

Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ)
Αλγόριθμοι & Πολυπλοκότητα ΙΙ (ΜΠΛΑ)

Εργαστήριο Λογικής και Επιστήμης Υπολογισμών
Εθνικό Μετσόβιο Πολυτεχνείο

2015-2016



Πληροφορίες Μαθήματος

Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ)

Αλγόριθμοι & Πολυπλοκότητα ΙΙ (ΜΠΛΥ), Λ4-Υπ.

- Διδάσκοντες: Σ. Ζάχος, Ά. Παγουρτζής
- Βοηθοί Διδασκαλίας: Α. Αντωνόπουλος, Α. Χαλκή
- Επιμέλεια Διαφανειών: Α. Αντωνόπουλος
- Δευτέρα: 17:00 - 19:00 (1.1.31, Παλιά Κτίρια ΗΜΜΥ, ΕΜΠ)
Πέμπτη: 15:00 - 17:00 (1.1.31, Παλιά Κτίρια ΗΜΜΥ, ΕΜΠ)
- Ώρες Γραφείου: Μετά από κάθε μάθημα, Παρασκευή 11:00-13:00
- Σελίδα: www.corelab.ntua.gr/courses/complexity/
- Βαθμολόγηση:
 - Διαγώνισμα: 6 μονάδες
 - Ασκήσεις: 2 μονάδες
 - Ομιλία: 2 μονάδες
 - Quizes : 2 μονάδες

Computational Complexity

Graduate Course

Computation and Reasoning Laboratory
National Technical University of Athens

2015-2016

Professors:

S. Zachos, Professor

A. Pagourtzis, Ass. Professor

Slides: Antonis Antonopoulos



This work is licensed under a Creative Commons Attribution-NonCommercial- NoDerivatives 4.0 International License.

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

Why Complexity?

- *Computational Complexity*: Quantifying the amount of computational resources required to solve a given task. Classify computational problems according to their inherent difficulty in complexity classes, and prove relations among them.
- *Structural Complexity*: “The study of the relations between various complexity classes and the global properties of individual classes. [...] The goal of structural complexity is a thorough understanding of the relations between the various complexity classes and the internal structure of these complexity classes.” [J. Hartmanis]

Decision Problems

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

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

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

Example

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

Optimization Problems

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

Example

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

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

A Model Discussion

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

Efficiently Computable \equiv Polynomial-Time Computable

Contents

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

Definition

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

- $Q = \{q_0, q_1, q_2, q_3, \dots, q_n, q_{\text{halt}}, q_{\text{yes}}, q_{\text{no}}\}$ is a finite set of states.
 - Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
 - $q_0 \in Q$ is the initial state.
 - $F \subseteq Q$ is the set of final states.
 - $\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{S, L, R\}$ is the transition function.
-
- A TM is a “programming language” with a single data structure (a tape), and a cursor, which moves left and right on the tape.
 - Function δ is the *program* of the machine.

Turing Machines and Languages

Definition

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

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

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

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

Definitions

Definition

If for a language L there is a TM M , which if $x \in L$ then $M(x) = \text{"yes"}$, and if $x \notin L$ then $M(x) \uparrow$, we call L **recursively enumerable**.

*By $M(x) \uparrow$ we mean that M does not halt on input x (it runs forever).

Theorem

If L is recursive, then it is recursively enumerable.

Proof: *Exercise*

Definition

If f is a function, $f : \Sigma^* \rightarrow \Sigma^*$, we say that a TM M computes f if, for any string $x \in \Sigma^*$, $M(x) = f(x)$. If such M exists, f is called a **recursive function**.

- Turing Machines can be thought as algorithms for solving string related problems.

Multitape Turing Machines

- We can extend the previous Turing Machine definition to obtain a Turing Machine with multiple tapes:

Definition

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

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

Bounds on Turing Machines

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

Definition

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

Definition

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

Multitape Turing Machines

Theorem

Given any k -tape Turing Machine M operating within time $T(n)$, we can construct a TM M' operating within time $\mathcal{O}(T^2(n))$ such that, for any input $x \in \Sigma^$, $M(x) = M'(x)$.*

Proof: See Th.2.1 (p.30) in [1].

- This is a strong evidence of the robustness of our model:
Adding a bounded number of strings does not increase their computational capabilities, and affects their efficiency only polynomially.

Linear Speedup

Theorem

Let M be a TM that decides $L \subseteq \Sigma^$, that operates within time $T(n)$. Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within time $T'(n) = \varepsilon T(n) + n + 2$.*

Proof: See Th.2.2 (p.32) in [1].

- If, for example, T is linear, i.e. something like cn , then this theorem states that the constant c can be made arbitrarily close to 1. So, it is fair to start using the $\mathcal{O}(\cdot)$ notation in our time bounds.
- A similar theorem holds for space:

Theorem

Let M be a TM that decides $L \subseteq \Sigma^$, that operates within space $S(n)$. Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within space $S'(n) = \varepsilon S(n) + 2$.*

Nondeterministic Turing Machines

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

Definition

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

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

Nondeterministic Turing Machines

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

Definition

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

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

Contents

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

Diagonalization

Theorem

The functions from \mathbb{N} to \mathbb{N} are uncountable.

Proof: Let, for the sake of contradiction that are countable:

ϕ_1, ϕ_2, \dots . Consider the following function: $f(x) = \phi_x(x) + 1$.

This function must appear somewhere in this enumeration, so let

$\phi_y = f(x)$. Then $\phi_y(x) = \phi_x(x) + 1$, and if we choose y as an argument, then $\phi_y(y) = \phi_y(y) + 1$. \square

Machines as strings

- It is obvious that we can represent a Turing Machine as a string: *just write down the description and encode it using an alphabet, e.g. $\{0, 1\}$.*
- We denote by $\lfloor M \rfloor$ the TM M 's representation as a string.
- Also, if $x \in \Sigma^*$, we denote by M_x the TM that x represents.

Keep in mind that:

- **Every string represents some Turing Machine.**
- **Every TM is represented by infinitely many strings.**

The Universal Turing Machine

- So far, our computational models are specified to solve a single problem.
- Turing observed that there is a TM that can simulate any other TM M , given M 's description as input.

Theorem

There exists a TM \mathcal{U} such that for every $x, w \in \Sigma^$, $\mathcal{U}(x, w) = M_w(x)$.*

Also, if M_w halts within T steps on input x , then $\mathcal{U}(x, w)$ halts within $CT \log T$ steps, where C is a constant independent of x , and depending only on M_w 's alphabet size number of tapes and number of states.

Proof: See section 3.1 in [1], and Th. 1.9 and section 1.7 in [2].

The Halting Problem

- Consider the following problem: “Given the description of a TM M , and a string x , will M halt on input x ? ” This is called the HALTING PROBLEM.
- **We want to compute this problem ! ! !** (Given a computer program and an input, will this program enter an infinite loop?)
- In language form: $H = \{ \langle M \rangle; x \mid M(x) \downarrow \}$, where “ \downarrow ” means that the machine halts, and “ \uparrow ” that it runs forever.

Theorem

H is recursively enumerable.

Proof: See Th.3.1 (p.59) in [1]

- In fact, H is not just a recursively enumerable language:
If we had an algorithm for deciding H , then we would be able to derive an algorithm for deciding any r.e. language (**RE**-complete).

The Halting Problem

- But....

Theorem

H is not recursive.

Proof:

See Th.3.1 (p.60) in [1]

- Suppose, for the sake of contradiction, that there is a TM M_H that decides H .
- Consider the TM D :

$D(\langle M \rangle) : \text{if } M_H(\langle M \rangle; \langle M \rangle) = \text{"yes"} \text{ then } \uparrow \text{ else "yes"}$
- What is $D(\langle D \rangle)$?
- If $D(\langle D \rangle) \uparrow$, then M_H accepts the input, so $\langle D \rangle; \langle D \rangle \in H$, so $D(D) \downarrow$.
- If $D(\langle D \rangle) \downarrow$, then M_H rejects $\langle D \rangle; \langle D \rangle$, so $\langle D \rangle; \langle D \rangle \notin H$, so $D(D) \uparrow$. \square

- Recursive languages are a *proper* subset of recursive enumerable ones.
- Recall that the complement of a language L is defined as:

$$\bar{L} = \{x \in \Sigma^* \mid x \notin L\} = \Sigma^* \setminus L$$

Theorem

- ① *If L is recursive, so is \bar{L} .*
- ② *L is recursive if and only if L and \bar{L} are recursively enumerable.*

Proof: Exercise

More Undecidability

- The HALTING PROBLEM, our first undecidable problem, was the first, but not the only undecidable problem. It spawns a wide range of such problems, via *reductions*.
- To show that a problem A is undecidable we establish that, if there is an algorithm for A , then there would be an algorithm for H , which is absurd.

Theorem

The following languages are not recursive:

- ① $\{M \mid M \text{ halts on all inputs}\}$
- ② $\{M; x \mid \text{There is a } y \text{ such that } M(x) = y\}$
- ③ $\{M; x \mid \text{The computation of } M \text{ uses all states of } M\}$
- ④ $\{M; x; y \mid M(x) = y\}$

Rice's Theorem

- The previous problems lead us to a more general conclusion:

**Any non-trivial property of
Turing Machines is undecidable**

- If a TM M accepts a language L , we write $L = L(M)$:

Theorem (Rice's Theorem)

Suppose that \mathcal{C} is a proper, non-empty subset of the set of all recursively enumerable languages. Then, the following problem is undecidable:

Given a Turing Machine M , is $L(M) \in \mathcal{C}$?

Rice's Theorem

Proof:

See Th.3.2 (p.62) in [1]

- We can assume that $\emptyset \notin \mathcal{C}$ (*why?*).
- Since \mathcal{C} is nonempty, $\exists L \in \mathcal{C}$, accepted by the TM M_L .
- Let M_H the TM deciding the HALTING PROBLEM for an arbitrary input x . For each $x \in \Sigma^*$, we construct a TM M as follows:

$M(y) : \text{if } M_H(x) = \text{"yes"} \text{ then } M_L(y) \text{ else } \uparrow$

- We claim that: $L(M) \in \mathcal{C}$ if and only if $x \in H$.

Proof of the claim:

- If $x \in H$, then $M_H(x) = \text{"yes"}$, and so M will accept y or never halt, depending on whether $y \in L$. Then the language accepted by M is exactly L , which is in \mathcal{C} .
- If $M_H(x) \uparrow$, M never halts, and thus M accepts the language \emptyset , which is not in \mathcal{C} . \square

Contents

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

Proof (*cont'd*):

Definition (Configuration Graph of a TM)

The configuration graph of M on input x , denoted $G(M, x)$, has as **vertices** all the possible configurations, and there is an **edge** between two vertices C and C' if and only if C' can be reached from C in one step, according to M 's transition function.

- So, we have reduced this simulation to REACHABILITY* problem (also known as S-T CONN), for which we know there is a poly-time ($\mathcal{O}(n^2)$) algorithm.
- So, the simulation takes $c_2 c_1^{2(\log n + S(n))} \sim k^{\log n + S(n)}$ steps. \square

*REACHABILITY: Given a graph G and two nodes $v_1, v_n \in V$, is there a path from v_1 to v_n ?

The essential Complexity Hierarchy

Definition

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

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

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

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

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

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

The essential Complexity Hierarchy

Definition

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

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

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

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

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

Certificate Characterization of NP

Definition

Let $R \subseteq \Sigma^* \times \Sigma^*$ a binary relation on strings.

- R is called **polynomially decidable** if there is a DTM deciding the language $\{x; y \mid (x, y) \in R\}$ in polynomial time.
- R is called **polynomially balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$, for some $k \geq 1$.

Theorem

Let $L \subseteq \Sigma^$ be a language. $L \in \mathbf{NP}$ if and only if there is a polynomially decidable and polynomially balanced relation R , such that:*

$$L = \{x \mid \exists y R(x, y)\}$$

- This y is called **succinct certificate**, or **witness**.

Proof:

See Pr.9.1 (p.181) in [1]

(\Leftarrow) If such an R exists, we can construct the following NTM deciding L :

“On input x , *guess* a y , such that $|y| \leq |x|^k$, and then test (in poly-time) if $(x, y) \in R$. If so, accept, else reject.” Observe that an accepting computation exists if and only if $x \in L$.

(\Rightarrow) If $L \in \mathbf{NP}$, then \exists an NTM N that decides L in time $|x|^k$, for some k . Define the following R :

“(x, y) $\in R$ if and only if y is an **encoding** of an accepting computation of $N(x)$.”

R is polynomially balanced and decidable (*why?*), so, given by assumption that N decides L , we have our conclusion. \square

Can creativity be automated?

As we saw:

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

If $P = NP$...

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

Complements of complexity classes

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

$$P \subseteq NP \cap coNP$$

- Note the *similarity* and the *difference* with $R = RE \cap coRE$.

Quantifier Characterization of Complexity Classes

Definition

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

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

- **P** = (\forall/\forall)
- **NP** = (\exists/\forall)
- **coNP** = (\forall/\exists)

Savitch's Theorem

- REACHABILITY \in NL.

See Ex.2.10 (p.48) in [1]

Theorem (Savitch's Theorem)

REACHABILITY \in DSPACE[$\log^2 n$]

Proof:

See Th.7.4 (p.149) in [1]

$PATH(x, y, i)$: "There is a path from x to y , of length $\leq 2^i$ ".

- We can solve REACHABILITY if we can compute $PATH(x, y, \lceil \log n \rceil)$, for any nodes $x, y \in V$, since any path in G can be at most $n \log n$ long.
- If $i = 0$, we can check whether $PATH(x, y, i)$.
- If $i \geq 1$:

forall nodes z test whether $PATH(x, z, i - 1)$ and $PATH(z, y, i - 1)$

Savitch's Theorem

Proof (*cont'd*):

- We generate all nodes z one after the other, *reusing* space.
- Once a z is generated, we add $(x, z, i - 1)$ to the tape, and start working on this recursively.
- If a negative answer is obtained to $PATH(x, z, i - 1)$, we erase this triple and move to the next z .
- If a positive answer is obtained to $PATH(x, z, i - 1)$, we erase the triple and move to $PATH(z, y, i - 1)$.
- If this is negative, we erase it and move to the next z .
- If it is positive, we compare it to (x, y, i) to check that this is the second recursive call, and then return a positive answer to $PATH(x, y, i)$.
- The work tape contains at any moment at most $\lceil \log n \rceil$, each of length at most $3 \log n$. \square

Savitch's Theorem

Corollary

NSPACE $[S(n)] \subseteq$ **DSPACE** $[S^2(n)]$, for any space-constructible function $S(n) \geq \log n$.

Proof:

- Let M be the nondeterministic TM to be simulated.
- We run the algorithm of Savitch's Theorem proof on the configuration graph of M on input x .
- Since the configuration graph has $c^{S(n)}$ nodes, $\mathcal{O}(S^2(n))$ space suffices. \square

Corollary

PSPACE = NSPACE

NL-Completeness

- In Complexity Theory, we “connect” problems in a complexity class with partial ordering relations, called **reductions**, which formalize the notion of “*a problem that is at least as hard as another*”.
- A reduction must be computationally weaker than the class in which we use it.

Definition

A language L_1 is **logspace reducible** to a language L_2 , denoted $L_1 \leq_l L_2$, if there is a function $f : \Sigma^* \rightarrow \Sigma^*$, computable by a DTM in $\mathcal{O}(\log n)$ space, such that for all $x \in \Sigma^*$:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

We say that a language L is **NL-complete** if it is in **NL** and for every $A \in \mathbf{NL}$, $A \leq_l L$.

NL-Completeness

Theorem

REACHABILITY is **NL**-complete.

Proof:

See Th.4.18 (p.89) in [2]

- We've argued why $REACHABILITY \in \mathbf{NL}$.
- Let $L \in \mathbf{NL}$, that is, it is decided by a $\mathcal{O}(\log n)$ NTM N .
- Given input x , we can construct the *configuration graph* of $N(x)$.
- We can assume that this graph has a *single* accepting node.
- We can construct this in logspace: Given configurations C, C' we can in space $\mathcal{O}(|C| + |C'|) = \mathcal{O}(\log |x|)$ check the graph's adjacency matrix if they are connected by an edge.
- It is clear that $x \in L$ if and only if the produced instance of $REACHABILITY$ has a "yes" answer. \square

Certificate Definition of NL

- We want to give a characterization of **NL**, similar to the one we gave for **NP**.
- A certificate may be polynomially long, so a logspace machine may not have the space to store it.
- So, we will assume that the certificate is provided to the machine on a separate tape that is **read once**.

Certificate Definition of NL

Definition

A language L is in **NL** if there exists a deterministic TM M with an additional special read-once input tape, such that for every $x \in \Sigma^*$:

$$x \in L \Leftrightarrow \exists y, |y| \in \text{poly}(|x|), M(x, y) = 1$$

where by $M(x, y)$ we denote the output of M where x is placed on its input tape, and y is placed on its special read-once tape, and M uses at most $\mathcal{O}(\log |x|)$ space on its read-write tapes for every input x .

- What if remove the read-once restriction and allow the TM's head to move back and forth on the certificate, and read each bit multiple times?

Immerman-Szelepcsényi

Theorem (The Immerman-Szelepcsényi Theorem)

REACHABILITY \in **NL**

Proof:

See Th.4.20 (p.91) in [2]

- It suffices to show a $\mathcal{O}(\log n)$ verification algorithm A such that: $\forall (G, s, t), \exists$ a polynomial certificate u such that: $A((G, s, t), u) = \text{"yes"}$ iff t is not reachable from s .
- A has read-once access to u .
- G 's vertices are identified by numbers in $\{1, \dots, n\} = [n]$
- C_i : "*The set of vertices reachable from s in $\leq i$ steps.*"
- Membership in C_i is easily certified:
- $\forall i \in [n]: v_0, \dots, v_k$ along the path from s to v , $k \leq i$.
- The certificate is at most polynomial in n .

The Immerman-Szelepcényi Theorem

Proof (*cont'd*):

- We can check the certificate using read-once access:
 - ① $v_0 = s$
 - ② for $j > 0$, $(v_{j-1}, v_j) \in E(G)$
 - ③ $v_k = v$
 - ④ Path ends within at most i steps
- We now construct two types of certificates:
 - ① A certificate that a vertex $v \notin C_i$, given $|C_i|$.
 - ② A certificate that $|C_i| = c$, for some c , given $|C_{i-1}|$.
- Since $C_0 = \{s\}$, we can provide the 2nd certificate to convince the verifier for the sizes of C_1, \dots, C_n
- C_n is the set of vertices *reachable* from s .

The Immerman-Szelepcényi Theorem

Proof (*cont'd*):

- Since the verifier has been convinced of $|C_n|$, we can use the 1st type of certificate to convince the verifier that $t \notin C_n$.

- **Certifying that $v \notin C_i$, given $|C_i|$**

The certificate is the list of certificates that $u \in C_i$, for every $u \in C_i$.

The verifier will check:

- ① Each certificate is valid
- ② Vertex u , given a certificate for u , is larger than the previous.
- ③ No certificate is provided for v .
- ④ The total number of certificates is exactly $|C_i|$.

The Immerman-Szelepcsényi Theorem

Proof (*cont'd*):

Certifying that $v \notin C_i$, given $|C_{i-1}|$

The certificate is the list of certificates that $u \in C_{i-1}$, for every $u \in C_{i-1}$

The verifier will check:

- ① Each certificate is valid
- ② Vertex u , given a certificate for u , is larger than the previous.
- ③ No certificate is provided for v or for a neighbour of v .
- ④ The total number of certificates is exactly $|C_{i-1}|$.

Certifying that $|C_i| = c$, given $|C_{i-1}|$

The certificate will consist of n certificates for each vertex.

The verifier will check all certificates, and count the vertices that have been certified to be in C_i . If $|C_i| = c$, it accepts. \square

The Immerman-Szelepcsényi Theorem

Corollary

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

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

Proof:

- Let $L \in \mathbf{NSPACE}[S(n)]$. We will show that $\exists S(n)$ space-bounded NTM \overline{M} deciding \overline{L} :
- \overline{M} on input x uses the above certification procedure on the *configuration graph* of M . \square

Corollary

$$\mathbf{NL} = \mathbf{coNL}$$

Contents

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

Proof (*cont'd*):

- We simulate $M_i^B(1^i)$ for $i^{\log i}$ steps.
- How do we answer the oracle questions “Is $x \in B$ ”?
- **If** $|x| < i$, we look for x in B_{i-1} .
- \rightarrow **If** $x \in B_{i-1}$, M_i^B goes to q_{YES}
 \rightarrow **Else** M_i^B goes to q_{NO}
- **If** $|x| \geq i$, M_i^B goes to q_{NO} , and $x \rightarrow X$.
- Suppose that after at most $i^{\log i}$ steps the machine *rejects*.
 - Then we define $B_i = B_{i-1} \cup \{x \in \{0, 1\}^* : |x| = i, x \notin X\}$
 so $1^i \in L$, and $L(M_i^B) \neq L$.
 - Why $\{x \in \{0, 1\}^* : |x| = i, x \notin X\} \neq \emptyset$? ?
- If the machine *accepts*, we define $B_i = B_{i-1}$, so that $1^i \notin L$.
- If the machine fails to halt in the allotted time, we set
 $B_i = B_{i-1}$, but we know that the same machine will appear in
 the enumeration with an index sufficiently large. □

The Limits of Diagonalization

- As we saw, an oracle can transfer us to an alternative computational “*universe*”.
(We saw a universe where $\mathbf{P} = \mathbf{NP}$, and another where $\mathbf{P} \neq \mathbf{NP}$)
- Diagonalization is a technique that relies in the facts that:
 - **TMs are (effectively) represented by strings.**
 - **A TM can simulate another without much overhead in time/space.**
- So, diagonalization or any other proof technique relies only on these two facts, holds also for *every* oracle.
- Such results are called **relativizing results**.
E.g., $\mathbf{P}^A \subseteq \mathbf{NP}^A$, for every $A \in \{0, 1\}^*$.
- The above two theorems indicate that \mathbf{P} vs. \mathbf{NP} is a **nonrelativizing** result, so diagonalization and any other relativizing method doesn't suffice to prove it.

The Classes $\mathbf{P}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}}$

- $\mathbf{P}^{\mathbf{SAT}}$ is the class of languages decided in pol time with a SAT oracle.
 - Polynomial number of queries
 - Queries computed adaptively
- SAT is **NP**-complete $\Rightarrow \mathbf{P}^{\mathbf{SAT}} = \mathbf{P}^{\mathbf{NP}}$
- $\mathbf{FP}^{\mathbf{NP}}$ is the class of **functions** that can be computed by a pol-time TM with a SAT oracle.
- We will try to determine the complexity of the Traveling Salesman Problem (TSP):
- Goal: $\text{MAX OUTPUT} \leq_m^P \text{MAX-WEIGHT SAT} \leq_m^P \text{TSP}$

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 **FP^{NP}**-complete.

Proof:

- MAX OUTPUT \in **FP^{NP}**.
- Let $F : \Sigma^* \rightarrow \Sigma^* \in$ **FP^{NP}** $\Rightarrow \exists$ poly-time TM $M^?$, s.t.
 $M^{\text{SAT}}(x) = F(x)$
- We'll show: $F \leq_m^p$ MAX OUTPUT:

FP^{NP}-complete Problems

Proof (cont'd):

- Reductions R and S (*log space computable*) s.t.:
 - $\forall x, R(x)$ is a instance of MAX OUTPUT
 - $S(\text{max output of } R(x)) \rightarrow F(x)$

NTM N :

Let $n = p^2(|x|)$, $p(\cdot)$, is the poly bound of SAT.

$N(1^n)$ generates x on a string.

M^{SAT} query state (ϕ_1):

- If $z_1 = 0$ (ϕ_1 unsat'd), then continue from q_{NO} .
- If $z_1 = 1$ (ϕ_1 sat'd), 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= $\underbrace{z_1 z_2 \dots 00}_n$

(*Successful computation*)

FP^{NP}-complete Problems

Proof (cont'd):

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.

Theorem

MAX-WEIGHT SAT is **FP^{NP}**-complete.

Proof:

MAX-WEIGHT SAT is in **FP^{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_m^P MAX-WEIGHT SAT:

FP^{NP}-complete Problems

Proof (cont.):

- $NTMN(1^n) \rightarrow \phi(N, m)$:
Any satisfying truth assignment of $\phi(N, m) \rightarrow$ 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 FP^{NP} -complete.

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

Definition

$\mathbf{P}^{\mathbf{NP}}[\log n]$ 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.

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

CLIQUE SIZE Definition

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

Theorem

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

Conclusion

- ① $\text{TSP}_{(D)}$ is **NP**-complete.
- ② TSP is **FP^{NP}**-complete.

And now,

- **P^{NP}** → **NP^{NP}** ?
- Oracles for **NP^{NP}** ?

The Polynomial Hierarchy

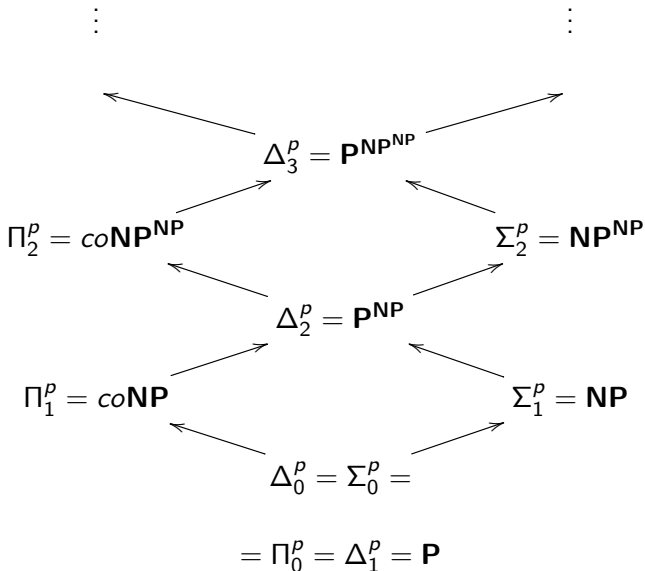
Polynomial Hierarchy Definition

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathbf{P}$
- $\Delta_{i+1}^P = \mathbf{P}^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = \mathbf{NP}^{\Sigma_i^P}$
- $\Pi_{i+1}^P = \mathbf{coNP}^{\Sigma_i^P}$
-

$$\mathbf{PH} \equiv \bigcup_{i \geq 0} \Sigma_i^P$$

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

The Polynomial Hierarchy



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 $L = \{x | \exists y : (x, y) \in R\} \in \mathbf{NP} \checkmark$
- 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 of 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

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

Corollary

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

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

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 $\mathbf{NP}=\mathbf{coNP}$, the Polynomial Hierarchy collapses to the first level.

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:

$$\exists X_1 \forall X_2 \exists X_3 \dots Q X_i \phi$$

, 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 $\Sigma_i^P = \Sigma_{i+1}^P$.

Theorem

PH \subseteq **PSPACE**

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

Relativized Results

Let's see how the inclusion of the Polynomial Hierarchy to Polynomial Space, and the inclusions of each level of **PH** to the next relativizes:

- $\mathbf{PH}^A \neq \mathbf{PSPACE}^A$ relative to *some* oracle $A \subseteq \Sigma^*$.
(Yao 1985, Håstad 1986)
- $\Pr_A[\mathbf{PH}^A \neq \mathbf{PSPACE}^A] = 1$
(Cai 1986, Babai 1987)
- $(\forall i \in \mathbb{N}) \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}$ relative to *some* oracle $A \subseteq \Sigma^*$.
(Yao 1985, Håstad 1986)
- $\Pr_A[(\forall i \in \mathbb{N}) \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}] = 1$
(Rossman-Servedio-Tan, 2015)

Contents

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

RP Class

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

Definition

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

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

We define

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

- Similarly we define the class **coRP**.

Quantifier Characterizations

- **RP** \subseteq **NP**, since every accepting “branch” is a certificate!
- **RP** \subseteq **BPP**, **coRP** \subseteq **BPP**
- **RP** = $(\exists^+/\forall) \subseteq (\exists/\forall) =$ **NP**
- **coRP** = $(\forall/\exists^+) \subseteq (\forall/\exists) =$ **coNP**

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

- i $\forall y \exists^+ z R(x, y, z) \Rightarrow \exists^+ C \forall y \bigvee_{z \in C} R(x, y, z)$
- ii $\forall z \exists^+ y R(x, y, z) \Rightarrow \forall C \exists^+ y \bigwedge_{z \in C} R(x, y, z)$

- 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, $L \in \mathbf{BPP}$. \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 $\text{coBPP} = \mathbf{BPP}$, we prove only $\mathbf{BPP} \subseteq \Sigma_2^P$.

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: $t_1, t_2, \dots, t_{p(n)} \in U$.

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

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

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

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

BPP and PH

Proof (cont.)

So, $T = (t_1, \dots, t_{p(n)})$ 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 .

$(x \in L) \Leftrightarrow (\exists T \text{ that cover all } U)$

So,

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

which is precisely the form of languages in $\Sigma_2\mathbf{P}$.

The last existential quantifier $(\exists(j \leq p(n))\dots)$ affects only polynomially many possibilities, so it doesn't "count" (can be tested in polynomial time by trying all t_j 's).



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 **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:

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

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

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

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

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

Proof: The machine M' does the following:

- Run $M(x)$ for every input x for $k = 8|x|^{2c+d}$ times, and obtain outputs $y_1, y_2, \dots, y_k \in \{0, 1\}$.
- 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, \dots, X_k are independent Boolean r.v.'s, with:

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

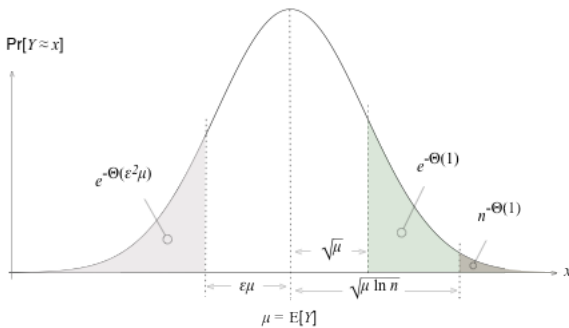
Applying a Chernoff Bound we obtain:

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



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 $\mathcal{O}(\rho/\varepsilon^2)$, where $\rho = \log(1/\delta)$.
- The probability that k is $\rho\sqrt{n}$ far from μn decays **exponentially** with ρ .



Intermission: Chernoff Bounds

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

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

Other useful form is:

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

- 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 $\Pr[M(x) = L(x)] \geq 1/2 + |x|^{-c}$ for every $x \in \{0, 1\}^*$. Then:

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

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

Semantic vs. Syntactic Classes

- Every NPTM defines some language in **NP**:
 $x \in L \Leftrightarrow \# \text{accepting paths} \neq 0$
- We can get an effective enumeration of all NPTMs, each deciding an **NP** language.
- But not every NPTM decides a language in **RP**:
 e.g., the NPTM that has *exactly one* accepting path.
- In this case, there is no way to tell whether the machine will always halt with the certified output. We call these classes **semantic**.
- So we have:
 - **Syntactic Classes** (like **P**, **NP**)
 - **Semantic Classes** (like **RP**, **BPP**, **NP** \cap **coNP**, **TFNP**)

Complete Problems for BPP?

- Any syntactic class has a “free” complete problem:

$$\{\langle M, x \rangle : M \in \mathcal{M} \ \& \ M(x) = \text{“yes”}\}$$

where \mathcal{M} is the class of TMs of the variant that defines the class

- In semantic classes, this complete language is usually *undecidable* (Rice’s Theorem).
- The defining property of **BPTIME** machines is **semantic!**
- If finally **P = 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 an NPTM M , such that for every $x \in \{0, 1\}^*$: $x \in L$ if and only if *more than half* of the computations of M on input x accept.

- Or, equivalently:

Definition

A language $L \in \mathbf{PP}$ if there exists a poly-time TM M and a polynomial $p \in \text{poly}(n)$, 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

- The defining property of **PP** is **syntactic**, any NPTM can define a language in **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?

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 $T : \mathbb{N} \rightarrow \mathbb{N}$ 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 **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

P_{/poly} is the class of languages that are decidable by polynomial size circuits families. That is,

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

Theorem (Nonuniform Hierarchy Theorem)

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

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

Turing Machines that take advice

Definition

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

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

contains 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 $\mathbf{P}_{/poly}$)

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

Proof: (\subseteq) Let $L \in \mathbf{P}_{/poly}$. 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 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

Uniform Families of Circuits

- We saw that $\mathbf{P}_{/poly}$ 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

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

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

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

Then, taking the union bound:

$$\begin{aligned} \Pr[\exists x \in \{0,1\}^n : M(x, r) \neq L(x)] &= \Pr \left[\bigcup_{x \in \{0,1\}^n} M(x, r) \neq L(x) \right] \leq \\ &\leq \sum_{x \in \{0,1\}^n} \Pr [M(x, r) \neq L(x)] < 2^{-n} + \dots + 2^{-n} = 1 \end{aligned}$$

So, $\exists r_n \in \{0,1\}^{\text{poly}(n)}$, s.t. $\forall x \in \{0,1\}^n: M(x, r_n) = L(x)$.

Using $\{r_n\}_{n \in \mathbb{N}}$ as advice string, we have the non-uniform machine.



Relationship among Complexity Classes

Theorem

The following are equivalent:

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

Corollary

Every sparse set has polynomial-size circuits.

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 $1 \leq a < b$:

$$\mathbf{BPTIME}(n^a)/1 \not\subseteq \mathbf{BPTIME}(n^b)/1$$

Theorem ([FST05])

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

$$\mathbf{RTIME}(n^b)/1 \not\subseteq \mathbf{RTIME}(n^a)/\log(n)^{1/2a}$$

Circuit Lower Bounds

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

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

- 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:

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

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 $\exp(\Omega(n^{1/(d-1)}))$, 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 $f : \{0, 1\}^n \rightarrow \{0, 1\}$ 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 $CLIQUE_{k,n} : \{0, 1\}^{\binom{n}{2}} \rightarrow \{0, 1\}$ the function that on input an adjacency matrix of an n -vertex graph G outputs 1 iff G contains a 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:*

Contents

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

Properties of Arthur-Merlin Games

Theorem

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

Proof:

Obvious from (2): $(\exists^+EA/AE) \subseteq (A^+E/EA)$. \square

Theorem

i $\mathbf{AM} \subseteq \Pi_2^p$

ii $\mathbf{MA} \subseteq \Sigma_2^p \cup \Pi_2^p$

Proof:

i) $\mathbf{AM} = (\forall E/AE) \subseteq (A^+E/EA) \subseteq (AE/EA) = \Pi_2^p$

ii) $\mathbf{MA} = (\exists^+EA/AE) \subseteq (EA/AE) = \Sigma_2^p$, and

$\mathbf{MA} \subseteq \mathbf{AM} \Rightarrow \mathbf{MA} \subseteq \Pi_2^p$. So, $\mathbf{MA} \subseteq \Sigma_2^p \cup \Pi_2^p$. \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

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

Example

Properties of Arthur-Merlin Games

Proof:

- The general case is implied by the generalization of BPP-Theorem **(1)** & **(2)**:
- $(Q_1 \exists^+ Q_2 / Q_3 \exists^+ Q_4) = (Q_1 \exists^+ \forall Q_2 / Q_3 \forall \exists^+ Q_4) = (Q_1 \forall \exists^+ Q_2 / Q_3 \exists^+ \forall Q_4)$ **(1')**
- $(Q_1 \exists \forall Q_2 / Q_3 \forall \exists^+ Q_4) \subseteq (Q_1 \forall \exists \forall Q_2 / Q_3 \exists^+ \forall Q_4)$ **(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 $\text{coNP} \subseteq \text{AM}$ (that is, if GI is NP -complete), then the Polynomial Hierarchy collapses at the second level, and $\text{PH} = \Sigma_2^P = \text{AM}$.

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

Then:

$$\Sigma_2^P = (\exists\forall/\forall\exists) \stackrel{\text{Hyp.}}{\subseteq} (\exists\forall/\exists\forall) \stackrel{(2)}{\subseteq} (\forall\exists/\forall\exists) = (\forall\exists/\exists\forall) = (\forall\exists/\forall\exists)$$

$$\text{AM} \subseteq (\forall\exists/\forall\exists) = \Pi_2^P. \quad \square \quad \Sigma_2^P = (\exists\forall/\forall\exists) \stackrel{\text{Hyp.}}{\subseteq} (\forall\exists/\exists\forall) \stackrel{(2)}{\subseteq} (\forall\exists/\forall\exists) = (\forall\exists/\exists\forall) = \text{AM} \subseteq (\forall\exists/\forall\exists) = \Pi_2^P. \quad \square$$

$$\Sigma_2^P = (\exists\forall/\forall\exists) \stackrel{\text{Hyp.}}{\subseteq} (\forall\exists/\exists\forall) \stackrel{(2)}{\subseteq} (\forall\exists/\forall\exists) = (\forall\exists/\exists\forall) = (\forall\exists/\forall\exists)$$

$$\text{AM} \subseteq (\forall\exists/\forall\exists) = \Pi_2^P. \quad \square \quad \Sigma_2^P = (\exists\forall/\forall\exists) \stackrel{\text{Hyp.}}{\subseteq} (\forall\exists/\exists\forall) \stackrel{(2)}{\subseteq} (\forall\exists/\forall\exists) = (\forall\exists/\exists\forall) = \text{AM} \subseteq (\forall\exists/\forall\exists) = \Pi_2^P. \quad \square$$

$$\stackrel{\text{Hyp.}}{\subseteq} (\forall\exists/\exists\forall) \stackrel{(2)}{\subseteq} (\forall\exists/\forall\exists)$$

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

$$\text{almost } \mathcal{C} = \left\{ L \mid \Pr_{A \in \{0,1\}^*} [L \in \mathcal{C}^A] = 1 \right\}$$

Theorem

- i $\text{almost } \mathbf{P} = \mathbf{BPP}$ [BG81]
- ii $\text{almost } \mathbf{NP} = \mathbf{AM}$ [NW94]
- iii $\text{almost } \mathbf{PH} = \mathbf{PH}$

Measure One Results

Theorem (Kurtz)

For almost every pair of oracles B, C :

- i $\mathbf{BPP} = \mathbf{P}^B \cap \mathbf{P}^C$
- ii $\mathbf{almostNP} = \mathbf{NP}^B \cap \mathbf{NP}^C$

Indicative Open Questions

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

The power of Interactive Proofs

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

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

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)

$$\mathbf{IP = PSPACE}$$

The power of Interactive Proofs

Lemma 1

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

Proof:

- If the Prover is an **NP**, or even a **PSPACE** machine, the lemma holds.
- But what if we have an omnipotent prover?
- On any input, the Prover chooses its messages in order to *maximize the probability of V's acceptance!*
- We consider prover as an **oracle**, by assuming wlog that his responses are one bit at a time.
- Th protocol has polynomially many rounds (say $N=n^c$), which bounds the messages and the random bits used.
- So, the protocol is described by a computation tree T :

The power of Interactive Proofs

Proof(cont'd):

- Vertices of T are V 's configurations.
- **Random Branches** (queries to the random tape)
- **Oracle Branches** (queries to the prover)
- For each fixed P , the tree T_P can be pruned to obtain only random branches.
- Let $\Pr_{opt}[E \mid F]$ the conditional probability given that the prover *always behaves optimally*.
- The acceptance condition is $m_N = 1$.
- For $y_i \in \{0, 1\}^N$ and $z_i \in \{0, 1\}$ let:

$$R_i = \bigwedge_{j=1}^i m_j = y_j$$

$$S_i = \bigwedge_{j=1}^i l_j = z_j$$

The power of Interactive Proofs

Proof(cont'd):



$$\Pr_{\text{opt}}[m_N = 1 \mid R_{i-1} \wedge S_{i-1}] =$$

$$\sum_{y_i} \max_{z_i} \Pr_{\text{opt}}[m_N = 1 \mid R_i \wedge S_i] \cdot \Pr_{\text{opt}}[R_i \mid R_{i-1} \wedge S_{i-1}]$$

- $\Pr_{\text{opt}}[R_i \mid R_{i-1} \wedge S_{i-1}]$ is **PSPACE**-computable, by simulating V .
- $\Pr_{\text{opt}}[m_N = 1 \mid R_i \wedge S_i]$ can be calculated by DFS on T .
- The probability of acceptance is

$$\Pr_{\text{opt}}[m_N = 1] = \Pr_{\text{opt}}[m_N = 1 \mid R_0 \wedge S_0]$$
- The prover can calculate its optimal move at any point in the protocol in **PSPACE** by calculating $\Pr_{\text{opt}}[m_N = 1 \mid R_i \wedge S_i]$ for $z_i \in \{0, 1\}$ and choosing its answer to be the value that gives the maximum. □

Warmup: Interactive Proof for UNSAT

Lemma 2

$$\text{PSPACE} \subseteq \text{IP}$$

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

Theorem

$$\text{coNP} \subseteq \text{IP}$$

- 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 ($\text{mod}N$) (where $N > 2^n \cdot 3^m$), in the following way:

Arithmetization

- Arithmetic generalization of a CNF Boolean Formula.

T	\longrightarrow	1
F	\longrightarrow	0
$\neg x$	\longrightarrow	$1 - x$
\wedge	\longrightarrow	\times
\vee	\longrightarrow	$+$

Example

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

- 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

Sends primality proof for N

$$q_1(x) = \sum p(x, x_2, \dots, x_n)$$

$$q_2(x) = \sum p(r_1, x, x_3, \dots, x_n)$$

$$q_n(x) = p(r_1, \dots, r_{n-1}, x)$$

Verifier

checks proof

checks if $q_1(0) + q_1(1) = 0$

← sends $r_1 \in \{0, \dots, N-1\}$

checks if $q_2(0) + q_2(1) = q_1(r_1)$

← sends $r_2 \in \{0, \dots, N-1\}$

⋮

checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$

picks $r_n \in \{0, \dots, N-1\}$

checks if $q_n(r_n) = p(r_1, \dots, r_n)$

Warmup: Interactive Proof for UNSAT

- If ϕ is **unsatisfiable**, then

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \dots, x_n) \equiv 0 \pmod{N}$$

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 $\{0, \dots, N-1\}$ 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

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \dots, x_n) \neq 0 \pmod{N}$$

- So, $p_1(0) + p_1(1) \neq 0$, 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, $\Pr[p_1(r_1) \neq q_1(r_1)] \geq 1 - \frac{m}{N}$.
- 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, $\Pr[p_n \neq q_n] \geq 1 - (n - 1)\frac{m}{N}$.
- If so, with probability at least $1 - \frac{m}{N}$ the verifier will reject since, $q_n(x)$ and $p(r_1, \dots, r_{n-1}, x)$ differ on at least that fraction of points.
- **The total probability that the verifier will accept is at most $\frac{nm}{N}$.**

Arithmetization of QBF

$$\begin{array}{l} \exists \longrightarrow \Sigma \\ \forall \longrightarrow \Pi \end{array}$$

Example

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

↓

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

Theorem

A closed QBF is true if and only if the 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 $\mathcal{O}(2^{2^n})$.

- 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

$$A' \neq 0 \pmod{p} \Leftrightarrow A \text{ is true.}$$

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 $\{0, 1\}^k \rightarrow \{0, 1\}$.
- Now, consider the language of all true quantified boolean formulas:

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

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

