Harvard CS 121 and CSCI E-121

Lecture 22: The P vs. NP Question and NP-completeness

Harry Lewis

November 19, 2013

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

Composite Numbers

- COMPOSITES $= \{w : w$ a composite number in binary $\}$.

  COMPOSITES $\in$ NP

  Not obviously in P, since an exhaustive search for factors can take time proportional to the value of w, which grows as $2^n =$ exponential in the size of w.

  Only in 2002 was it shown that COMPOSITES $\in$ P (equivalently, PRIMES $\in$ P).

Boolean logic

Boolean formulas

Def: A Boolean formula (B.F.) is either:

- a "Boolean variable" $x, y, z, \ldots$
- $(\alpha \vee \beta)$ where α, β are B.F.'s.
- $(\alpha \wedge \beta)$ where α, β are B.F.'s.
- $\neg\alpha$ where α is a B.F.

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

[Omitting redundant parentheses]

Boolean satisfiability

Def: A truth assignment is a mapping
a : Boolean variables $\rightarrow \{0, 1\}$. [$0 =$ false, $1 =$ true]

The $\{0, 1\}$ value of a B.F. γ on a truth assignment a is given by the usual rules of logic:

- If γ is a variable x, then $\gamma(a) = a(x)$.
- If $\gamma = (\alpha \vee \beta)$, then $\gamma(a) = 1$ iff $\alpha(a) = 1$ or $\beta(a) = 1$.
- If $\gamma = (\alpha \wedge \beta)$, then $\gamma(a) = 1$ iff $\alpha(a) = 1$ and $\beta(a) = 1$.
- If $\gamma = \neg\alpha$, then $\gamma(a) = 1$ iff $\alpha(a) = 0$.

a satisfies γ (sometimes written $a \models \gamma$) iff $\gamma(a) = 1$.

In this case, γ is satisfiable. If no a satisfies γ, then γ is unsatisfiable.

Boolean Satisfiability

SAT $= \{\alpha : \alpha$ is a satisfiable Boolean formula$\}$.

Prop: SAT $\in$ NP

A "similar" problem in P: 2-SAT

A 2-CNF formula is one that looks like

$$(x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg x)$$

i.e., a conjunction of clauses, each of which is the disjunction of 2 literals (or 1 literal, since $(x) \equiv (x \vee x)$)

2-SAT $=$ the set of satisfiable 2-CNF formulas.

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

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

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

Of course, this gets us nowhere near P.

Is $\text{P} = \text{NP}$?

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

- Many apparently hard DECISION PROBLEMS are actually easy!
- Thousands of important languages can be decided in polynomial time, e.g.
  - SATISFIABILITY
  - TRAVELLING SALESMAN
  - HAMILTONIAN CIRCUIT
  - MAP COLORING
- But not just DECISION problems but also SEARCH and OPTIMIZATION problems would also be easy if P = NP

If P = NP, then many apparently hard SEARCH PROBLEMS are 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.
- Proof: Use binary search!

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"
  - "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.

How to solve Optimization problems quickly if P=NP

- Use binary search to find the maximum value M such that, for some x, $f(x) \geq M$
- Any such question "is there an x such that $f(x) \geq M$?" can be answered quickly because the corresponding DECISION problem is easy
- Then search for such an x
- which is easy to do since SEARCH is easy

If P $=$ NP, Secure Cryptography becomes impossible

- **Cryptography:** Every 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.
  - Take CS127.

If P = NP, Artificial Intelligence becomes easy

- **Artificial Intelligence:** "Learning" is easy.
  - 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) if P $=$ NP.
  - 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, 24 years before NP was defined

[$\phi(n) =$ time required for a TM to determine whether a formula 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?

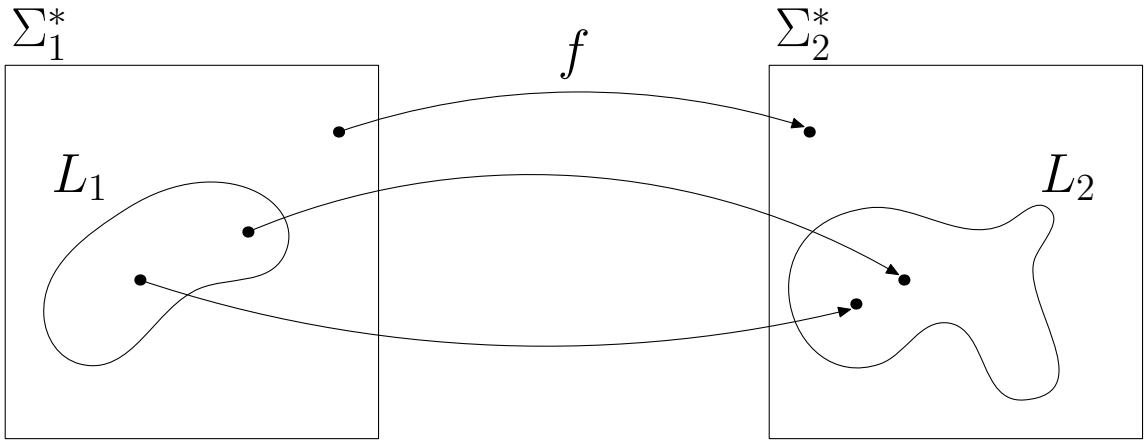
- **Q:** If P $\neq$ NP, can we conclude anything about any specific problems?

- **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 $L_2 \in \mathsf{P}$, then $L_1 \in \mathsf{P}$.
- **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

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

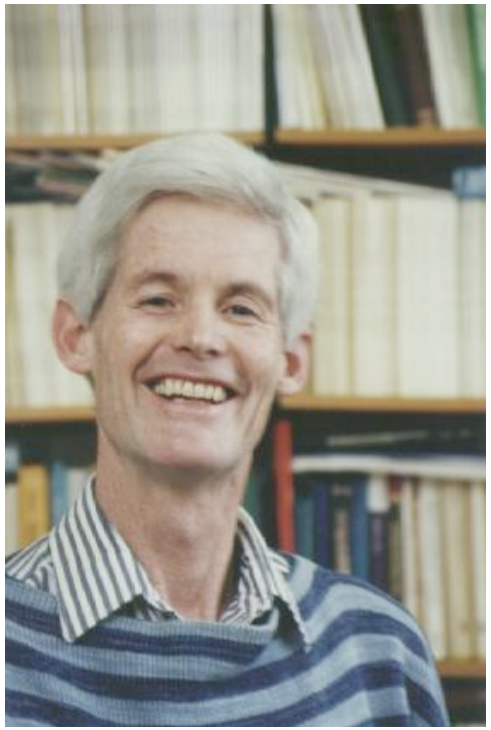
NP-Completeness

- **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.



Cook, Symposium on Theory of Computing, 1971

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly

Levin, "Universal Search Problems," 1973

ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX *1973* *Вып. 3*

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

"Several well-known large-scale problems of the sequential search type are discussed, and it is proved that those problems can be solved only in the time that it takes to solve any problems of the indicated type, in general."