DD 2445 - Complexity Theory

Jakob Nordström
Theory Group
School of Computer Science and Communication
www.csc.kth.se/~jakobn
jakobn@kth.se

But don't send e-mail! Sign up at Pizza!
piazza.com/kth.se/fall2017/dd2445

Course webpage
www.csc.kth.se/dd2445/complexity

Textbook
Sanjeev Arora & Boaz Barak
Computational Complexity: A Modern Approach

Will sometimes follow textbook (mostly 1st half)
Sometimes deviate (mostly 2nd half of course)

Default: Supposed to read chapter(s) referenced for lecture

Will this be on the exam? Yes, if it helps you solve the problem sets :-)

\[\text{L1}\]
Examination

Problem sets: determines grade
Hand in by the deadlines

Peer evaluation: pass/fail

Top grade A: paper presentation
Also PhD level version - talk to me if you have questions

Goal: get you to work in depth with course material
Cannot learn by just reading - need to get your hands dirty
Computational complexity theory:
What is efficiently computable in practice (given the fact that resources are limited)

Computational

- 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, etc.
  - 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 efficiently computable?

Many fundamental open problems
1. Is solving a problem harder than checking a solution? (Avoiding exhaustive search)

2. Can randomness speed up computation? (If computers 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 quickly 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 read all the input, let alone store it?
1. Probably yes - big (yet) 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 computed models)
4. Sometimes yes, a lot. Sometimes no, not at all. Lots of research in TCS group at KTH.
5. Theoretical model says yes. Not clear if physically realizable.
6. Definitely yes. Almost all of modern crypto builds on this. (Also connections to 2)
7. Amazingly yes! Very connected to 5.
8. Amazingly yes! Connections to crypto.
9. Yes, sometimes:   - sublinear-time algorithms
                     - streaming algorithms
10. 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 (A), i.e., $P \neq \text{NP}$, and study what follows in "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 Aho-Barak (unless stated otherwise), in particular in Ch 0

Will be slightly more relaxed regarding matters of presentation (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 not a proof. Important to fill in missing details (when reading and when solving pieces)

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

Computational problem

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.
   \((x_1 \lor x_2 \lor x_3) \land \overline{(x_1 \lor x_2)} \land (\overline{x_1} \lor x_2) \land (\overline{x_2} \lor \overline{x_3})\)
Function problem

\[ f : \Sigma^* \rightarrow \Sigma^* \]

Given \( x \), compute \( f(x) \)

We will focus on simplified version

Decision problem

\[ f : \Sigma^* \rightarrow \{ \text{yes}, \text{no} \} \quad (\text{or} \rightarrow \{0, 1\}) \]

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

\[ f : \Sigma^* \rightarrow \{ \text{yes}, \text{no} \} \leftrightarrow \text{Language} \]

\[ L = \{ x \in \Sigma^* | f(x) = \text{yes} \} \]

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

\[
f(n) = \mathcal{O}(g(n)) \quad \text{if exists/constants } c, N
\]
\[
s.t. \text{ for } n \geq N \text{ it holds that } f(n) \leq c \cdot g(n)
\]

\[
f(n) = \Omega(g(n)) \quad \text{... } f(n) \geq c \cdot g(n)
\]

\[
f(n) = \Theta(g(n)) \quad \text{if } f(n) = \mathcal{O}(g(n)) \text{ and } f(n) = \Omega(g(n))
\]

\[
f(n) = \omega(g(n)) \quad \text{if } \forall c > 0 \quad \exists N \quad \text{ s.t. } n \geq N \implies f(n) \geq c \cdot g(n)
\]

\[
f(n) = \alpha(g(n)) \quad \text{if } \forall K > 0 \exists N \quad \text{ s.t. } n \geq N \implies f(n) \leq K \cdot g(n)
\]
Efficiency in what model? 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 2.

Informally:
- A program of TM or set of states of TM
- 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 \( q_h \in Q. \)
**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**

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

The language $${	ext {HALT}}$$ is not decidable/computable by any TM
Proof. By contradiction.

Suppose that \( H \) is a TM that decides \( \text{HALT} \).

We can construct another TM 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 \( H(M,M) = \text{"yes"} \) then
  - while true
    - end while
- else // \( H(M,M) = \text{"no"} \)
  - halt

What does \( H' \) do when given input \( M = H' \)?

a) \( H' \) halts on \( H' \) \( \Rightarrow \)
   \( H(H',H') = \text{"yes"} \) \( \Rightarrow \) \( H' \) gets stuck in while loop

b) \( H' \) does not halt on \( H' \) \( \Rightarrow \)
   \( H(H',H') = \text{"no"} \) \( \Rightarrow \) \( H' \) halts

Contradiction. Hence \( H \) doesn't exist. \( \square \)