Harvard CS 121 and CSCI E-207
Lecture 22: The P vs. NP Question and NP-completeness

Salil Vadhan

November 20, 2012

- Reading: Sipser §7.4, §7.5.
- For "culture": *Computers and Intractability: A Guide to the Theory of NP-completeness*, by Garey & Johnson.

P vs. NP

- We would like to solve problems in NP efficiently.
- We know P $\subseteq$ NP.
- Problems in P can be solved “fairly” quickly.
- What is the relationship between P and NP?

NP and Exponential Time

Claim: NP $\subseteq \bigcup_k \text{TIME}(2^{n^k})$

Proof:

Of course, this gets us nowhere near P.

Is P $=$ NP?

i.e., do all the NP problems have polynomial time algorithms?

It doesn't "feel" that way but as of today there is no NP problem that has been proven to require exponential time!

The Strange, Strange World if P $=$ NP

Thousands of important languages can be decided in polynomial time, e.g.

- SATISFIABILITY
- TRAVELLING SALESMAN
- HAMILTONIAN CIRCUIT
- MAP COLORING
- $\vdots$

If P $=$ NP, then Searching becomes easy

Every "reasonable" search problem could be solved in polynomial time.

- "reasonable" $\equiv$ solutions can be recognized in polynomial time (and are of polynomial length)
- SAT SEARCH: Given a satisfiable boolean formula, find a satisfying assignment.
- FACTORING: Given a natural number (in binary), find its prime factorization.
- NASH EQUILIBRIUM: Given a two-player "game", find a Nash equilibrium.
- $\vdots$

If P $=$ NP, Optimization becomes easy

Every "reasonable" optimization problem can be solved in polynomial time.

- Optimization problem $\equiv$ "maximize (or minimize) $f(x)$ subject to certain constraints on x" (AM 121)
- "Reasonable" $\equiv$ "f and constraints are poly-time"
- MIN-TSP: Given a TSP instance, find the shortest tour.
- SCHEDULING: Given a list of assembly-line tasks and dependencies, find the maximum-throughput scheduling.
- PROTEIN FOLDING: Given a protein, find the minimum-energy folding.
- CIRCUIT MINIMIZATION: Given a digital circuit, find the smallest equivalent circuit.

If P $=$ NP, Secure Cryptography becomes impossible

Every polynomial-time encryption algorithm can be “broken” in polynomial time.

- “Given an encryption z, find the corresponding decryption key K and message m” is an NP search problem.
- Thus modern cryptography seeks to design encryption algorithms that cannot be broken under the *assumption* that certain NP problems are hard to solve (e.g. FACTORING).
- Take CS 220r.

If P $=$ NP, Artificial Intelligence becomes easy

Machine learning is an NP search problem

- Given many examples of some concept (e.g. pairs (image1, “dog”), (image2, “person”), ...), classify new examples correctly.
- Turns out to be equivalent to finding a short “classification rule” consistent with examples.
- Take CS228.

If P $=$ NP, Even Mathematics Becomes Easy!

Mathematical proofs can always be found in polynomial time (in their length).

- SHORT PROOF: Given a mathematical statement S and a number n (in unary), decide if S has a proof of length at most n (and, if so, find one).
- An NP problem!
- cf. letter from Gödel to von Neumann, 1956.

Library of Congress

Gödel's Letter to Von Neumann, 55 years ago

[$\phi(n) =$ time required for a TM to determine whether a mathematical statement has a proof of length n]

...

If there really were a machine with $\phi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$) this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. ...

It would be interesting to know, for instance, the situation concerning the determination of primality of a number and how strongly in general the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search. ...

The World if P $\neq$ NP?

Q: If P $\neq$ NP, can we conclude anything about any specific problems?

Idea: Try to find a "hardest" NP language.

- Just like A_{TM} was the "hardest" Turing-recoginizable language.
- Want $L \in$ NP such that $L \in$ P iff every NP language is in P.

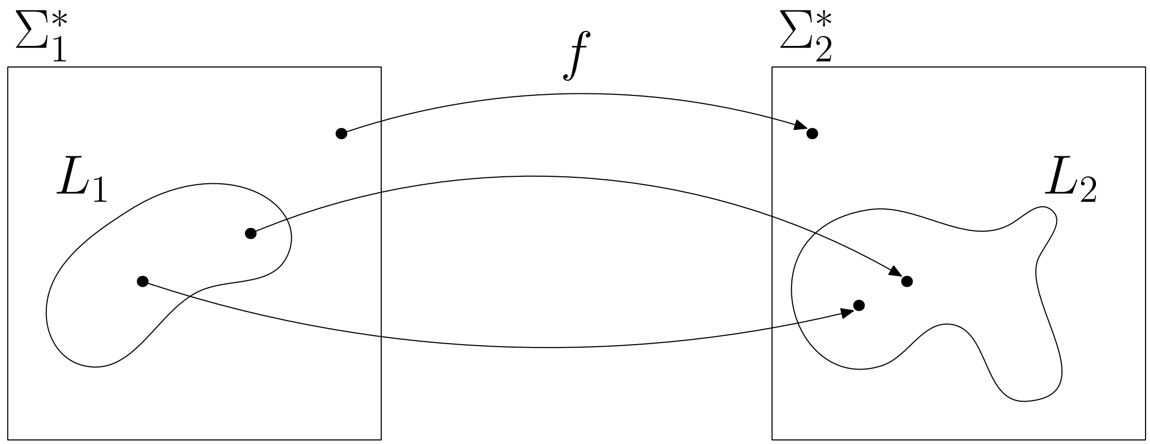
Polynomial-time Reducibility

Def: $L_1 \leq_{\mathsf{P}} L_2$ iff there is a polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ s.t. for every $x \in \Sigma_1^*$, $x \in L$ iff $f(x) \in L_2$.

Proposition: If $L_1 \leq_{\mathsf{P}} L_2$ and $L_2 \in$ P, then $L_1 \in$ P.

Proof:

$L_1 \leq_{\mathsf{P}} L_2$



$x \in L_1 \Rightarrow f(x) \in L_2$

$x \notin L_1 \Rightarrow f(x) \notin L_2$

f computable in polynomial time

$L_2 \in \mathsf{P} \Rightarrow L_1 \in \mathsf{P}$.

NP-Completeness

Def: L is NP-complete iff

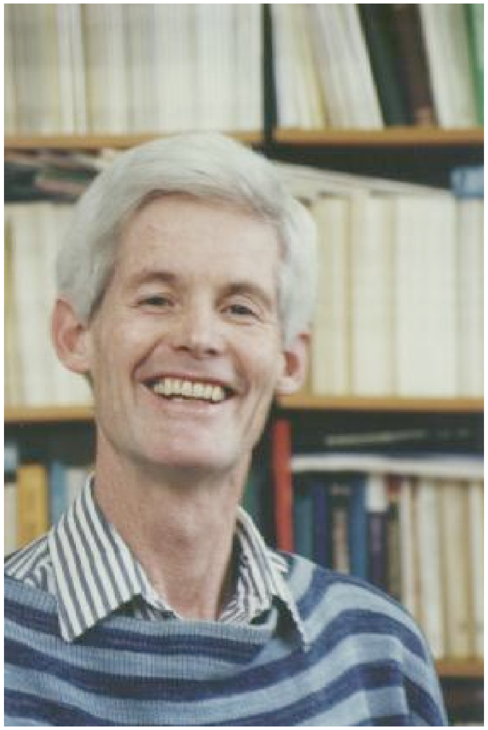
1. $L \in$ NP and
2. Every language in NP is reducible to L in polynomial time. ("L is NP-hard")

Prop: Let L be any NP-complete language.
Then P $=$ NP *if and only if* $L \in$ P.

Cook–Levin Theorem (Stephen Cook 1971, Leonid Levin 1973)

Theorem: SAT (Boolean satisfiability) is NP-complete.

Proof: Need to show that every language in NP reduces to SAT (!) Proof later.



More NP-complete problems

From now on we prove NP-completeness using:

Lemma: If we have the following

- L is in NP
- $L_0 \leq_{\mathsf{P}} L$ for some NP-complete L_0

Then L is NP-complete.

Proof:

3-SAT

Def: A Boolean formula is in 3-CNF if it is of the form:

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where each clause C_i is a disjunction ("or") of 3 literals:

$$C_i = (C_{i1} \vee C_{i2} \vee C_{i3})$$

where each literal C_{ij} is either

- a variable x, or
- the negation of a variable, $\neg x$.

e.g. $(x \vee y \vee z) \wedge (\neg x \vee \neg u \vee w) \wedge (u \vee u \vee u)$

3-SAT is the set of satisfiable 3-CNF formulas.

3-SAT is NP-complete

Proof: Show that SAT $\leq_{\mathrm{P}}$ 3-SAT.

1. Given an arbitrary Boolean formula, e.g.

$$\begin{array}{l} F = (\neg((x \vee \neg y) \wedge (z \vee w)) \vee \neg x). \\ \qquad\quad 1 \qquad\; 2\;\, 3 \quad\; 4 \quad\;\; 5 \qquad\; 6\;\, 7 \end{array}$$

2. Number the operators.

3. Select a new variable a_i for each operator.
   The variable a_i is supposed to mean "the subformula rooted at operator i is true."

4. Write a formula stating the relation between each subformula and its children subformulas.

Reduction of SAT to 3-SAT, continued

For example, where

$$F = (\neg((x \vee \neg y) \wedge (z \vee w)) \vee \neg x),$$
$$\quad\quad 1 \quad\quad 2\ 3 \quad 4 \quad\quad 5 \quad\quad 6\ 7$$

$$F_1 = \left(\begin{array}{llll} & (a_3 \equiv \neg y) & \wedge & (a_7 \equiv \neg x) \\ \wedge & (a_2 \equiv x \vee a_3) & \wedge & (a_1 \equiv \neg a_4) \\ \wedge & (a_5 \equiv z \vee w) & \wedge & (a_6 \equiv a_1 \vee a_7) \\ \wedge & (a_4 \equiv a_2 \wedge a_5) & & \end{array} \right)$$

5. Let k be the number of the main operator/subformula of F. (Note: $k = 6$ in the example)

Write F_1 in 3-CNF to obtain F_2

- **Fact:** Every function $f : \{0,1\}^k \to \{0,1\}$ can be written as a k-CNF and as a k-DNF (OR of ANDs).
  [albeit with possibly 2^k clauses]

- **Proof:**

Output of the reduction: $a_k \wedge F_2$.

Q: Does this prove that every Boolean formula can be converted to 3-CNF?

In contrast, 2-SAT $\in$ P

Method (resolution):

1. If x and $\neg x$ are both clauses, then not satisfiable

   e.g. $(x) \wedge (z \vee y) \wedge (\neg x)$

2. If $(x \vee y) \wedge (\neg y \vee z)$ are both clauses, add clause $(x \vee z)$ (which is implied).

3. Repeat. If no contradiction emerges $\Rightarrow$ satisfiable.

$O(n^2)$ repetitions of step 2 since only 2 literals/clause.

Proof of correctness: omitted