Part III
Introduction to Computational Complexity

Lectured by Timothy Gowers

Artur Avameri

Contents

0 Introduction

A good book for the course is the first few chapters of *Computational Complexity: A modern approach* by Arora and Barack.

0.1 Computational problems

A problem in complexity theory is somewhat different from a mathematical problem – it is more like a class of problems.

Example 0.1. Given: A graph G with n vertices and $x, y \in V(G)$. Problem: Is there a path from x to y? This is a decision problem (yes/no answer).

A problem has a variable input and an output. If the output belongs to $\{0,1\}$, i.e. {no, yes}, then the problem is called a **decision problem**. Write $\{0,1\}^*$ for the set of $\bigcup_{n=1}^{\infty}\{0,1\}^n$. Then a decision problem can be encoded as a **Boolean function**, that is, a function $f : \{0,1\}^* \to \{0,1\}$.

The set $\{x \in \{0,1\}^* \mid f(x) = 1\}$ is called the **language** defined by f.

0.2 Turing machines

A **Turing machine** formalizes the notion of an algorithm. There are many ways to formalize it – we (handwavily) describe a few.

A k–**tape Turing machine** consists of several ingredients. The first is a collection of k **tapes**, where a tape is an infinite sequence of **cells**. There is also a finite set A called the **alphabet**, and each cell contains an element of A. There is also a **head** which is in a **state** (an element of a finite set S of states) and in a **position** in each state. S contains two special states, S_{init} and S_{halt}.

A state is a function that takes as input an element of $A^k \times S$ and outputs an element of $A^k \times S \times \{L, N, R\}^k$. If S is this "transition function", then the machine rewrites $(a_1, \ldots, a_k)$ according to the A^k component of the image, changes the state of the head according to the S component, and shifts each tape according to the $\{L, N, R\}^k$ component.

One tape is designated as the input tape and is never changed, another is the output tape. All tapes other than the input tape start full of zeroes. If the machine reaches the state S_{halt}, it stops. If the input is x and the output is y, we say that the machine computed y given input x.

Variants assume that $A = \{0,1\}$; that $k = 1$ (with a different convention about input–output tapes); that tapes are two–sided, etc.

1 Some complexity classes

Definition 1.1. The complexity class P consists of all Boolean functions (i.e. problems) $f : \{0,1\}^* \to \{0,1\}$ such that there exists a Turing machine T and a polynomial p such that for every $x \in \{0,1\}^*$, T computes $f(x)$ in at most $p(|x|)$ steps, where $|x| = m$ if $x \in \{0,1\}^m$.

Example 1.1. The problem st-CONN (input: directed graph G and two vertices s, t; output: 1 if there is a directed path from s to t) belongs to P.

Example 1.2. Input: positive integers m, n and output: mn. This is polynomial in the number of digits.

23 Jan 2024, Lecture 2

We now talk about NP, which stands for nondeterministic polynomial time.

Example 1.3. Input: a graph G with n vertices. Output: 1 iff G contains a Hamilton cycle (a cycle that contains every vertex).

Loosely, a nondeterministic algorithm is one that doesn't fully specify what it does. It outputs $f(x) = 1$ if there is some sequence of choices that leads to $f(x) = 1$. More formally, a **nondeterministic Turing machine** is one that has not one but two transition functions. At each step, it applies one or the other. We say that it computes f if $f(x) = 1 \iff$ there is some sequence of choices that leads to output 1 when the input is x.

Definition 1.2. NP is the class of functions computable in polynomial time by a nondeterministic Turing machine.

Definition 1.3 (Alternative definition)**.** $f \in NP$ if there is a polynomial p and a function $g \in P$ such that for every $x \in \{0,1\}^* \iff \exists y \in \{0,1\}^{p(|x|)}$ such that $g(x, y) = 1$.

To see that this is equivalent, observe first that if f satisfies the second definition, then we can write down y nondeterministically and apply g. In the other direction, y encodes choices made by the NDTM, so given y, the computation can be done deterministically.

Big open problem: does $P = NP$?

Definition 1.4 (co–NP)**.** $f \in$ co–NP $\iff 1 - f \in NP$.

Alternatively, $f \in$ co–NP $\iff$ there exists a polynomial p and $g \in P$ such that $\forall x \in \{0,1\}^*$, $f(x) = 1 \iff \forall y \in \{0,1\}^{p(|x|)}, g(x, y) = 1$.

For example, primality testing is in both NP and co–NP (for co–NP, we need to find a polynomial time algorithm that verifies a number is prime, see Ex. Sheet 1).

1.1 The polynomial hierarchy

Definition 1.5. Define Σ_0^P and Π_0^P to be P. If Σ_k^P and Π_k^P have been defined, then $f \in \Sigma_{k+1}^P \iff \exists$ a polynomial p and $g \in \Pi_k^P$ such that $f(x) = 1 \iff \exists y, g(x, y) = 1$.

Also $f \in \Pi_{k+1}^P \iff \exists$ a polynomial p and $g \in \Sigma_k^P$ such that $f(x) = 1 \iff \forall y, g(x, y) = 1$ (here $y \in \{0, 1\}^{p(x)}$).

Example 1.4. $f \in \Sigma_5^P \iff \exists h \in P$ such that

$$f(x) = 1 \iff \exists y_1 \, \forall y_2 \exists y_3 \, \forall y_4 \exists y_5, h(y_1, y_2, y_3, y_4, y_5) = 1.$$

We define PH (the polynomial hierarchy) as

$$\text{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^P \cup \Pi_k^P.$$

Proposition 1.1. If $P = NP$, then $P = \text{PH}$.

Proof. Note first that if $P = NP$, then $P =$ co–NP. If $f \in \Sigma_{k+1}^P$, then $\exists g \in \Pi_k^P$ such that $f(x) = 1 \iff \exists y \; g(x, y) = 1$. By induction, $g \in P$, so $f \in NP$, so $f \in P$. The proof for Π_{k+1}^P is similar, just look at the negation. □

Exercises on Ex. Sheet 1:

- If $NP =$ co–NP, then PH $= NP =$ co–NP.
- If $\Sigma_k^P = \Sigma_{k+1}^P$ or $\Sigma_k^P = \Pi_k^P$, then PH $= \Sigma_k^P$.

PSPACE. This consists of functions that can be computed by a Turing machine that uses only a polynomial amount of tape.

Proposition 1.2. $NP \subset$ PSPACE.

Again a reminder: all proofs involving Turing machines are really more proof sketches.

Proof. If $f(x) = 1 \iff \exists y \; g(x, y) = 1$ for $g \in P$, then compute f by a brute–force search. All we need to remember is which y we've got to (in any sensible ordering) and whether we've found a value of y that works. Note importantly that we can reuse the space needed to compute each $g(x, y)$. □

Exercise on Ex. Sheet 1: PH $\subset$ PSPACE.

There are functions in PSPACE but not in PH. Good examples are games: "there exists a move I can play such that for any move you play there exists a move that I can play...". For example, take a game of Go with board size n going to infinity to get an example.

EXPTIME. This is the class of functions that can be computed in time $\exp(O(n^k))$ for some k.