

# Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ)

## Υπολογιστική Πολυπλοκότητα

Εργαστήριο Λογικής και Επιστήμης Υπολογισμών  
Εθνικό Μετσόβιο Πολυτεχνείο

2018-2019




1

# Computational Complexity

Graduate Course

Antonis Antonopoulos

Computation and Reasoning Laboratory  
National Technical University of Athens

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

[e148abab0c9923b3cc835507f68f4caf404710e9](https://creativecommons.org/licenses/by-nc-nd/4.0/)

3

# Πληροφορίες Μαθήματος

## Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ)

### Υπολογιστική Πολυπλοκότητα (ΑΛΜΑ)

- Διδάσκοντες: Σ. Ζάχος, Ά. Παγουρτζής
- Βοηθοί Διδασκαλίας: Α. Αντωνόπουλος, Α. Χαλκή
- Επιμέλεια Διαφανειών: Α. Αντωνόπουλος
- Δευτέρα: 17:00 - 20:00 (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 13:00-14:00
- Σελίδα: <http://courses.corelab.ntua.gr/complexity>
- Βαθμολόγηση:
  - Διαγώνισμα: 6 μονάδες
  - Ασκήσεις: 2 μονάδες
  - Ομιλία: 2 μονάδες
  - Quiz: 1 μονάδα

2

## 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 J. Katz, **Notes on Complexity Theory**, 2011, University of Maryland

4

# Contents

- **Introduction**
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

- **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]

5

## 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  $\Sigma$  ( $|\Sigma| \geq 2$ ).
- 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  $\Sigma$ , we associate the following language with the decision problem  $\Pi$ :

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

## 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  $G$ ?

7

6

## Search Problems

- Have answers of the form of an **object**.
- **Relation**  $R(x, y)$  connecting instances  $x$  with answers (objects)  $y$  we wish to find for  $x$ .
- Given instance  $x$ , find a  $y$  such that  $(x, y) \in R$ .

## Example

**FACTORING:** Given integer  $N$ , find its prime decomposition

$$N = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}.$$

8

### 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  $\pi$  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**

### What have we learned?

- Computational Complexity classifies problems into classes, and studies the relations and the structure of these classes.
- We have decision problems with boolean answer, or function/optimization problems which output an object as an answer.
- Given some nice properties of polynomials, we identify polynomial-time algorithms as efficient algorithms.

## Contents

- Introduction
- **Turing Machines**
- Undecidability
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

### Definition

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

- $Q = \{q_0, q_1, q_2, q_3, \dots, q_n, q_{yes}, q_{no}\}$  is a finite set of states.
- $\Sigma$  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  $\delta$  is the **program** of the machine.

13

### 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.

15

## 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**.

14

## 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  $M = (Q, \Sigma, \delta, q_0, F)$ :

- $Q = \{q_0, q_1, q_2, q_3, \dots, q_n, q_{halt}, q_{yes}, q_{no}\}$  is a finite set of states.
- $\Sigma$  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.

16

## 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  $M$ .

17

## 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  $S'(n) = \varepsilon S(n) + 2$ .

19

## 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.*

18

## 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.
- $\Sigma$  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**.

20

# 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**.

21

# Contents

- Introduction
- Turing Machines
- **Undecidability**
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

23

## What have we learned?

- A recursive language is decided by a TM.
- A recursive enumerable language is accepted by a TM that halts only if  $x \in L$ .
- Multiple tape TMs can be simulated by a one-tape TM with quadratic overhead.
- Linear speedup justifies the  $\mathcal{O}(\cdot)$  notation.
- Nondeterministic TMs move in “parallel universes”, making different choices simultaneously.
- A Deterministic TM computation is a *path*.
- A Nondeterministic TM computation is a *tree*, i.e. exponentially many paths ran simultaneously.

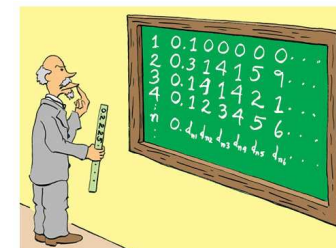
22

# Diagonalization



*Suppose there is a town with just one barber, who is male. In this town, the barber shaves all those, and only those, men in town who do not shave themselves. Who shaves the barber?*

Diagonalization is a technique that was used in many different cases:



George showed it wouldn't fit in.

24

## Diagonalization

### Theorem

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

**Proof:** Let, for the sake of contradiction that are countable:  $\phi_1, \phi_2, \dots$ . 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  $\phi_y(y) = \phi_y(y) + 1$ .  $\square$

- Using the same argument:

### Theorem

*The functions from  $\{0, 1\}^*$  to  $\{0, 1\}$  are uncountable.*

25

## 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  $U$  such that for every  $x, w \in \Sigma^*$ ,  $U(x, w) = M_w(x)$ . Also, if  $M_w$  halts within  $T$  steps on input  $x$ , then  $U(x, w)$  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].

27

## 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  $\perp M \perp$  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 TM.**
- **Every TM is represented by infinitely many strings.**

- There exists (at least) an uncomputable function from  $\{0, 1\}^*$  to  $\{0, 1\}$ , since the set of all TMs is countable.

26

## 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:  $H = \{\perp M \perp; x \mid M(x) \downarrow\}$ , 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).

28

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

29

# 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  $\{M; x; y \mid M(x) = y\}$

31

- 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

- Let  $E(M) = \{x \mid (q_0, \triangleright, \varepsilon) \xrightarrow{M^*} (q, y \sqcup x \sqcup, \varepsilon)\}$
- $E(M)$  is the language *enumerated* by  $M$ .

## Theorem

$L$  is recursively enumerable iff there is a TM  $M$  such that  $L = E(M)$ .

30

# 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  $L(M) \in \mathcal{C}$ ?

32























## Karp reductions vs logspace reductions

### Theorem

A logspace reduction is a polynomial-time reduction.

### Proof:

See Th.8.1 (p.160) in [1]

- Let  $M$  the logspace reduction TM.
- $M$  has  $2^{\mathcal{O}(\log n)}$  possible configurations.
- The machine is deterministic, so *no configuration can be repeated* in the computation.
- So, the computation takes  $\mathcal{O}(n^k)$  time, for some  $k$ .

□

## Circuits and CVP

### Definition (CVP)

Circuit Value Problem (CVP): Given a circuit  $C$  and an assignment  $x$  to its variables, determine whether  $C(x) = 1$ .

- $\text{CVP} \in \mathbf{P}$ .

### Example

REACHABILITY  $\leq_m^{\ell}$  CVP: Graph  $G \rightarrow$  circuit  $R(G)$ :

- The gates are of the form:
  - $g_{i,j,k}$ ,  $1 \leq i, j \leq n$ ,  $0 \leq k \leq n$ .
  - $h_{i,j,k}$ ,  $1 \leq i, j, k \leq n$
- $g_{i,j,k}$  is **true** iff there is a path from  $i$  to  $j$  without intermediate nodes bigger than  $k$ .
- $h_{i,j,k}$  is **true** iff there is a path from  $i$  to  $j$  without intermediate nodes bigger than  $k$ , and  $k$  is used.

## Circuits and CVP

### 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 of incoming edges) 2.
- The vertices labeled with  $\neg$  have *fan-in* 1.
- 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.

## Circuits and CVP

### Example

- Input gates:  $g_{i,j,0}$  is **true** iff  $(i = j \text{ or } (i, j) \in E(G))$ .
- For  $k = 1, \dots, n$ :  $h_{i,j,k} = (g_{i,k,k-1} \wedge g_{k,j,k-1})$
- For  $k = 1, \dots, n$ :  $g_{i,j,k} = (g_{i,j,k-1} \vee h_{i,j,k})$
- The output gate  $g_{1,n,n}$  is **true** iff there is a path from 1 to  $n$  using no intermediate paths above  $n$  (sic).
- We also can compute the reduction in logspace: go over all possible  $i, j, k$ 's and output the appropriate edges and sorts for the variables  $(1, \dots, 2n^3 + n^2)$ .























