L1 I

JAKOB NORDSTRÖM
THEORY GROUP, KTH CSC
www.csc.kth.se/~jakobn
jakobn@kth.se

But... Don't send e-mail
Sign up at Piazza
piazza.com/kth.se/fall2013/dd2446

Course webpage
www.csc.kth.se/utbildning/kth/kurser/DD2446/kpl13

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

We will sometimes follow textbook, sometimes deviate
Default: Students supposed to read chapter(s) referenced for lecture

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

Examination
Problem sets: (Hand in by the deadlines) determines grade A-E
Peer evaluation: pass/fail
Goal: get you to work in depth with material
Cannot just learn by reading — need to get hands dirty

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

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 finding a solution as easy as recognizing one? (avoiding exhaustive search)

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

(3) Can every efficient algorithm be converted into one that uses a tiny amount of memory?

(4) Can every sequential algorithm be efficiently parallelized?

(5) Can hard problems become significantly easier if algorithms are allowed to make mistakes on a small number of (unusual) inputs?

(6) Can hard problems become significantly easier if algorithms don't need to give optimal but only approximate solutions?

(7) Can we use quantum mechanics to build faster computers?

(8) Can computationally hard problems actually be useful for computation?

(9) Can proofs be verified by quick, random sampling? Is it possible to give proofs that reveal absolutely nothing other than truth of statement?

(10) Can we compute solutions to problems that are so huge we don't even have time to read all of the input? That we can't store it?

1: Probably no — don't know
2: Probably no — don't know
3: Probably no — don't know
4: Probably no — don't know
5: Yes, sometimes [NOT COVERED IN COURSE]
6: Sometimes, sometimes not
Area of expertise of Theory Group
7: Theoretical models say yes
Unclear whether realizable [NOT COVERED IN COURSE]
8: Definitely yes! All our modern crypto builds on this
9: Yes and yes! Connected to crypto
10: Yes, sometimes
- sublinear-time algorithms
- streaming algorithms

On ~~many~~ of these questions there is consensus...

But for many (most?) we don't know how to prove what we believe

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

Fascinating and exciting questions
Broad implications far outside of computer science.

But in order to study these questions need some 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 not 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

(3) Given Boolean formula, find satisfying truth value assignment.

Function problem

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

Given x, compute $f(x)$

We will focus on simplified version

Decision problem

$f : \Sigma^* \to \{yes, no\}$ (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

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

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

$f(n) = O(g(n))$ 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 $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$

$f(n) = o(g(n))$ if $\forall \varepsilon > 0$ $\exists N$ 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 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

- Q [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

  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) Descriptions of TMs can be represented as strings, and given as inputs to other TMs

(3) There is a universal Turing machine that can simulate any other TM given its string representation. Simulation runs in time $O(T) \log T$ - very efficient.

This gives us that there are natural undecidable problems

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

Thm HALT is not computable by any TM.

Proof Let H be a TM that decides HALT.

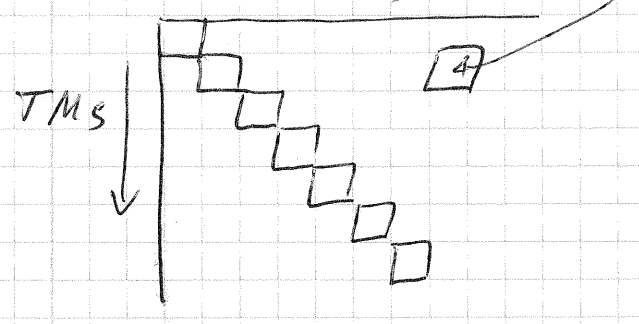
Define new TM H' that does the following on input M
- if H accepts (M, M) then H' gets stuck in a while-loop forever
- if H rejects (M, M), then halt

What does H' do when given itself as input?

a) H' halts on $H' \Rightarrow H$ rejects $(H', H') \Rightarrow H'$ didn't halt on H'

b) H' loops $\Rightarrow H$ accepts $(H', H') \Rightarrow H'$ does halt. ∎

Diagonalization



Build a TM that disagrees with table in diagonal
⇒ cannot be one of TMs in the table

Another example: Given set of polynomial equations with integer coefficients, do the equations have an integer solution?

Even if computable, might be infeasible in practice. Classify how hard various functions are

COMPLEXITY CLASS set of functions that can be computed within some given resource bounds

Def $T: \mathbb{N} \to \mathbb{N}$ function

A language L is in $DTIME(T(n))$ if there is a TM that runs in time $c \cdot T(n)$ for some constant c and decides L

Def $P = \bigcup_{k=1}^{\infty} DTIME(n^k)$

Note (a) P defined for decision problems

(b) Running time measured in # bits in input

Church-Turing Thesis

> Every physically realizable computation device can be simulated by a Turing Machine

Not a ~~theorem~~ mathematical theorem — it couldn't be — but consistent with what we currently know about nature

Strong/Extended Church-Turing Thesis

> Anything efficiently computable on any device is efficiently computable by a TM (i.e., with polynomial overhead).

(Might not be true if quantum computers can be built)

So we think of P as capturing what is efficiently computable

Interlude

Given language / problem L,
would like to

a) give algorithm A deciding L
b) prove that no algorithm can do better than A

Many successes for (a).
Task (b) seems almost completely beyond reach! (Because it's hard — how can you *prove* that some totally weird algorithm out there can't do better.)

What to do?

(1) Restrict model, e.g. focusing on what "non-weird" algorithms do and prove that no such algorithm can do better

(2) At least relate how hard different problems are compared to each other via reductions

- shows connections
- helps us to expand our intuition about what to expect

REDUCTIONS

L_1 reduces to L_2, $L_1 \leq L_2$, if
$\exists$ (efficient) algorithm that computes
same function g s.t.

$$x \in L_1 \implies g(x) \in L_2$$
$$x \notin L_1 \implies g(x) \notin L_2$$

Positive use

Have Efficient algorithm for L_2
Given new problem L_1,
Reduce L_1 to L_2 and solve it
Ex bipartite matching $\leq$ max flow

Negative use

Believe that L_1 is hard
Given new problem L_2
Reduce L_1 to L_2
Shows that L_2 must be as hard as L_1

IS P A REASONABLE MODEL OF EFFICIENTLY SOLVABLE PROBLEMS?

L1 XIV

Pros

- Composes well – efficient programs can call efficient subroutines and stay efficient
- Exponents of the polynomials in running times are often small
- Reasonable agreement with practice

Cons

- Worst-case scenario too strict – what if difficulty depends on some pathological instance never seen in practice?
  - Not clear what this means
  - Attempts at average-case complexity
- Polynomial time too slow – the small exponents we observe is because that's the kind of algorithms we can understand and discover

  Also, huge data sets can make even quadratic or linear time infeasible
  - There is research into this
  - But P still relevant class

- What about other physical models that might
  (a) be continuous, not discrete
  - still need to measure, and to deal with noise
  (b) use randomness (say, radioactive decay)
  - doesn't seem to make
  (c) use quantum mechanics
  - jury is out...

Cons (continued)

L1 XV

- Decision problem framework is too limited
  - Yes, sometimes. But surprisingly often not. We'll see an example next lecture.

SUMMING UP

TURING MACHINES GENERAL WAY TO MODEL COMPUTATION (although we don't want to get stuck in low-level details)

SOME NATURAL FUNCTIONS NOT COMPUTABLE AT ALL (HALTING PROBLEM)

IDENTIFY "EFFICIENTLY SOLVABLE" WITH COMPLEXITY CLASS P — ON BALANCE, SEEMS LIKE REASONABLE (AND SUCCESSFUL!) DEFINITION

NEXT TIME

NP, NP-COMPLETENESS, AND BEYOND (CHAPTER 2)