# Randomness and non-uniformity

## JASS 2006 Course 1: Proofs and Computers

Felix Weninger

TU München

April 2006

# Outline

1. Randomized computation
   - Concepts of randomized algorithms
   - Randomized complexity classes
   - Random sources
2. Non-uniformity
   - Computation with advice
   - Non-uniform polynomial time
   - On **P** vs. **NP**

# Randomized algorithms

- Usage of random sources
- Probability of error (incorrect result)
- Application of randomized algorithms:
  - Decision problems (e.g. primality tests)
  - Function problems (e.g. factorization)
  - Scientific computing (e.g. numerical simulation, Monte Carlo quadrature)
  - ...
- Analysis of randomized algorithms: important application for probability theory
  $\rightarrow$ Randomized algorithms also called *probabilistic*
- This section of the talk focuses on *time* and *error* bounds of randomized algorithms for *decision problems*.

# Example: Polynomial identity testing

Given two polynomials $p_1, p_2 \in \mathbb{F}[x]$, $\deg(p_1), \deg(p_2) \leq d$, decide whether $p_1 \equiv p_2$!
Equivalent: Decide whether $p_1 - p_2 \equiv 0$!

## Algorithm

1. **choose** $S \subset \mathbb{F}$, $x \in S$
2. $y := p_1(x) - p_2(x)$
3. **if** $(y = 0)$ **return** "$p_1 \equiv p_2$" **else return** "$p_1 \not\equiv p_2$"

# Analysis of the algorithm

- If $p_1 \equiv p_2$, the algorithm always outputs $p_1 \equiv p_2$.
- If $p_1 \not\equiv p_2$, it answers incorrectly iff $x$ is a root of $p_1 - p_2$.
  $\Rightarrow$ Probability for incorrect answer: $\leq \frac{d}{|S|}$
- Polynomial running time with bounded error probability: *Monte Carlo* algorithm

Discussion:

- Use of the algorithm is pointless if $p_1$ and $p_2$ are explicitly given (e.g. as a list of coefficients).
- Provably, the algorithm also works for *multivariate polynomials* $\in \mathbb{F}[x_1, \ldots, x_m]$.
- Important application: *Determinants of symbolic matrices* (implicitly given multivariate polynomials)
  Evaluation of determinant: $O(n^3)$; symbolic computation: no known deterministic polynomial-time algorithm!

# Classification of randomized algorithms

- The four cases for a randomized algorithm $A$ deciding $L$:

| | $A(x) = 1$ | $A(x) = 0$ |
|---|---|---|
| $x \in L$ | | *false negative* $(p_1)$ |
| $x \notin L$ | *false positive* $(p_2)$ | |

- $p_1 = 0$ or $p_2 = 0$: $A$ is called a *one-sided error* algorithm
  - $p_1 = 0$: “no“ answer definitely correct
  - $p_2 = 0$: “yes“ answer definitely correct
- Otherwise: $A$ is called a *two-sided error* algorithm (neither answer is definitely correct).
- “Pathological“ example: Deciding $L$ by coin toss is obviously a two-sided error algorithm with $p_1 = p_2 = \frac{1}{2}$.

# **NP** from a probabilistic point of view

- Informal notion of nondeterministic computation: Choosing from possible computation steps uniformly at random
- $\rightarrow$ Basic idea: Consider computations as "events" in the sense of probability theory!
- *Standardized nondeterministic Turing machines* (SNDTM):
  - Computation tree: *full binary tree* of depth $f(|x|)$ (where $x$ is the input and $f$ is the machine's time bound)
  - Theorem: If an arbitrary NDTM decides $L$ within time $f(|x|)$, so does a SNDTM within time $O(f(|x|))$.

  $\rightarrow$ Easy probabilistic analysis: All computation "events" have probability $2^{-f(|x|)}$

# **NP** from a probabilistic point of view (2)

Consider a NDTM deciding $L \in \mathbf{NP}$ in polynomial time $p(|x|)$:

- Zero probability of false positive (if $x \notin L$, all computations are required to reject).
- Probability of false negative: probably as high as $1 - 2^{-p(|x|)}$! (only one accepting computation required if $x \in L$)

**Idea:** Define a subset of **NP** such that it is guaranteed that a NDTM deciding a language $L$ in this class has a "decent" amount of accepting connections if $x \in L$!

# The class **RP**

**RP**: “randomized polynomial time“

**Definition**

A language $L$ is in **RP** if there exists a SNDTM $M$ deciding $L$ and a polynomial $p$, such that for every input $x$, $M$ halts after $p(|x|)$ steps and the following holds:

1. $x \in L \Rightarrow \textbf{prob}[M(x) = 0] \leq \frac{1}{2}$ (false negative)
2. $x \notin L \Rightarrow \textbf{prob}[M(x) = 1] = 0$ (false positive)

# Invariance of the constant

The constant $\frac{1}{2}$ is arbitrary. Any constant $0 < \epsilon < 1$ results in the same complexity class.

## Example

Let $M'$ be a SNDTM deciding $L$ with $\textbf{prob}[M'(x) = 0] \leq \frac{2}{3}$ for any $x \in L$. We build a TM $M$ from $M'$ that runs the following procedure (*amplification*):

1. Invoke $M'(x)$ three times.
2. Accept $x$ iff $M'$ has accepted $x$ at least once.

For $x \in L$, $\textbf{prob}[M(x) = 0] \leq \left(\frac{2}{3}\right)^3 = \frac{8}{27} \leq \frac{1}{2}$ while $M$ still rejects any $x \notin L$.

Clearly the probability of false negatives exponentially reduces in the number of executions of an **RP** algorithm!

# Some additional notes on **RP**

- Similar constructions: $\frac{1}{2}$ can also be replaced by
  - a fixed inverse polynomial $q(|x|)^{-1}$ ("negligible error probability")
  - or even $1 - q(|x|)^{-1}$ ("noticeable success probability")
- Note the fundamental difference between the latter and the definition of **NP** (*exponentially* small fraction of accepting computations for $x \in L$)!
- The definition of **RP** is "asymmetric". *Is* **RP** *closed under complement?*

# The class **coRP**

The (open) question whether **RP** is closed under complement is motivation for the definition of **coRP**, as follows:

**Definition**

A language $L$ is in **coRP** if there exists a SNDTM $M$ deciding $L$ and a polynomial $p$, such that for every input $x$, $M$ halts after $p(|x|)$ steps and the following holds:

1. $x \in L \Rightarrow \mathbf{prob}[M(x) = 0] = 0$ (false negative)
2. $x \notin L \Rightarrow \mathbf{prob}[M(x) = 1] \leq \frac{1}{2}$ (false positive)

Obviously, **coRP** $\subseteq$ **coNP**.
The famous Miller-Rabin primality test is a **coRP** algorithm.

# Las Vegas Algorithms

Consider the set of languages $\mathbf{RP} \cap \mathbf{coRP}$:

- A language $L \in \mathbf{RP} \cap \mathbf{coRP}$ has two probabilistic polynomial algorithms:
  - $A_1$: no false positives (**RP**)
  - $A_2$: no false negatives (**coRP**).
- Run $A_1$ and $A_2$ in parallel, for $k$ times.
- For $x \notin L$, we do not get a definitive result if and only if $A_2$ keeps returning "probably $x \in L$" ($x \in L$: vice versa).
- Probability for this case: $2^{-k}$.
- After a finite number of steps (average case: polynomial), we have a definite result: *Las Vegas* algorithms

# The class **ZPP**

- Complexity class for problems with Las Vegas algorithms: **ZPP** ("**z**ero **p**robability of error **p**olynomial time")
- Typical problem: PRIMES ($O(\log^3 n)$) Las Vegas algorithm; **RP** algorithm found by Adleman and Huang in 1987)

## Definition

$$\mathbf{ZPP} := \mathbf{RP} \cap \mathbf{coRP}$$

# The class BPP

We are looking for an appropriate complexity class for problems which have efficient two-sided error algorithms: "**b**ounded **p**robability of error **p**olynomial time".

## Definition

A language $L$ is in **BPP** if there exists a SNDTM $M$ deciding $L$ and a polynomial $p$, such that for every input $x$, $M$ halts after $p(|x|)$ steps and the following holds:

$$\mathbf{prob}[M(x) = \chi_L(x)] \geq \frac{3}{4},$$

where $\chi_L(x)$ is the *characteristic function* of $L$.

Informally: "$M$ decides $L$ by clear majority".

# Notes on BPP

- Again, the constant $\frac{3}{4}$ is arbitrary and can be replaced by any constant $\frac{1}{2} < \epsilon < 1$ or even by $\frac{1}{2} + q(|x|)^{-1}$ for a fixed polynomial $q$.
- Comparing **BPP** with **NP**, we get:

| | **BPP** | **NP** |
|---|---|---|
| $x \in L$ | $\textbf{prob}[M(x) = 1] \geq \frac{3}{4}$ | $\textbf{prob}[M(x) = 1] > 0$ |
| $x \notin L$ | $\textbf{prob}[M(x) = 0] \geq \frac{3}{4}$ | $\textbf{prob}[M(x) = 0] = 1$ |

  Therefore it is not clear at all whether **BPP** $\subseteq$ **NP** or vice versa. This is in fact an unresolved problem. However it is considered unlikely that **NP** $\subseteq$ **BPP** (why?)
- We will get into **BPP** $\overset{?}{\subseteq}$ **NP** later.

# The problem MAJSAT

Does the majority of truth assignments satisfy a boolean expression $\varphi$ with $n$ variables?

- If $\varphi \in L$, there might be only $2^{n-1}+1$ satisfying truth assignments.
- That means: The obvious SNDTM accepts such $\varphi \in L$ with a probability as low as $\frac{1}{2}+2^{-n}$.
- Therefore, **BPP** is probably not an appropriate complexity class for MAJSAT.
- Furthermore, there seems to be no *succinct certificate* for $\varphi$, so MAJSAT is not even likely to be in **NP**.

# Defining an appropriate class for MAJSAT

We want languages to be decided by “simple majority“:

**Definition**

A language $L$ is in **PP**' if there exists a SNDTM $M$ deciding $L$ and a polynomial $p$, such that for every input $x$, $M$ halts after $p(|x|)$ steps and the following holds:

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

Still, this definition does not capture the difficulty of MAJSAT!

# The class PP

We are going one step further:

**Definition**

A language $L$ is in **PP** if there exists a SNDTM $M$ deciding $L$ and a polynomial $p$, such that for every input $x$, $M$ halts after $p(|x|)$ steps and the following holds:

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

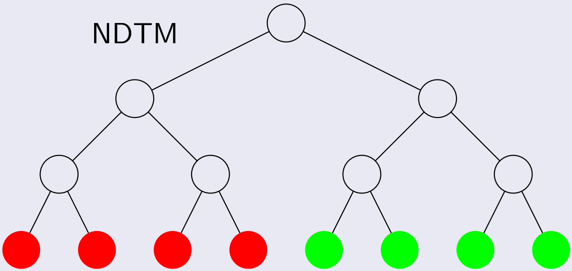
This is perhaps the weakest possible definition for a probabilistic algorithm: "**p**robabilistic **p**olynomial time".

# Discussion of **PP**

**Theorem**

**NP** $\subseteq$ **PP**.

**Proof.**




□

# Efficient experimentation

The following question is most important in analyzing probabilistic algorithms:

*How often do you have to repeat the algorithm so that you can consider the result to be "correct" with reasonable confidence?*

- **RP** algorithm: Repeat the algorithm $n$ times $\longrightarrow$
  - At least one "no" answer occurs $\longrightarrow$ "no" is correct
  - Otherwise: **prob**[$n$ "yes" answers are incorrect] $\leq 2^{-n}$.
  - Similar for **coRP**.
- Two-sided error algorithms: Neither answer is surely correct! Obvious solution: Take the "majority vote" of $n$ runs. *Problem: Estimate the error probability of this procedure!*

# The Chernov bound for probabilistic algorithms

**Lemma**

*Let $A$ be a two-sided error algorithm that anwers correctly with probability $\frac{1}{2} + \epsilon$. Let $Y$ denote the number of correct answers after $n$ independent executions of $A$: $Y$ is a* binomial random variable*. Then, for any* $0 < \epsilon < \frac{1}{2}$,

$$\textbf{prob}\left[Y \leq \frac{n}{2}\right] \leq e^{-\frac{\epsilon^2 n}{6}}.$$

$\longrightarrow$ Choose $n = \frac{c}{\epsilon^2}$ with an appropriate $c$.

**Corollary**

**BPP** *can be efficiently (that is in polynomial time) experimented.*
**PP** *(with $\epsilon$ probably exponentially small) cannot.*

1. Randomized computation
   - Concepts of randomized algorithms
   - Randomized complexity classes
   - Random sources

2. Non-uniformity
   - Computation with advice
   - Non-uniform polynomial time
   - On **P** vs. **NP**

# Sources of randomness

- *Hardware random number generators*: Use “external“ randomness found in
  - physical processes (nuclear decay detected by Geiger counters, images from Lava lamps);
  - interrupts from I/O devices;
  - swap files; ...
- *Pseudorandom number generators*: Deterministic algorithms
  - Generate a “long“ sequence of “random“ numbers from a “short“ *seed*
  - “Quality“: Given uniform distribution on the seeds, how uniform does the output “look“?
  - Examples: Linear congruential generators (simple, standard in most computer systems, but poor quality), Mersenne twister (complex, used for numerical simulation)

# Hardware random number generators

- Properties of a *perfect* random source:
  - *Independency* (i.e. the value of bit $x_i$ is not influenced by the values of $x_1 \ldots x_{i-1}$)
  - *Fairness* (i.e. $\mathbf{prob}[x_i = 1] = \frac{1}{2}$).
- Physical processes tend to produce *dependent* bit sequences.
- This fact leads to the concept of *slightly* random sources.

# Slightly random sources

## Definition

**($\delta$-random source)** Let $0 < \delta \leq \frac{1}{2}$, and let $p : \{0,1\}^* \rightarrow [\delta, 1-\delta]$ be an arbitrary function. A $\delta-$random source $S_p$ is a sequence of bits $x_1 \ldots x_n$ such that, for $1 \leq i \leq n$,

$$\mathbf{prob}[x_i = 1] = p(x_1 \ldots x_{i-1})$$

- Slightly random sources *cannot* drive a **BPP** algorithm! (Notion of slightly random source as an *adversary*)
- Nevertheless: Simulation of **BPP** algorithms using a $\delta$-random source is possible with *cubic* loss of efficiency (Vazirani 1985; Papadimitriou 1994)

# Pseudorandom number generators

- Linear congruential generators of the form

$$x_{n+1} = (ax_n + b) \mod m$$

  are fast, but fail many statistical tests for uniformity!
- Our notion of "pseudorandom number generator" (PRNG): random sequence that looks uniform to any *efficient observer*
- *Cryptographically secure* PRNG (CSPRNG)
  - "unpredictable", but polynomial running time: key requirement in cryptography!
  - $\rightarrow$ use *one-way functions*: "easy" to compute, "hard" to invert
  - Existence of such functions: only conjectured ($\rightarrow$ discrete logarithm)!

# Derandomization of **BPP**

- *Naive approach*: Iterate over all *random strings* and take majority vote
  $\Rightarrow$ deterministic algorithm, 100% correctness, but exponential running time!
- *Non-trivial derandomization*: Take subset of all random strings such that majority is preserved!

The connection to PRNGs:

**Theorem**

*If there exists a PRNG $G$ that turns a seed of size $m(n) \ll n$ into a pseudorandom sequence of length $n$, then* **BPP** *can be derandomized in DTIME(time$(G) \cdot 2^m$).*

# Derandomization approaches

- First derandomization approach: Use CSPRNG
  $\Rightarrow$ sub-exponential derandomization of **BPP** under assumption of one-way functions (Yao 1982).
- Second approach: *Nisan-Wigderson PRNG* (NWPRNG)
  - use *any hard function* (superpolynomial running time of PRNG)
  - 1994: sub-exponential derandomization

Complete (polynomial) derandomization using NWPRNG and hardness assumption in terms of circuits ($\rightarrow$ next section):

**Theorem (Impagliazzo and Wigderson, 1997)**

If there is a language $L \in \mathbf{E} := \bigcup_c \mathrm{DTIME}(2^{cn})$ which, for almost all inputs of size $n$, requires Boolean circuits of size $2^{\epsilon n}$ for some $\epsilon > 0$, then $\mathbf{BPP} = \mathbf{P}$.

# The $\mathbf{BPP} \stackrel{?}{=} \mathbf{P}$ question

- Problem: Proving lower bound for circuit size seems to be extremely hard!
- "Hardness vs. randomness" paradigm: Either there exist provably hard functions or randomness extends the class of efficient algorithms (Wigderson 2002).
- Conjecture: $\mathbf{BPP} = \mathbf{P}$

Question: If $\mathbf{BPP} = \mathbf{P}$, is the concept of probabilistic computation useless?
Papadimitriou 1994: **P** may be the class of problems with efficient algorithms, *deterministic polynomial or not*.

# Summary

- From the natural, but unrealistic model of nondeterministic computation, we derived the plausible concept of randomized computation.
- We classified algorithms according to their bounds on error probability and gave a notion which algorithms can be efficiently experimented.
- We had a look at the implementation of randomized algorithms. We introduced a concept of non-ideal, but plausible hardware random sources and its impact on randomized computability.
- Finally, we discussed pseudorandom number generators and the idea of *complete derandomization*.

1. Randomized computation
   - Concepts of randomized algorithms
   - Randomized complexity classes
   - Random sources

2. Non-uniformity
   - Computation with advice
   - Non-uniform polynomial time
   - On **P** vs. **NP**

# Turing machines with advice

- Another view of randomized computation: *deterministic* Turing machine that takes an additional random string as input
- Generalization: arbitrary *advice strings*, *one* for *all* inputs of length $n$

## Definition

A language $L$ is in $\mathbf{P}/f(n)$ if there exists a polynomial-time two-input deterministic Turing machine $M$, a complexity function $f(n)$ and a sequence $(a_n)$ of advice strings such that:

- $\forall n : |a_n| \leq f(n)$ (advice is space-bounded)
- $\forall x \in \{0,1\}^n : M(a_n, x) = \chi_L(x)$ ($M$ decides $L$ using $a_n$ as advice)

# The class $\mathbf{P}/poly$

- *Non-uniformity* in the definition of $\mathbf{P}/f(n)$: No specification of $(a_n)$!
- Advice of exponential size is pointless (why?)
- Perhaps the most important subset of $\mathbf{P}/f(n)$:

**Definition**

$$\mathbf{P}/poly := \bigcup_k \mathbf{P}/n^k$$

It is clear that $\mathbf{P} \subseteq \mathbf{P}/poly$.

# Boolean circuits

## Definition

A *Boolean circuit* is a dag $(V, E)$ with a labelling function $s : V \to \{\neg, \vee, \wedge, x_1, \dots, x_n, 0, 1, \text{out}\}$, such that

- $s(v) = \neg \Rightarrow \deg^+(v) = 1$ (NOT gate)
- $s(v) = \vee$ or $s(v) = \wedge \Rightarrow \deg^+(v) = 2$ (AND/OR gates)
- $s(v) = x_1, \dots, x_n, 0, 1 \Rightarrow \deg^+(v) = 0$ (input)
- $s(v) = \text{out} \Rightarrow \deg^-(v) = 0$ (output)
- The labels $x_1, \dots, x_n, \text{out}$ are used exactly once.

A boolean circuit $C$ with inputs $x_1 \dots x_n$ is usually more succinct than an equivalent boolean expression $\varphi(x_1 \dots x_n)$ ("shared expressions").

# Circuit complexity

Given a string $x$ in binary encoding, what is the *size* (number of gates) of a Boolean circuit $C$ which has $\chi_L(x)$ as output for some language $L$ ("$C$ decides $L$")?

## Definition

A language $L \subseteq \{0,1\}^*$ has polynomial circuits if there exists a sequence $(C_n)$ of Boolean circuits and a polynomial $p$ such that:

- $\forall n : \text{size}(C_n) \leq p(n)$
- $C_n$ has $n$ inputs, and the output of $C_n$ is $\chi_L(x)$ $\forall x \in \{0,1\}^n$.

Non-uniformity again: We do not specify how to construct $C_n$!

# The connection to $\mathbf{P}/poly$

## Theorem

*A language $L$ has polynomial circuits iff $L \in \mathbf{P}/poly$.*

## Proof sketch

- "$\Rightarrow$": Use as advice strings binary encodings of $C_n \Rightarrow$ polynomial advice length; CIRCUIT VALUE is **P**-complete.
- "$\Leftarrow$": Given polynomial-time TM $M$ with polynomial advice strings $a_n$, "hard-wire" them into to $M'$. Encode the *computation matrix* of $M'$, which represents the input/output string over time, as a Boolean circuit (input gates: initial string; output gate: acceptance indicator). Show that this circuit has polynomial size (hint: show that the matrix entries are logarithmic in respect to $n$).

# The power of $\mathbf{P}/poly$

**Theorem (Adleman)**

$\mathbf{BPP} \subseteq \mathbf{P}/poly$.

**Proof.**

Proof idea: We want to use random strings $r$ as advice strings (one for all inputs of length $n$).
Let $L \in \mathbf{BPP}$ be decided by a TM $M$ that is time-bounded by $p(n)$.
Let $\mathrm{bad}(x) := \{r \in \{0,1\}^{p(n)} : M(x,r) \neq \chi_L(x)\}$. W.l.o.g.: $M$ has error probability $\frac{1}{3^n} \Rightarrow \mathbf{prob}_{r \in \{0,1\}^{p(n)}}[r \in \mathrm{bad}(x)] = \frac{1}{3^n}$. Thus:

$$\mathbf{prob}\left[r \in \bigcup_{x \in \{0,1\}^n} \mathrm{bad}(x)\right] \leq \sum_{x \in \{0,1\}^n} \mathbf{prob}\left[r \in \mathrm{bad}(x)\right] = \frac{2^n}{3^n} < 1$$

This implies the existence of at least one “good“ $r$. □

# The power of $\mathbf{P}/poly$

## Theorem

$\mathbf{P}/poly$ contains non-recursive languages.

## Proof.

1. Claim: Every *unary language* $L \subseteq \{1\}^*$ is in $\mathbf{P}/poly$.

   Proof: Define as advice string $a_n := \begin{cases} 1 & 1^n \in L \\ 0 & \text{otherwise} \end{cases}$

2. Claim: There are non-recursive unary languages.
   Proof: Given any non-recursive $L \subseteq \{0,1\}^*$, define

$$U := \{1^n \mid \text{binary expansion of } n \text{ is in } L\}$$

□

# Introducing uniformity

- Clearly $\mathbf{P}/poly$ is an unrealistic model of computation!
- Idea: Consider languages decided by *uniform* Boolean circuits, i.e. circuits constructed by polynomially time-bounded (or logarithmically space-bounded) Turing machines!

"Unfortunately":

**Theorem**

*A language $L \subseteq \{0,1\}^*$ has uniform polynomial circuits iff $L \in \mathbf{P}$.*

Note: By giving a uniform description of the advice strings in the proof that $\mathbf{BPP} \subseteq \mathbf{P}/poly$, we would have proven that $\mathbf{P} = \mathbf{BPP}$!

*So what is left?*

# Sparse languages

## Definition

A language $L \subseteq \{0,1\}^*$ is *sparse* if there exists a polynomial $p$ such that

$$\forall n : |L \cap \{0,1\}^n| \leq p(n)$$

Otherwise, $L$ is *dense*.

## Example

Every unary language is sparse. Every known **NP**-complete language is dense.

## Lemma

*Every sparse language is in* $\mathbf{P}/poly$.

# Sparse languages and $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

**Theorem (Fortune)**

$\mathbf{P} = \mathbf{NP}$ iff every $L \in \mathbf{NP}$ Karp-reduces to a sparse language.

**Definition (informal)**

A language $L$ *Cook-reduces* to $L'$ iff $L$ can be decided in polynomial time, using polynomially many queries of the type "$x \in L'$?" to an oracle for $L'$.

Claim: A Karp reduction is a special case of a Cook reduction.

**Theorem (Karp and Lipton)**

$\mathbf{NP} \subseteq \mathbf{P}/poly$ iff every $L \in \mathbf{NP}$ Cook-reduces to a sparse language.

If $\mathbf{NP} \not\subseteq \mathbf{P}/poly$, then $\mathbf{P} \neq \mathbf{NP}$.

# Summary

- We defined computation with advice and the class **P**/*poly* of languages decided by polynomial-time deterministic Turing machines with advice of polynomial length. We saw that there is a strong connection to circuit complexity.
- We proposed that **P**/*poly* provides an upper bound for efficient computation, as it contains **BPP**.
- However, it also contains undecidable languages because of the lack of uniformity in the advice.
- We introduced the concept of uniformity and showed that this reduces **P**/*poly* to **P**.
- Finally, we saw that nevertheless **P**/*poly* is of great theoretical interest. We had a look at a proof that $\mathbf{P} \neq \mathbf{NP}$ under a reasonable conjecture.

# References

- Rajeev Motwani and Prabhakar Raghavan: Randomized Algorithms. Cambridge University Press, 1995.
- Christos H. Papadimitriou: Computational Complexity. Addison Wesley, 1994.
- Avi Wigderson and Irit Dinur: Derandomizing Pseudorandomness - A Survey (2002).
- Oded Goldreich: Introduction to Complexity Theory - Lecture Notes (1999).