DD2445 LECTURE 2

L2 I

First lecture - recap

Practicalities

Discussion about computation and computational complexity

Basic definitions

Computational model

TURING MACHINE

Q Set of states

Σ finite-size alphabet (think $\{0,1\}$)

Tapes Read-only input tape

Write-only output tape

Read-write work tapes

Used to solve DECISION PROBLEMS

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

- Is there a path from s to t in graph G?
- Is the Boolean formula F satisfiable?

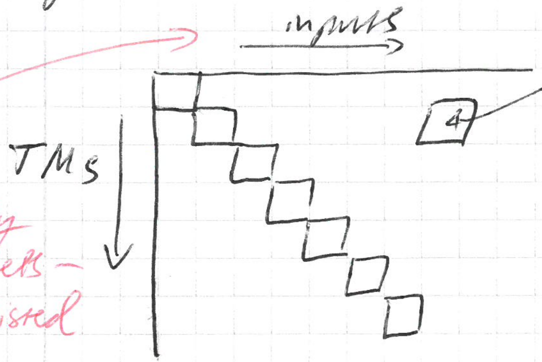
Solve decision problem $f : \Sigma^* \to \{yes, no\}$ $\iff$ DECIDE LANGUAGE $L = \{ x \in \Sigma^* \mid f(x) = yes \}$

There is a UNIVERSAL TURING MACHINE that can simulate any other TM efficiently given its description as a string.

It is undecidable whether given TM M halts on input x

What happened in the proof?

Diagonalization



box(M,x) does M halt on x?

countably infinite sets — can be listed

Using assumption that halting problem can be decided

Build a TM that disagrees with table on diagonal
$\Rightarrow$ cannot be one of TMs in the table
Hence assumption must be wrong

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

FROM NOW ON AND FOR REST OF COURSE
Focus on computable problems

COMPLEXITY CLASS set of functions that can be computed within same 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
$$\boxed{P = \bigcup_{k=1}^{\infty} DTIME(n^k)}$$

Note (a) P defined for decision problems

(b) Running time measured in # bits in input

Church-Turing Thesis

$\boxed{\text{Every physically realizable computational device can be simulated by a Turing Machine}}$

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

Strong / Extended Church-Turing Thesis

$\boxed{\text{Anything efficiently computable on any device is efficiently computable by a TM (i.e., with polynomial overhead).}}$

(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 some 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

L2 VI

IS P A REASONABLE MODEL OF EFFICIENTLY SOLVABLE PROBLEMS?

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 matter

(c) use quantum mechanics
  - jury is out...

Cons (continued)

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

SUMMING UP SO FAR

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 ~~ON THE~~ AGENDA

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

L2 VIII

Solving v.s. verifying

Doing an exam requires coming up with solutions — can be hard

Grading just involves verifying correctness — much easier (hopefully... usually...)

Complexity class P

Efficiently solvable problems (in polynomial time)

Complexity class NP

Problems for which solutions can be verified efficiently

DEF 1 Language L is in NP if $\exists$ polynomial p and poly-time TM M (verifier) s.t.

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

u is a certificate or witness for x

EXAMPLES

① TRAVELLING SALESMAN

n cities, $\binom{n}{2}$ pairwise distances d_{ij}

length constraint k

Is there a tour visiting every city exactly one and having length $\leq k$

② SUBSET SUM

n numbers $A_1, \ldots, A_n$ and number T, is there subset $S \subseteq [n]$ s.t. $\sum_{i \in S} A_i = T$

③ LINEAR PROGRAMMING

m linear inequalities $a_1 u_1 + a_2 u_2 + \ldots + a_n u_n \leq b$

$a_i, b \in \mathbb{Q}$, is there an assignment to the u_i satisfying all inequalities?

④ 0/1 INTEGER PROGRAMMING

Same as above, but assignments to u_i in $\{0,1\}$

⑤ GRAPH ISOMORPHISM

Given two graphs G_1, G_2, are they isomorphic? I.e., Is there a bijection $\pi: V(G_1) \to V(G_2)$ s.t. $(u,v) \in E(G_1)$ iff $(\pi(u), \pi(v)) \in E(G_2)$

⑥ COMPOSITE NUMBERS

Given $N \in \mathbb{N}$, decide if N is composite (i.e., not a prime)

(7) FACTORING

Given $N, L, U \in \mathbb{N}$, does N have a prime factor p with $L \leq p \leq U$

(8) CONNECTIVITY

Given graph G and vertices s, t, are s and t connected in G

(9) CNFUNSAT

Given a CNF formula $F = \bigwedge_{i=1}^{m} C_i$ for clauses C_i on the form $x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4$, does every truth value assignment fail to satisfy at least one clause?

---

(1) TSP: witness: tour

complexity: hardest problem in NP: NP-complete

(2) witness: subset

complexity: NP-complete

(3) witness: assignment

complexity: For long best-by worst-case exptime Simplex

Khachiyan '79: $\in P$

Karmarkar '84: Efficient

(4) witness: assignment ; complexity NP-complete

(5) witness: bijection cplx: not believed to be in P

— " — NP-complete

(6) witness: factor

known to be solvable with randomness [Miller '76, Rabin '80]

In P! [AKS '04]

(7) witness: prime factor

not believed to be in P (RSA breaks down)

not believed to be NP-complete

(8) witness: path   cplx in P (do BFS)

(9) witness: ?

in coNP – see later

PROPOSITION 3 $P \subseteq NP \subseteq EXP$

DEF 2 EXP (or EXPTIME) $= \bigcup_{c=1}^{\infty} DTIME(2^{n^c})$

Proof $P \subseteq NP$: pick witnesses of length 0.

$NP \subseteq EXP$: At most exp many witness candidates, try all in poly-time per candidate.

Prop 2 is state-of-the art (sadly).

One of the Millennium Problems: $P \stackrel{?}{=} NP$

Most (but not all) believe $P \neq NP$.

L2 : XII

Original definition of NP uses nondeterminism (the "N" in "NP")

Nondeterministic TM

Each line in program has two variants (TM has two transition functions)

At each step, TM arbitrarily chooses one

Think of it as flipping a "golden coin" that always comes up "the best way"

NTM / NDTM accepts x if at least one possible sequence of choices leads to accept; otherwise rejects

NDTM/NTM runs in time $T(n)$ if all possible sequences of choices terminate within time $T(n)$.

Aside: accept means either

(a) write 1 on output tape, or

(b) reach special state q_{accept}

DEF 4 L is in $NTIME(T(n))$ if $\exists\, c>0$ and $c \cdot T(n)$-time NDTM M such that for every $x \in \{0,1\}^*$ $x \in L$ iff $M(x) = 1$

LEMMA 5 $NP = \bigcup_{c \in \mathbb{N}} NTIME(n^c)$

Proof Need to prove

(1) $L \in NP \Rightarrow L \in \bigcup_{c \in \mathbb{N}} NTIME(n^c)$

(2) $L \in \bigcup_{c \in \mathbb{N}} NTIME(n^c) \Rightarrow L \in NP$

(1) There is some witness. Let NDTM write down a witness nondeterministically, then verify (by running verifier for L)

$x \in L \Rightarrow \exists$ good witness $\Rightarrow$ accept

$x \notin L \Rightarrow$ no good witness $\Rightarrow$ reject

Runs in poly-time, since verifier poly-time and witness poly-length.

(2) Let the witness be a good sequence of "golden coin tosses". Simulate NDTM with this sequence to check acceptance.

Wouldn't it be great to have an NDTM on your desk?
Not physically realizable (as far as we know)
But useful theoretical model, e.g. for exhaustive search

SUMMING UP LECTURE 2

Efficient computation

Defined as deterministic polynomial time
Can be debated
But on the whole seems like fruitful definition

Church–Turing thesis + extended versions

TM "universal model" of computation
Consistent with our knowledge so far

Reductions

Use to solve problems
– " – relate hardness of problems

NP

Problems with efficiently verifiable solutions

$P \subseteq NP \subseteq EXP$

Inclusions widely believed to be strict

$P \overset{?}{\neq} NP$ *the* open problem of complexity theory

N in NP stands for "nondeterministic computation" – computation with "golden coin flips"