COMPUTABILITY AND COMPLEXITY

JAKOB NORDSTRÖM
ALGORITHM & COMPLEXITY SECTION
DEPARTMENT OF COMPUTER SCIENCE (DIKU)
http://www.jakobnordstrom.se

Course webpage

www.jakobnordstrom.se/teaching/CoCo23/

Announcements, problem set submissions, and questions will be on Absalon

Textbook

Sanjeev Arora & Boaz Barak
Computational Complexity: A Modern Approach

Plus some other material towards the end of the course

Prepare for the lecture by reading the chapter(s) specified

By default, read full chapter.
Will try to make clear if parts of chapter can be skipped

Problem sets

Will appear on course webpage and Absalon. Expect 4 psets.

Collaboration in groups encouraged But solutions should be written individually and from scratch

The best way to learn this type of material is not just by reading, but by getting your hands dirty working on problems

Problem sets graded pass/fail

You have to pass all problem sets to take the exam

If you fail, there will be one resubmission per problem set

Computational complexity theory:
What is efficiently computable in practice
(given the fact that resources are limited)

Computation

- Informal understanding since ancient times
  "write down symbols following certain rules"
- 1st half of 20th century: precise, mathematical definition
- Invention of (electronic) computer
- Now computers omnipresent
- But goes beyond computers – computation happens in many other ways
  - biology (DNA etc)
  - neuroscience
  - physics, et cetera
  - economics – markets compute equilibrium price

All of this seems to be captured by one computational model (spoiler alert: Turing machine)

Interesting question: What is computable in this model? Answer: not everything
will see example.

Even more interesting question: What is <u>efficiently</u> computable?

Many fundamental open problems

L 1
III

(1) Is solving a problem harder than checking a solution?
(Avoiding exhaustive search)

(2) Can randomness speed up computation?
(If computer can flip fair coins)

(3) Can any efficient algorithm be converted to one that uses tiny amount of memory?

(4) Can every sequential algorithm be efficiently parallelized?

(5) Can hard problems become significantly easier if it's OK to give not optimal but only approximate solutions?

(6) Can quantum mechanics be used to build faster computers?

(7) Can computationally hard problems be useful to solve computational problems efficiently?

(8) Can proofs be verified by quick, random sampling of just a few bits

(9) Is it possible to give proofs that reveal absolutely nothing other than the truth of the statement?

(10) Is it possible to compute solutions to problems that are so large we ~~cannot even~~ don't have time to read all the input?
Or can't even store it?

L1 IV

1. Probably yes — big(gest) open problem in TCS (and all of math)
   Assume "yes" as axiom? (cf. gravity)

2. Probably no — don't know for sure, but quite strange things would happen otherwise

3. Probably no, but this is also big open problem
   (Can prove no in bounded models)

4. — " —

5. Sometimes yes, a lot. Sometimes no, not at all. Lots of research in TCS group at KTH

6. Theoretical model says yes.
   Not clear if physically realizable [NOT COVERED IN COURSE]

7. Definitely yes! Almost all of modern crypto builds on this. (Also connections to 2)

8. Amazingly yes! Very connected to 5

9. Amazingly yes! Connections to crypto

10. Yes, sometimes
    - sublinear-time algorithms
    - streaming algorithm

On many of these questions there is consensus...

But for most we don't know how to prove what we believe

... And we could be wrong (has happened before)

Fascinating and exciting questions with implications far outside of computer science

Two approaches

(A) Concrete, unconditional lower bounds
"Low-level" computational complexity
Consider bounded models of computation

(B) Connections between computational problems and notions
E.g. assume answer "yes" to (1), i.e., $P \neq NP$, and study what follows
"High-level" computational complexity

But in order to study these questions need sure, formal footing.
(Should be mostly review of things you know/have known)

Will try to follow notation in Arora-Barak (unless stated otherwise), in particular in Ch 0

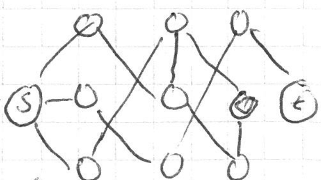
Will be slightly more relaxed regarding matters of representation (but important to know this can be formalized)

On a related note: Will sometimes sketch proofs or focus on getting main idea across. This is <u>not</u> a proof. Important to fill in missing details (when reading and when solving psets)

Represent objects (numbers, graphs, formulas) as strings $\{0,1\}^*$ (or in Σ^* for other alphabet Σ)

Computational problem [Arora-Barak uses Γ]

1. Given graph G, vertices s, t, find path in G from s to t



2. Given integer n, find prime factors
   25 957
3. Given Boolean formula, find satisfying truth value assignment.

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$$

Function problem

$$f : \Sigma^* \to \Sigma^*$$

Given x, compute $f(x)$

We will focus on simplified version

Decision problem

$$f : \Sigma^* \to \{yes, no\} \qquad (\text{or} \to \{1, 0\})$$

(1') Is there a path from s to t in G

(2') Is there a prime factor of n less than k

(3') Is the Boolean formula satisfiable

Much cleaner to work with
Often doesn't matter – efficient solution to decision problem yields solution to function problem

Historical terminology

$$\boxed{\begin{array}{c}\text{Decision problem} \\ f : \Sigma^* \to \{yes, no\}\end{array}} \iff \boxed{\begin{array}{c}\text{Language} \\ L \subseteq \Sigma^* \\ L = \{x \in \Sigma^* \mid f(x) = yes\}\end{array}}$$

We say that an algorithm that computes f DECIDES L

Aside: encoding issues

1) Implicitly assume we've agreed on encoding of inputs and outputs
   - can be important in practice
   - Usually not in this course
   - Avoid silly encodings, e.g. unary

2) Some strings are not valid encodings ("syntax errors") – treat as "no" instances

---

Measure efficiency as # basic operations as function of input length
- ignore constants depending on low-level details
- look at asymptotic behaviour as input size grows

$f(n) = O(g(n))$ if exists positive constants c, N s.t. for $n \geq N$ it holds that $f(n) \leq c \cdot g(n)$

$f(n) = \Omega(g(n))$ ... $f(n) \geq c \cdot g(n)$

$f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

$f(n) = o(g(n))$ if $\forall \varepsilon > 0$ $\exists$ $\cancel{c}, N$ s.t. $n \geq N$ $f(n) \leq \varepsilon \cdot g(n)$

$f(n) = \omega(g(n))$ if $\forall K > 0\ \exists N$ s.t. $n \geq N \Rightarrow f(n) \geq K \cdot g(n)$.

Efficiency in <u>what model</u>? Turing machine. Seems to be able to simulate all physically realizable computational methods with little overhead.

Important model. Important to understand. But a nuisance to program TMs...
So we will just give brief overview — read details in Ch 1

Informally

- Q $\boxed{\text{program of TM}}$ (counter) or set of states of TM
- Γ alphabet (symbols) finite size
- Tapes
  - input tape — read-only, contains input
  - work tapes
  - output tape — write-only

Read/write heads on tapes

At each step

- read symbols on tapes
- write symbols to all (non-input) tapes and move heads
- go to new state

Running time = # steps

Compute a function

write value on output tape, then move to halting state $q_{halt} \in Q$.

FACTS

(1) Model very robust to tweaks
- change of alphabet
- # tapes (from just 1 and up)

(2) Description of TM can be written as string and given as input to other TMs

(3) Hence, there is a UNIVERSAL TURING MACHINE that can simulate any other TM given its string representation
If original TM runs in time T, then simulation runs in time $O(T \log T)$ — very efficient

---

From this follows that there are uncomputable/undecidable problems

$HALT = \{ (M, x) \mid TM\ M \text{ halts on input } x \}$

THM The language HALT is not decidable / computable by any TM

Proof By contradiction.

Suppose that H is a TM that decides HALT

We can construct another TM H' that simulates H as a subroutine

Then we can feed H' to H with a suitable input.

All legitimate, so if reach contradiction, then H can't exist.

TM H' with input M

```
if H(M,M) = "yes" then
    while true
    endwhile
else      // H(M,M) = "no"
    halt
```

What does H' do when given input M = H'?

a) H' halts on H' ⇒
   H(H', H') = "yes" ⇒ H' gets stuck in while loop

b) H' does not halt on H' ⇒
   H(H', H') = "no" ⇒ H' halts

Contradiction. Hence H doesn't exist $\square$