is $O(n^2)$.
- The answer on termination is expected to be correct.

This is an (admittedly silly) example of a Las Vegas algorithm

The class ZPP (Las-Vegas Algorithms)

The Class ZPP: Defn 1

The class ZPP is the class of problems for which a PTM exists such that

- If “accept” or “reject” is returned it is correct
- The run time is expected poly-time.

So a Las Vegas algorithm may not terminate!

The Class ZPP: Defn 2

The class ZPP is the class of problems with PTMs such that

- Runs in poly-time.
- Returns “accept”, “reject” or “dont know”
- Answer is either correct or “dont know”
- “Dont know” returned with probability at most $1/2$.

The class ZPP (Las-Vegas Algorithms)

The Class ZPP: Defn 1

The class ZPP is the class of problems for which a PTM exists such that

- If “accept” or “reject” is returned it is correct
- The run time is expected poly-time.

So a Las Vegas algorithm may not terminate!

The Class ZPP: Defn 2

The class ZPP is the class of problems with PTMs such that

- Runs in poly-time.
- Returns “accept”, “reject” or “dont know”
- Answer is either correct or “dont know”
- “Dont know” returned with probability at most $1/2$.

Definition 1 $=$ Definition 2

Let A_1 be a definition 1 PTM and A_2 be a definition 2 PTM.

From A_1 we can make a definition 2 PTM A_2' as

- Run A_1 for twice its expected run time.
- If it does not terminate, stop it and return “dont know”

From A_2 we can make a definition 2 PTM A_1' as

- Run A_2
- If it returns “dont know” run it again
- Repeat until successfull.

Exercise: Fill in the missing steps in these equivalences.

ZPP $= (\text{RP} \cap \text{co-RP})$

Let $\mathcal{L}$ be a language in $(\text{RP} \cap \text{co-RP})$

- $\exists$ PTM RP algorithm A accepting $\mathcal{L}$.
- $\exists$ PTM co-RP algorithm B accepting $\mathcal{L}$.

Create Las Vegas algorithm C as

- Run A. If it returns “accept” answer “accept”
- Run B. If it returns “reject” answer “reject”
- Repeat until one answers.

So $(\text{RP} \cap \text{co-RP}) \subset \text{ZPP}$

ZPP $= (\mathrm{RP} \cap \text{co-RP})$

Let $\mathcal{L}$ be a language in ZPP.

- $\exists$ PTM Las Vegas algorithm C accepting $\mathcal{L}$.

Create an RP algorithm for $\mathcal{L}$ via

- Run C for twice its expected run time.
- If gives an answer return it.
- Otherwise return "reject"

Similarly can create a co-RP algorithm.

Thus ZPP $\subset (\mathrm{RP} \cap \text{co-RP})$.

De-Randomization

A major current problem in complexity is to remove randomness completely.

Random numbers are an expensive resource, so we want to eliminate their usage as much as possible.

If we could eliminate randomness completely we would have BPP $=$ P.

One way to eliminate randomness is to use pseudo-random functions.

- Exhibiting yet another link between complexity theory and crypto.