DD2445 LECTURE 4

So far

Computational model (Turing machine)

Cplx class P - efficiently solvable (decision) problems

Cplx class NP - efficiently verifiable (decision) problems

NP-complete problems SAT

coNP

And above... polynomial hierarchy

EXP, NEXP

If EXP ≠ NEXP, then P ≠ NP

Proof: PADDING

Next collection of topics on the agenda

- Is it true that more time makes it possible to solve more problems?
- Are there complexity classes between P and NP?
- Diagonalization
- Oracles

How to prove that two cplx classes are different?

Find a language in one class that is not in the other

Every language $L \in \mathcal{C}$ decided by some TM M_L that runs within resource bound specified by $\mathcal{C}$

Separate $\mathcal{C}_1$ and $\mathcal{C}_2$ by finding TM M running within resource bounds specified by $\mathcal{C}_1$ that differs from every TM in $\mathcal{C}_2$ on at least one input

Then

$$L = \{ x \mid M(x) = 1 \}$$

is a language separating $\mathcal{C}_1$ and $\mathcal{C}_2$

$$L \in \mathcal{C}_1 \setminus \mathcal{C}_2$$

Essentially only known tool to do this:

DIAGONALIZATION

DIAGONALIZATION

Recall: Turing machine specified by
- finite alphabet Γ (symbols)
- finite set of possible states Q
- transition function (program) mapping
  $Q \times$ {read symbols on tapes} to
  $Q \times$ {written symbols on tapes} $\times$ {head movements}

Can agree on some encoding of TMs as (finite) binary strings. Let's use encoding such that
(a) exists "stop marker", and padding with more bits after stop marker has no effect but encodes same machine.
(b) "syntax error" encoding identified with trivial TM that immediately halts and rejects, say.

Then

1. Every string $x \in \{0,1\}^*$ represents a TM M_x. Given $i \in \mathbb{N}$ write M_i to denote Turing machine encoded by i written in binary
2. Every TM M is represented by infinitely many strings / infinitely many integers
3. This representation is efficient in that given x, we can simulate M_x on the universal Turing machine $\mathcal{U}$ with at most a logarithmic overhead.

Write table with rows and columns indexed by integers.

Interpret: Rows $\Leftrightarrow$ TMs

Columns $\Leftrightarrow$ inputs

	1	2	3	4	5
M_1					
M_2					
M_3					
M_4					
$\vdots$					

(i,j) contains $M_i(j)$ output of TM M_i on input j (in binary)

~~Cons~~ Construct TM by walking diagonally downwards to the left, making sure at least one mismatch per row $\Rightarrow$ Contradiction; such TM can't exist.

TIME HIERARCHY THEOREM

If f, g time-constructible functions satisfying $f(n) \log f(n) = o(g(n))$, then

$$\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n))$$

$\subsetneq$ strict subset containment. Also $\subset$ $\subsetneq$

Time-constructible?

Technical condition which we won't go into

All "natural" functions $f(n) \geq n$ that you can think of are time-constructible

E.g. $f(n) = n \log n$, $f(n) = n^2$, $f(n) = 2^n$ etc

Will prove:

TIME HIERARCHY THEOREM, VANILLA VERSION

$$\mathrm{DTIME}(n) \subsetneq \mathrm{DTIME}(n^{1.5})$$

Proof Let D be following TM:

On input x, run universal TM $\mathcal{U}$ for $|x|^{1.4}$ steps to simulate execution of M_x on x

If $\mathcal{U}$ halts with output $b \in \{0,1\}$,

output opposite answer $1-b$

Else output 0.

How set time bound for TM? E.g.

- compute $|x|$
- then compute $|x|^{1.4}$ & store in counter
- then fill dedicated worktape with special marker symbol, until counter decreased to 0.
- Now move back to start of work tape, start simulation, and at every step move right on "timer tape"
- abort if ever see non-marker symbol on timer tape

D decides some language, namely $L_D = \{x \mid D(x) = 1\}$

D runs in time $\sim n^{1.4}$ by construction (log factor for simulation does not change this)

Hence $L_D \in DTIME(n^{1.5})$ (by same margin).

We claim $L_D \notin DTIME(n)$

For contradiction, assume $\exists\, M$ that on any x runs in $\leq c|x|$ steps (for some fixed c) and outputs $D(x)$.

M can be simulated on any x in time $O(|x| \log |x|)$ by $\mathcal{U}$. $\leq c' \cdot |x| \log |x|$ for some c'

Fix large enough N s.t. $n^{1.4}$ is larger than this if $n \geq N$.

Pick some x of length $\geq N$ s.t.
$M_x = M$ (possible by (2) above)

Then – on input x, D will simulate M on x
– M will have time to terminate and output $M(x)$
– By def of D we have $D(x) = 1 - M(x) \neq M(x)$
– But M decides L_D by assumption, so $M(x) = D(x)$

Contradiction. Hence no such M exists, QED ☒

There is also a time hierarchy theorem for nondeterministic computation

NONDETERMINISTIC TIME HIERARCHY THEOREM
If f, g are time-constructible functions, satisfying $f(n+1) = o(g(n))$, then

$$\mathrm{NTIME}(f(n)) \subsetneq \mathrm{NTIME}(g(n))$$

Proof more subtle. Will skip this.

Most problems studied in NP are known either to be in P or to be NP-complete.
So can it be that every problem in NP is either in P or NP-complete?
(Results of that flavour known as DICHOTOMY THEOREMS.)

Answer If $P = NP$, then yes (trivially).
If $P \neq NP$, then no, in very strong sense.

What lies between P and NP?

If $P = NP$, nothing (clearly)

But what if $P \neq NP$?

LADNER'S THEOREM

If $P \neq NP$, then there exists a strict, infinite hierarchy of complexity classes between P and NP

Guided exercise for problem set

- Prove vanilla version of this statement
- Most of details can be found in textbook
- Want you to go through the proof and make sure you understand it
- Write nice, complete exposition aimed at student finishing ADK, say
- So practice also writing and presentation skills.

LADNER'S THEOREM, VANILLA VERSION

If $P \neq NP$, then there exists a language $L \in NP \setminus P$ that is not NP-complete

Caveat: This language L looks quite contrived... But interesting to know it exists.

Main idea: Padding.

Let $P: \mathbb{N} \to \mathbb{N}$ be some function such that $P(n)$ is computable in time polynomial in n.

Define SAT_P to be (CNF)SAT with all size-n formulas φ padded with $n^{P(n)}$ 1's

$$SAT_P = \{ \varphi 0 1^{n^{P(n)}} \mid \varphi \in CNFSAT \text{ and } n = |\varphi| \}$$

That is: given string x, scan from back until first 0. Let φ be everything before that 0. Set $n = |\varphi|$ = length of this. Have string 1^k after 0.
$x \in SAT_P$ if (a) $\varphi \in CNFSAT$ and (b) $k = n^{P(n)}$

OBSERVATIONS

- If $P(n) \in O(1)$, then SAT_P NP-complete
- If $P(n) = \Omega(n/\log n)$, then $SAT_P \in P$

Proof: Problem set

Want to choose padding function in some clever way so that SAT_P is too hard to be in P (assuming $P \neq NP$) but too easy to be NP-complete (because the padding gives extra time)

Here is our padding function

$\underline{H(n)}$

```
if n ≤ 4
   return 1
else
   i := 0 ;  failed := TRUE
   while i < log log n and failed
      failed := FALSE ; i := i+1 ;
      for all x ∈ {0,1}* with |x| ≤ log n
         simulate M_i on x for i·|x|^i steps
         if M_i didn't terminate
            failed := TRUE
         else
            let b := output of M_i(x)
            split x = φ 0 1^k and
               let s := |φ|
            check that b = 1 if and only if
                  φ ∈ CNFSAT and k = s^{H(s)}
            else
                  failed := TRUE
      endfor
   endwhile
   return i
```

Recursive call

Checking if M_i decides SAT_H correctly on all strings of at most logarithmic size

CLAIMS ABOUT H

1. H is well-defined (i.e., the algorithm computes a specific function)
2. $H(n)$ is computed in time polynomial in n
3. $SAT_H \in P$ if and only if $H(n) = O(1)$ [i.e., there exists a K such that $\forall n\ H(n) \leq K$]
4. If $SAT_H \notin P$ then $H(n) \to \infty$ as $n \to \infty$

Proofs: Problem set (plus read Arora-Barak)

Now assume $P \neq NP$

(i) Suppose $SAT_H \in P$.
Then we can show that $CNFSAT \in P$
But CNFSAT NP-complete. Contradiction.

(ii) Suppose SAT_H NP-complete
Then we can reduce CNFSAT to SAT_H efficiently
But if so can compose reductions and compress CNFSAT instance so much that they are solvable in polynomial time.
Contradiction.

Detailed proof: Problem set

Are there more interesting and natural non-NP-complete languages in $NP \setminus P$?

Obviously, we don't know

But FACTORING and GRAPH ISOMORPHISM are candidates (though graph isomorphism not so much any longer)