

Complexity Theory

Slides based on *S.Aurora, B.Barak. Complexity Theory: A Modern Approach.*

Ahto Buldas

e-mail: `Ahto.Buldas@ut.ee`

home: `http://home.cyber.ee/ahtbu`

phone: 51 02 160

Textbooks

Upcoming textbook:

1. Sanjeev Aurora and Boaz Barak. Complexity Theory: A Modern Approach.

<http://www.cs.princeton.edu/theory/complexity/>

2. Steven Rudich, Avi Wigderson (Editors). Computational Complexity Theory. 2004.

Some citations ...

As long as a branch of science offers an abundance of problems, so long it is alive; a lack of problems foreshadows extinction or the cessation of independent development. David Hilbert, 1900.

The subject of my talk is perhaps most directly indicated by simply asking two questions: first, is it harder to multiply than to add? and second, why?...I (would like to) show that there is no algorithm for multiplication computationally as simple as that for addition, and this proves something of a stumbling block. Alan Cobham, 1964.

Computation

... is the process of producing an output from a set of inputs in a finite number of steps.

Some computational problems:

- *Multiplication*: Given two integer numbers, compute their product.
- *Solving Linear Equations*: Find a solution to a set of n linear equations over n variables.
- *Dinner Party Problem*: Given a list of acquaintances and a list of containing all pairs of individuals who are not on speaking terms with each other, find the largest set of acquaintances you can invite to a dinner party such that you do not invite any two who are not on speaking terms.

Formal Models of Computation

In the beginning of the 20th century, computation was simply a person writing numbers on a note pad following certain rules.

In the middle of the 20th century, the notion of computation was made much more precise.

Different formal models of computation were discovered: Turing machines, lambda calculus, cellular automata, pointer machines, bouncing billiards balls, Conways Game of life, etc.

All those models are *universal*—each is capable of implementing all computations that we can conceive of on any other model.

This universality motivated the invention of the standard electronic computer, which is capable of executing all possible programs.

Computation as a Major Scientific Concept

Generalizing from models such as cellular automata, scientists have come to view many natural phenomena as computational processes.

The understanding of reproduction in living things was triggered by the discovery of self-reproduction in computational machines (von Neumann, Pauli etc.).

Intriguing fact: Pauli predicted the existence of a DNA-like substance in cells almost a decade before Watson and Crick discovered it.

Some scientists suggested that the entire universe may be viewed as a giant computer.

Such physical theories have been used in the past decade to design a model for *quantum computation*.

Computability and Complexity

From 1930s to the 1950s, researchers focused on the theory of *computability* and showed that several interesting computational tasks are inherently uncomputable: no computer can solve them without going into infinite loops (i.e., never halting) on certain inputs.

Computability will not be our focus here.

We focus on issues of *computational efficiency*— computational complexity theory is concerned with how much computational resources are required to solve a given task.

Some Key Questions in Complexity Theory

- Do some tasks inherently require exhaustive search? (P vs NP)
- Can algorithms use randomness to speed up computation?
- Can problems be solved quicker if only approximate solutions are required?
- Can we use hard problems to construct unbreakable ciphers?
- Can we use (very counterintuitive) quantum mechanical properties to solve hard problems faster?
- Can we generate mathematical proofs automatically?

At about 40 years of age, Complexity Theory still does not completely answer to any of these questions!

Efficiency of Multiplication

Consider the task of multiplying two integers a and b ... and two algorithms:

- *Repeated Addition*: add a to itself $b - 1$ times.
- *Grade-School Algorithm*:

$$\begin{array}{r}
 423 \cdot 577 \\
 \hline
 2961 \\
 25610 \\
 214200 \\
 \hline
 244071
 \end{array}$$

Size n of the input—the number of digits.

For multiplying two n -digit numbers, the repeated addition uses $n \cdot 10^{n-1}$ additions, while the grade school algorithm uses $2 \cdot n^2$ additions.

The fastest known algorithm ($n \log n$) uses the Fast Fourier Transform.

Efficiency of Solving Linear Equations

The classic Gaussian elimination algorithm (named after Gauss but already known in some form to Chinese mathematicians of the first century) uses $O(n^3)$ basic arithmetic operations to solve n equations over n variables.

In the late 1960s, Strassen found a more efficient algorithm that uses roughly $O(n^{2.81})$ operations, and ...

The best known algorithm takes $O(n^{2.376})$ operations.

Efficiency of Solving the Dinner Party Problem

Obvious algorithm: try all possible subsets from the largest to the smallest, and stop after a subset that does not include “unfriendly” pairs of guests.

Running time = the number of subsets = 2^n .

To organize a 70-person party, supercomputers would spend thousands of years ...

Surprisingly: We still do not know significantly better algorithms!

We have reasons to suspect that no efficient algorithm exists for this task, which is also called the *independent set problem*, and which, together with thousands of other important problems, is NP-complete.

The famous **P** vs **NP** question asks whether or not any of these problems has an efficient algorithm.

Proving Non-Existence of Efficient Algorithms

How to prove that the current algorithm is *the best*?

- How to show that the $O(n \log n)$ multiplication algorithm can never be improved,
- i.e. how to show that multiplication is harder than addition which takes $O(n)$ steps?

We can also try to prove that there is no algorithm for the dinner party task that takes $< 2^{n/10}$ steps.

We cannot verify all algorithms because there are infinitely many of them!

Hence, we have to mathematically *prove* that there is no better algorithm.

Impossibility results have been very fruitful (Independence of Euclid's 4th axiom, Impossibility of solving algebraic equations in radicals, etc.)

Conventions (i)

Whole numbers: A whole number is a number in the set $\mathbb{Z} = \{0, \pm 1, \pm 2, \dots\}$. A number denoted by one of the letters i, j, k, ℓ, m, n is always assumed to be whole. If $n \geq 1$, then we denote by $[n]$ the set $\{1, \dots, n\}$.

Integer Functions: For a real number x , we denote by $\lceil x \rceil$ the smallest $n \in \mathbb{Z}$ such that $n \geq x$ and by $\lfloor x \rfloor$ the largest $n \in \mathbb{Z}$ such that $n \leq x$. Whenever we use a real number in a context requiring a whole number, the operator $\lceil \rceil$ is implied.

Logarithms: We denote by $\log x$ the logarithm of x to the base 2.

Conventions (ii)

Sufficiently large n : A condition holds for sufficiently large n if it holds for every $n \geq N$ for some number N (for example, $2^n > 100n^2$ for sufficiently large n).

Sums: We use expressions such as $\sum_i f(i)$ when the range of values i takes is obvious from the context.

Strings and vectors: If u is a string or vector, then u_i denotes the value of the i -th symbol/coordinate of u .