Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 12: Randomised Computation

Aalto University
School of Science
Department of Computer Science

Spring 2019

Agenda

- Modelling randomised computation
- Probabilistic complexity classes
- Example: Polynomial identity testing
- Error reduction

Solving Hard Problems: Randomness

- **There are intractable problems that we don't know how to solve in polynomial time**
  - How to deal with such problems in practice?
- **One possible approach: Allow *random choices***
  - **Basic idea:** allow the program to flip coins
  - When does this this help? (Or does it help at all?)

Randomised Computation

- **Real world contains *random* phenomena**
  - Randomness is not captured by deterministic Turing machines
- **What happens if we add randomness to Turing machines?**
  - Randomness is widely used in computation, e.g. simulations
  - Random algorithms can be simpler and more efficient for some problems
  - However, in many (most? all?) cases it turns out that randomness can be eliminated by some *derandomisation technique*

Probabilistic Turing Machines

- **A *probabilistic Turing machine* M is a Turing machine with following special features:**
  - M has two *transition functions* δ_1 and δ_2
  - M always outputs 1 (*accept*) or 0 (*reject*)

- **An *execution* of a probabilistic Turing machine M:**
  - Start from the starting state as normal
  - At each step, apply δ_1 with probability $1/2$ and δ_2 with probability $1/2$

- **The output $M(x) \in \{0, 1\}$ is a *random variable***

Probabilistic Turing Machines

Definition

We say that a probabilistic Turing machine M runs in time $T(n)$ if M halts on input $x \in \{0,1\}^*$ in $T(|x|)$ steps regardless of the random choices.

- **If PTM runs in time t, there are 2^t possible branches**
  - Each branch is selected with probability $1/2^t$
  - $\Pr[M(x) = 1]$ is the *fraction* of branches accepting

Randomised Acceptance and Errors

- **For probabilistic Turing machines, we allow machines to output wrong answer for some random choices**
  - Depending on the exact formulation, we get different complexity classes
- **Possible options for resolving this:**
  - Allow *false negatives*, but no false positives
  - Allow *false positives*, but no false negatives
  - Allow both false negatives and false positives
  - Don't allow errors, but require that the *expected running time* is bounded

RTIME and RP: One-sided error

Definition (Randomised time)

The class RTIME$(T(n))$ is the set of languages L for which there exists a probabilistic Turing machine M and a constant $c > 0$ such that M runs in time $c \cdot T(n)$, and

- for all $x \in L$, we have $\Pr[M(x) = 1] \geq 2/3$, and
- for all $x \notin L$, we have $\Pr[M(x) = 1] = 0$.

Definition (Randomised polynomial time)

$$\mathsf{RP} = \bigcup_{d=1}^{\infty} \mathsf{RTIME}(n^d)$$

RP: Properties and Relationships

- RP **algorithms are called** ***Monte Carlo*** **algorithms**
- **Complementary class:** coRP
  - **Yes-instances:** accepted always
  - **No-instances:** rejected with probability $\geq 2/3$
- **Relationships and completeness**
  - P $\subseteq$ RP $\cap$ coRP
  - RP $\subseteq$ NP
  - coRP $\subseteq$ coNP
  - No known complete problems for RP and coRP

Expected Running Time

Definition (Expected running time)

Let M be a probabilistic Turing Machine. Let $T_{M,x}$ be a random variable whose value is the running time of M on x. We say that M has *expected running time* $T(n)$ if $\mathrm{E}[T_{M,x}] \leq T(|x|)$ for all $x \in \{0,1\}^*$.

ZTIME and ZPP: Zero-sided error

Definition (zero-error probabilistic time)

The class ZTIME$(T(n))$ is the set of languages L for which there exists a probabilistic Turing machine M with expected running time $T(n)$ such that whenever M halts on input $x \in \{0,1\}^*$, we have that $M(x) = 1$ if and only if $x \in L$.

Definition (Zero-error probabilistic polynomial time)

$$\mathsf{ZPP} = \bigcup_{d=1}^{\infty} \mathsf{ZTIME}(n^d)$$

ZPP: Properties and Relationships

- ZPP **algorithms are called *Las Vegas* algorithms**
- ZPP $=$ RP $\cap$ coRP
  - **Basic idea "$\supseteq$":** perform repeated runs of both the RP and the coRP algorithm until one of them gives a definitive answer
  - **Basic idea "$\subseteq$":** run ZPP algorithm for polynomial time, use default answer if the ZPP algorithm does not stop

BPTIME and BPP: Two-sided error

Definition (Bounded-error probabilistic time)

The class BPTIME$(T(n))$ is the set of languages L for which there exists a probabilistic Turing machine M and a constant $c > 0$ such that M runs in time $c \cdot T(n)$, and

- for all $x \in L$, we have $\Pr[M(x) = 1] \geq 2/3$, and
- for all $x \notin L$, we have $\Pr[M(x) = 0] \geq 2/3$.

Definition (Bounded-error probabilistic polynomial time)

$$\text{BPP} = \bigcup_{d=1}^{\infty} \text{BPTIME}(n^d)$$

BPP: Properties and Relationships

- **Relationships and completeness**
  - RP $\subseteq$ BPP
  - coRP $\subseteq$ BPP
  - BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$
  - No known complete problems for BPP
- **Proving separations for** BPP **seems difficult**
  - We don't even know if BPP $\neq$ NEXP!
  - On the other hand, it is known that if NP $\subseteq$ BPP, then PH $= \Sigma_2^p$

Polynomial Identity Testing

- A polynomial is *identically zero* if and only if its monomial representation equals 0
- **Example:**

$$\begin{aligned} & -xy+(x-y)(x^2+y)+x^2(y-x)+y^2 \\ = & -xy+x^3+xy-yx^2-y^2+x^2y-x^3+y^2 \\ = & -xy+xy-x^3+x^3-yx^2+x^2y-y^2+y^2=0 \end{aligned}$$

  is identically zero
- Two polynomials, p and q over variables $x_1,\ldots,x_n$, are *equal* iff the polynomial $p-q$ is identically zero

Polynomial Identity Testing

- One can obtain a Monte Carlo algorithm for checking whether a polynomial is not identically zero by using the *Schwartz-Zippel lemma*:

Lemma (Schwartz-Zippel)

Let $p(x_1,...,x_n)$ be a multivariate polynomial with total degree $d \geq 0$ over a field $\mathbb{F}$. Assume that p is not identically zero. Let S be a finite subset of $\mathbb{F}$ and let $r_1, r_2, ..., r_n$ be selected randomly from S. Then

$$\Pr\left[p(r_1, r_2, \ldots, r_n) = 0\right] \leq d / |S| .$$

- No deterministic polynomial time algorithm for this task is known

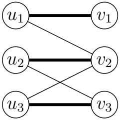
Perfect Matching

Definition (Perfect matching)

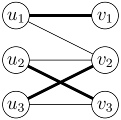
- **Instance:** Bipartite graph $B = (U, V, E)$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $E \subseteq U \times V$.
- **Question:** Is there a set $E' \subseteq E$ of n edges such that for any two distinct edges $(u, v), (u', v') \in E'$, $u \neq u'$ and $v \neq v'$ (i.e., is there a *perfect matching*)?

- A perfect matching can be seen as a permutation π of $1, \ldots, n$ such that $(u_i, v_{\pi(i)}) \in E$ for all $u_i \in U$

Example (perfect matchings as permutations)



$\pi_1(1)=1$
$\pi_1(2)=2$
$\pi_1(3)=3$



$\pi_1(1)=1$
$\pi_1(2)=3$
$\pi_1(3)=2$

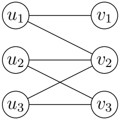
Perfect Matching

- **Perfect matching is related to the *determinant***
  - Given a graph G, construct an $n \times n$ matrix A^G, where the element $a_{i,j}$ is a variable x_{ij} if $(u_i, v_j) \in E$ and 0 otherwise.
  - Determinant of A^G is

$$\det A^G = \sum_{\pi} \operatorname{sgn}(\pi) \prod_{i=1}^{n} a_{i,\pi(i)}$$

where π ranges over permutations of n

Example (perfect matchings and determinants)



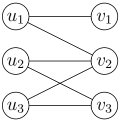
$$A^G = \begin{pmatrix} x_{1,1} & x_{1,2} & 0 \\ 0 & x_{2,2} & x_{2,3} \\ 0 & x_{3,2} & x_{3,3} \end{pmatrix}$$

$$\det A^G = x_{1,1}x_{2,2}x_{3,3} - x_{1,1}x_{2,3}x_{3,2}$$

Perfect Matching

- **Determinant of A^G tells us about the existence of a perfect matching**
  - Bipartite graph G has a perfect matching if and only if there is a term for which $a_{i,\pi(i)} \neq 0$ for all $i = 1, \ldots, n$.
  - Hence, G has a perfect matching if and only if $\det A^G$ is not identically 0.

Example (perfect matchings and determinants)



$$A^G = \begin{pmatrix} x_{1,1} & x_{1,2} & 0 \\ 0 & x_{2,2} & x_{2,3} \\ 0 & x_{3,2} & x_{3,3} \end{pmatrix}$$

$$\det A^G = x_{1,1}x_{2,2}x_{3,3} - x_{1,1}x_{2,3}x_{3,2}$$

Perfect Matching

- Testing whether $\det A^G$ is identically 0 for a symbolic matrix A^G containing variables can be done by using a randomised algorithm via Schwartz-Zippel lemma

Randomised algorithm for perfect matching

Given an $n \times n$ matrix $A^G(x_1, \ldots, x_m)$ with $m \leq n^2$ variables:

- Choose m random integers $i_1, \ldots, i_m$ (between 0 and M)
- Compute $\det A^G(i_1, \ldots, i_m)$ (by Gaussian elimination)
- If $\det A^G(i_1, \ldots, i_m) \neq 0$, then return *yes*
- If $\det A^G(i_1, \ldots, i_m) = 0$, then return *no*

- **Accepts yes-instances with probability $1 - n/M$**
- **Rejects no-instances always**

BPP Error Reduction

Theorem

Let $L \subseteq \{0,1\}^*$ *be a language, and assume that there is a polynomial-time PTM* M *such that for every* $x \in \{0,1\}^*$*, we have*

$$\Pr[M(x) = L(x)] \geq 1/2 + |x|^{-c}$$

for constant $c > 1$*. Then for every constant* $d > 0$*, there is a polynomial-time PTM* M' *such that for every* $x \in \{0,1\}^*$*, we have*

$$\Pr[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}.$$

- **Implies that** $r = 2/3$ **in the definition of** BPP **can be replaced by any constant** $r > 1/2$. (In fact even by a function that approaches $1/2$ at most polynomially.)

BPP Error Reduction: Proof

- **Machine M' does the following on input $x \in \{0,1\}^*$:**
  - Run $M(x)$ for $k = 8\left|x\right|^{2c+d}$ times to obtain outputs $y_1, y_2, \ldots, y_k$
  - Output majority of $y_1, y_2, \ldots, y_k$

- **We need to show that probability of the wrong answer is exponentially small**
  - Define random variable X_i so that X_i is 0 if $y_i = L(x)$, and 1 otherwise
  - $\sum_{i=1}^{k} X_i$ counts the number of *wrong answers*
  - We want to prove that $\Pr\left[\sum_{i=1}^{k} X_i \geq k/2\right] \leq 1 - 2^{-\left|x\right|^d}$
  - For this, we use the *Chernoff bound*

Chernoff Bound

Theorem (Chernoff bound)

Suppose that $X_1, \ldots, X_k$ are independent random variables taking the values 1 and 0 with probabilities p and $1-p$, respectively, and consider their sum $X = \sum_{i=1}^{k} X_i$. Then for all $0 \leq \delta \leq 1$,

$$\Pr\left[X \geq (1+\delta)pk\right] \leq e^{-\frac{\delta^2}{3}pk}.$$

BPP Error Reduction: Proof

- **We now apply Chernoff bound to random variables X_i:**
  - Random variables X_i are independent
  - $p = 1/2 - |x|^{-c}$
  - We set $\delta = |x|^{-c}/2$
  - Then $(1+\delta)pk < k/2$
  - Thus $\Pr\left[\sum_{i=1}^{k} X_i \geq k/2\right] \leq \Pr\left[\sum_{i=1}^{k} X_i \geq (1+\delta)pk\right]$

- **By the Chernoff bound, we have**

$$\Pr\left[\sum_{i=1}^{k} X_i \geq (1+\delta)pk\right] \leq e^{-\frac{\delta^2}{3}pk} \leq 2^{-|x|^d}$$

Error Reduction

- **Error reduction for** BPP **can be used to prove** BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$
  - **Basic idea:** since we can make acceptance probability exponentially small, there is a very small certificate for accepting or rejecting states
  - Can be checked in Σ_2^p
  - Need some non-trivial technical details
- **Error reduction works also for** RP **and** coRP
  - Success probability $|x|^{-c}$ is enough
  - Easier to prove, no need for Chernoff bound

Probabilistic and Quantum Computation

- **Strong Church-Turing thesis:** *any physically realisable system can be simulated by a Turing machine with polynomial overhead*
  - Would require that BPP $=$ P
  - This sounds surprising, but may well be the case (or not)

- **What about *quantum computation*?**
  - Quantum polynomial time BQP
  - Best known quantum algorithms beat best known randomised algorithms for some problems
  - **Known:** BPP $\subseteq$ BQP $\subseteq$ PSPACE

Lecture 12: Summary

- Monte Carlo algorithms: RP and coRP
- Las Vegas algorithms: ZPP
- BPP
- Polynomial Identity Testing
- Error reduction