First lecture - recap

Practicalities
Discussion about computation and computational complexity

Basic definitions

Computational model
TURING MACHINE
A set of states
$\Sigma$ finite-size alphabet (think \{0,1\})
Tapes read-only input tape
write-only output tape
read-write work tapes

Used to solve DECISION PROBLEMS
$f : \Sigma^* \rightarrow \{\text{yes, no}\}$

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

Solve decision problem

$\text{DECIDE LANGUAGE}$

$f : \Sigma^* \rightarrow \{\text{yes, no}\} \quad \iff \quad L = \{ x \in \Sigma^* | f(x) = \text{yes} \}$

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

Build a TM that disagrees with table in diagonal

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 share resource bounds.

**Def** \( T: \mathbb{N} \to \mathbb{N} \) function

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

**Def** \( P = \bigcup_{k=1}^{\infty} \text{DTIME} (n^k) \)
Church-Turing Thesis

Every physically realizable computational device can be simulated by a Turing machine.

Not a 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 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

- sheds connections
- helps us to expand our intuition about what to expect
REDUCTIONS

$D_1$ reduces to $D_2$, $D_1 \leq D_2$, if

\[ x \in D_1 \Rightarrow g(x) \in D_2 \]

\[ x \notin D_1 \Rightarrow g(x) \notin D_2 \]

Positive use

Have efficient algorithm for $D_2$

Given new problem $D_1$

Reduce $D_1$ to $D_2$ and solve it

Ex: Bipartite matching $\leq$ max flow

Negative use

Believe that $D_1$ is hard

Given new problem $D_2$

Reduce $D_1$ to $D_2$

Shows that $D_2$ must be as hard as $D_1$

$G = (V,E)$, exists $k$-clique?

$\forall u, v \in V$

$\forall x, y \in V$

$\forall u, v \in V$

$\forall x, y \in V$

$i \in [k]$ if

$u = v \in E$

$i \neq j \in [k]$ if $(u,v) \notin E$
How hard is this formula for state-of-the-art SAT solvers? Open

Captures (other) state-of-the-art algorithm for actually solving clique in practice.
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
  - Averages 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 datasets 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
  - Yet, 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)
Solving vs. Verifying

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

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

Complexity class \( \mathsf{P} \)

Efficiently solvable problems
(in polynomial time)

Complexity class \( \mathsf{NP} \)

Problems for which solutions can be verified efficiently

**DEF**

Language \( L \) is in \( \mathsf{NP} \) if

\( \exists \) polynomial \( p \) and poly-time

TM \( M \) (verifier) s.t.

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

\( u \) 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 \( S \subseteq \{1, \ldots, n\} \) s.t.
   \( \sum_{i \in S} A_i = T \)

3. LINEAR PROGRAMMING
   \( m \) linear inequalities \( a_{11} u_1 + a_{12} u_2 + \ldots + a_{1n} u_n \leq b_1, \ldots, b \in \mathbb{Q} \),
   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
   \( \Pi : V(G_1) \rightarrow V(G_2) \) 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. CNF UNSAT
Given a CNF formula $F = \bigwedge_{i=1}^{m} C_i$ for clauses $C_i$ on the form $x \lor \neg x_2 \lor x_3 \lor x_4$, does every truth value assignment fail to satisfy at least one clause?

1. TSP: witness: tour
   complexity: hardest problem in $NP \cap BPP$; $NP$-complete

2. witness: subset
   complexity: $NP$-complete

3. witnesses: assignment
   complexity: For large cases, $NP \cap BPP \subset \text{time } C^{\sqrt{n}}$
   Khachiyan, 1979: $\neq \mathbb{P}$
   Karmarkar, 1984: Efficient

4. witnesses: assignment; complexity: $NP$-complete

5. witnesses: bijection split; not believed to be in $P$

6. witness: factor
   known to be solvable with randomness
   [Miller '76, Karp '80]
   in $P!$ [AKS '04]
7. Witness: prime factors
   not believed to be in P (RSA breaks down)
   not believed to be NP-complete.

8. Witness: path cyle in P (do BFS)

9. Witness: ?
   in coNP - see later

Proposition 3: \( P \leq NP \leq EXP \)

\[ \text{DEF 2 EXP (or EXPTIME) } = \cup_{c=1}^{\infty} \text{DTIME}(2^{n^c}) \]

Proof:
\( P \leq NP \): pick witnesses of length 0, candidates
\( NP \leq EXP \): at most exp many witnesses, try all in poly-time per candidate.

Prop 2 is state-of-the-art (sadly).
One of the Millennium Problems: \( P = NP \)
Most (but not all) believe \( P \neq NP \).
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 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 a language $L$ is in NTIME($T(n)$) if $\exists C > 0$
and $c \cdot T(n)$ - time NDTM such that
for every $x \in \{0,1\}^*$, $x \in L$ iff $M(x) = 1$
Lemma 5 \( NP = \bigcup_{\text{con}} \text{NTIME}(n^c) \)

Proof

(1) \( L \in NP \Rightarrow L \in \bigcup_{\text{con}} \text{NTIME}(n^c) \)

(2) \( L \in \text{NTIME} \bigcup_{\text{con}} \text{NTIME}(n^c) \Rightarrow L \in NP \)

(2) There is some witness. Let \( \text{NDTM} \) write down a witness nondeterministically, then verify (by running verifier for \( L \))

\( x \in L \Rightarrow \exists \) good witness \( \Rightarrow \) accept

\( x \not\in L \Rightarrow \) 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 \( \text{NDTM} \) with this sequence to check acceptance.

Wouldn't it be great to have an \( \text{NDTM} \) on your desk?

Not physically realizable (as far as we know).
But useful theoretical model, e.g. for exhaustive search.
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

\[ P \leq NP \leq EXP \]
Including widely believed to be strict
\[ P \neq NP \] the open problem of complexity theory

\( \neg \) in \( NP \) stands for "nondeterministic computation" — computation with "golden confusions"