E0 224: Computational Complexity Theory

Naman Mishra

August 2025

Contents

Lectures

The course

Instructor: Chandan Saha

Lecture 1.
Wednesday
August 6

Resources:

(1) Computational Complexity: A Modern Approach by Sanjeev Arora and Boaz Barak
(We'll closely follow this book)

(2) Computational Complexity Theory by Steven Rudich and Avi Wigderson (Editors)

(3) Mathematics and Computation by Avi Wigderson

(4) Boolean Function Complexity by Stasys Jukna

(5) Gems of Theoretical Computer Science by Schoening and Pruim

(6) The Nature of Computation by Moore and Mertens

(7) The Complexity Theory Companion by Hemaspaandra and Ogihara

(8) Online lecture notes… (take a look at this webpage)

Evaluation:

(45%) Three assignments, one posted at the end of each month, with two weeks for completion. The submission will be via email, as a LaTeX-generated PDF file. You may freely use all resources and tools, and confer with each other, so long as you list these out.

(25%) Midterm exam

(30%) Final exam

Classify computational problems based on the amount of resources required by algorithms to solve them.

Problems

Problems come in various flavors.

Decision problem

Search problem

- Search for a prime between n and $2n$.

Counting problem

- Count the number of

Optimization problem

- Find a minimum size *vertex cover* in a graph.
- Optimize a linear function subject to *linear inequality constraints*. (Linear programming)

Algorithms

Algorithms are methods for solving problems, studied via formal *models of computation*, such as Turing machines.

Resources

- **Time:** Number of bit operations
- **Space:** Number of memory cells required
- **Randomness:** Number of random bits used
- **Communication:** Number of bits sent over a network

Roadmap

Structural complexity The classes P, NP, coNP, NP-completeness, et cetera. The computation must be space bounded. There is an entire polynomial hierarchy. huh?

- How hard is it to check if the largest independent set in G has size k?

- How hard is it to check if there is a circuit of size k that computes the same Boolean function as a given Boolean circuit?

Circuit complexity The internal workings of an algorithm can be viewed as a *Boolean circuit*, yet another nice combinatorial model of computation closely related to Turing machines. The size, depth and width of a circuit correspond to the sequential, parallel and space somplexity, respectively, of the algorithm it represents.

Proving $\mathsf{P} \neq \mathsf{NP}$ also reduces to showing circuit lower bounds, that is, showing the existence of Boolean functions that are hard to compute by small circuits.

Randomness We get probabilistic complexity classes such as BPP, RP, coRP, et cetera.

Access to random bits can help improve computational complexity, but to what extent? Quicksort has expected running time $\Theta(n \log n)$, but worst-case time $\Theta(n^2)$.

Counting complexity The class $\#\mathsf{P}$.

- How hard is it to count the number of perfect matchings in a graph?
- How hard is it to count the number of cycles in a graph?

Approximation A hardness of approximation result looks like the following.

> **Theorem .1** (Hastad, 1997)**.** *If there exists, for some $\varepsilon > 0$, a polynomial-time algorithm to compute an assignment that satisfies at least $7/8 + \varepsilon$ fraction of the clauses of an input* $3\mathsf{SAT}$*, then* $\mathsf{P} = \mathsf{NP}$*.*

In contrast, there is a polynomial-time algorithm to compute an assignment that satisfies at least $7/8$ fraction of the clauses.

Another example is that of probabilistically checkable proofs (PCPs).

Chapter I

Turing machines

Turing called them a-machines (automatic machines). Church, his doctoral advisor, named them after him.

A turing machine consists of memory tape(s) and a finite set of rules.

Definition I.1. A k-tape Turing machine M is described by a tuple (Γ, Q, δ) such that

(1) M has k one-sided memory tapes (input/work/output) with *heads*;

(2) Γ is a finite alphabet, including a special *blank* symbol $\flat$. Each memory cell contains an element of Γ.

(3) Q is a finite set of *states* with two special states: q_0 and q_∞.

(4) δ is a function from $Q \times \Gamma^k \times \{-1, 0, 1\}^k$.

The starting configuration is assumed to contain the inut string on the input tape at its beginning, following by trailing $\flat$ symbols. All other tapes contain only $\flat$s. The heads are all positioned at the beginning of each tape.

A step of computation is performed by applying δ. Once the machine enters the state q_∞, it halts computation.

Let $f\colon \{0,1\}^* \to \{0,1\}^*$, $T\colon \mathbb{N} \to \mathbb{N}$ and M a Turing machine on the alphabet $\{0,1\}$.

Definition I.2 (Computation and running time)**.** M computes f if for every $x \in \{0,1\}^*$, M halts with $f(x)$ on its output tape once began with x on its input tape.

This computation is in T time if for every $x \in \{0,1\}^*$, M halts within $T(|x|)$ steps.

In this course, we will almost always deal with Turing machines that halt on every input, and computational problems that can be solved by a Turing machine.

Definition I.3 (Time constructible function)**.** A function $T\colon \mathbb{N} \to \mathbb{N}$ is *time constructible* if $T(n) \geq n$ and there is a Turing machine that computes the function that maps x (expressed in binary) to $T(|x|)$ (expressed in binary) in $O(T(|x|))$ time.

For example, $(\cdot)^2$, $2^{\cdot}$ and $(\cdot)\log(\cdot)$ are all time constructible.

Theorem I.4 (Binary alphabets suffice)**.** *Let $f\colon 2^* \to 2^*$ and $T\colon \mathbb{N} \to \mathbb{N}$ be a time constructible function.*

If a Turing machine M over an alphabet Γ, then there exists another Turing machine M' that computes f in time $4\log_2|\Gamma|T(n)$ using $\{0, 1, \flat\}$ as the alphabet.

Theorem I.5 (One tape suffices)**.** *Let $f\colon 2^* \to 2^*$ and $T\colon \mathbb{N} \to \mathbb{N}$ be a time constructible function.*

If a Turing machine M with k tapes computes f in $T(n)$, then there exists a Turing machine M' with 1 tape that computes f in $kT(n)^2$ time.

I.1 Universal Turing machine

Every Turing machine can be represented by a finite string over $\{0, 1\}$. Conversely, every string over $\{0, 1\}$ represents some Turing machine, by mapping initially invalid representations to the trivial Turing machine. Finally, if we allow padding with zeroes, each Turing machine has infinitely many representations. For a binary string α, we will let M_α denote the Turing machine encoded by it.

Theorem I.6 (Universal Turing machine)**.** *There exists a Turing machine U that computes $M_\alpha(x)$ for every input $\alpha|x$.*

Modern day electronic computers are physical realizations of universal Turing machines.