Technische Universität Dresden

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 23: Probabilistic Complexity Classes (2)

Sergei Obiedkov

Knowledge-Based Systems

TU Dresden, 12 Jan 2026

More recent versions of this slide deck might be available.
For the most current version of this course, see
`https://iccl.inf.tu-dresden.de/web/Complexity_Theory/en`

Review: PP and BPP

Definition 21.4: A language **L** is in Polynomial Probabilistic Time (PP) if there is a PTM $\mathcal{M}$ such that:

- there is a polynomial function f such that $\mathcal{M}$ will always halt after $f(|w|)$ steps on all input words w,
- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] > \frac{1}{2}$,
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{2}$.

Definition 21.11: A language **L** is in Bounded-Error Polynomial Probabilistic Time (BPP) if there is a PTM $\mathcal{M}$ such that:

- there is a polynomial function f such that $\mathcal{M}$ will always halt after $f(|w|)$ steps on all input words w,
- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \geq \frac{2}{3}$,
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{3}$.

Review: Polynomial Identity Testing in BPP

Algorithm: For a polynomial $p(x_1, \ldots, x_m)$ encoded by a circuit of size $\leq n$

- Randomly select a number $k \in \{1, \ldots, 2^{2n}\}$.
- Randomly select $a_1, \ldots, a_n \in \{1, \ldots, 10 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits).
- Evaluate the circuit modulo k to compute $p(a_1, \ldots, a_m) \mod k$.
- Repeat this experiment for $4n$ times and accept iff all the outcomes are 0.

This leads to a constant error probability of < 0.5 for polynomials that are non-zero (which can be amplified to be $\leq \frac{1}{3}$) and an error probability of 0 for polynomials that are.

BPP and other classes

The neighbours of BPP

We have already observed that P $\subseteq$ BPP.

Moreover, since PP uses less strict conditions on probabilities, we immediately get

$$\text{BPP} \subseteq \text{PP} \subseteq \text{PSpace}.$$

Another interesting result is the following:

Theorem 23.1 (Adleman's[1] Theorem): BPP $\subseteq$ P$_{/\mathrm{poly}}$

(remember that we also know that P $\subseteq$ P$_{/\mathrm{poly}}$ but not whether NP $\subseteq$ P$_{/\mathrm{poly}}$)

[1]) Adleman is the A in RSA.

Proving Adleman's Theorem

Theorem 23.1 (Adleman's Theorem): $\mathsf{BPP} \subseteq \mathsf{P}_{/\mathrm{poly}}$

Proof: By Theorem 21.13, any language in BPP is decided by a PTM $\mathcal{M}$ with error probability $\leq \frac{1}{2^{n+1}}$, for an input of size n. Moreover, $\mathcal{M}$ uses a polynomial (in n) number m of random bits $r \in \{0, 1\}^m$ (verifier perspective on PTMs).

- String r is bad for input $w \in \{0, 1\}^n$ if $\mathcal{M}$ returns the wrong answer on w for random bits r; otherwise r is good for w.
- Since the error probability $\leq \frac{1}{2^{n+1}}$, there are $\leq \frac{2^m}{2^{n+1}}$ bad strings for any w.
- In total, for all 2^n inputs, there are $\leq 2^n \frac{2^m}{2^{n+1}} = \frac{2^m}{2}$ bad strings.
- Therefore, there are strings r that are good for all inputs.

Take one such universally good string $\hat{r}$; build a circuit for a deterministic verifier TM of inputs $w\#r$ as in Theorem 19.7; hardwire $\hat{r}$ as input for the certificate. $\square$

BPP and the Polynomial Hierarchy

Recall: The polynomial hierarchy can be defined via

- polynomial-time ATMs with a constant-bounded number of alternations
- oracle (N)TMs using oracles for lower levels of the hierarchy

Example: $\Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}} = \mathsf{NP}^{\mathsf{coNP}}$. These are languages decided by polynomial-time ATMs starting in an existential state and possibly alternating to a universal state.

At first sight, there seems to be no connection to BPP, yet:

Theorem 23.2 (Sipser-Gács-Lautemann Theorem): $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$

Notes:

- Sipser showed BPP $\subseteq$ PH; Gács strengthened the result; Lautemann gave the readable proof we will present – all in 1983.
- Stronger inclusions are known, but no separation from other classes we have covered is.

Proving Sipser-Gács-Lautemann (1)

Theorem 23.2 (Sipser-Gács-Lautemann Theorem): BPP $\subseteq \Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$

Proof: Overall proof outline:

- We will show that BPP $\subseteq \Sigma_2^{\mathrm{P}}$. This implies coBPP $\subseteq \Pi_2^{\mathrm{P}}$ and hence BPP $\subseteq \Pi_2^{\mathrm{P}}$, since BPP is closed under complement.
- We will show the inclusion for an arbitrary language **L** $\in$ BPP.
- For such **L**, there is a PTM $\mathcal{M}$ with the following features:
  - $\mathcal{M}$ runs in time $p(n)$ for some polynomial p, using $p(n)$ random bits;
  - $\mathcal{M}$ accepts **L** with error probability $\leq 2^{-n}$ (using probability amplification as in Theorem 21.13).

  We can view the computation of $\mathcal{M}$ as a deterministic polytime computation over an input of length n and an additional string of $p(n)$ random bits, as before.
- The key to the proof is the extreme difference between acceptance and rejection:
  - either $\geq (1 - 2^{-n})2^{p(n)}$ of random vectors $r \in \{0, 1\}^{p(n)}$ lead to acceptance,
  - or only $\leq 2^{-n}2^{p(n)} = 2^{p(n)-n}$ of random vectors $r \in \{0, 1\}^{p(n)}$ lead to acceptance.

  $\leadsto$ we want to tell the two situations apart in Σ_2^{P}

Proving Sipser-Gács-Lautemann (2)

Theorem 23.2 (Sipser-Gács-Lautemann Theorem): BPP $\subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$

Proof (continued): Idea for telling apart acceptance and rejection:

- For input w, let $S_w \subseteq \{0,1\}^{p(n)}$ be the set of all random vectors r such that $\mathcal{M}$ accepts w when using r.
- $S_w \subseteq \{0,1\}^{p(n)}$ is either almost all of $\{0,1\}^{p(n)}$ or a tiny fraction thereof.
- We consider "shifted copies" of S_w, created by some uniform bit-flipping S_w vectors:
  - If S_w is large, then polynomially many such copies can cover all of $\{0,1\}^{p(n)}$.
  - If S_w is small, then polynomially many copies are too small to cover $\{0,1\}^{p(n)}$.
- Making a "shifted copy":
  for some $u \in \{0,1\}^{p(n)}$, set $S_w \oplus u = \{r \oplus u \mid r \in S_w\}$, where $\oplus$ is XOR (sum mod 2).
- Number of shifted copies: we will use $k = \left\lceil \frac{p(n)}{n} \right\rceil + 1$ copies (a polynomial number).

We will show that k shifts can cover $\{0,1\}^{p(n)}$ if and only if S_w is "large".

Proving Sipser-Gács-Lautemann (3)

Theorem 23.2 (Sipser-Gács-Lautemann Theorem): BPP $\subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$

Proof (continued):

Claim 1: If $|S_w| \leq 2^{p(n)-n}$, then, for every set of $k = \left\lceil \frac{p(n)}{n} \right\rceil + 1$ vectors $u_1, \ldots, u_k \in \{0,1\}^{p(n)}$, we have $\bigcup_{i=1}^{k}(S_w \oplus u_i) \subsetneq \{0,1\}^{p(n)}$.

The result follows from the cardinalities of the involved sets:

Using $|S_w \oplus u_i| = |S_w|$, we obtain

$$\left|\bigcup_{i=1}^{k}(S_w \oplus u_i)\right| \leq k|S_w| \leq \left(\left\lceil \frac{p(n)}{n} \right\rceil + 1\right) 2^{p(n)-n} = \frac{\left(\left\lceil \frac{p(n)}{n} \right\rceil + 1\right)}{2^n} 2^{p(n)} = o(2^{p(n)}).$$

Therefore, the claim holds for sufficiently large n.
This suffices, since inputs of shorter length can surely be decided in Σ_2^{P} as well.

Proving Sipser-Gács-Lautemann (4)

Theorem 23.2 (Sipser-Gács-Lautemann Theorem): $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$

Proof (continued):

Claim 2: If $|S_w| \geq (1 - 2^{-n})2^{p(n)}$, then there is a set of $k = \left\lceil \frac{p(n)}{n} \right\rceil + 1$ vectors $u_1, \ldots, u_k \in \{0,1\}^{p(n)}$ such that $\bigcup_{i=1}^{k}(S_w \oplus u_i) = \{0,1\}^{p(n)}$.

We argue that, for independently and randomly chosen $u_1, \ldots, u_k$, we have $\Pr\left[\bigcup_{i=1}^{k}(S_w \oplus u_i) = \{0,1\}^{p(n)}\right] > 0$. The claim follows from this.

For a particular $r \in \{0,1\}^{p(n)}$, we compute

$$\Pr\left[r \notin \bigcup_{i=1}^{k}(S_w \oplus u_i)\right] \overset{(a)}{=} \Pi_{i=1}^{k}\Pr\left[r \notin (S_w \oplus u_i)\right] \overset{(b)}{\leq} \Pi_{i=1}^{k} 2^{-n} = 2^{-nk} = 2^{-n\left(\left\lceil \frac{p(n)}{n} \right\rceil + 1\right)} < 2^{-p(n)},$$

since: (a) u_i are selected independently; (b) $\Pr\left[r \notin (S_w \oplus u_i)\right] = \Pr\left[r \oplus u_i \notin S_w\right] \leq 2^{-n}$.

Therefore: $\Pr\left[\text{there is } r \in \{0,1\}^{p(n)} \setminus \bigcup_{i=1}^{k}(S_w \oplus u_i)\right] < 2^{p(n)} \cdot 2^{-p(n)} = 1$. In particular, there is at least one choice of $u_1, \ldots, u_k$ where this event does not occur, i.e., where all r are in $\bigcup_{i=1}^{k}(S_w \oplus u_i)$.

Proving Sipser-Gács-Lautemann (5)

Theorem 23.2 (Sipser-Gács-Lautemann Theorem): $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$

Proof (continued): In summary, we have shown:

- If S_w is “small”, then there are no vectors $u_1, \ldots, u_k$ such that $\bigcup_{i=1}^{k}(S_w \oplus u_i) = \{0,1\}^{p(n)}$.
- If S_w is “large”, then there are vectors $u_1, \ldots, u_k$ such that $\bigcup_{i=1}^{k}(S_w \oplus u_i) = \{0,1\}^{p(n)}$.

Hence, we can check the acceptance of $\mathcal{M}$ by computing if the following holds true:

$$\exists u_1, \ldots, u_k . \forall r \in \{0,1\}^{p(n)} . r \in \bigcup_{i=1}^{k}(S_w \oplus u_i).$$

Using the DTM version of PTMs, this becomes

$$\exists u_1, \ldots, u_k . \forall r \in \{0,1\}^{p(n)} . \bigvee_{i=1}^{k} \mathcal{M} \text{ accepts } w \text{ for random vector } r \oplus u_i.$$

This is a Σ_2^{P} computation. $\square$

Hierarchy Theorems for BPP

The Time Hierarchy Theorems for deterministic and non-deterministic Turing machines show that, when given (sufficiently) more time, such TMs can solve more problems. In particular:

- P $\neq$ ExpTime
- NP $\neq$ NExpTime

The proofs were based on diagonalisation arguments that enabled TMs with more time to deliberately differ from all TMs with less time.

Unfortunately, no such arguments are known for BPP:

- The difficulty of applying diagonalisation arguments is related to the semantic definition of BPP.
- Currently, we don't even know if BPP $\neq$ NExpTime!

Relationship of BPP and P

We know P $\subseteq$ BPP $\subseteq$ PP $\subseteq$ PSpace but not even if BPP $\neq$ NExpTime.

However, most experts expect that . . .

BPP is equal to P!

- Many BPP algorithms have been de-randomised successfully.
- BPP $=$ P is equivalent to the existence of strong pseudo-random number generators, which many experts consider likely.

Further probabilistic classes

Types of errors

We have defined BPP by restricting the probability of error to $\leq \frac{1}{3}$.

However, there are two types of errors:

- False positives: the PTM accepts a word that is not in the language
- False negatives: the PTM rejects a word that is in the language

Common BPP algorithms can often avoid one of these errors:

Example 23.3: Our previous algorithm for polynomial identity testing aimed to decide **ZeroP**. For inputs $w \in$ **ZeroP**, the algorithm accepted with probability 1 (no false negatives). Uncertainty only occurred for inputs $w \notin$ **ZeroP** (false positives were possible, though unlikely).

Randomised Polynomial Time

Excluding false positives/negatives from BPP leads to classes with one-sided error:

Definition 23.4: A language **L** is in Randomised Polynomial Time (RP) if there is a PTM $\mathcal{M}$ satisfying the following conditions:

- there is a polynomial function f such that $\mathcal{M}$ halts after $f(|w|)$ steps on every input word w;
- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \geq \frac{2}{3}$;
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] = 0$.

Definition 23.5: A language **L** is in coRP if its complement is in RP, i.e., if there is a polynomially time-bounded PTM $\mathcal{M}$ satisfying the following conditions:

- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] = 1$;
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{3}$.

Example 23.6: **ZeroP** $\in$ coRP.

Probability amplification for RP and coRP

It is clear from the definitions that RP $\subseteq$ BPP and coRP $\subseteq$ BPP.

Hence, we can apply Theorem 21.13 to amplify the output probability.

However, the situation for one-sided error classes is actually much simpler:

Theorem 23.7: Consider a language **L** and a polynomially time-bounded PTM $\mathcal{M}$ for which there is a constant $c > 0$ such that, for every word $w \in \Sigma^*$,

- if $w \in$ **L**, then $\Pr[\mathcal{M}$ accepts $w] \geq |w|^{-c}$;
- if $w \notin$ **L**, then $\Pr[\mathcal{M}$ accepts $w] = 0$.

Then, for every constant $d > 0$, there is a polynomially time-bounded PTM $\mathcal{M}'$ such that

- if $w \in$ **L**, then $\Pr[\mathcal{M}'$ accepts $w] \geq 1 - 2^{-|w|^d}$;
- if $w \notin$ **L**, then $\Pr[\mathcal{M}'$ accepts $w] = 0$.

Proof: Much simpler than for BPP (exercise). $\square$

RP and NP

The asymmetric acceptance conditions of RP reminds us of NP, since already “some” accepting runs are enough to prove acceptance.

Indeed, we get

Theorem 23.8: RP $\subseteq$ NP

Proof: If $\mathcal{M}$ satisfies the RP acceptance conditions for **L**, then $\mathcal{M}$ can be considered as an NTM that accepts **L** with respect to the usual non-deterministic acceptance conditions. Indeed, $\mathcal{M}$ has an accepting run on input $|w|$ if and only if $w \in$ **L**. $\square$

Similarly, we find coRP $\subseteq$ coNP.

Recall: While RP $\subseteq$ BPP, we do not know whether BPP $\subseteq$ NP.

Zero-sided error

Instead of admitting a possibly false answer (positive or negative), one can also require the correct answer while making some concessions on runtime:

Definition 23.9: A PTM $\mathcal{M}$ has expected runtime $f : \mathbb{N} \to \mathbb{R}$ if, for any input w, the expectation $E[T_w]$ of the number T_w of steps taken by $\mathcal{M}$ on input w is $T_w \leq f(|w|)$.

ZPP is the class of all languages for which there is a PTM $\mathcal{M}$ that

- returns the correct answer whenever it halts;
- has expected runtime f for some polynomial function f.

ZPP is for zero-error probabilistic polynomial time.

Note: In general, algorithms that produce correct results while giving only probabilistic guarantees on resource usage are called Las Vegas algorithms, as opposed to Monte Carlo algorithms, which have guaranteed resource bounds but probabilistic correctness (as in the case of BPP).

Zero-sided vs. one-sided error

In spite of the different approaches of expected error vs. expected runtime, we find a close relation between ZPP, RP, and coRP:

Theorem 23.10: ZPP = RP ∩ coRP

Proof: ZPP ⊆ RP: Given a ZPP algorithm $\mathcal{M}$, construct an RP algorithm by running $\mathcal{M}$ for three times the expected (polynomial) runtime t. If it stops, return the same answer; if it times out, reject.

- For any random variable X and $c > 0$, Markov's inequality implies $\Pr[X \geq cE[X]] \leq \frac{E[X]}{cE[X]} = \frac{1}{c}$.
- Hence, the probability of $\mathcal{M}$ running for $\geq 3t$ is $\leq \frac{1}{3}$.
- Therefore, the probability of a false negative (due to a timeout) is $\leq \frac{1}{3}$.

ZPP ⊆ coRP is dual; we just have to accept after timeout.

Zero-sided vs. one-sided error

In spite of the different approaches of expected error vs. expected runtime, we find a close relation between ZPP, RP, and coRP:

Theorem 23.10: ZPP = RP ∩ coRP

Proof: ZPP ⊇ RP ∩ coRP: Assume we have an RP algorithm $\mathcal{A}$ and a coRP algorithm $\mathcal{B}$ for the same language **L**. To obtain a ZPP algorithm, we run $\mathcal{A}$ and $\mathcal{B}$ on input w:

- If $\mathcal{A}$ accepts, accept.
- If $\mathcal{B}$ rejects, reject.
- If $\mathcal{A}$ rejects and $\mathcal{B}$ accepts, repeat the experiment.

Since RP has no false positives and coRP has no false negatives, this can only return the correct answer.

The probability of repetition is $\leq \frac{1}{3}$, since it requires one of the algorithms to be in error.

Hence, the probability of k repetitions is $\leq 3^{-k}$, for an expected runtime of $\leq \sum_{k\geq 0} \frac{(k+1)p}{3^k}$, where p is the combined (polynomial) runtime of $\mathcal{A}$ and $\mathcal{B}$. This is polynomial. $\square$

A Randomised Algorithm for **Max-3SAT** and Its Derandomisation

3SAT in Monte Carlo

Let ϕ be a 3CNF with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.

Algorithm: For each x_i, choose a value with probability $\frac{1}{2}$.

- If ϕ is unsatisfiable, the algorithm will detect this.
- If ϕ is satisfiable, the algorithm finds a satisfying assignment with probability $\geq 2^{-n}$.

Max-3SAT

Let ϕ be a 3CNF with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.[1]
Problem: Find an assignment that satisfies as many clauses as possible.

Algorithm: For each x_i, choose a value with probability $\frac{1}{2}$.

- S is a random variable equal to the number of satisfied clauses.
- S_i is an indicator random variable that is 1 when C_i is satisfied.

$$S = \sum_{i=1}^{m} S_i \qquad E[S] = \sum_{i=1}^{m} E[S_i] = \sum_{i=1}^{m} \Pr[S_i = 1] = \frac{7}{8}m$$

In expectation, the algorithm computes a 7/8-approximate solution.

[1] All three variables in each C_i are different.

Max-3SAT

Let ϕ be a 3CNF with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.[1]

Algorithm: For each x_i, choose a value with probability $\frac{1}{2}$.

The expected number of satisfied clauses is $\frac{7}{8}m$ for any ϕ.

Hence, every 3CNF ϕ with m clauses has an assignment satisfying at least $\frac{7}{8}m$ clauses.

Let's run the algorithm until we find one.

[1] All three variables in each C_i are different.

MAX-3SAT in Las Vegas

Algorithm: For each x_i, choose a value with probability $\frac{1}{2}$. Repeat until at least $\frac{7}{8}m$ clauses are satisfied.

Let p and p_i be the probabilities of satisfying, respectively, at least $\frac{7}{8}m$ and exactly i clauses with a random assignment.

$$\begin{aligned}\frac{7}{8}m = \sum_{i=0}^{m} ip_i &= \sum_{0\le i<\frac{7}{8}m} ip_i \quad + \sum_{\frac{7}{8}m\le i\le m} ip_i \\ &\le m' \sum_{0\le i<\frac{7}{8}m} p_i \quad + m \sum_{\frac{7}{8}m\le i\le m} p_i \\ &\le m(1-p) \quad + mp \quad \le m' + mp\end{aligned}$$

$$m' = \max\{n \in \mathbb{N} \mid n < \frac{7}{8}m\}$$

Then, $mp \ge \frac{7}{8}m - m' \ge \frac{1}{8}$ and $p \ge \frac{1}{8m}$.
The expected number of iterations is $\le 8m$.

Great expectations

- When we choose a random assignment for a 3CNF ϕ with m clauses, the expected number of satisfied clauses $E(\phi) \geq \frac{7}{8}m$.
- When we select a value for one variable, the expectation may change.
- If we make sure it never decreases, we will have at least $\frac{7}{8}m$ satisfied clauses after we assign values to all the variables.

Great expectations

Let ϕ_0 be ϕ with $x = 0$ and ϕ_1 be ϕ with $x = 1$.
Then

$$E(\phi) = \frac{1}{2}E(\phi_0) + \frac{1}{2}E(\phi_1).$$

Since

$$E(\phi) \geq \frac{7}{8}m,$$

we have $E(\phi_0) \geq \frac{7}{8}m$ or $E(\phi_1) \geq \frac{7}{8}m$.

Choose a value i that gives $E(\phi_i) \geq \frac{7}{8}m$ and fix it for x!

But how do we compute expectations?

What happens to clauses when we select a value?

$x := 0$

Before	**After**
$(p \lor q \lor r)$	$(p \lor q \lor r)$
$(x \lor y \lor z)$	$(y \lor z)$
$(\neg x \lor y \lor z)$	1
(x)	0

$x := 1$

Before	**After**
$(p \lor q \lor r)$	$(p \lor q \lor r)$
$(x \lor y \lor z)$	1
$(\neg x \lor y \lor z)$	$(y \lor z)$
$(\neg x)$	0

What is the probability of making a clause true?

$(x \vee y \vee z)$	$7/8$
$(x \vee y)$	$3/4$
(x)	$1/2$
0	0
1	1

- We can compute $E(\phi)$ by summing up these values for all the clauses.
- If ϕ has n variables, we only need to do this for n formulas each having m clauses.
- No randomness in the running time!

Summary and Outlook

Complexity relationships: see board (or make your own drawing).

Probabilistic classes with ones-sided error – RP and coRP – are common.

ZPP defines random computations with zero-sided error but probabilistic runtime.

Many experts believe (or expect) that

$$\mathrm{P} = \mathrm{ZPP} = \mathrm{RP} = \mathrm{coRP} = \mathrm{BPP} \subsetneq \mathrm{PP}.$$

What's next?

- Quantum computing
- Interactive Proofs
- Solving NP-hard problems
- Examinations