Theoretical Computer Science (ECE)
Algorithms and Complexity II (MPLA)

Computation and Reasoning Laboratory
National Technical University of Athens

2013-2014

2st Part

Oracles - Polynomial Hierarchy - Randomization - Nonuniform Complexity - Interaction - Counting Complexity

Professors:
S. Zachos, Professor
A. Pagourtzis, Ass. Professor

TA-Slides: Antonis Antonopoulos

Bibliography

Textbooks

1. C. Papadimitriou, **Computational Complexity**, Addison Wesley, 1994
2. S. Arora, B. Barak, **Computational Complexity: A Modern Approach**, Cambridge University Press, 2009
3. O. Goldreich, **Computational Complexity: A Conceptual Perspective**, Cambridge University Press, 2008

Lecture Notes

1. L. Trevisan, **Lecture Notes in Computational Complexity**, 2002, UC Berkeley
2. E. Allender, M. Loui, and K. Regan, **Three chapters for the CRC Handbook on Algorithms and Theory of Computation** (M.J. Atallah, ed.), (Boca Raton: CRC Press, 1998).

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- **Oracles & Optimization Problems**
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Counting Complexity

Introduction

TSP Versions

1. *TSP (D)*
2. *EXACT TSP*
3. *TSP COST*
4. *TSP*

$(1) \leq_P (2) \leq_P (3) \leq_P (4)$

DP Class Definition

Definition

A language L is in the class **DP** if and only if there are two languages $L_1 \in \mathbf{NP}$ and $L_2 \in co\mathbf{NP}$ such that $L = L_1 \cap L_2$.

- **DP** is *not* $\mathbf{NP} \cap co\mathbf{NP}$!
- Also, **DP** is a *syntactic* class, and so it has complete problems.

***SAT-UNSAT* Definition**

Given two Boolean expressions ϕ, ϕ', both in 3CNF. Is it true that ϕ is satisfiable and ϕ' is not?

Complete Problems for DP

Theorem

SAT-UNSAT is **DP**-complete.

Proof

- Firstly, we have to show it is in **DP**.
  So, let:
  $L_1 = \{(\phi, \phi'): \phi \text{ is satisfiable}\}$.
  $L_2 = \{(\phi, \phi'): \phi' \text{ is unsatisfiable}\}$.
  It is easy to see, $L_1 \in \mathbf{NP}$ and $L_2 \in co\mathbf{NP}$, thus $L \equiv L_1 \cap L_2 \in \mathbf{DP}$.
- For completeness, let $L \in \mathbf{DP}$. We have to show that $L \leq_P SAT\text{-}UNSAT$. $L \in \mathbf{DP} \Rightarrow L = L_1 \cap L_2$, $L_1 \in \mathbf{NP}$ and $L_2 \in co\mathbf{NP}$.
  SAT **NP**-complete$\Rightarrow \exists R_1 : L_1 \leq_P SAT$ and $R_2 : \overline{L_2} \leq_P SAT$.
  Hence, $L \leq_P SAT\text{-}UNSAT$, by $R(x) = (R_1(x), R_2(x))$

Complete Problems for DP

Theorem

EXACT TSP is **DP**-complete.

Proof

- $EXACT\ TSP \in \mathbf{DP}$, by $L_1 \equiv TSP \in \mathbf{NP}$ and $L_2 \equiv TSP\ COMPLEMENT \in co\mathbf{NP}$
- Completeness: we'll show that $SAT\text{-}UNSAT \leq_P EXACT\ TSP$.
  $3SAT \leq_P HP$: $(\phi, \phi') \to (G, G')$
  Broken Hamilton Path (*2 node-disjoint paths that cover all nodes*)
  Almost Satisfying Truth Assignement (*satisfies all clauses except for one*)

Complete Problems for DP

Proof

We define distances:

1. If $(i,j) \in$ E(G) or E(G'): $d(i,j) \equiv 1$
2. If $(i,j) \notin$ E(G), but i and j $\in$ V(G): $d(i,j) \equiv 2$
3. Otherwise: $d(i,j) \equiv 4$

Let n be the size of the graph.

1. If ϕ and ϕ' satisfiable, then $optCost = n$
2. If ϕ and ϕ' **un**satisfiable, then $optCost = n + 3$
3. If ϕ satisfiable and ϕ' not, then $optCost = n + 2$
4. If ϕ' satisfiable and ϕ not, then $optCost = n + 1$

"**yes**" instance of *SAT-UNSAT* $\Leftrightarrow$ $optCost = n + 2$

Let $B \equiv n + 2$!

Other DP-complete problems

Also:

- *CRITICAL SAT*: Given a Boolean expression ϕ, is it true that it's **un**satisfiable, but deleting any clause makes it satisfiable?
- *CRITICAL HAMILTON PATH*: Given a graph, is it true that it has **no** Hamilton path, but addition of any edge creates a Hamilton path?
- *CRITICAL 3-COLORABILITY*: Given a graph, is it true that it is **not** 3-colorable, but deletion of any node makes it 3-colorable?

are **DP**-complete!

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.

Theorem

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

Proof

Take A to be a **PSPACE**-complete language. Then:
$\mathbf{PSPACE} \subseteq \mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{NPSPACE} \subseteq \mathbf{PSPACE}$.

Theorem

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

The Classes P^{NP} and FP^{NP}

Alternative DP Definition

DP is the class of languages that can be decided by an oracle machine which makes 2 queries to a *SAT* oracle, and accepts iff the 1st answer is **yes**, and the 2nd is **no**.

- $\mathbf{P}^{SAT}$ is the class of languages decided in pol time with a *SAT* oracle.
  - Polynomial number of queries
  - Queries computed adaptively
- *SAT* **NP**-complete $\Rightarrow$ $\mathbf{P}^{SAT} = \mathbf{P}^{\mathbf{NP}}$
- $\mathbf{FP}^{\mathbf{NP}}$ is the class of functions that can be computed by a pol-time TM with a *SAT* oracle.
- Goal: $MAX\ OUTPUT \leq_P MAX\text{-}WEIGHT\ SAT \leq_P SAT$

FP^{NP}-complete Problems

MAX OUTPUT Definition

Given NTM N, with input 1^n, which halts after $\mathcal{O}(n)$,with output a string of length n. Which is the largest output,of any computation of N on 1^n?

Theorem

MAX OUTPUT is $\mathbf{FP^{NP}}$-complete.

Proof

MAX OUTPUT $\in \mathbf{FP^{NP}}$.
Let $F : \Sigma^* \to \Sigma^* \in \mathbf{FP^{NP}} \Rightarrow \exists$ pol-time TM $M^?$, s.t. $M^{SAT}(x) = F(x)$. We'll show: $F \leq$*MAX OUTPUT*!
Reductions R and S (*log space computable*) s.t.:

- $\forall x, R(x)$ is a instance of *MAX OUTPUT*
- $S(\text{max output of } R(x)) \to F(x)$

FP^{NP}-complete Problems

Proof (cont.)
NTM N:
Let $n = p^2(|x|)$, $p(\cdot)$, is the pol bound of SAT.
$N(1^n)$ generates x on a string.
M^{SAT} query state (ϕ_1):

- If $z_1 = 0$ (ϕ_1 unsat), then continue from q_{NO}.
- If $z_1 = 1$ (ϕ_1 sat), then guess assignment T_1:
  - If test succeeds, continue from q_{YES}.
  - If test fails, output$=0^n$ and **halt**. (Unsuccessful computation)

Continue to all guesses (z_i), and **halt**, with output$= \underbrace{z_1 z_2 \ldots 00}_{n}$

(Successful computation)

FP^{NP}-complete Problems

Proof (cont.)

We claim that the successful computation that outputs the largest integer, correspond to a correct simulation:
Let j the smallest integer,s.t.: $z_j = 0$, while ϕ_j was satisfiable.
Then, $\exists$ another successful computation of N, s.t.: $z_j = 1$.
The computations agree to the first $j - 1$ digits,$\Rightarrow$ the 2^{nd} represents a larger number.
The S part: $F(x)$ can be read off the end of the largest output of N.

FP^{NP}-complete Problems

MAX-WEIGHT SAT Definition

Given a set of clauses, each with an integer weight, find the truth assignment that satisfies a set of clauses with the most total weight.

FP^{NP}-complete Problems

MAX-WEIGHT SAT Definition

Given a set of clauses, each with an integer weight, find the truth assignment that satisfies a set of clauses with the most total weight.

Theorem

MAX-WEIGHT SAT is $\mathbf{FP}^{\mathbf{NP}}$-complete.

Proof

MAX-WEIGHT SAT is in $\mathbf{FP}^{\mathbf{NP}}$: By binary search, and a *SAT* oracle, we can find the largest possible total weight of satisfied clauses, and then, by setting the variables 1-1, the truth assignment that achieves it.

MAX OUTPUT$\leq$*MAX-WEIGHT SAT*:

FP^{NP}-complete Problems

Proof (cont.)

- $NTMN(1^n) \to \phi(N, m)$:
  Any satisfying truth assignment of $\phi(N, m) \to$ legal comp. of $N(1^n)$
- Clauses are given a huge weight (2^n), so that any t.a. that aspires to be optimum satisfy all clauses of $\phi(N, m)$.
- Add more clauses: (y_i): $i = 1, ..n$ with weight 2^{n-i}.
- Now, optimum t.a. must *not* represent any legal computation, but this which produces the *largest* possible output value.
- S part: From optimum t.a. of the resulting expression (or the weight), we can recover the optimum output of $N(1^n)$.

FP^{NP}-complete Problems

And the main result:

Theorem

TSP is $\mathbf{FP^{NP}}$-complete.

FP^{NP}-complete Problems

And the main result:

Theorem

TSP is $\mathbf{FP^{NP}}$-complete.

Corollary

TSP COST is $\mathbf{FP^{NP}}$-complete.

The Class $P^{NP[\log n]}$

Definition

$\mathbf{P}^{\mathbf{NP[logn]}}$ is the class of all languages decided by a polynomial time oracle machine, which on input x asks a total of $\mathcal{O}(\log|x|)$ SAT queries.

- $\mathbf{FP}^{\mathbf{NP[logn]}}$ is the corresponding class of functions.

The Class $P^{NP[\log n]}$

Definition

$\mathbf{P^{NP[logn]}}$ is the class of all languages decided by a polynomial time oracle machine, which on input x asks a total of $\mathcal{O}(\log|x|)$ SAT queries.

- $\mathbf{FP^{NP[logn]}}$ is the corresponding class of functions.

CLIQUE SIZE Definition

Given a graph, determine the size of his *largest* clique.

The Class $P^{NP[\log n]}$

Definition

$\mathbf{P^{NP[logn]}}$ is the class of all languages decided by a polynomial time oracle machine, which on input x asks a total of $\mathcal{O}(\log|x|)$ *SAT* queries.

- $\mathbf{FP^{NP[logn]}}$ is the corresponding class of functions.

CLIQUE SIZE Definition

Given a graph, determine the size of his *largest* clique.

Theorem

CLIQUE SIZE is $\mathbf{FP^{NP[logn]}}$-complete.

Conclusion

1. *TSP (D)* is **NP**-complete.
2. *EXACT TSP* is **DP**-complete.
3. *TSP COST* is $\mathbf{FP}^{\mathbf{NP}}$-complete.
4. *TSP* is $\mathbf{FP}^{\mathbf{NP}}$-complete.

And now,

- $\mathbf{P}^{\mathbf{NP}} \rightarrow \mathbf{NP}^{\mathbf{NP}}$?
- Oracles for $\mathbf{NP}^{\mathbf{NP}}$?

The Polynomial Hierarchy

Polynomial Hierarchy Definition

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

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

Basic Theorems

Theorem

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Π_{i-1}^p and

$$L = \{x : \exists y, s.t. : (x, y) \in R\}$$

Proof (by Induction)

- For $i = 1$
  $\{x; y : (x, y) \in R\} \in \mathbf{P}$,so $L = \{x | \exists y : (x, y) \in R\} \in \mathbf{NP}$
- For $i > 1$
  If $\exists R \in \Pi_{i-1}^p$, we must show that $L \in \Sigma_i^p \Rightarrow$
  $\exists$ NTM with Σ_{i-1}^p oracle: NTM(x) guesses a y and asks Π_{i-1}^p oracle whether $(x, y) \notin R$.

Basic Theorems

Proof (cont.)

- If $L \in \Sigma_i^p$, we must show the existence or R.
  $L \in \Sigma_i^p \Rightarrow \exists$ NTM M^K, $K \in \Sigma_{i-1}^p$, which decides L.
  $K \in \Sigma_{i-1}^p \Rightarrow \exists S \in \Pi_{i-2}^p : (z \in K \Leftrightarrow \exists w : (z, w) \in S)$
  We must describe a relation R (we know: $x \in L \Leftrightarrow$ accepting comp of $M^K(x)$)
  Query Steps: "yes" $\rightarrow z_i$ has a certificate w_i st $(z_i, w_i) \in S$.
  So, $R(x) =$"$(x, y) \in R$ *iff y records an accepting computation of* $M^?$ *on* x *, together with a certificate* w_i *for each* **yes** *query* z_i *in the computation.*"
  We must show $\{x; y : (x, y) \in R\} \in \Pi_{i-1}^p$.

Basic Theorems

Corollary

Let L be a language , and $i \geq 1$. $L \in \Pi_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Σ_{i-1}^p and

$$L = \{x : \forall y, |y| \leq |x|^k, s.t. : (x, y) \in R\}$$

Corollary

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced, polynomially-time decicable $(i+1)$-ary relation R such that:

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 ... Q y_i, s.t. : (x, y_1, ..., y_i) \in R\}$$

where the i^{th} quantifier Q is $\forall$, if i is even, and $\exists$, if i is odd.

Basic Theorems

Theorem

If for some $i \geq 1$, $\Sigma_i^p = \Pi_i^p$, then for all $j > i$:

$$\Sigma_j^p = \Pi_j^p = \Delta_j^p = \Sigma_i^p$$

Or, the polynomial hierarchy *collapses* to the i^{th} level.

Proof

It suffices to show that: $\Sigma_i^p = \Pi_i^p \Rightarrow \Sigma_{i+1}^p = \Sigma_i^p$
Let $L \in \Sigma_{i+1}^p \Rightarrow \exists R \in \Pi_i^p$: $L = \{x | \exists y : (x, y) \in R\}$
Since $\Pi_i^p = \Sigma_i^p \Rightarrow R \in \Sigma_i^p$
$(x, y) \in R \Leftrightarrow \exists z : (x, y, z) \in S$, $S \in \Pi_{i-1}^p$.
Thus, $x \in L \Leftrightarrow \exists y; z : (x, y, z) \in S$, $S \in \Pi_{i-1}^p$, which means $L \in \Sigma_i^p$.

Basic Theorems

Corollary

If $\mathbf{P}=\mathbf{NP}$, or even $\mathbf{NP}=\mathrm{co}\mathbf{NP}$, the Polynomial Hierarchy collapses to the first level.

Basic Theorems

Corollary

If **P**=**NP**, or even **NP**=co**NP**, the Polynomial Hierarchy collapses to the first level.

MINIMUM CIRCUIT Definition

Given a Boolean Circuit C, is it true that there is no circuit with fewer gates that computes the same Boolean function

- *MINIMUM CIRCUIT* is in Π_2^p, and not known to be in any class below that.
- It is open whether *MINIMUM CIRCUIT* is Π_2^p-complete.

Theorem

If *SAT* has Polynomial Circuits, then the Polynomial Hierarchy collapses to the second level.

Basic Theorems

$QSAT_i$ Definition

Given expression ϕ, with Boolean variables partitioned into i sets X_i,is ϕ satisfied by the overall truth assignment of the expression:

$$\exists X_1 \forall X_2 \exists X_3 \ldots . . Q X_i \phi$$

, where Q is $\exists$ if i is *odd*, and $\forall$ if i is even.

Theorem

For all $i \geq 1$ $QSAT_i$ is Σ_i^p-complete.

Basic Theorems

Theorem

If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

Proof

Let L is **PH**-complete.
Since $L \in \mathbf{PH}$, $\exists i \geq 0 : L \in \Sigma_i^p$.
But any $L' \in \Sigma_{i+1}^p$ reduces to L. Since PH is closed under reductions, we imply that $L' \in \Sigma_i^p$, so $\Sigma_i^p = \Sigma_{i+1}^p$.

Basic Theorems

Theorem

If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

Proof

Let L is **PH**-complete.
Since $L \in \mathbf{PH}$, $\exists i \geq 0 : L \in \Sigma_i^p$.
But any $L' \in \Sigma_{i+1}^p$ reduces to L. Since PH is closed under reductions, we imply that $L' \in \Sigma_i^p$, so $\Sigma_i^p = \Sigma_{i+1}^p$.

Theorem

$\mathbf{PH} \subseteq \mathbf{PSPACE}$

- $\mathbf{PH} \stackrel{?}{=} \mathbf{PSPACE}$ (Open). If it was, then **PH** has complete problems, so it collapses to some finite level.

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & Optimization Problems
- **Randomized Computation**
- Non-Uniform Complexity
- Interactive Proofs
- Counting Complexity

Warmup: Randomized Quicksort

Deterministic Quicksort

Input: A list L of integers;

If $n \leq 1$ then return L.

Else {

- let $i = 1$;
- let L_1 be the sublist of L whose elements are $< a_i$;
- let L_1 be the sublist of L whose elements are $= a_i$;
- let L_1 be the sublist of L whose elements are $> a_i$;
- Recursively Quicksort L_1 and L_3;
- return $L = L_1 L_2 L_3$;

Warmup: Randomized Quicksort

Randomized Quicksort

Input: A list L of integers;

If n $\leq$ 1 then return L.

Else {

- choose a random integer i, $1 \leq i \leq n$;
- let L_1 be the sublist of L whose elements are $< a_i$;
- let L_1 be the sublist of L whose elements are $= a_i$;
- let L_1 be the sublist of L whose elements are $> a_i$;
- Recursively Quicksort L_1 and L_3;
- return $L = L_1L_2L_3$;

Warmup: Randomized Quicksort

- Let T_d the *max* number of comparisons for the Deterministic Quicksort:

$$T_d \geq T_d(n-1) + \mathcal{O}\left(n\right)$$

$$\Downarrow$$

$$T_d(n) = \Omega(n^2)$$

Warmup: Randomized Quicksort

- Let T_d the *max* number of comparisons for the Deterministic Quicksort:

$$T_d \geq T_d(n-1) + \mathcal{O}(n)$$

$$\Downarrow$$

$$T_d(n) = \Omega(n^2)$$

- Let T_r the *expected* number of comparisons for the Randomized Quicksort:

$$T_r \geq \frac{1}{n} \sum_{j=0}^{n-1} [T_r(j) - T_r(n-1-j)] + \mathcal{O}(n)$$

$$\Downarrow$$

$$T_r(n) = \mathcal{O}(n \log n)$$

Warmup: Polynomial Identity Testing

1. Two polynomials are equal if they have the same coefficients for corresponding powers of their variable.
2. A polynomial is *identically zero* if all its coefficients are equal to the additive identity element.
3. How we can test if a polynomial is identically zero?

Warmup: Polynomial Identity Testing

1. Two polynomials are equal if they have the same coefficients for corresponding powers of their variable.
2. A polynomial is *identically zero* if all its coefficients are equal to the additive identity element.
3. How we can test if a polynomial is identically zero?
4. We can choose uniformly at random $r_1, \dots, r_n$ from a set $S \subseteq \mathbb{F}$.
5. We are wrong with a probability at most:

Theorem (Schwartz-Zippel Lemma)

Let $Q(x_1, \dots, x_n) \in \mathbb{F}[x_1, \dots, x_n]$ be a multivariate polynomial of total degree d. Fix any finite set $S \subseteq \mathbb{F}$, and let $r_1, \dots, r_n$ be chosen indepedently and uniformly at random from S. Then:

$$\mathbf{Pr}[Q(r_1, \dots, r_n) = 0 | Q(x_1, \dots, x_n) \neq 0] \leq \frac{d}{|S|}$$

Warmup: Polynomial Identity Testing

Proof:

(*By Induction on n*)

- For $n = 1$: $\mathbf{Pr}[Q(r) = 0 | Q(x) \neq 0] \leq d/|S|$
- For n:

$$Q(x_1, \dots, x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2, \dots, x_n)$$

where $k \leq d$ is the *largest* exponent of x_1 in Q.
$deg(Q_k) \leq d - k \Rightarrow \mathbf{Pr}[Q_k(r_2, \dots, r_n) = 0] \leq (d-k)/|S|$
Suppose that $Q_k(r_2, \dots, r_n) \neq 0$. Then:

$$q(x_1) = Q(x_1, r_2, \dots, r_n) = \sum_{i=0}^{k} x_1^i Q_i(r_2, \dots, r_n)$$

$deg(q(x_1)) = k$, and $q(x_1) \neq 0$!

Warmup: Polynomial Identity Testing

Proof (cont'd):
The base case now implies that:

$$\mathbf{Pr}[q(r_1) = Q(r_1, \ldots, r_n) = 0] \leq k/|S|$$

Thus, we have shown the following two equalities:

$$\mathbf{Pr}[Q_k(r_2, \ldots, r_n) = 0] \leq \frac{d-k}{|S|}$$

$$\mathbf{Pr}[Q_k(r_1, r_2, \ldots, r_n) = 0 | Q_k(r_2, \ldots, r_n) \neq 0] \leq \frac{k}{|S|}$$

Using the following identity: $\mathbf{Pr}[\mathcal{E}_1] \leq \mathbf{Pr}[\mathcal{E}_1|\overline{\mathcal{E}}_2] + \mathbf{Pr}[\mathcal{E}_2]$ we obtain that the requested probability is no more than the sum of the above, which proves our theorem! $\square$

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 $T : \mathbb{N} \to \mathbb{N}$, 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 $R : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ 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.

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 $co\mathbf{RP}$.

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{NP}$, since every accepting "branch" is a certificate!
- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$
- $\mathbf{RP} = (\exists^+/\forall)$

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**

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**

Quantifier Characterizations

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

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

Theorem (Decisive Characterization of BPP)

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

Quantifier Characterizations

Proof:

- Let $L \in \mathbf{BPP}$. Then, by definition, there exists a polynomial-time computable predicate Q and a polynomial q such that for all x's of length n:

$$x \in L \Rightarrow \exists^{+} y \ Q(x,y)$$

$$x \notin L \Rightarrow \exists^{+} y \ \neg Q(x,y)$$

Swapping Lemma

1. $\forall y \exists^{+} z \ R(x,y,z) \Rightarrow \exists^{+} C \forall y \ \bigvee_{z \in C} R(x,y,z)$
2. $\forall z \exists^{+} y \ R(x,y,z) \Rightarrow \forall C \exists^{+} y \ \bigwedge_{z \in C} R(x,y,z)$

- By the above Lemma: $x \in L \Rightarrow \exists^{+} z \ Q(x,z) \Rightarrow \forall y \exists^{+} z \ Q(x, y \oplus z) \Rightarrow \exists^{+} C \forall y \, [\exists (z \in C) \ Q(x, y \oplus z)]$, where C denotes (as in the Swapping's Lemma formulation) a set of $q(n)$ strings, each of length $q(n)$.

Quantifier Characterizations

Proof (cont'd):

- On the other hand, $x \notin L \Rightarrow \exists^+ y \neg Q(x,z) \Rightarrow \forall z \exists^+ y \neg Q(x, y \oplus z) \Rightarrow \forall C \exists^+ y\, [\forall (z \in C) \neg Q(x, y \oplus z)]$.
- Now, we only have to assure that the appeared predicates $\exists z \in C \;\; Q(x, y \oplus z)$ and $\forall z \in C \; \neg Q(x, y \oplus z)$ are computable in polynomial time
- Recall that in Swapping Lemma's formulation we demanded $|C| \leq p(n)$ and that for each $v \in C: \;\; |v| = p(n)$. This means that we seek if a string of polynomial length *exists*, or if the predicate holds *for all* such strings in a set with polynomial cardinality, procedure which can be surely done in polynomial time.

Quantifier Characterizations

Proof (cont'd):

- Conversely, if $L \in (\exists^+\forall/\forall\exists^+)$, for each string w, $|w| = 2p(n)$, we have $w = w_1 w_2$, $|w_1| = |w_2| = p(n)$. Then:
  $x \in L \Rightarrow \exists^+ y \forall z\ R(x,y,z) \Rightarrow \exists^+ w\ R(x, w_1, w_2)$
  $x \notin L \Rightarrow \forall y \exists^+ z\ R(x,y,z) \Rightarrow \exists^+ w\ \neg R(x, w_1, w_2)$
- So, $L \in$ **BPP**. $\square$

- The above characterization is *decisive*, in the sense that if we replace $\exists^+$ with $\exists$, the two predicates are still complementary (i.e. $R_1 \Rightarrow \neg R_2$), so they still define a complexity class.
- In the above characterization of **BPP**, if we replace $\exists^+$ with $\exists$, we obtain very easily a well-known result:

Corollary (Sipser-Gács Theorem)

$$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$$

BPP and PH

Theorem (Sipser-Gács)

$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Proof (*Lautemann*)
Because $co\mathbf{BPP} = \mathbf{BPP}$,we prove only $\mathbf{BPP} \subseteq \Sigma_2\mathbf{P}$.
Let $L \in \mathbf{BPP}$ (L is accepted by "clear majority").
For $|x| = n$, let $A(x) \subseteq \{0,1\}^{p(n)}$ be the set of *accepting* computations.
We have:

- $x \in L \Rightarrow |A(x)| \geq 2^{p(n)}\left(1 - \frac{1}{2^n}\right)$
- $x \notin L \Rightarrow |A(x)| \leq 2^{p(n)}\left(\frac{1}{2^n}\right)$

Let U be the set of all bit strings of length $p(n)$.
For $a, b \in U$, let $a \oplus b$ be the XOR:
$a \oplus b = c \Leftrightarrow c \oplus b = a$, so "$\oplus b$" is 1-1.

BPP and PH

Proof (cont.)

For $t \in U$, $A(x) \oplus t = \{a \oplus t : a \in A(x)\}$ (translation of $A(x)$ by t). We imply that: $|A(x) \oplus t| = |A(x)|$

If $x \in L$, consider a *random* (drawing $p^2(n)$ bits) sequence of translations: $t_1, t_2, .., t_{p(n)} \in U$.

For $b \in U$, these translations *cover* b, if $b \in A(x) \oplus t_j$, $j \leq p(n)$.

$b \in A(x) \oplus t_j \Leftrightarrow b \oplus t_j \in A(x) \Rightarrow \mathbf{Pr}[b \notin A(x) \oplus t_j]=\frac{1}{2^n}$

$\mathbf{Pr}[\text{b is } \textbf{not} \text{ covered by any } t_j]=2^{-np(n)}$

$\mathbf{Pr}[\exists \text{ point that is } \textbf{not} \text{ covered}]\leq 2^{-np(n)}|U| = 2^{-(n-1)p(n)}$

BPP and PH

Proof (cont.)

So, $T = (t_1, .., t_{p(n)})$ has a positive probability that it covers all of U.
If $x \notin L, |A(x)|$ is exp small,and (for large n) there's not T that cover all U.
$(x \in L) \Leftrightarrow (\exists T$ that cover all $U)$
So,

$$L = \{x | \exists (T \in \{0,1\}^{p^2(n)}) \forall (b \in U) \exists (j \leq p(n)) : b \oplus t_j \in A(x)\}$$

which is precisely the form of languages in $\Sigma_2\mathbf{P}$.
The last existential quantifier $(\exists (j \leq p(n)) ...)$ affects only polynomially many possibilities,so it doesn't "count" (can by tested in polynomial time by trying all t_j's).

$\square$

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:

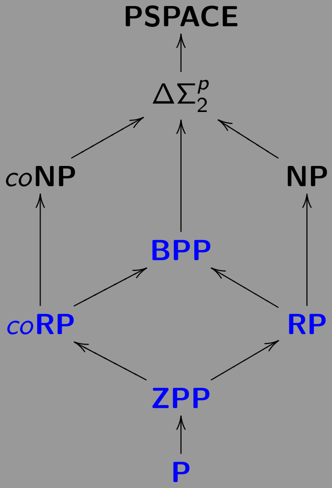
$$\mathbf{ZPP} = \bigcup_{c \in \mathbb{N}} \mathbf{ZTIME}[n^c]$$

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 $\Delta\Sigma_2^p = \Sigma_2^p \cap \Pi_2^p = \mathbf{NP}^{\mathbf{NP}} \cap co\mathbf{NP}^{\mathbf{NP}}$)

BPP and PH



$$
\begin{tikzcd}
 & \textbf{PSPACE} & \\
 & (\exists\forall/\forall\exists) \cap (\forall\exists/\exists\forall) \arrow[u] & \\
(\forall/\exists) \arrow[ur] & & (\exists/\forall) \arrow[ul] \\
 & (\exists^+/\exists^+) \arrow[uu] & \\
(\forall/\exists^+) \arrow[uu] \arrow[ur] & & (\exists^+/\forall) \arrow[uu] \arrow[ul] \\
 & (\forall/\forall) \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

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 $x \in \{0,1\}^*$:

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

Proof: The machine M' does the following:

- `Run` $M(x)$ `for every input` x `for` $k = 8|x|^{2c+d}$ `times, and obtain outputs` $y_1, y_2, \ldots, y_k \in \{0,1\}$.
- `If the majority of these outputs is 1, return 1`
- `Otherwise, return 0.`

We define the r.v. X_i for every $i \in [k]$ to be 1 if $y_i = L(x)$ and 0 otherwise.

$X_1, X_2, \ldots, X_k$ are indepedent Boolean r.v.'s, with:

$$\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] \geq p = \frac{1}{2} + |x|^{-c}$$

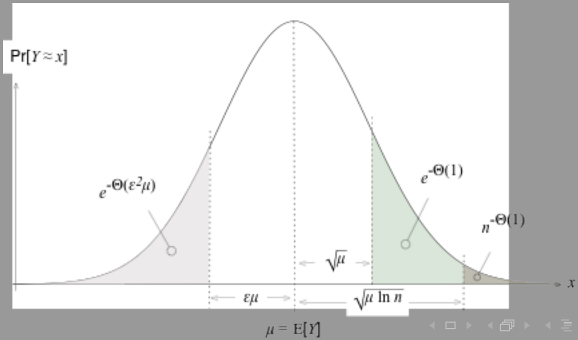
Applying a Chernoff Bound we obtain:

$$\mathbf{Pr}\left[|\sum_{i=1}^{k} X_i - pk| > \delta pk \right] < e^{-\frac{\delta^2}{4} pk} = e^{-\frac{1}{4|x|^{2c}} \frac{1}{2} 8|x|^{2c+d}} \leq 2^{-|x|^d}$$

$\square$

Intermission: Chernoff Bounds

- *How many* samples do we need in order to estimate μ up to an *error* of $\pm\varepsilon$ with *probability* at least $1-\delta$?
- Chernoff Bound tells us that this number is $\mathcal{O}\left(\rho/\varepsilon^2\right)$, where $\rho = \log(1/\delta)$.
- The probability that k is $\rho\sqrt{n}$ far from μn decays **exponentially** with ρ.



Intermission: Chernoff Bounds

$$\mathbf{Pr}\left[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu\right] \leq \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu}$$

$$\mathbf{Pr}\left[\sum_{i=1}^{n} X_i \leq (1-\delta)\mu\right] \leq \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu}$$

Other useful form is:

$$\mathbf{Pr}\left[\left|\sum_{i=1}^{n} X_i - \mu\right| \geq c\mu\right] \leq 2e^{-\min\{c^2/4, c/2\}\cdot\mu}$$

- This probability is bounded by $2^{-\Omega(\mu)}$.

Error Reduction for BPP

- From the above we can obtain the following interesting corollary:

Corollary

For $c > 0$, let $\mathbf{BPP}_{1/2+n^{-c}}$ denote the class of languages L for which there is a polynomial-time PTM M satisfying $\mathbf{Pr}[M(x) = L(x)] \geq 1/2 + |x|^{-c}$ for every $x \in \{0,1\}^*$. Then:

$$\mathbf{BPP}_{1/2+n^{-c}} = \mathbf{BPP}$$

- Obviously, $\exists^+ = \exists^+_{1/2+\varepsilon} = \exists^+_{2/3} = \exists^+_{3/4} = \exists^+_{0.99} = \exists^+_{1-2^{-p(|x|)}}$

Complete Problems for BPP?

- The defining property of **BPTIME** machines is **semantic**!
- We cannot test whether a TM can accept every input string with probability $\geq 2/3$ or with $\leq 1/3$ (**why?**)
- In contrast, the defining property of **NP** is **syntactic**!
- We have:
  - **Syntactic Classes**
  - **Semantic Classes**
- If finally $\mathbf{P} = \mathbf{BPP}$, then **BPP** will have complete problems!!

Complete Problems for BPP?

- The defining property of **BPTIME** machines is **semantic**!
- We cannot test whether a TM can accept every input string with probability $\geq 2/3$ or with $\leq 1/3$ (**why?**)
- In contrast, the defining property of **NP** is **syntactic**!
- We have:
  - **Syntactic Classes**
  - **Semantic Classes**
- If finally $\mathbf{P} = \mathbf{BPP}$, then **BPP** will have complete problems!!

- For the same reason, in semantic classes we cannot prove Hierarchy Theorems using Diagonalization.

The Class PP

Definition

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

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

- Or, more "syntactically":

Definition

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

$$x \in L \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\right\}\right| \geq \frac{1}{2} \cdot 2^{p(|x|)}$$

The Class PP

- Due to the lack of a gap between the two cases, we cannot amplify the probability with polynomially many repetitions, as in the case of **BPP**.
- **PP** is closed under complement.
- A breakthrough result of R. Beigel, N. Reingold and D. Spielman is that **PP** is closed under *intersection*!

The Class PP

- Due to the lack of a gap between the two cases, we cannot amplify the probability with polynomially many repetitions, as in the case of **BPP**.
- **PP** is closed under complement.
- A breakthrough result of R. Beigel, N. Reingold and D. Spielman is that **PP** is closed under *intersection*!
- The syntactic definition of **PP** gives the possibility for *complete problems*:
- Consider the problem `MAJSAT`:
  Given a Boolean Expression, is it true that the majority of the 2^n truth assignments to its variables (that is, at least $2^{n-1}+1$ of them) satisfy it?

The Class PP

Theorem

MAJSAT is **PP***-complete!*

- `MAJSAT` is not likely in **NP**, since the (*obvious*) certificate is not very succinct!

The Class PP

Theorem

MAJSAT is **PP***-complete!*

- MAJSAT is not likely in **NP**, since the (*obvious*) certificate is not very succinct!

Theorem

$$\mathbf{NP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$$

The Class PP

Theorem

$\mathtt{MAJSAT}$ is **PP***-complete!*

- $\mathtt{MAJSAT}$ is not likely in **NP**, since the (*obvious*) certificate is not very succinct!

Theorem

$$\mathbf{NP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$$

Proof:

It is easy to see that **PP** $\subseteq$ **PSPACE**:

We can simulate any **PP** machine by enumerating all strings y of length $p(n)$ and verify whether **PP** machine accepts. The **PSPACE** machine accepts if and only if there are more than $2^{p(n)-1}$ such y's (by using a counter).

The Class PP

Proof (cont'd):

Now, for **NP** $\subseteq$ **PP**, let $A \in$ **NP**. That is, $\exists p \in poly(n)$ and a poly-time and balanced predicate R such that:

$$x \in A \;\Leftrightarrow\; (\exists y, |y| = p(|x|)) : R(x, y)$$

Consider the following TM:

M accepts input (x, by), with $|b| = 1$ and $|y| = p(|x|)$, if and only if $R(x, y) = 1$ or $b = 1$.

- If $x \in A$, then $\exists$ at least one y s.t. $R(x, y)$. Thus, $\mathbf{Pr}[M(x) \text{ accepts}] \geq 1/2 + 2^{-(p(n)+1)}$.
- If $x \notin A$, then $\mathbf{Pr}[M(x) \text{ accepts}] = 1/2$.

$\square$

Other Results

Theorem

If $\mathbf{NP} \subseteq \mathbf{BPP}$*, then* $\mathbf{NP} = \mathbf{RP}$*.*

Other Results

Theorem

If **NP** $\subseteq$ **BPP**, *then* **NP** $=$ **RP**.

Proof:

- **RP** is closed under $\leq_m^p$-reducibility.
- It suffices to show that if $\mathtt{SAT} \in$ **BPP**, then $\mathtt{SAT} \in$ **RP**.
- Recall that $\mathtt{SAT}$ has the **self-reducibility** property: $\phi(x_1, \dots, x_n)$: $\phi \in \mathtt{SAT} \Leftrightarrow (\phi|_{x_1=0} \in \mathtt{SAT} \ \vee \ \phi|_{x_1=1} \in \mathtt{SAT})$.
- $\mathtt{SAT} \in$ **BPP**: $\exists$ PTM M computing $\mathtt{SAT}$ with error probability bounded by $2^{-|\phi|}$.
- We can use the *self-reducibility* of $\mathtt{SAT}$ to produce a truth assignment for ϕ as follows:

Other Results

Proof (cont'd):

Input: A Boolean formula ϕ with n variables
If $M(\phi) = 0$ **then** reject ϕ;
For $i = 1$ to n
$\rightarrow$ **If** $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=0}) = 1$ **then** let $\alpha_i = 0$
$\rightarrow$ **ElseIf** $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=1}) = 1$ **then** let $\alpha_i = 1$
$\rightarrow$ **Else** reject ϕ and halt;
If $\phi|_{x_1=\alpha_1,\ldots,x_n=\alpha_n} = 1$ **then** accept F
Else reject F

Other Results

Proof (cont'd):

`Input:` `A Boolean formula` ϕ `with` n `variables`
If $M(\phi) = 0$ **then** `reject` ϕ;
For $i = 1$ `to` n
$\rightarrow$ **If** $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=0}) = 1$ **then** `let` $\alpha_i = 0$
$\rightarrow$ **ElseIf** $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=1}) = 1$ **then** `let` $\alpha_i = 1$
$\rightarrow$ **Else** `reject` ϕ `and halt;`
If $\phi|_{x_1=\alpha_1,\ldots,x_n=\alpha_n} = 1$ **then** `accept` F
Else `reject` F

- Note that M_1 accepts ϕ *only if* a t.a. $t(x_i) = \alpha_i$ is found.
- Therefore, M_1 never makes mistakes if $\phi \notin \mathtt{SAT}$.
- If $\phi \in \mathtt{SAT}$, then M rejects ϕ on each iteration of the loop w.p. $2^{-|\phi|}$.
- So, $\mathbf{Pr}[M_1 \text{ accepting } x] = (1 - 2^{-|\phi|})^n$, which is greater than $1/2$ if $|\phi| \geq n > 1$. $\square$

Relativized Results

Theorem

Relative to a random oracle A, $\mathbf{P}^A = \mathbf{BPP}^A$. That is,

$$\mathbf{Pr}_A[\mathbf{P}^A = \mathbf{BPP}^A] = 1$$

Also,

- $\mathbf{BPP}^A \subsetneq \mathbf{NP}^A$, relative to a *random* oracle A.
- There exists an A such that: $\mathbf{P}^A \neq \mathbf{RP}^A$.
- There exists an A such that: $\mathbf{RP}^A \neq co\mathbf{RP}^A$
- There exists an A such that: $\mathbf{RP}^A \neq \mathbf{NP}^A$.

Relativized Results

Theorem

Relative to a random oracle A, $\mathbf{P}^A = \mathbf{BPP}^A$. That is,

$$\mathbf{Pr}_A[\mathbf{P}^A = \mathbf{BPP}^A] = 1$$

Also,

- $\mathbf{BPP}^A \subsetneq \mathbf{NP}^A$, relative to a *random* oracle A.
- There exists an A such that: $\mathbf{P}^A \neq \mathbf{RP}^A$.
- There exists an A such that: $\mathbf{RP}^A \neq co\mathbf{RP}^A$
- There exists an A such that: $\mathbf{RP}^A \neq \mathbf{NP}^A$.

Corollary

There exists an A such that:

$$\mathbf{P}^A \neq \mathbf{RP}^A \neq \mathbf{NP}^A \not\subseteq \mathbf{BPP}^A$$

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & Optimization Problems
- Randomized Computation
- **Non-Uniform Complexity**
- Interactive Proofs
- Counting Complexity

Boolean Circuits

- A Boolean Circuit is a natural model of *nonuniform* computation, a generalization of hardware computational methods.

- A non-uniform computational model allows us to use a different "algorithm" to be used for every input size, in contrast to the standard (or *uniform*) Turing Machine model, where the same T.M. is used on (infinitely many) input sizes.

- Each circuit can be used for a fixed input size, which limits or model.

Definition (Boolean circuits)

For every $n \in \mathbb{N}$ an n-input, single output Boolean Circuit C is a directed acyclic graph with n sources and *one* sink.

- All nonsource vertices are called *gates* and are labeled with one of $\wedge$ (and), $\vee$ (or) or $\neg$ (not).
- The vertices labeled with $\wedge$ and $\vee$ have *fan-in* (i.e. number or incoming edges) 2.
- The vertices labeled with $\neg$ have *fan-in* 1.
- The *size* of C, denoted by $|C|$, is the number of vertices in it.
- For every vertex v of C, we assign a value as follows: for some input $x \in \{0,1\}^n$, if v is the i-th input vertex then $val(v) = x_i$, and otherwise $val(v)$ is defined recursively by applying v's logical operation on the values of the vertices connected to v.
- The *output* $C(x)$ is the value of the output vertex.
- The *depth* of C is the length of the longest directed path from an input node to the output node.

- To overcome the fixed input length size, we need to allow families (or sequences) of circuits to be used:

Definition

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A $T(n)$*-size circuit family* is a sequence $\{C_n\}_{n\in\mathbb{N}}$ of Boolean circuits, where C_n has n inputs and a single output, and its size $|C_n| \leq T(n)$ for every n.

- These infinite families of circuits are defined arbitrarily: There is **no** pre-defined connection between the circuits, and also we haven't any "guarantee" that we can construct them efficiently.
- Like each new computational model, we can define a complexity class on it by imposing some restriction on a *complexity measure*:

Definition

We say that a language L is in $\textbf{SIZE}(T(n))$ if there is a $T(n)$-*size circuit family* $\{C_n\}_{n\in\mathbb{N}}$, such that $\forall x \in \{0,1\}^n$:

$$x \in L \Leftrightarrow C_n(x) = 1$$

Definition

$\mathbf{P}_{/\mathbf{poly}}$ is the class of languages that are decidable by polynomial size circuits families. That is,

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c\in\mathbb{N}} \textbf{SIZE}(n^c)$$

Theorem (Nonuniform Hierarchy Theorem)

For every functions $T, T' : \mathbb{N} \to \mathbb{N}$ *with* $\frac{2^n}{n} > T'(n) > 10T(n) > n$,

$$\textbf{SIZE}(T(n)) \subsetneq \textbf{SIZE}(T'(n))$$

Turing Machines that take advice

Definition

Let $T, a : \mathbb{N} \to \mathbb{N}$. The class of languages decidable by $T(n)$-time Turing Machines with $a(n)$ bits of advice, denoted

$$\mathbf{DTIME}\,(T(n)/a(n))$$

containts every language L such that there exists a sequence $\{a_n\}_{n\in\mathbb{N}}$ of strings, with $a_n \in \{0,1\}^{a(n)}$ and a Turing Machine M satisfying:

$$x \in L \Leftrightarrow M(x, a_n) = 1$$

for every $x \in \{0,1\}^n$, where on input (x, a_n) the machine M runs for at most $\mathcal{O}(T(n))$ steps.

Turing Machines that take advice

Theorem (Alternative Definition of $\mathbf{P}_{/\mathbf{poly}}$)

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c,d \in \mathbb{N}} \mathbf{DTIME}(n^c / n^d)$$

Turing Machines that take advice

Theorem (Alternative Definition of $\mathbf{P}_{/\mathbf{poly}}$)

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c,d\in\mathbb{N}} \mathbf{DTIME}(n^c/n^d)$$

Proof: $(\subseteq)$ Let $L \in \mathbf{P}_{/\mathbf{poly}}$. Then, $\exists\{C_n\}_{n\in\mathbb{N}} : C_{|x|} = L(x)$.
We can use C_n 's encoding as an advice string for each n.

Turing Machines that take advice

Theorem (Alternative Definition of $\mathbf{P}_{/\mathbf{poly}}$)

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c,d \in \mathbb{N}} \mathbf{DTIME}(n^c/n^d)$$

Proof: ($\subseteq$) Let $L \in \mathbf{P}_{/\mathbf{poly}}$. Then, $\exists \{C_n\}_{n \in \mathbb{N}} : C_{|x|} = L(x)$.
We can use C_n 's encoding as an advice string for each n.
($\supseteq$) Let $L \in \mathbf{DTIME}(n^c/n^d)$. Then, since $\texttt{CVP}$ is $\mathbf{P}$-complete, we construct for every n a circuit D_n such that, for $x \in \{0,1\}^n, a_n \in \{0,1\}^{a(n)}$:

$$D_n(x, a_n) = M(x, a_n)$$

Then, let $C_n(x) = D_n(x, a_n)$ (We hard-wire the advice string!)
Since $a(n) = n^d$, the circuits have polynomial size. $\square$.

Theorem

$$\mathbf{P} \subsetneq \mathbf{P}_{/\mathbf{poly}}$$

- For "$\subseteq$", recall that $\mathtt{CVP}$ is $\mathbf{P}$-complete.
- But why proper inclusion?
- Consider the following language:

$$\mathtt{U} = \{1^n | n\text{'s binary expression encodes a pair } < M, x > \text{ s.t. } M(x)\downarrow\}$$

- It is easy to see that $\mathtt{U} \in \mathbf{P}_{/\mathbf{poly}}$, but....

Theorem (Karp-Lipton Theorem)

If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{PH} = \Sigma_2^p$.

Theorem (Meyer's Theorem)

If $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{EXP} = \Sigma_2^p$.

Uniform Families of Circuits

- We saw that $\mathbf{P}_{/\mathbf{poly}}$ contains an undecidable language.
- The root of this problem lies in the "weak" definition of such families, since it suffices that $\exists$ a circuit family for L.
- We haven't a way (or an algorithm) to construct such a family.
- So, may be useful to restric or attention to families we can construct efficiently:

Theorem (P-Uniform Families)

A circuit family $\{C_n\}_{n\in\mathbb{N}}$ is $\mathbf{P}$-uniform if there is a polynomial-time T.M. that on input 1^n outputs the description of the circuit C_n.

- But...

Theorem

A language L is computable by a $\mathbf{P}$-uniform circuit family iff $L \in \mathbf{P}$.

Theorem

$$\mathbf{BPP} \subset \mathbf{P}_{/\mathbf{poly}}$$

Proof: Recall that if $L \in \mathbf{BPP}$, then $\exists$ PTM M such that:

$$\mathbf{Pr}_{r \in \{0,1\}^{poly(n)}}[M(x,r) \neq L(x)] < 2^{-n}$$

Then, taking the union bound:

$$\mathbf{Pr}[\exists x \in \{0,1\}^n : M(x,r) \neq L(x)] = \mathbf{Pr}\left[\bigcup_{x \in \{0,1\}^n} M(x,r) \neq L(x)\right] \leq$$

$$\leq \sum_{x \in \{0,1\}^n} \mathbf{Pr}[M(x,r) \neq L(x)] < 2^{-n} + \cdots + 2^{-n} = 1$$

So, $\exists r_n \in \{0,1\}^{poly(n)}$, s.t. $\forall x \{0,1\}^n$: $M(x,r) = L(x)$.
Using $\{r_n\}_{n \in \mathbb{N}}$ as advice string, we have the non-uniform machine.

$\square$

Theorem

The following are equivalent:

1. $A \in \mathbf{P}_{/\mathbf{poly}}$.
2. *There exists a sparse set* S *such that* $A \leq_T^P S$.

Corollary

Every sparse set has polynomial-size circuits.

Definition (Circuit Complexity or Worst-Case Hardness)

For a finite Boolean Function $f : \{0,1\}^n \rightarrow \{0,1\}$, we define the (circuit) *complexity* of f as the size of the smallest Boolean Circuit computing f (that is, $C(x) = f(x), \forall x \in \{0,1\}^n$).

Definition (Average-Case Hardness)

The minimum S such that there is a circuit C of size S such that:

$$\mathbf{Pr}\left[C(x) = f(x)\right] \geq \frac{1}{2} + \frac{1}{S}$$

is called the (average-case) hardness of f.

Hierarchies for Semantic Classes with advice

- We have argued why we can't obtain Hierarchies for semantic measures using classical diagonalization techniques. But using small advice we can have the following results:

Theorem ([Bar02], [GST04])

For $a, b \in \mathbb{R}$, with $1 \leq a < b$:

$$\mathbf{BPTIME}(n^a)/1 \subsetneq \mathbf{BPTIME}(n^b)/1$$

Theorem ([FST05])

For any $1 \leq a \in \mathbb{R}$ there is a real $b > a$ such that:

$$\mathbf{RTIME}(n^b)/1 \subsetneq \mathbf{RTIME}(n^a)/\log(n)^{1/2a}$$

Circuit Lower Bounds

- The significance of proving lower bounds for this computational model is related to the famous "**P** vs **NP**" problem, since:

$$\mathbf{NP} \setminus \mathbf{P}_{/\mathbf{poly}} \neq \emptyset \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

- But...after decades of efforts, The best lower bound for an **NP** language is $5n - o(n)$, proved very recently (2005).
- There are better lower bounds for some special cases, i.e. some restricted classes of circuits, such as: bounded depth circuits, monotone circuits, and bounded depth circuits with "counting" gates.

Definition

Let $PAR : \{0,1\}^n \rightarrow \{0,1\}$ be the *parity* function, which outputs the modulo 2 sum of an n-bit input. That is:

$$PAR(x_1, ..., x_n) \equiv \sum_{i=1}^{n} x_i \pmod 2$$

Theorem

For all constant d, PAR has no polynomial-size circuit of depth d.

- The above result (improved by Håstad and Yao) gives a relatively tight lower bound of $\exp\left(\Omega(n^{1/(d-1)})\right)$, on the size of n-input PAR circuits of depth d.

Definition

For $x, y \in \{0,1\}^n$, we denote $x \preceq y$ if every bit that is 1 in x is also 1 in y. A function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if $f(x) \leq f(y)$ for every $x \preceq y$.

Definition

A Boolean Circuit is *monotone* if it contains only AND and OR gates, and no NOT gates. Such a circuit can only compute monotone functions.

Theorem (Monotone Circuit Lower Bound for `CLIQUE`)

Denote by $CLIQUE_{k,n} : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ the function that on input an adjacency matrix of an n-vertex graph G outputs 1 iff G contains an k-clique. There exists some constant $\epsilon > 0$ such that for every $k \leq n^{1/4}$, there is no monotone circuit of size less than $2^{\epsilon\sqrt{k}}$ that computes $CLIQUE_{k,n}$.

- So, we proved a significant lower bound ($2^{\Omega(n^{1/8})}$)
- The significance of the above theorem lies on the fact that there was some alleged connection between monotone and non-monotone circuit complexity (e.g. that they would be polynomially related). Unfortunately, Éva Tardos proved in 1988 that the gap between the two complexities is exponential.
- Where is the problem finally?
  Today, we know *that a result for a lower bound using such techniques would imply the inversion of strong one-way functions:*

*Natural Proofs [Razborov, Rudich 1994]

Definition

Let $\mathcal{P}$ be the predicate:

> *"A Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ doesn't have n^c-sized circuits for some $c \geq 1$."*

$\mathcal{P}(f) = 0, \forall f \in \mathbf{SIZE}(n^c)$ for a $c \geq 1$. We call this n^c-*usefulness*.

A predicate $\mathcal{P}$ is natural if:

- There is an algorithm $M \in \mathbf{E}$ such that for a function $g : \{0,1\}^n \rightarrow \{0,1\}$: $M(g) = \mathcal{P}(g)$.
- For a random function g: $\mathbf{Pr}[\mathcal{P}(g) = 1] \geq \frac{1}{n}$

Theorem

If strong one-way functions exist, then there exists a constant $c \in \mathbb{N}$ such that there is no n^c-useful natural predicate $\mathcal{P}$.

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & Optimization Problems
- Randomized Computation
- Non-Uniform Complexity
- **Interactive Proofs**
- Counting Complexity

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,...)

Warmup: Interactive Proofs with deterministic Verifier

Definition (Deterministic Proof Systems)

We say that a language L has a k-round deterministic interactive proof system if there is a deterministic Turing Machine V that on input $x, \alpha_1, \alpha_2, \ldots, \alpha_i$ runs in time polynomial in $|x|$, and can have a k-round interaction with any TM P such that:

- $x \in L \Rightarrow \exists P: \ \langle V, P\rangle(x) = 1$ (*Completeness*)
- $x \notin L \Rightarrow \forall P: \ \langle V, P\rangle(x) = 0$ (*Soundness*)

The class **dIP** contains all languages that have a k-round deterministic interactive proof system, where p is polynomial in the input length.

- $\langle V, P\rangle(x)$ denotes the output of V at the end of the interaction with P on input x, and α_i the exchanged strings.
- The above definition does not place limits on the computational power of the Prover!

Warmup: Interactive Proofs with deterministic Verifier

- But...

Theorem

$$\mathbf{dIP} = \mathbf{NP}$$

Proof: Trivially, $\mathbf{NP} \subseteq \mathbf{dIP}$. ✓
Let $L \in \mathbf{dIP}$:

- A certificate is a transcript $(\alpha_1, \dots, \alpha_k)$ causing V to accept, i.e. $V(x, \alpha_1, \dots, \alpha_k) = 1$.
- We can efficiently check if $V(x) = \alpha_1$, $V(x, \alpha_1, \alpha_2) = \alpha_3$ etc...
  - If $x \in L$ such a transcript exists!
  - Conversely, if a transcript exists, we can define define a proper P to satisfy: $P(x, \alpha_1) = \alpha_2$, $P(x, \alpha_1, \alpha_2, \alpha_3) = \alpha_4$ etc., so that $\langle V, P\rangle(x) = 1$, so $x \in L$.
- So $L \in \mathbf{NP}$! $\square$

Probabilistic Verifier: The Class IP

- We saw that if the verifier is a simple deterministic TM, then the interactive proof system is described precisely by the class **NP**.
- 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 $G_1 \cong G_2$.

- $\texttt{GI}$: Given two graphs G_1, G_2, decide if they are isomorphic.
- $\texttt{GNI}$: 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

<u>Verifier</u>: 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.
<u>Prover</u>: Identifies which of G_1, G_2 was used to produce H.
Let G_j be the graph. Sends j to V.
<u>Verifier</u>: Accept if $i = j$. Reject otherwise.

- If $G_1 \ncong G_2$, then the powerfull prover can (nondeterministively) 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.

Babai's Arthur-Merlin Games

Definition (Extended (FGMSZ89))

An Arhur-Merlin Game is a pair of interactive TMs A and M, and a predicate R such that:

- On input x, exactly $2q(|x|)$ messages of length $m(|x|)$ are exchanged, $q, m \in poly(|x|)$.
- A goes first, and at iteration $1 \leq i \leq q(|x|)$ chooses u.a.r. a string r_i of length $m(|x|)$.
- M's reply in the i^{th} iteration is $y_i = M(x, r_1, \ldots, r_i)$ (M's strategy).
- For every M', a **conversation** between A and M' on input x is $r_1 y_1 r_2 y_2 \cdots r_{q(|x|)} y_{q(|x|)}$.
- The set of all conversations is denoted by $CONV_x^{M'}$, $|CONV_x^{M'}| = 2^{q(|x|)m(|x|)}$.

Babai's Arthur-Merlin Games

Definition (*cont'd*)

- The predicate R maps the input x and a conversation to a Boolean value.
- The set of accepting conversations is denoted by $ACC_x^{R,M}$, and is the set:

$$\{r_1 \cdots r_q | \exists y_1 \cdots y_q \; s.t. \; r_1 y_1 \cdots r_q y_q \in CONV_x^M \wedge R(r_1 y_1 \cdots r_q y_q) = 1\}$$

- A language L has an Arthur-Merlin proof system if:
  - **There exists** a strategy for M, such that for all $x \in L$: $\frac{ACC_x^{R,M}}{CONV_x^M} \geq \frac{2}{3}$ (*Completeness*)
  - **For every** strategy for M, and for every $x \notin L$: $\frac{ACC_x^{R,M}}{CONV_x^M} \leq \frac{1}{3}$ (*Soundness*)

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 $\rightarrow$ Prover**
- **Arthur $\rightarrow$ 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** $\rightarrow$ **Prover**
- **Arthur** $\rightarrow$ **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 $p \in poly(n)$:

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

- So,

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

Properties of Arthur-Merlin Games

- $\mathbf{MA} \subseteq \mathbf{AM}$
- $\mathbf{MA}[1] = \mathbf{NP}$, $\mathbf{AM}[1] = \mathbf{BPP}$
- **AM** could be intuitively approached as the probabilistic version of **NP** (usually denoted as $\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP}$).
- $\mathbf{AM} \subseteq \Pi_2^p$ and $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$.
- $\mathbf{NP}^{\mathbf{BPP}} \subseteq \mathbf{MA}$, $\mathbf{MA}^{\mathbf{BPP}} = \mathbf{MA}$, $\mathbf{AM}^{\mathbf{BPP}} = \mathbf{AM}$ and $\mathbf{AM}^{\Delta\Sigma_1^p} = \mathbf{AM}^{\mathbf{NP} \cap co\mathbf{NP}} = \mathbf{AM}$
- If we consider the complexity classes $\mathbf{AM}[k]$ (the languages that have Arthur-Merlin proof systems of a bounded number of rounds, they form an hierarchy:

$$\mathbf{AM}[0] \subseteq \mathbf{AM}[1] \subseteq \cdots \subseteq \mathbf{AM}[k] \subseteq \mathbf{AM}[k+1] \subseteq \cdots$$

- Are these inclusions proper ? ? ?

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

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

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

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

Properties of Arthur-Merlin Games

Theorem

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

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

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathbf{N} \cdot \mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$ (*the last inclusion holds by quantifier contraction*). Also, $(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.

ii) Similarly, $\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$. Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

1. $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
2. $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathbf{N}\cdot\mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$ (*the last inclusion holds by quantifier contraction*). Also, $(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.

ii) Similarly,
$\mathbf{AM} = \mathbf{BP}\cdot\mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$.
Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

1. $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
2. $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathbf{N}\cdot\mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$ (*the last inclusion holds by quantifier contraction*). Also, $(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.

ii) Similarly,
$\mathbf{AM} = \mathbf{BP}\cdot\mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$.
Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

1. (i) $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
2. (ii) $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ (1) (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ (2)

i) $\mathbf{MA} = \mathbf{N}\cdot\mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \stackrel{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$ (*the last inclusion holds by quantifier contraction*). Also, $(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.

ii) Similarly,
$\mathbf{AM} = \mathbf{BP}\cdot\mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$.
Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

$$\mathbf{MA} \subseteq \mathbf{AM}$$

Proof:
Obvious from (2): $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$. $\square$

Theorem

1. $\mathbf{AM} \subseteq \Pi_2^p$
2. $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$

Proof:
i) $\mathbf{AM} = (\forall\exists/\exists^+\forall) \subseteq (\forall\exists/\exists\forall) = \Pi_2^p$
ii) $\mathbf{MA} = (\exists\forall/\forall\exists^+) \subseteq (\exists\forall/\forall\exists) = \Sigma_2^p$, and
$\mathbf{MA} \subseteq \mathbf{AM} \Rightarrow \mathbf{MA} \subseteq \Pi_2^p$. So, $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(1)}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(1)}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(1)}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(1)}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(1)}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- The Arthur-Merlin Hierarchy collapses at its second level:

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(1)}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Proof:

- The general case is implied by the generalization of BPP-Theorem (1) & (2):
- $(\mathbf{Q_1}\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\exists^+\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\forall\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ $(\mathbf{1'})$
- $(\mathbf{Q_1}\exists\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) \subseteq (\mathbf{Q_1}\forall\exists\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ $(\mathbf{2'})$
- Using the above we can easily see that the Arthur-Merlin Hierarchy collapses at the second level. (*Try it!*) $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

If co$\mathbf{NP} \subseteq \mathbf{AM}$ *(that is, if* $\mathtt{GI}$ *is* $\mathbf{NP}$*-complete), then the Polynomial Hierarchy collapses at the second level, and* $\mathbf{PH} = \Sigma_2^p = \mathbf{AM}$.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if* $\mathtt{GI}$ *is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$
Then:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if* $\mathcal{GI}$ *is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$
Then:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if* GI *is* **NP***-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM***.*

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

If co$\mathbf{NP} \subseteq \mathbf{AM}$ *(that is, if* $\mathtt{GI}$ *is* $\mathbf{NP}$*-complete), then the Polynomial Hierarchy collapses at the second level, and* $\mathbf{PH} = \Sigma_2^p = \mathbf{AM}$.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$
Then:
$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

Measure One Results

- $\mathbf{P}^A \neq \mathbf{NP}^A$, for almost all oracles A.
- $\mathbf{P}^A = \mathbf{BPP}^A$, for almost all oracles A.
- $\mathbf{NP}^A = \mathbf{AM}^A$, for almost all oracles A.

Definition

$$almost\mathcal{C} = \left\{ L | \mathbf{Pr}_{A \in \{0,1\}^*} \left[L \in \mathcal{C}^A \right] = 1 \right\}$$

Theorem

1. $almost\mathbf{P} = \mathbf{BPP}$ *[BG81]*
2. $almost\mathbf{NP} = \mathbf{AM}$ *[NW94]*
3. $almost\mathbf{PH} = \mathbf{PH}$

Measure One Results

Theorem (Kurtz)

For almost every pair of oracles B, C:

1. (i) $\mathbf{BPP} = \mathbf{P}^B \cap \mathbf{P}^C$
2. (ii) $almost\mathbf{NP} = \mathbf{NP}^B \cap \mathbf{NP}^C$

Indicative Open Questions

- Does exist an oracle separating **AM** from *almost***NP**?
- Is *almost***NP** contained in some finite level of Polynomial-Time Hierarchy?
- *Motivated by [BHZ]*: If $co\mathbf{NP} \subseteq almost\mathbf{NP}$, does it follow that **PH** collapses?

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 **BPP** $\subseteq \Sigma_2^p$, and many researchers believe that **P** $=$ **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)

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

The power of Interactive Proofs

Lemma 1

$$\mathbf{IP} \subseteq \mathbf{PSPACE}$$

Warmup: Interactive Proof for UNSAT

Lemma 2

$$\mathbf{PSPACE} \subseteq \mathbf{IP}$$

- For simplicity, we will construct an Interactive Proof for `UNSAT` (a *co***NP**-complete problem), showing that:

Theorem

$$co\mathbf{NP} \subseteq \mathbf{IP}$$

- Let N be a prime.
- We will translate a **formula** ϕ with m clauses and n variables $x_1, \dots, x_n$ to a **polynomial** p over the field $(modN)$ (where $N > 2^n \cdot 3^m$), in the following way:

Arithmetization

- Arithmetic generalization of a CNF Boolean Formula.

$$\begin{array}{rcl} \mathtt{T} & \longrightarrow & 1 \\ \mathtt{F} & \longrightarrow & 0 \\ \neg x & \longrightarrow & 1 - x \\ \wedge & \longrightarrow & \times \\ \vee & \longrightarrow & + \end{array}$$

Example

$$(x_3 \vee \neg x_5 \vee x_{17}) \wedge (x_5 \vee x_9) \wedge (\neg x_3 \vee x_4)$$
$$\downarrow$$
$$(x_3 + (1 - x_5) + x_{17}) \cdot (x_5 + x_9) \cdot ((1 - x_3) + (1 - x_4))$$

- Each literal is of degree 1, so the polynomial p is of degree at most m.
- Also, $0 < p < 3^m$.

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$ picks $r_n \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$ picks $r_n \in \{0, \ldots, N-1\}$ checks if $q_n(r_n) = p(r_1, \ldots, r_n)$

Warmup: Interactive Proof for UNSAT

- If ϕ is **unsatisfiable**,then

$$\sum_{x_1\in\{0,1\}}\sum_{x_2\in\{0,1\}}\cdots\sum_{x_n\in\{0,1\}} p(x_1,\ldots,x_n)\equiv 0\ (modN)$$

  and the protocol will succeed.
- Also, the arithmetization can be done in polynomial time, and if we take $N=2^{\mathcal{O}(n+m)}$, then the elements in the field can be represented by $\mathcal{O}(n+m)$ bits, and thus an evaluation of p in any point of $\{0,\ldots,N-1\}$ can be computed in polynomial time.
- We have to show that if ϕ is satisfiable, then the verifier will **reject** with high probability.
- If ϕ is satisfiable, then $\sum_{x_1\in\{0,1\}}\sum_{x_2\in\{0,1\}}\cdots\sum_{x_n\in\{0,1\}} p(x_1,\ldots,x_n)\neq 0\ (modN)$

- So, $p_1(01) + p_1(1) \neq 0$, so if the prover send p_1 we 're done.
- If the prover send $q_1 \neq p_1$, then the polynomials will agree on at most m places. So, $\mathbf{Pr}\left[p_1(r_1) \neq q_1(r_1)\right] \geq 1 - \frac{m}{N}$.
- If indeed $p_1(r_1) \neq q_1(r_1)$ and the prover sends $p_2 = q_2$, then the verifier will reject since $q_2(0) + q_2(1) = p_1(r_1) \neq q_1(r_1)$.
- Thus, the prover must send $q_2 \neq p_2$.
- We continue in a similar way: If $q_i \neq p_i$, then with probability at least $1 - \frac{m}{N}$, r_i is such that $q_i(r_i) \neq p_i(r_i)$.
- Then, the prover must send $q_{i+1} \neq p_{i+1}$ in order for the verifier not to reject.
- At the end, if the verifier has not rejected before the last check, $\mathbf{Pr}\left[p_n \neq q_n\right] \geq 1 - (n-1)\frac{m}{N}$.
- If so, with probability at least $1 - \frac{m}{N}$ the verifier will reject since, $q_n(x)$ and $p(r_1, \ldots, r_{n-1}, x)$ differ on at least that fraction of points.
- The total probability that the verifier will accept if at most $\frac{nm}{N}$.

Arithmetization of QBF

$$\begin{array}{ccc} \exists & \longrightarrow & \sum \\ \forall & \longrightarrow & \prod \end{array}$$

Example

$$\forall x_1 \exists x_2 [(x_1 \wedge x_2) \vee \exists x_3 (\bar{x}_2 \wedge x_3)]$$

$$\downarrow$$

$$\prod_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \left[(x_1 \cdot x_2) + \sum_{x_3 \in \{0,1\}} (1 - x_2) \cdot x_3 \right]$$

Theorem

A closed QBF is true if and only if tha value of its arithmetic form is non-zero.

Arithmetization of QBF

- If a QBF is `true`, its value could be quite large:

Theorem

Let A be a closed QBF of size n. Then, the value of its arithmetic form cannot exceed $\mathcal{O}\left(2^{2^n}\right)$.

- Since such numbers cannot be handled by the protocol, we reduce them modulo some -smaller- prime p:

Theorem

Let A be a closed QBF of size n. Then, there exists a prime p of length polynomial in n, such that its arithmetization

$$A' \neq 0(mod p) \Leftrightarrow A \text{ is true.}$$

Arithmetization of QBF

- A QBF with all the variables quantified is called **closed**, and can be evaluated to either `True` or `False`.
- An **open** QBF with $k > 0$ free variables can be interpreted as a boolean function $\{0,1\}^k \to \{0,1\}$.
- Now, consider the language of all true quantified boolean formulas:

$$\mathtt{TQBF} = \{\Phi | \Phi \text{ is a true quantified Boolean formula}\}$$

- It is known that `TQBF` is a **PSPACE**-complete language!
- So, if we have a interactive proof protocol recognizing `TQBF`, then we have a protocol for every **PSPACE** language.

Protocol for TQBF

- Given a quantified formula

$$\Psi = \forall x_1 \exists x_2 \forall x_3 \cdots \exists x_n \ \phi(x_1, \ldots, x_n)$$

we use *arithmetization* to construct the polynomial P_ϕ. Then, $\Psi \in \texttt{TQBF}$ if and only if

$$\prod_{b_1 \in \{0,1\}^*} \sum_{b_2 \in \{0,1\}^*} \prod_{b_3 \in \{0,1\}^*} \cdots \sum_{b_n \in \{0,1\}^*} P_\phi(b_1, \ldots, b_n) \neq 0$$

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 $\exists \pi \in \{0,1\}^* : \mathbf{Pr}[V^{\pi}(x) = 1] = 1$
- *Soundness*: If $x \notin L$, then $\forall \pi \in \{0,1\}^* : \mathbf{Pr}[V^{\pi}(x) = 1] \leq \frac{1}{2}$

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:

$$\mathbf{PCP}(0,0) = \textbf{?}$$
$$\mathbf{PCP}(0, poly) = \textbf{?}$$
$$\mathbf{PCP}(poly, 0) = \textbf{?}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, poly) = \textbf{?}$$
$$\mathbf{PCP}(poly, 0) = \textbf{?}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, \mathit{poly}) = \mathbf{NP}$$
$$\mathbf{PCP}(\mathit{poly}, 0) = \mathbf{?}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, poly) = \mathbf{NP}$$
$$\mathbf{PCP}(poly, 0) = co\mathbf{RP}$$

Main Results

- Obviously:

$$\mathbf{PCP}(0,0) = \mathbf{P}$$
$$\mathbf{PCP}(0, poly) = \mathbf{NP}$$
$$\mathbf{PCP}(poly, 0) = co\mathbf{RP}$$

- 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.

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & Optimization Problems
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- **Counting Complexity**

Why counting?

- So far, we have seen two versions of problems:
  - Decision Problems (if a solution *exists*)
  - Function Problems (if a solution can be *produced*)
- A very important type of problems in Complexity Theory is also:
  - Counting Problems (*how many* solution exist)

Example (#SAT)

Given a Boolean Expression, compute the number of different truth assignments that satisfy it.

- Note that if we can solve #SAT in polynomial time, we can solve SAT also.
- Similarly, we can define #HAMILTON PATH, #CLIQUE, etc.

Basic Definitions

Definition (#P)

A function $f : \{0,1\}^* \to \mathbb{N}$ is in #**P** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing Machine M such that for every $x \in \{0,1\}^*$:

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$$

- The definition implies that $f(x)$ can be expressed in $poly(|x|)$ bits.
- Each function f in #**P** is equal to the number of paths from an initial configuration to an accepting configuration, or **accepting paths** in the configuration graph of a poly-time NDTM.
- $\mathbf{FP} \subseteq \#\mathbf{P} \subseteq \mathbf{PSPACE}$
- If $\#\mathbf{P} = \mathbf{FP}$, then $\mathbf{P} = \mathbf{NP}$.
- If $\mathbf{P} = \mathbf{PSPACE}$, then $\#\mathbf{P} = \mathbf{FP}$.

- In order to formalize a notion of completeness for $\#\mathbf{P}$, we must define proper reductions:

Definition (Cook Reduction)

A function f is $\#\mathbf{P}$-complete if it is in $\#\mathbf{P}$ and every $g \in \#\mathbf{P}$ is in $\mathbf{FP}^g$.

- As we saw, for each problem in **NP** we can define the associated counting problem: If $A \in \mathbf{NP}$, then $\#A(x) = |\{y \in \{0,1\}^{p(|x|)} : \ R_A(x,y) = 1\}| \in \#\mathbf{P}$

- In order to formalize a notion of completeness for $\#\mathbf{P}$, we must define proper reductions:

Definition (Cook Reduction)

A function f is $\#\mathbf{P}$-complete if it is in $\#\mathbf{P}$ and every $g \in \#\mathbf{P}$ is in $\mathbf{FP}^g$.

- As we saw, for each problem in **NP** we can define the associated counting problem: If $A \in \mathbf{NP}$, then $\#A(x) = |\{y \in \{0,1\}^{p(|x|)} : \ R_A(x,y) = 1\}| \in \#\mathbf{P}$
- We now define a more strict form of reduction:

Definition (Parsimonious Reduction)

We say that there is a parsimonious reduction from $\#A$ to $\#B$ if there is a polynomial time transformation f such that for all x:

$$|\{y : \ R_A(x,y) = 1\}| = |\{z : \ R_B(f(x),z) = 1\}|$$

Completeness Results

Theorem

$\#$CIRCUIT SAT *is* $\#\mathbf{P}$*-complete.*

Proof:

- Let $f \in \#\mathbf{P}$. Then, $\exists M, p$:
  $f = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$.
- Given x, we want to construct a circuit C such that:

$$|\{z : \; C(z)\}| = |\{y : \; y \in \{0,1\}^{p(|x|}, M(x,y) = 1\}|$$

- We can construct a circuit $\hat{C}$ such that on input x, y simulates $M(x,y)$.
- We know that this can be done with a circuit with size about the square of M's running time.
- Let $C(y) = \hat{C}(x,y)$. $\square$

Completeness Results

Theorem

*#SAT is #**P**-complete.*

Proof:

- We reduce #CIRCUIT SAT to #SAT:
- Let a circuit C, with $x_1, \ldots, x_n$ input gates and $1, \ldots, m$ gates.
- We construct a Boolean formula ϕ with variables $x_1, \ldots, x_n, g_1, \ldots, g_m$, where g_i represents the output of gate i.
- A gate can be complete described by simulating the output for each of the 4 possible inputs.
- In this way, we have reduced C to a formula ϕ with $n + m$ variables and $4m$ clauses. $\square$

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0, 1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $\{x_i, y_i\} \in E$ iff $A_{i,j} = 1$.

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0,1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$ and $\{x_i, y_i\} \in E$ iff $A_{i,j} = 1$.
- The term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ has a perfect matching.

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0, 1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $\{x_i, y_i\} \in E$ iff $A_{i,j} = 1$.
- The term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ has a perfect matching.
- So, in this case $perm(A)$ is the number of perfect matchings in the corresponding graph!

Valiant's Theorem

Theorem (Valiant's Theorem)

*PERMANENT is #**P**-complete.*

- Notice that the decision version of `PERMANENT` is in **P** ! !

Quantifiers vs Counting

- An imporant open question in the 80s concerned the relative power of Polynomial Hierarchy and $\#\mathbf{P}$.
- Both are natural generalizations of **NP**, but it seemed that their features were not directly comparable to each other.
- But, in 1989, S. Toda showed the following theorem:

Quantifiers vs Counting

- An imporant open question in the 80s concerned the relative power of Polynomial Hierarchy and $\#\mathbf{P}$.
- Both are natural generalizations of **NP**, but it seemed that their features were not directly comparable to each other.
- But, in 1989, S. Toda showed the following theorem:

Theorem (Toda's Theorem)

$$\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}[1]}$$

The Class $\oplus\mathbf{P}$

Definition

A language L is in the class $\oplus\mathbf{P}$ if there is a NDTM M such that for all strings x, $x \in L$ iff the *number of accepting paths* on input x is odd.

- The problems $\oplus$`SAT` and $\oplus$`HAMILTON PATH` are $\oplus\mathbf{P}$-complete.
- $\oplus\mathbf{P}$ is closed under complement.

Theorem

$$\mathbf{NP} \subseteq \mathbf{RP}^{\oplus\mathbf{P}}$$