Introduction to Computational Complexity

Antonis Antonopoulos

Computation and Reasoning Laboratory
National Technical University of Athens

Contents

Decision Problems

- Have answers of the form "*yes*" or "*no*"
- Encoding: each instance x of the problem is represented as a *string* of an alphabet Σ ($|\Sigma| \geq 2$).
- Decision problems have the form "Is x in L?", where L is a *language*, $L \subseteq \Sigma^*$.

- So, for an encoding of the input, using the alphabet Σ, we associate the following language with the decision problem Π:

$$L(\Pi) = \{x \in \Sigma^* \mid x \text{ is a representation of a "yes" instance of the problem } \Pi\}$$

Example

- Given a number x, is this number prime? ($x \overset{?}{\in} \texttt{PRIMES}$)
- Given graph G and a number k, is there a clique with k (or more) nodes in G?

Optimization Problems

- For each instance x there is a **set of Feasible Solutions** $F(x)$.
- To each $s \in F(x)$ we map a positive integer $c(x)$, using **the objective function** $c(s)$.
- We search for the solution $s \in F(x)$ which minimizes (or maximizes) the objective function $c(s)$.

Example

- The **Traveling Salesperson Problem** (`TSP`):
  Given a finite set $C = \{c_1, \ldots, c_n\}$ of cities and a distance $d(c_i, c_j) \in \mathbb{Z}^+, \forall (c_i, c_j) \in C^2$, we ask for a permutation π of C, that minimizes this quantity:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

A Model Discussion

- There are many computational models (RAM, Turing Machines etc).
- The **Church-Turing Thesis** states that all computation models are equivalent. That is, every computation model can be simulated by a Turing Machine.
- In Complexity Theory, we consider **efficiently computable** the problems which are solved (aka the languages that are decided) in **polynomial number of steps** (*Edmonds-Cobham Thesis*).

Efficiently Computable $\equiv$ Polynomial-Time Computable

Contents

Definition

A Turing Machine M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \ldots, q_n, q_{\text{halt}}, q_{\text{yes}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{S, L, R\}$ is the transition function.

- A TM is a “programming language” with a single data structure (a tape), and a cursor, which moves left and right on the tape.
- Function δ is the *program* of the machine.

Turing Machines and Languages

Definition

Let $L \subseteq \Sigma^*$ be a language and M a TM such that, for every string $x \in \Sigma^*$:

- If $x \in L$, then $M(x) =$ "yes"
- If $x \notin L$, then $M(x) =$ "no"

Then we say that **M decides L**.

- We can alternatively say that $M(x) = \chi_L(x)$, where $\chi_L(\cdot)$ is the *characteristic function* of L (if we consider 1 as "yes" and 0 as "no").
- If L is decided by some TM M, then L is called a **recursive language**.

Bounds on Turing Machines

- We will characterize the “performance” of a Turing Machine by the amount of *time* and *space* required on instances of size n, when these amounts are expressed as a function of n.

Definition

Let $T : \mathbb{N} \to \mathbb{N}$. We say that machine M operates within time $T(n)$ if, for any input string x, the time required by M to reach a final state is at most $T(|x|)$. Function T is a **time bound** for M.

Definition

Let $S : \mathbb{N} \to \mathbb{N}$. We say that machine M operates within space $S(n)$ if, for any input string x, M visits at most $S(|x|)$ locations on its work tapes (excluding the input tape) during its computation. Function S is a **space bound** for M.

Nondeterministic Turing Machines

- We will now introduce an **unrealistic** model of computation:

Definition

A Turing Machine M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \ldots, q_n, q_{\text{halt}}, q_{\text{yes}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\delta : (Q \setminus F) \times \Gamma \to Pow(Q \times \Gamma \times \{S, L, R\})$ is the transition **relation**.

Nondeterministic Turing Machines

- In this model, an input is accepted if <u>there is</u> *some sequence* of nondeterministic choices that results in "yes".
- An input is rejected if there is *no sequence* of choices that lead to acceptance.
- Observe the similarity with recursively enumerable languages.

Definition

We say that M operates within bound $T(n)$, if for every input $x \in \Sigma^*$ and every sequence of nondeterministic choices, M reaches a final state within $T(|x|)$ steps.

- The above definition requires that M does not have computation paths longer than $T(n)$, where $n = |x|$ the length of the input.
- The amount of time charged is the *depth* of the **computation tree**.

Contents

Parameters used to define complexity classes:

- Model of Computation (Turing Machine, RAM, Circuits)
- Mode of Computation (Deterministic, Nondeterministic, Probabilistic)
- Complexity Measures (*Time, Space, Circuit Size-Depth*)
- Other Parameters (Randomization, Interaction)

Our first complexity classes

Definition

Let $L \subseteq \Sigma^*$, and $T, S : \mathbb{N} \to \mathbb{N}$:

- We say that $L \in \mathbf{DTIME}[T(n)]$ if there exists a TM M deciding L, which operates within the *time* bound $\mathcal{O}(T(n))$, where $n = |x|$.
- We say that $L \in \mathbf{DSPACE}[S(n)]$ if there exists a TM M deciding L, which operates within *space* bound $\mathcal{O}(S(n))$, that is, for any input x, requires space at most $S(|x|)$.
- We say that $L \in \mathbf{NTIME}[T(n)]$ if there exists a *nondeterministic* TM M deciding L, which operates within the time bound $\mathcal{O}(T(n))$.
- We say that $L \in \mathbf{NSPACE}[S(n)]$ if there exists a *nondeterministic* TM M deciding L, which operates within space bound $\mathcal{O}(S(n))$.

Our first complexity classes

- The above are **Complexity Classes**, in the sense that they are sets of languages.
- All these classes are parameterized by a function T or S, so they are *families* of classes (for each function we obtain a complexity class).

Definition (Complement of a complexity class)

For any complexity class $\mathcal{C}$, $co\mathcal{C}$ denotes the class: $\{\overline{L} \mid L \in \mathcal{C}\}$, where $\overline{L} = \Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$.

- We want to define "reasonable" complexity classes, in the sense that we want to "compute more problems", given more computational resources.

Constructible Functions

Definition (Time-Constructible Function)

A nondecreasing function $T : \mathbb{N} \rightarrow \mathbb{N}$ is **time constructible** if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto \llcorner T(|x|) \lrcorner$ in time $T(n)$.

Definition (Space-Constructible Function)

A nondecreasing function $S : \mathbb{N} \rightarrow \mathbb{N}$ is **space-constructible** if $S(n) > \log n$ and there is a TM M that computes $S(|x|)$ using $S(|x|)$ space, given x as input.

- The restriction $T(n) \geq n$ is to allow the machine to read its input.
- The restriction $S(n) > \log n$ is to allow the machine to “remember” the index of the cell of the input tape that it is currently reading.
- Also, if $f_1(n)$, $f_2(n)$ are time/space-constructible functions, so are $f_1 + f_2$, $f_1 \cdot f_2$ and $f_1^{f_2}$.

Constructible Functions

Theorem (Hierarchy Theorems)

Let t_1, t_2 be time-constructible functions, and s_1, s_2 be space-constructible functions. Then:

1. *If* $t_1(n)\log t_1(n) = o(t_2(n))$, *then* $\mathbf{DTIME}(t_1) \subsetneq \mathbf{DTIME}(t_2)$.
2. *If* $t_1(n+1) = o(t_2(n))$, *then* $\mathbf{NTIME}(t_1) \subsetneq \mathbf{NTIME}(t_2)$.
3. *If* $s_1(n) = o(s_2(n))$, *then* $\mathbf{DSPACE}(s_1) \subsetneq \mathbf{DSPACE}(s_2)$.
4. *If* $s_1(n) = o(s_2(n))$, *then* $\mathbf{NSPACE}(s_1) \subsetneq \mathbf{NSPACE}(s_2)$.

- So, we have the hierachy:

$$\mathbf{DTIME}[n] \subsetneq \mathbf{DTIME}[n^2] \subsetneq \mathbf{DTIME}[n^3] \subsetneq \cdots$$

- We will later see that the class containing the problems we can efficiently solve (recall the Edmonds-Cobham Thesis) is the class $\mathbf{P} = \bigcup_{c\in\mathbb{N}} \mathbf{DTIME}[n^c]$.

- Hierarchy Theorems tell us how classes of the same kind relate to each other, when we vary the complexity bound.
- The most interesting results concern relationships between classes of different kinds:

Theorem

Suppose that $T(n), S(n)$ are time-constructible and space-constructible functions, respectively.Then:

1. $\mathbf{DTIME}[T(n)] \subseteq \mathbf{NTIME}[T(n)]$
2. $\mathbf{DSPACE}[S(n)] \subseteq \mathbf{NSPACE}[S(n)]$
3. $\mathbf{NTIME}[T(n)] \subseteq \mathbf{DSPACE}[T(n)]$
4. $\mathbf{NSPACE}[S(n)] \subseteq \mathbf{DTIME}[k^{\log n+S(n)}]$

Corollary

$$\mathbf{NTIME}[T(n)] \subseteq \bigcup_{c>1} \mathbf{DTIME}[c^{T(n)}]$$

The essential Complexity Hierarchy

Definition

$$\mathbf{L} = \mathbf{DSPACE}[\log n]$$

$$\mathbf{NL} = \mathbf{NSPACE}[\log n]$$

$$\mathbf{P} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[n^c]$$

$$\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}[n^c]$$

$$\mathbf{PSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{DSPACE}[n^c]$$

$$\mathbf{NPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}[n^c]$$

The essential Complexity Hierarchy

Definition

$$\mathbf{EXP} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[2^{n^c}]$$

$$\mathbf{NEXP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}[2^{n^c}]$$

$$\mathbf{EXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{DSPACE}[2^{n^c}]$$

$$\mathbf{NEXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}[2^{n^c}]$$

The essential Complexity Hierarchy

Definition

$$\mathbf{EXP} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[2^{n^c}]$$

$$\mathbf{NEXP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}[2^{n^c}]$$

$$\mathbf{EXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{DSPACE}[2^{n^c}]$$

$$\mathbf{NEXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}[2^{n^c}]$$

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{NPSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}$$

Can creativity be automated?

As we saw:

- Class **P**: Efficient *Computation*
- Class **NP**: Efficient *Verification*
- So, if we can efficiently verify a mathematical proof, can we create it efficiently?

If $P = NP$...

- For every mathematical statement, and given a page limit, we would (quickly) generate a proof, if one exists.
- Given detailed constraints on an engineering task, we would (quickly) generate a design which meets the given criteria, if one exists.
- Given data on some phenomenon and modeling restrictions, we would (quickly) generate a theory to explain the date, if one exists.

See "A. Wigderson: Knowledge, Creativity and **P** versus **NP**"

Complements of complexity classes

- Deterministic complexity classes are in general closed under complement ($co\mathbf{L} = \mathbf{L}$, $co\mathbf{P} = \mathbf{P}$, $co\mathbf{PSPACE} = \mathbf{PSPACE}$).
- Complements of non-deterministic complexity classes are very interesting:
- The class $co\mathbf{NP}$ contains all the languages that have **succinct disqualifications** (the analogue of *succinct certificate* for the class **NP**). The "no" instance of a problem in $co\mathbf{NP}$ has a short proof of its being a "no" instance.
- So:

$$\boxed{\mathbf{P} \subseteq \mathbf{NP} \cap co\mathbf{NP}}$$

- Note the *similarity* and the *difference* with $\mathbf{R} = \mathbf{RE} \cap co\mathbf{RE}$.

Quantifier Characterization of Complexity Classes

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y \ R(x, y)$
- $x \notin L \Rightarrow Q_2 y \ \neg R(x, y)$

- $\mathbf{P} = (\forall/\forall)$
- $\mathbf{NP} = (\exists/\forall)$
- $co\mathbf{NP} = (\forall/\exists)$

Savitch's Theorem

Theorem (Savitch's Theorem)

$$\mathbf{PSPACE} = \mathbf{NPSPACE}$$

The Immerman-Szelepscényi Theorem

Theorem

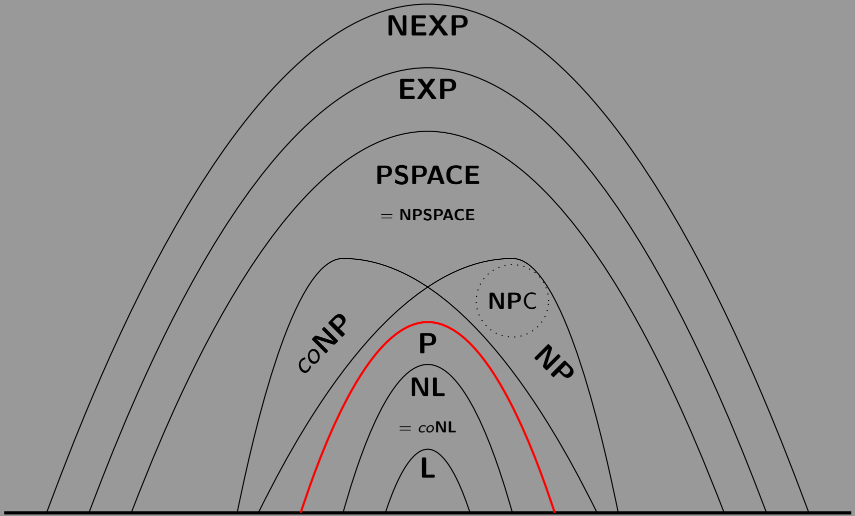
For every space constructible $S(n) > \log n$:

$$\mathbf{NSPACE}[S(n)] = co\mathbf{NSPACE}[S(n)]$$

Corollary

$$\mathbf{NL} = co\mathbf{NL}$$

Our Complexity Hierarchy Landscape



Contents

Oracle TMs and Oracle Classes

Definition

A Turing Machine $M^?$ with *oracle* is a multi-string deterministic TM that has a special string, called **query string**, and three special states: $q_?$ (*query state*), and q_{YES}, q_{NO} (*answer states*). Let $A \subseteq \Sigma^*$ be an arbitrary language. The computation of oracle machine M^A proceeds like an ordinary TM except for transitions from the query state:
From the $q_?$ moves to either q_{YES}, q_{NO}, depending on whether the current query string is in A or not.

- The answer states allow the machine to use this answer to its further computation.
- The computation of $M^?$ with oracle A on iput x is denoted as $M^A(x)$.

Oracle TMs and Oracle Classes

Definition

Let $\mathcal{C}$ be a time complexity class (deterministic or nondeterministic).
Define $\mathcal{C}^A$ to be the class of all languages decided by machines of the same sort and time bound as in $\mathcal{C}$, only that the machines have now oracle A. Also, we define: $\mathcal{C}_1^{\mathcal{C}_2} = \bigcup_{L \in \mathcal{C}_2} \mathcal{C}_1^L$.

For example, $\mathbf{P}^{\mathbf{NP}} = \bigcup_{L \in \mathbf{NP}} \mathbf{P}^L$. Note that $\mathbf{P}^{\mathrm{SAT}} = \mathbf{P}^{\mathbf{NP}}$.

Theorem

There exists an oracle A for which $\mathbf{P}^A = \mathbf{NP}^A$

Theorem

There exists an oracle B for which $\mathbf{P}^B \neq \mathbf{NP}^B$

The Polynomial Hierarchy

Polynomial Hierarchy Definition

- $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathbf{P}$
- $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
- $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$
- $\Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$
- $$\mathbf{PH} \equiv \bigcup_{i \geqslant 0} \Sigma_i^p$$

- $\Sigma_0^p = \mathbf{P}$
- $\Delta_1^p = \mathbf{P}$, $\Sigma_1^p = \mathbf{NP}$, $\Pi_1^p = co\mathbf{NP}$
- $\Delta_2^p = \mathbf{P}^{\mathbf{NP}}$, $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$, $\Pi_2^p = co\mathbf{NP}^{\mathbf{NP}}$

$$\begin{tikzcd}
\vdots & & \vdots \\
& \Delta_3^p = \mathbf{P}^{\mathbf{NP}^{\mathbf{NP}}} \arrow[ul] \arrow[ur] & \\
\Pi_2^p = co\mathbf{NP}^{\mathbf{NP}} \arrow[ur] & & \Sigma_2^p = \mathbf{NP}^{\mathbf{NP}} \arrow[ul] \\
& \Delta_2^p = \mathbf{P}^{\mathbf{NP}} \arrow[ul] \arrow[ur] & \\
\Pi_1^p = co\mathbf{NP} \arrow[ur] & & \Sigma_1^p = \mathbf{NP} \arrow[ul] \\
& \Delta_0^p = \Sigma_0^p = \arrow[ul] \arrow[ur] & \\
& = \Pi_0^p = \Delta_1^p = \mathbf{P} &
\end{tikzcd}$$

Contents

Probabilistic Turing Machines

- A Probabilistic Turing Machine is a TM as we know it, but with access to a "random source", that is an extra (read-only) tape containing *random-bits*!
- Randomization on:
  - **Output** (one or two-sided)
  - **Running Time**

Definition (Probabilistic Turing Machines)

A Probabilistic Turing Machine is a TM with two transition functions δ_0, δ_1. On input x, we choose in each step with probability $1/2$ to apply the transition function δ_0 or δ_1, indepedently of all previous choices.

- We denote by $M(x)$ the *random variable* corresponding to the output of M at the end of the process.
- For a function $T : \mathbb{N} \rightarrow \mathbb{N}$, we say that M runs in $T(|x|)$-time if it halts on x within $T(|x|)$ steps (*regardless of the random choices it makes*).

BPP Class

Definition (BPP Class)

For $T : \mathbb{N} \to \mathbb{N}$, let $\mathbf{BPTIME}[T(n)]$ the class of languages L such that there exists a PTM which halts in $\mathcal{O}(T(|x|))$ time on input x, and $\mathbf{Pr}[M(x) = L(x)] \geq 2/3$.
We define:

$$\mathbf{BPP} = \bigcup_{c \in \mathbb{N}} \mathbf{BPTIME}[n^c]$$

- The class **BPP** represents our notion of efficient (randomized) computation!
- We can also define **BPP** using certificates:

BPP Class

Definition (Alternative Definition of BPP)

A language $L \in \mathbf{BPP}$ if there exists a poly-time TM M and a polynomial $p \in poly(n)$, such that for every $x \in \{0,1\}^*$:

$$\mathbf{Pr}_{r \in \{0,1\}^{p(n)}}[M(x,r) = L(x)] \geq \frac{2}{3}$$

- $\mathbf{P} \subseteq \mathbf{BPP}$
- $\mathbf{BPP} \subseteq \mathbf{EXP}$
- The "**P** vs **BPP**" question.

Quantifier Characterizations

- Proper formalism (*Zachos et al.*):

Definition (Majority Quantifier)

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a predicate, and ε a rational number, such that $\varepsilon \in \left(0, \frac{1}{2}\right)$. We denote by $(\exists^+ y, |y| = k)R(x, y)$ the following predicate:

> *"There exist at least* $\left(\frac{1}{2} + \varepsilon\right) \cdot 2^k$ *strings* y *of length* m *for which* $R(x, y)$ *holds."*

We call $\exists^+$ the *overwhelming majority* quantifier.

- $\exists_r^+$ means that the fraction r of the possible certificates of a certain length satisfy the predicate for the certain input.

Quantifier Characterizations

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y \; R(x, y)$
- $x \notin L \Rightarrow Q_2 y \; \neg R(x, y)$

- $\mathbf{P} = (\forall/\forall)$
- $\mathbf{NP} = (\exists/\forall)$
- $co\mathbf{NP} = (\forall/\exists)$
- $\mathbf{BPP} = (\exists^+/\exists^+) = co\mathbf{BPP}$

RP Class

- In the same way, we can define classes that contain problems with one-sided error:

Definition

The class $\mathbf{RTIME}[T(n)]$ contains every language L for which there exists a PTM M running in $\mathcal{O}(T(|x|))$ time such that:

- $x \in L \Rightarrow \mathbf{Pr}[M(x) = 1] \geq \frac{2}{3}$
- $x \notin L \Rightarrow \mathbf{Pr}[M(x) = 0] = 1$

We define

$$\mathbf{RP} = \bigcup_{c \in \mathbb{N}} \mathbf{RTIME}[n^c]$$

- Similarly we define the class *co***RP**.

Quantifier Characterizations

- **RP** $\subseteq$ **NP**, since every accepting "branch" is a certificate!
- **RP** $\subseteq$ **BPP**, *co***RP** $\subseteq$ **BPP**

- **RP** $= (\exists^+/\forall)$

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{NP}$, since every accepting "branch" is a certificate!
- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$

- $\mathbf{RP} = (\exists^+/\forall) \subseteq (\exists/\forall) = \mathbf{NP}$

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{NP}$, since every accepting "branch" is a certificate!
- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$

- $\mathbf{RP} = (\exists^+/\forall) \subseteq (\exists/\forall) = \mathbf{NP}$
- $co\mathbf{RP} = (\forall/\exists^+) \subseteq (\forall/\exists) = co\mathbf{NP}$

Quantifier Characterizations

- **RP** $\subseteq$ **NP**, since every accepting "branch" is a certificate!
- **RP** $\subseteq$ **BPP**, *co***RP** $\subseteq$ **BPP**

- **RP** $= (\exists^+/\forall) \subseteq (\exists/\forall) =$ **NP**
- *co***RP** $= (\forall/\exists^+) \subseteq (\forall/\exists) =$ *co***NP**

Theorem (Decisive Characterization of BPP)

$$\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$$

ZPP Class

- And now something completely different:
- What is the random variable was the running time and not the output?

ZPP Class

- And now something completely different:
- What is the random variable was the running time and not the output?
- We say that M has expected running time $T(n)$ if the expectation $\mathbf{E}[T_{M(x)}]$ is at most $T(|x|)$ for every $x \in \{0,1\}^*$. ($T_{M(x)}$ is the running time of M on input x, and it is a **random variable**!)

Definition

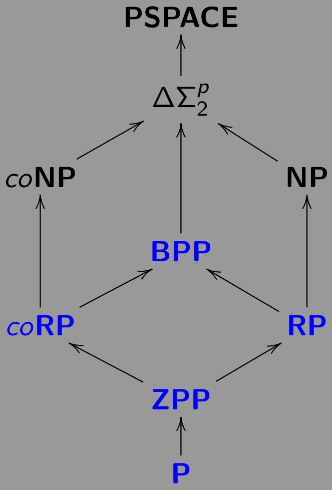
The class $\mathbf{ZTIME}[T(n)]$ contains all languages L for which there exists a machine M that runs in an expected time $\mathcal{O}(T(|x|))$ such that for every input $x \in \{0,1\}^*$, whenever M halts on x, the output $M(x)$ it produces is exactly $L(x)$. We define:

$$\mathbf{ZPP} = \bigcup_{c \in \mathbb{N}} \mathbf{ZTIME}[n^c]$$

ZPP Class

- The output of a **ZPP** machine is always correct!
- The problem is that we aren't sure about the running time.
- We can easily see that $\mathbf{ZPP} = \mathbf{RP} \cap co\mathbf{RP}$.

- The next Hasse diagram summarizes the previous inclusions: (Recall that $\Delta\Sigma_2^p = \Sigma_2^p \cap \Pi_2^p = \mathbf{NP}^{\mathbf{NP}} \cap co\mathbf{NP}^{\mathbf{NP}}$)



PSPACE

$\Delta\Sigma_2^p$

*co***NP**

NP

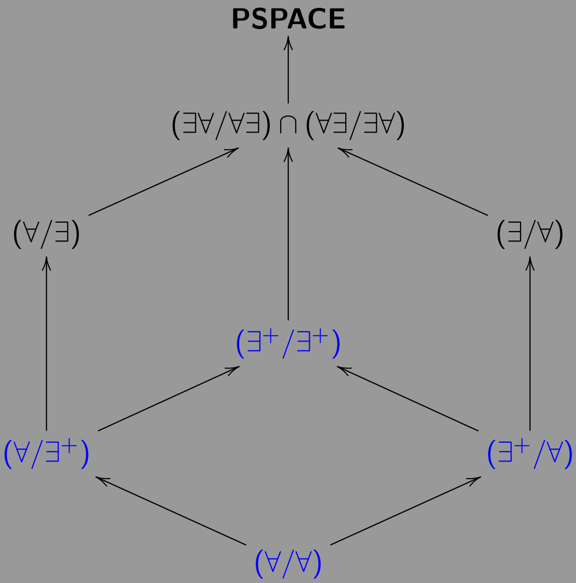
BPP

*co***RP**

RP

ZPP

P



PSPACE

$(\exists\forall/\forall\exists) \cap (\forall\exists/\exists\forall)$

$(\forall/\exists)$

$(\exists/\forall)$

$(\exists^+/\exists^+)$

$(\forall/\exists^+)$

$(\exists^+/\forall)$

$(\forall/\forall)$

Error Reduction for BPP

Theorem (Error Reduction for BPP)

Let $L \subseteq \{0,1\}^*$ be a language and suppose that there exists a poly-time PTM M such that for every $x \in \{0,1\}^*$:

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

Then, for every constant $d > 0$, $\exists$ poly-time PTM M' such that for every $x \in \{0,1\}^*$:

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

Contents

Introduction

"Maybe Fermat had a proof! But an important party was certainly missing to make the proof complete: the verifier. Each time rumor gets around that a student somewhere proved **P** = **NP**, *people ask "Has Karp seen the proof?" (they hardly even ask the student's name). Perhaps the verifier is most important that the prover." (from [BM88])*

- The notion of a mathematical proof is related to the certificate definition of **NP**.
- We enrich this scenario by introducing **interaction** in the basic scheme:
  The person (or TM) who verifies the proof asks the person who provides the proof a series of "queries", before he is convinced, and if he is, he provide the certificate.

Introduction

- The first person will be called **Verifier**, and the second **Prover**.
- In our model of computation, Prover and Verifier are interacting Turing Machines.
- We will categorize the various proof systems created by using:
    - various TMs (nondeterministic, probabilistic etc)
    - the information exchanged (private/public coins etc)
    - the number of TMs (IPs, MIPs,...)

Probabilistic Verifier: The Class IP

- Now, we let the *verifier* be probabilistic, i.e. the verifier's queries will be computed using a probabilistic TM:

Definition (Goldwasser-Micali-Rackoff)

For an integer $k \geq 1$ (that may depend on the input length), a language L is in $\mathbf{IP}[k]$ if there is a probabilistic polynomial-time T.M. V that can have a k-round interaction with a T.M. P such that:

- $x \in L \Rightarrow \exists P : Pr[\langle V, P\rangle(x) = 1] \geq \frac{2}{3}$ $(Completeness)$
- $x \notin L \Rightarrow \forall P : Pr[\langle V, P\rangle(x) = 1] \leq \frac{1}{3}$ $(Soundness)$

Probabilistic Verifier: The Class IP

Definition

We also define:

$$\mathbf{IP} = \bigcup_{c \in \mathbb{N}} \mathbf{IP}[n^c]$$

- The "output" $\langle V, P \rangle(x)$ is a random variable.
- We'll see that **IP** is a very large class! ($\supseteq$ **PH**)
- As usual, we can replace the completeness parameter $2/3$ with $1 - 2^{-n^s}$ and the soundness parameter $1/3$ by 2^{-n^s}, without changing the class for any fixed constant $s > 0$.
- We can also replace the completeness constant $2/3$ with 1 (**perfect completeness**), without changing the class, but replacing the soundness constant $1/3$ with 0, is equivalent with a *deterministic verifier*, so class **IP** collapses to **NP**.

Interactive Proof for Graph Non-Isomorphism

Definition

Two graphs G_1 and G_2 are *isomorphic*, if there exists a permutation π of the labels of the nodes of G_1, such that $\pi(G_1) = G_2$. If G_1 and G_2 are isomorphic, we write $G_1 \cong G_2$.

- `GI`: Given two graphs G_1, G_2, decide if they are isomorphic.
- `GNI`: Given two graphs G_1, G_2, decide if they are *not* isomorphic.

- Obviously, $\texttt{GI} \in \mathbf{NP}$ and $\texttt{GNI} \in co\mathbf{NP}$.
- This proof system relies on the Verifier's access to a *private* random source which cannot be seen by the Prover, so we confirm the crucial role the private coins play.

Interactive Proof for Graph Non-Isomorphism

Verifier: Picks $i \in \{1, 2\}$ uniformly at random.
Then, it permutes randomly the vertices of G_i to get a new graph H. Is sends H to the Prover.
Prover: Identifies which of G_1, G_2 was used to produce H.
Let G_j be the graph. Sends j to V.
Verifier: Accept if $i = j$. Reject otherwise.

Interactive Proof for Graph Non-Isomorphism

Verifier: Picks $i \in \{1,2\}$ uniformly at random.
Then, it permutes randomly the vertices of G_i to get a new graph H. Is sends H to the Prover.
Prover: Identifies which of G_1, G_2 was used to produce H.
Let G_j be the graph. Sends j to V.
Verifier: Accept if $i = j$. Reject otherwise.

- If $G_1 \ncong G_2$, then the powerfull prover can (nondeterministically) guess which one of the two graphs is isomprphic to H, and so the Verifier accepts with probability 1.
- If $G_1 \cong G_2$, the prover can't distinguish the two graphs, since a random permutation of G_1 looks exactly like a random permutation of G_2. So, the best he can do is guess randomly one, and the Verifier accepts with probability (at most) 1/2, which can be reduced by additional repetitions.

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

- **Merlin → Prover**
- **Arthur → Verifier**

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

- **Merlin → Prover**
- **Arthur → Verifier**
- Also, the class **MA** consists of all languages L, where there's an interactive proof for L in which the prover first sending a message, and then the verifier is "tossing coins" and computing its decision by doing a deterministic polynomial-time computation involving the input, the message and the random output.

Public vs. Private Coins

Theorem

$$\mathtt{GNI} \in \mathbf{AM}[2]$$

Theorem

For every $p \in poly(n)$:

$$\mathbf{IP}\left(p(n)\right) = \mathbf{AM}(p(n) + 2)$$

- So,

$$\mathbf{IP}[poly] = \mathbf{AM}[poly]$$

Properties of Arthur-Merlin Games

$$\begin{tikzcd}
 & \mathbf{NP} \arrow[r] & \mathbf{MA} \arrow[rr] \arrow[dr] & & \Sigma_2^p \\
 & & & \mathbf{AM} \arrow[dddr] & \\
\mathbf{P} \arrow[uur] \arrow[r] \arrow[ddr] & \mathbf{BPP} \arrow[uur] \arrow[ddr] & & & \\
 & & & co\mathbf{AM} \arrow[uuur] & \\
 & co\mathbf{NP} \arrow[r] & co\mathbf{MA} \arrow[ur] \arrow[rr] & & \Pi_2^p
\end{tikzcd}$$

Properties of Arthur-Merlin Games

- Proper formalism (*Zachos et al.*):

Definition (Majority Quantifier)

Let $R : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ be a predicate, and ε a rational number, such that $\varepsilon \in \left(0, \frac{1}{2}\right)$. We denote by $(\exists^+ y, |y| = k)R(x,y)$ the following predicate:

> *"There exist at least* $\left(\frac{1}{2} + \varepsilon\right) \cdot 2^k$ *strings* y *of length* m *for which* $R(x,y)$ *holds."*

We call $\exists^+$ the *overwhelming majority* quantifier.

- $\exists^+_r$ means that the fraction r of the possible certificates of a certain length satisfy the predicate for the certain input.
- Obviously, $\exists^+ = \exists^+_{1/2+\varepsilon} = \exists^+_{2/3} = \exists^+_{3/4} = \exists^+_{0.99} = \exists^+_{1-2^{-p(|x|)}}$

Properties of Arthur-Merlin Games

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y \; R(x, y)$
- $x \notin L \Rightarrow Q_2 y \; \neg R(x, y)$

- So: $\mathbf{P} = (\forall/\forall)$, $\mathbf{NP} = (\exists/\forall)$, $co\mathbf{NP} = (\forall/\exists)$
  $\mathbf{BPP} = (\exists^+/\exists^+)$, $\mathbf{RP} = (\exists^+/\forall)$, $co\mathbf{RP} = (\forall/\exists^+)$

Properties of Arthur-Merlin Games

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y \; R(x,y)$
- $x \notin L \Rightarrow Q_2 y \; \neg R(x,y)$

- So: $\mathbf{P} = (\forall/\forall)$, $\mathbf{NP} = (\exists/\forall)$, $co\mathbf{NP} = (\forall/\exists)$
  $\mathbf{BPP} = (\exists^+/\exists^+)$, $\mathbf{RP} = (\exists^+/\forall)$, $co\mathbf{RP} = (\forall/\exists^+)$

Arthur-Merlin Games

$$\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP} = (\exists^+\exists/\exists^+\forall)$$

$$\mathbf{MA} = \mathcal{N} \cdot \mathbf{BPP} = (\exists\exists^+/\forall\exists^+)$$

- Similarly: $\mathbf{AMA} = (\exists^+\exists\exists^+/\exists^+\forall\exists^+)$ etc.

The power of Interactive Proofs

- As we saw, **Interaction** alone does not gives us computational capabilities beyond **NP**.
- Also, **Randomization** alone does not give us significant power (we know that $\mathbf{BPP} \subseteq \Sigma_2^p$, and many researchers believe that $\mathbf{P} = \mathbf{BPP}$, which holds under some plausible assumptions).
- How much power could we get by their *combination*?
- We know that for fixed $k \in \mathbb{N}$, $\mathbf{IP}[k]$ collapses to

  $$\mathbf{IP}[k] = \mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP}$$

  a class that is "close" to **NP** (under similar assumptions, the non-deterministic analogue of **P** vs. **BPP** is **NP** vs. **AM**.)
- If we let k be a polynomial in the size of the input, how much more power could we get?

The power of Interactive Proofs

- Surprisingly:

Theorem (L.F.K.N. & Shamir)

$$\mathbf{IP} = \mathbf{PSPACE}$$

Epilogue: Probabilistically Checkable Proofs

- But if we put a **proof** instead of a Prover?

Epilogue: Probabilistically Checkable Proofs

- But if we put a **proof** instead of a Prover?
- The alleged proof is a string, and the (probabilistic) verification procedure is given direct (oracle) access to the proof.
- The verification procedure can access only *few* locations in the proof!
- We parameterize these Interactive Proof Systems by two complexity measures:
  - **Query** Complexity
  - **Randomness** Complexity
- The effective proof length of a PCP system is upper-bounded by $q(n) \cdot 2^{r(n)}$ (in the non-adaptive case). (How long can be in the adaptive case?)

PCP Definitions

Definition

PCP Verifiers Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-**PCP** verifier if there is a probabilistic polynomial-time algorithm V (the verifier) satisfying:

- *Efficiency*: On input $x \in \{0,1\}^*$ and given random oracle access to a string $\pi \in \{0,1\}^*$ of length at most $q(n) \cdot 2^{r(n)}$ (which we call the proof), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then, it accepts or rejects. Let $V^\pi(x)$ denote the random variable representing V's output on input x and with random access to π.
- *Completeness*: If $x \in L$, then $\exists \pi \in \{0,1\}^* : \mathbf{Pr}\left[V^\pi(x) = 1\right] = 1$
- *Soundness*: If $x \notin L$, then $\forall \pi \in \{0,1\}^* : \mathbf{Pr}\left[V^\pi(x) = 1\right] \leq \frac{1}{2}$

We say that a language L is in $\mathbf{PCP}[r(n), q(n)]$ if L has a $(\mathcal{O}(r(n)), \mathcal{O}(q(n)))$-**PCP** verifier.

Main Results

- Obviously:

$$\mathbf{PCP}[0,0] = \textbf{?}$$
$$\mathbf{PCP}[0,\mathit{poly}] = \textbf{?}$$
$$\mathbf{PCP}[\mathit{poly},0] = \textbf{?}$$

Main Results

- Obviously:

  $$\mathbf{PCP}[0,0] = \mathbf{P}$$
  $$\mathbf{PCP}[0, poly] = \textbf{?}$$
  $$\mathbf{PCP}[poly, 0] = \textbf{?}$$

Main Results

- Obviously:

  $$\mathbf{PCP}[0,0] = \mathbf{P}$$
  $$\mathbf{PCP}[0, poly] = \mathbf{NP}$$
  $$\mathbf{PCP}[poly, 0] = \mathbf{?}$$

Main Results

- Obviously:

  $$\mathbf{PCP}[0,0] = \mathbf{P}$$
  $$\mathbf{PCP}[0, \mathit{poly}] = \mathbf{NP}$$
  $$\mathbf{PCP}[\mathit{poly}, 0] = \mathit{co}\mathbf{RP}$$

Main Results

- Obviously:

$$\mathbf{PCP}[0,0] = \mathbf{P}$$
$$\mathbf{PCP}[0, \mathit{poly}] = \mathbf{NP}$$
$$\mathbf{PCP}[\mathit{poly}, 0] = \mathit{co}\mathbf{RP}$$

- A suprising result from Arora, Lund, Motwani, Safra, Sudan, Szegedy states that:

The PCP Theorem

$$\mathbf{NP} = \mathbf{PCP}[\log n, 1]$$

Main Results

- The restriction that the proof length is at most $q2^r$ is inconsequential, since such a verifier can look on at most this number of locations.
- We have that $\mathbf{PCP}[r(n), q(n)] \subseteq \mathbf{NTIME}[2^{\mathcal{O}(r(n))}q(n)]$, since a NTM could guess the proof in $2^{\mathcal{O}(r(n))}q(n)$ time, and verify it deterministically by running the verifier for all $2^{\mathcal{O}(r(n))}$ possible choices of its random coin tosses. If the verifier accepts for all these possible tosses, then the NTM accepts.