Computability and Complexity (CoCo) 2024

Lecturers

Jakob Nordström (main instructor)
Srikanth Srinivasan
Amir Yehudayoff

Algorithms & Complexity Section
Department of Computer Science (DIKU)

Teaching assistant

Shuo Pang

Course webpage

https://jakobnordstrom.se/teaching/CoCo24/
The course webpage contains all practical information about the course.
Announcements, problem sets submissions, questions, and discussions will be dealt with on Absalon.

Course material

Computational Complexity: A Modern Approach by Sanjeev Arora & Boaz Barak
Plus probably some other material towards the end of the course.
Advice: Prepare for lectures by skimming the chapter(s) specified in advance.

Problem sets

4 problem sets will be posted on the course webpage and on Absalon.
See course webpage for (tentative) deadlines and problem set rules.
Collaborations in groups of 2-3 encouraged.
But all solutions should be written **individually** and **from scratch** without sharing any material (or generating it from the Internet).
The best way to learn this type of material is not just by reading, but by "getting your hands dirty" working on problems.
Expect to have to think for quite some time about some of the problems.

COMPUTATIONAL COMPLEXITY THEORY
What is efficiently computable in practice
(given 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 (laptops, cellphones, Internet)
- But computation is about much more than computers — happens in many other ways
  - biology (e.g., DNA)
  - chemistry (e.g., chemical reactions)
  - neuroscience
  - physics
  - economics (e.g., markets computing equilibrium prices)
  - social networks

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.
Exist perfectly well-defined mathematical functions that cannot be computed

Even more interesting question: What is EFFICIENTLY computable?

What this course is about
As we will see, many fascinating open problems (and some answers)

SOME QUESTIONS

1. Is solving a problem harder than checking a solution (avoiding exhaustive search)
2. Can randomness help solve more problems faster? (If computers can flip fair coins)
3. Can any time-efficient algorithm be optimized to use tiny amounts of memory?
4. Can every sequential algorithm be efficiently parallelized? (Time T, P processors $\Rightarrow$ speed-up T/P?)
5. Can hard problems become significantly easier if we are willing to accept suboptimal, 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 complicated mathematical proofs be verified by quick, random sampling of just a few bits?
9. Is it possible to provide proofs that reveal absolutely nothing other than the truth of the statement proven?
10. Is it possible to compute solutions to problems so large that we don't have time to read them or space to store them?

SOME ATTEMPTS AT ANSWERS

1. Probably yes, but big(gest) open problem in TCS (and all of math) P vs NP
   Assume "yes" as axiom? (c.f. gravity)

2. Non-interactively, probably no, — don't know for sure, but strange things would happen otherwise P vs BPP
   Interactive computation: Randomness very helpful, as we will see

3. Probably no, but big open problem
   P v.s. L

4. ~~Again~~, probably no, but big open problem
   P vs NC

5. Assuming $P \neq NP$:
   - Sometimes a lot easier
   - Sometimes not at all

6. Theoretical models seem to predict yes
   Not clear if physically realizable
   (And recent "quantum supremacy" experiments debated)

7. Definitely YES! Almost all of modern cryptography builds on this
   (and also has connections to 2)

(8) Amazingly, YES! Very connected to (5)

(9) Again amazingly, YES! Connections to modern cryptography

(10) Yes, sometimes. Buzzwords:
- sublinear-time algorithms
- streaming algorithms

For many of these questions, there is scientific consensus...

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

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

Fascinating and exciting questions with implications far outside of computer science

Two approaches:

(A) Concrete, unconditional results for bounded computational models
"Low-level" computational complexity

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

In order to do rigorous study of computation, need formal mathematical setting defining
- what is a computer?
- what does it mean to solve a problem efficiently?
- what is a computational problem?

$\boxed{\text{COMPUTATIONAL PROBLEM}}$

Represent objects as strings in some alphabet $\boxed{\Sigma}$

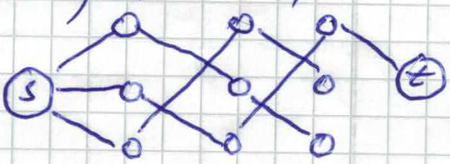
Think of Σ as
- 0 or 1
- a-zA-Z0-9_-()[]<>
- ASCII or UTF-8

Choose Σ as convenient. Does not matter as long as size finite $|\Sigma| < \infty$

STRING $s \in \Sigma^*$ sequence of 0 or more characters from Σ

Some example problems:

(1) Given graph G, vertices s, t, find path in G from s to t (or report none exist)



(2) Given integer N, find prime factors $N = 25957$

(3) Given propositional logic formula, find satisfying assignment (or report none exist)

$$(x_1 \vee x_2 \vee x_3) \vee (\neg x_1 \vee \neg x_2) \vee (\neg x_1 \vee \neg x_3) \vee (\neg x_2 \vee \neg x_3)$$

(4) Given integers $A_1, \ldots, A_n$ and target T, find subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} A_i = T$ (or report none exist) $\{2, 3, 5, 7\}$, target 11

Encoding issues

1) We can clearly encode all these problems as strings in some way
2) From now on, assume we've agreed on some reasonable encoding
   - Details of course matter in practice
   - Not really important for our discussion
   - Avoid silly encodings, e.g., unary (5 encoded as "11111")

$\boxed{\text{SEARCH PROBLEM}}$

Given $x \in \Sigma^*$ encoding problem instance, compute solution $y \in \Sigma^*$

Ex some path; some satisfying assignment

$\boxed{\text{FUNCTION PROBLEM}}$

If every problem has a unique answer, this defines function $f : \Sigma^* \to \Sigma^*$

Ex prime factors sorted in increasing order; lexicographically smallest path from s to t viewed as string in Σ^*

We will simplify even more

$\boxed{\text{DECISION PROBLEM}}$

Consider functions $\boxed{f : \Sigma^* \to \{0, 1\}}$

Think of $\boxed{\begin{array}{l} 0 = \text{"no"} \\ 1 = \text{"yes"} \end{array}}$

Examples

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

(2') Is there a prime factor of N of size at most U?

(3') Is the given formula satisfiable?

(4') Is there a subset of $\{A_1, \ldots, A_n\}$ summing to the target T?

Much cleaner to work with mathematically

Often does not matter – efficient algorithm for decision problem will give efficient algorithm for search problem

(Can you work this out for the problems above?)

$\boxed{\text{DECISION PROBLEMS AND LANGUAGES}}$

Decision problem		Language $\mathcal{L} \subseteq \Sigma^*$
$f : \Sigma^* \to \{yes, no\}$	$\Leftrightarrow$	$\mathcal{L} = \{x \in \Sigma^* \mid f(x) = yes\}$

$f(x) = 1 = yes$ x is "yes instance"
$f(x) = 0 = no$ x is "no instance"

Historical terminology
Rich literature
No way of avoiding it...

We say that an algorithm that computes f DECIDES $\mathcal{L}$

What do we do with strings that are not valid encodings?

Ex If a graph is a list of the edges, is there a path from s to t in

$(s,a), (a,b),)c,b,a(, (b,t))$?

"Syntax error" – Define $f(s) = no$ for strings s that are not valid encodings

$\boxed{\text{EFFICIENT COMPUTATION}}$

What can be computed within reasonable limits on resources such as
- computation time
- computation memory
- computation energy
- network communication

Most important measure for us: $\boxed{\text{TIME}}$

Measure how the number of operations needed scale with the size of the input
- ignore constants depending on low-level details
- look at asymptotic behaviour as input size grows

$\boxed{\text{ASYMPTOTIC NOTATION}}$

$T(n)$ is $O(g(n))$ if exist positive constants, c, N s.t. for $n \geq N$ it holds that $T(n) \leq c \cdot g(n)$

$T(n)$ is $\Omega(g(n))$ if exist positive constants c, N s.t. for $n \geq N$ it holds that $T(n) \geq c \cdot g(n)$

$T(n)$ is $\Theta(g(n))$ if $T(n)$ is $O(g(n))$ and $\Omega(g(n))$

$T(n)$ is $o(g(n))$ if for all $\varepsilon > 0$ exists N s.t. for $n \geq N$ $\quad T(n) \leq \varepsilon \cdot g(n)$

$T(n)$ is $\omega(g(n))$ if for all $K > 0$ exists N s.t. for $n \geq N$ $\quad T(n) \geq K \cdot g(n)$

But what is our computational model?

The **TURING MACHINE**

- Seems to be able to simulate all physically realizable computational methods with little overhead
- But very simple, so mathematically nice to work with
- But so simple and stupid that they are very annoying to deal with
  - CLRS completely skips details (for those of you who took AADS)
  - We will follow Chapter 1 in Arora-Barak, but will be brief and informal

TURING MACHINE (TM)

- Fixed alphabet Σ (of finite size)
- Program Q (or "~~set~~ of states" of TM)
- Tapes
  - input tape, contains input, read-only (after input, special EOF/blank symbol)
  - work/output tape (initialized to EOF/blank symbols)

  Tapes have a starting position but no end
- Read-write heads positioned on tapes (start in starting position)

(Can have more tapes if we want
Does not really matter — see Arora-Barak)

At each time step the TM:

(a) reads symbols at current position on all tapes
(b) writes symbol to work tapes
(c) move tape head left or right one step (or stand still)
(d) jump to new state $q' \in Q$

(b)–(d) depend on
- current state q
- ~~positions~~ symbols read on tapes in (a)

Special state $\boxed{q_{halt}}$ — TM stops

Running time # steps before reaching q_{halt}

To compute a function:
- write value on output tape
- then move to q_{halt}

Ex TM that decides whether # 1s in binary string odd

ALWAYS ~~q_{start}~~: Read symbol s on input tape, move input head right

q_{start}: If $s = 0$ go to q_{start}
If $s = 1$ go to q_{odd}
If $s = EOF$ write 0 on output and go to q_{halt}

q_{odd}: If $s = 0$ go to q_{odd}
If $s = 1$ go to q_{start}
If $s = EOF$ write 1 on output and go to q_{halt}

FACTS ABOUT TURING MACHINES

1. Expressive: Can do "normal things" efficiently, so we can allow ourselves to write pseudocode (including subroutine calls)

2. Robust to tweaks
   - change of alphabet
   - adding more work tapes

3. Description of TM can be written as string and given as input to other TM

4. There is a $\boxed{\text{UNIVERSAL TURING MACHINE}}$ that can simulate any other Turing machine M given its string representation.

   This is EFFICIENT — if original TM M runs in time T, then simulation runs in time $O(T \log T)$

All of this needs ~~proving~~, of course, but we do not have time or patience for this now... So take it on faith.

From this it follows that there are UNCOMPUTABLE / UNDECIDABLE PROBLEMS!

Perfectly well-defined mathematical functions that cannot be correctly computed by any algorithm

HALTING PROBLEM

Fix alphabet Σ

Fix some way of encoding Turing machines

Consider the language

$$\boxed{HALT = \{ \langle M, x \rangle \mid \text{Turing machine } M \text{ halts on input } x \}}$$

(As before, if M or x is not valid, then $\langle M, x \rangle$ is a no instance)

THEOREM The language HALT is not decidable by any Turing machine

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

This is all legitimate, so if we reach a contradiction, then H cannot exist

TM H' with input M

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

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

a) H' halts on H' $\Longrightarrow$
   $H(H', H') = $ yes $\Longrightarrow$
   H' gets stuck in infinite loop

b) H' does not halt on H' $\Longrightarrow$
   $H(H', H') = $ no $\Longrightarrow$
   H' halts

Contradiction! Hence H does not exist ☒

Another example:
Given set of polynomials with integer coefficients, do these equations have a common integral solution?

Undecidable!

Does not mean that no instance of these problems can ever be solved. But no algorithm can:
- always terminate
- always give correct answer