P, NP, and NP-Completeness

Siddhartha Sen

Questions: sssix@cs.princeton.edu

Some figures obtained from Introduction to Algorithms, 2nd ed., by CLRS

Tractability

Polynomial time (p-time) = O(n^k), where n is the input size and k is a constant

Problems solvable in p-time are considered **tractable**

NP-complete problems have no known p-time solution, considered **intractable**

Tractability

Difference between tractability and intractability can be slight

Can find shortest path in graph in O(m + $n$lg$n$) time, but finding longest simple path is NP-complete

Can find satisfiable assignment for 2-CNF formula in O(n) time, but for 3-CNF is NP-complete:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

Outline

- Complexity classes P, NP
  - Formal-language framework
- NP-completeness
  - Hardest problems in NP
- Reductions: $A \leq B$
  - NP-completeness reductions

Formal-language framework

Alphabet Σ = finite set of symbols

Language *L* over Σ is any subset of strings in Σ^*

We'll focus on $\Sigma = \{0, 1\}$

- L = {10, 11, 101, 111, 1011, …} is language of primes

Decision problems

A **decision problem** has a yes/no answer

Different, but related to **optimization problem**, where trying to maximize/minimize a value

Any decision problem Q can be viewed as language: $L = \{x \in \{0,1\}^* : Q(x) = 1\}$

Q **decides** *L*: every string in *L* accepted by Q, every string not in *L* rejected

Example of a decision problem

PATH = $\{\langle G, u, v, k\rangle : G = (V, E)$ is an undirected graph, $u,v \in V$, $k \geq 0$ is an integer, and $\exists$ a path from u to v in G with $\leq k$ edges$\}$

Encoding of input $\langle G, u, v, k\rangle$ is important! We express running times as function of input size

Corresponding optimization problem is SHORTEST-PATH

Complexity class P

P = $\{L \subseteq \{0, 1\}^* : \exists$ an algorithm A that decides L in p-time$\}$

PATH $\in$ P

Polynomial-time verification

Algorithm A **verifies** language L if

$L = \{x \in \{0, 1\}^* : \exists\, y \in \{0, 1\}^* \text{ s.t. } A(x, y) = 1\}$

Can verify PATH given input $\langle G, u, v, k\rangle$ and path from u to v

PATH $\in$ P, so verifying and deciding take p-time

For some languages, however, verifying much easier than deciding

SUBSET-SUM: Given finite set S of integers, is there a subset whose sum is exactly t?

Complexity class NP

Let A be a p-time algorithm and k a constant:

NP = $\{L \in \{0, 1\}^* : \exists$ a certificate y, $|y| = O(|x|^k)$, and an algorithm A s.t. $A(x, y) = 1\}$

SUBSET-SUM $\in$ NP

P vs. NP

Not much is known, unfortunately

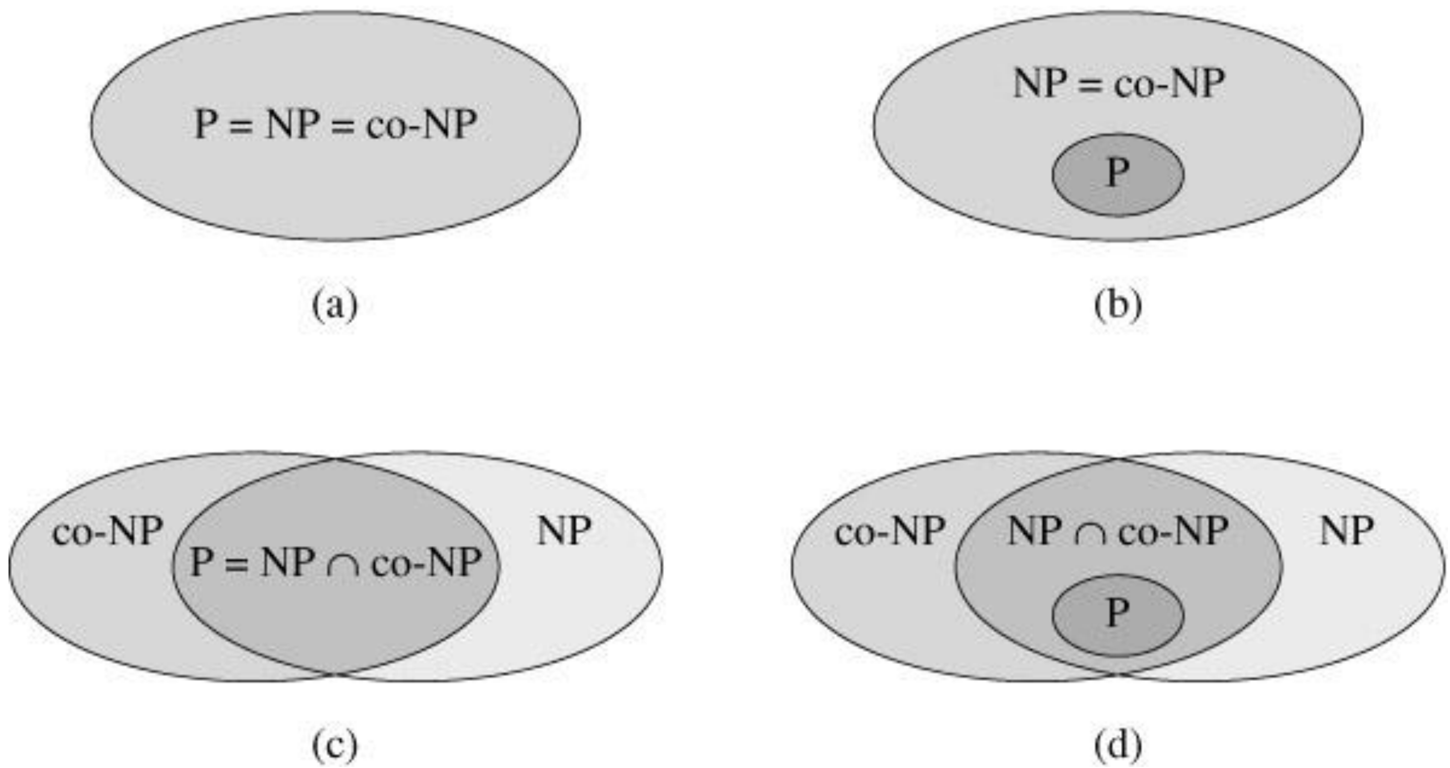
Can think of NP as the ability to appreciate a solution, P as the ability to produce one

P $\subseteq$ NP

Don't even know if NP closed under complement, i.e. NP = co-NP?

Does $L \in$ NP imply $\bar{L} \in$ NP?

P vs. NP



Comparing hardness

NP-complete problems are the “hardest” in NP:
if any NP-complete problem is p-time solvable, then all problems in NP are p-time solvable

How to formally compare easiness/hardness of problems?

Reductions

Reduce language L_1 to L_2 via function f:

1. Convert input x of L_1 to instance $f(x)$ of L_2
2. Apply decision algorithm for L_2 to $f(x)$

Running time = time to compute f + time to apply decision algorithm for L_2

Write as $L_1 \leq L_2$

Reductions show easiness/hardness

To show L_1 is easy, reduce it to something we know is easy (e.g., matrix mult., network flow, etc.)

$L_1 \leq easy$

Use algorithm for easy language to decide L_1

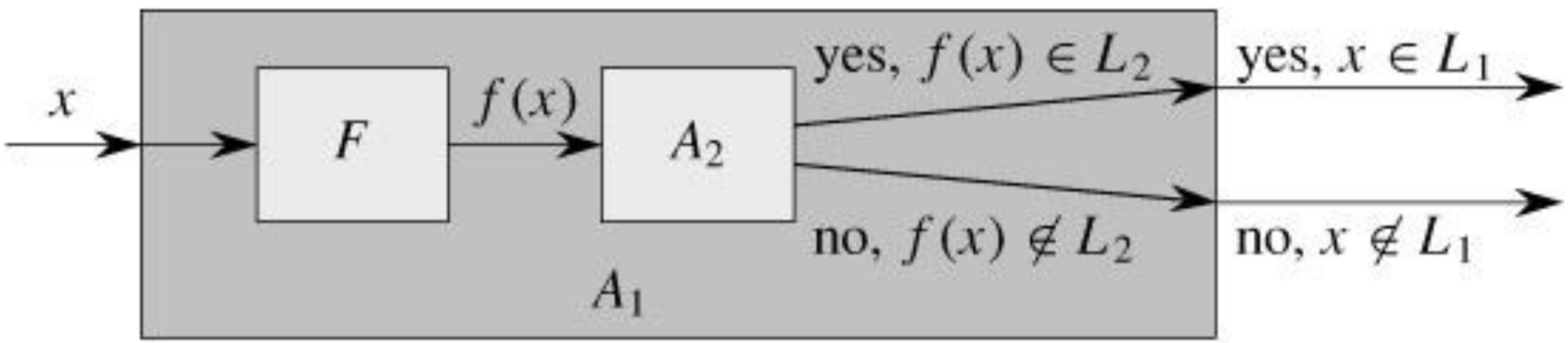
To show L_1 is hard, reduce something we know is hard to it (e.g., NP-complete problem):

$hard \leq L_1$

If L_1 was easy, *hard* would be easy too

Polynomial-time reducibility

L_1 is **p-time reducible** to L_2, or $L_1 \leq_p L_2$, if $\exists$ a p-time computable function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ s.t. for all $x \in \{0, 1\}^*$, $x \in L_1$ iff $f(x) \in L_2$



Lemma. If $L_1 \leq_p L_2$ and $L_2 \in$ P, then $L_1 \in$ P

Complexity class NPC

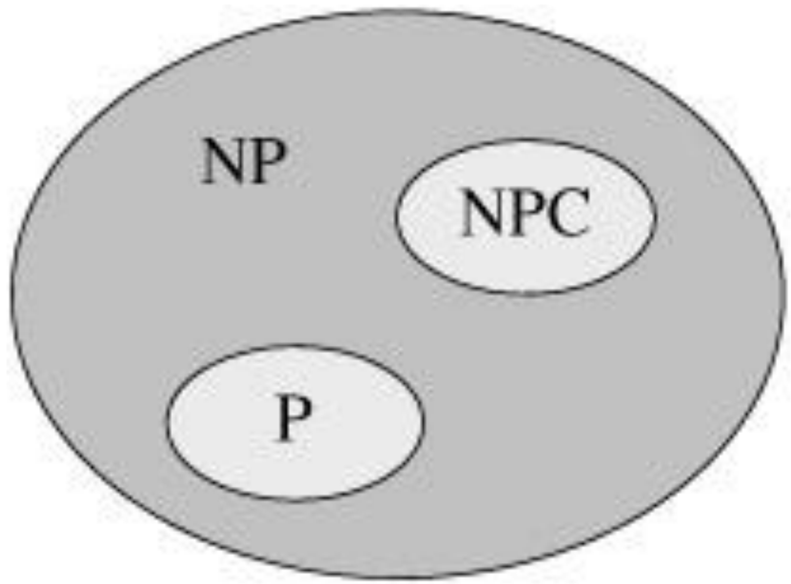
A language $L \subseteq \{0, 1\}^*$ is **NP-complete** if:

1. $L \in$ NP, and
2. $L' \leq_p L$ for every $L' \in$ NP, i.e. L is **NP-hard**

Lemma. If L is language s.t. $L' \leq_p L$ where $L' \in$ NPC, then L is NP-hard. If $L \in$ NP, then $L \in$ NPC.

Theorem. If any NPC problem is p-time solvable, then P = NP.

P, NP, and NPC



NPC reductions

Lemma. If L is language s.t. $L' \leq_p L$ where $L' \in$ NPC, then L is NP-hard. If $L \in$ NP, then $L \in$ NPC.

This gives us a recipe for proving any $L \in$ NPC:

1. Prove $L \in$ NP
2. Select $L' \in$ NPC
3. Describe algorithm to compute f mapping every input x of L' to input $f(x)$ of L
4. Prove f satisfies $x \in L'$ iff $f(x) \in L$, for all $x \in \{0, 1\}^*$
5. Prove computing f takes p-time

Bootstrapping

Need one language in NPC to get started

SAT = $\{\langle \phi \rangle : \phi$ is a satisfiable boolean formula$\}$

Can the variables of ϕ be assigned values in {0, 1} s.t. ϕ evaluates to 1?

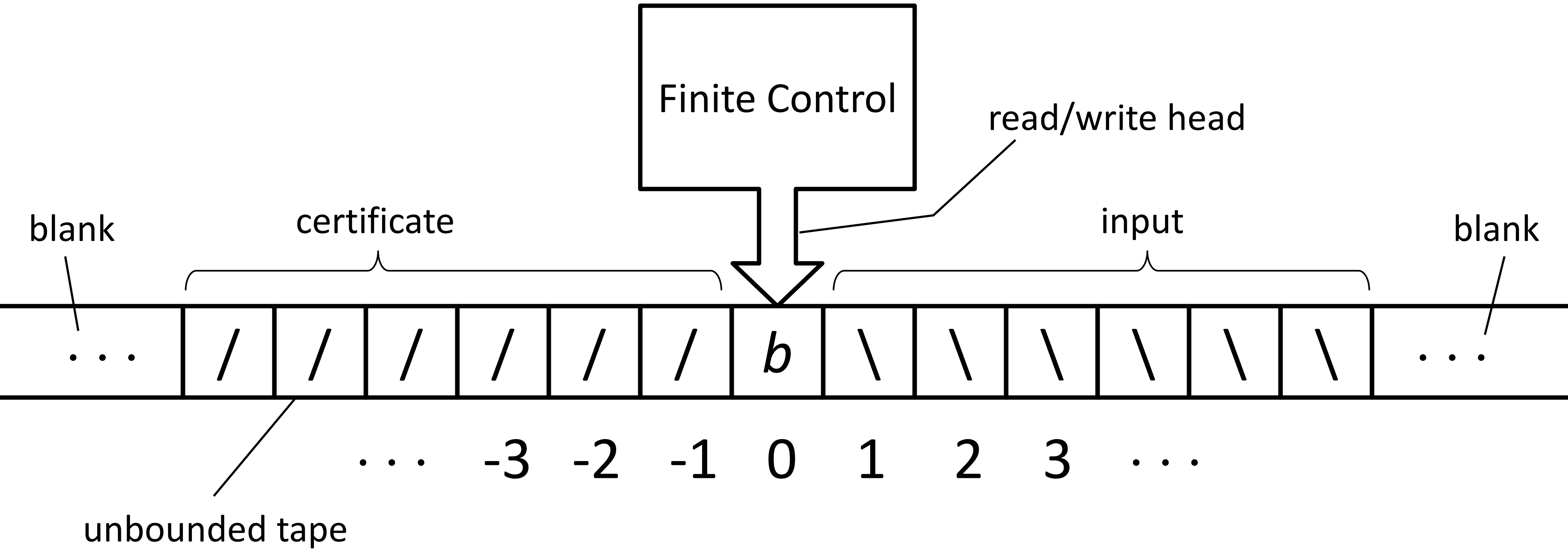
Cook-Levin theorem

Theorem. SAT $\in$ NPC.

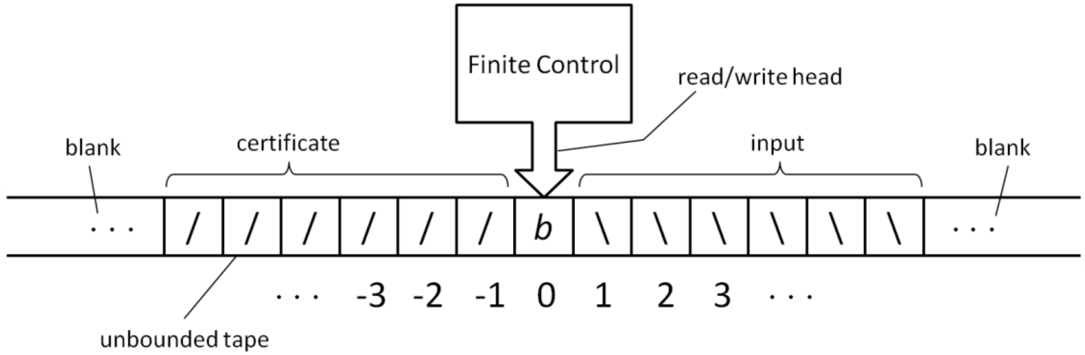
Proof. SAT $\in$ NP since certificate is satisfying assignment of variables. To show SAT is NP-hard, must show every $L \in$ NP is p-time reducible to it.

Idea: Use p-time verifier A(x,y) of L to construct input ϕ of SAT s.t. verifier says "yes" iff ϕ satisfiable

Verifier: Turing Machine



Church-Turing thesis: Everything computable is computable by a Turing machine



In one step, can write a symbol, move head one position, change state

What to do is based on state and symbol read

Fixed # of states: start state, “yes” state, (“no” state); fixed # of tape symbols, including blank

Explicit worst-case p-time bound $p(n)$

Proof plan

Given $L \in \text{NP}$ we have Turing machine that implements verifier A(x,y)

Input x, $|x| = n$, is "yes" instance iff for some certificate y, machine reaches "yes" state within $p(n)$ steps from start state

- Loops in "yes" state if gets there earlier

Construct $\phi = f(x)$ that is satisfiable iff this happens

- x is fixed and used to construct $f(x)$, but y is unspecified

Variables in ϕ

States: $1,\ldots, w$ // 1 = start, w = “yes”

Symbols: $1,\ldots, z$ // 1 = blank, rest input
// symbols like ‘0’ and ‘1’

Tape cells: $-p(n),\ldots, 0,\ldots, p(n)$

Time: $0, 1,\ldots, p(n)$

Variables:

h_{it}: true if head on tape cell i at time t,
$-p(\mathrm{n}) \leq i \leq p(n)$, $0 \leq t \leq p(n)$

s_{jt}: true if state j at time t,
$1 \leq j \leq w$, $0 \leq t \leq p(n)$

c_{ikt}: true if tape cell i holds symbol k at time t,
$-p(n) \leq i \leq p(n)$, $1 \leq k \leq z$, $0 \leq t \leq p(n)$

What does ϕ need to say?

At most one state, head position, and symbol per cell at each time:

$\neg h_{it} \vee \neg h_{i't}$, $\quad i \neq i'$, all t

$\neg s_{jt} \vee \neg s_{j't}$, $\quad j \neq j'$, all t

$\neg c_{ikt} \vee \neg c_{ik't}$, $\quad k \neq k'$, all i, all t

Correct initial state, head position, and tape contents:

$h_{00} \wedge s_{10} \wedge c_{010} \wedge c_{1k_10} \wedge c_{2k_20} \wedge \ldots \wedge c_{n\mathrm{k}_n0} \wedge c_{(n+1)10} \wedge \ldots \wedge c_{p(n)10}$

Input is $k_1,\ldots, k_n$, followed by blanks to right

Correct final state:

$s_{wp(n)}$

Correct transitions: e.g., if machine in state *j* reads *k*, it then writes *k'*, moves head right, and changes to state *j'*:

$$s_{jt} \wedge h_{it} \wedge c_{ikt} \Rightarrow s_{j'(t+1)} \wedge h_{(i+1)(t+1)} \wedge c_{ik'(t+1)}, \text{ all } i, t$$

Unread tape cells are unaffected:

$$h_{it} \wedge c_{i'kt} \Rightarrow c_{i'k(t+1)}, \quad i \neq i', \text{ all } k, t$$

Wrapping up

Any proof that gives "yes" execution gives satisfying assignment, and vice versa

Also ϕ contains $O(p(n)^2)$ variables, $O(p(n)^2)$ clauses

$\Rightarrow$ SAT $\in$ NPC

Now that we are bootstrapped, much easier to prove other $L \in$ NPC

Recall recipe for NPC proofs

1. Prove $L \in$ NP
2. Select $L' \in$ NPC
3. Describe algorithm to compute f mapping every input x of L' to input $f(x)$ of L
4. Prove f satisfies $x \in L'$ iff $f(x) \in L$, for all $x \in \{0, 1\}^*$
5. Prove computing f takes p-time

3-CNF-SAT $\in$ NPC

3-CNF-SAT = $\{\langle \phi \rangle : \phi$ is a satisfiable 3-CNF boolean formula$\}$

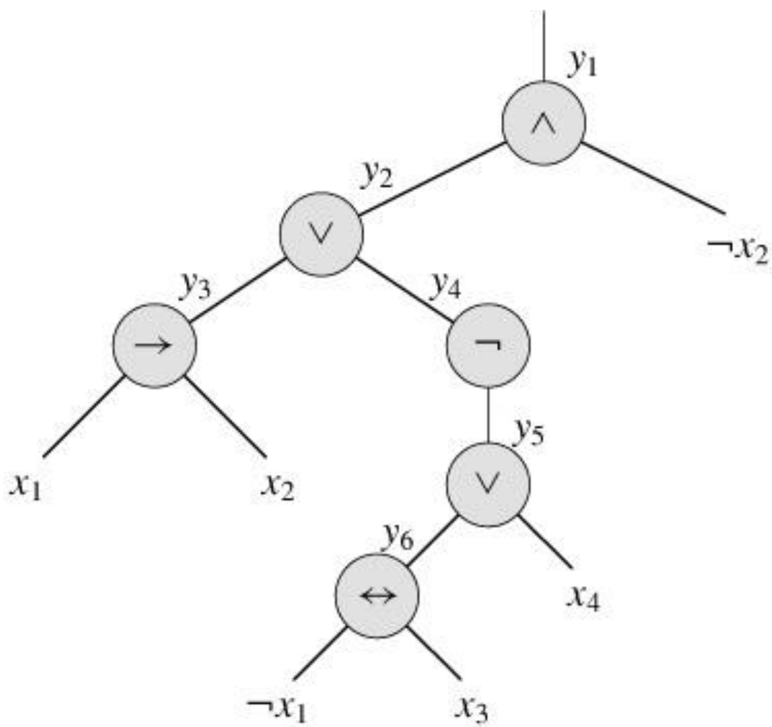
ϕ is **3-CNF** if it is AND of **clauses**, each of which is OR of three **literals** (variable or negation)

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

Proof. Show SAT $\leq_p$ 3-CNF-SAT

Given input of SAT, construct binary parse tree, introduce variable y_i for each internal node

E.g., $\phi = ((x_1 \Rightarrow x_2) \wedge \neg((\neg x_1 \Leftrightarrow x_3) \vee x_4)) \vee \neg x_2$



Rewrite as AND of root and clauses describing operation of each node:

$$
\begin{aligned}
\phi' \quad = \quad y_1 \quad & \wedge \quad (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
& \wedge \quad (y_2 \leftrightarrow (y_3 \vee y_4)) \\
& \wedge \quad (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\
& \wedge \quad (y_4 \leftrightarrow \neg y_5) \\
& \wedge \quad (y_5 \leftrightarrow (y_6 \vee x_4)) \\
& \wedge \quad (y_6 \leftrightarrow (\neg x_1 \rightarrow x_3))
\end{aligned}
$$

Each clause has at most three literals

Write truth table for each clause, e.g. for

$\phi'_1 = (y_1 \Leftrightarrow (y_2 \wedge \neg x_2))$:

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

Write DNF (OR of ANDs) for $\neg\phi'_1$:

$\neg\phi'_1 = (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee \ldots$

Use DeMorgan's laws to convert to CNF:

$\phi''_1 = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge \ldots$

If any clause has < three literals, augment with dummy variables p, q

$$(l_1 \vee l_2) \Leftrightarrow (l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$$

Resulting 3-CNF formula is satisfiable iff original SAT formula is satisfiable

CLIQUE $\in$ NPC

CLIQUE = $\{\langle G, k\rangle$: graph $G = (V, E)$ has clique of size $k\}$

Naïve algorithm runs in $\Omega(k^2 \times {}_{|V|}C_k)$

Proof. Show 3-CNF-SAT $\leq_p$ CLIQUE

Given formula $\phi = c_1 \wedge c_2 \wedge \ldots \wedge c_k$, construct input of CLIQUE:

For each $c_r = (l_1^r \vee l_2^r \vee l_3^r)$, place v_1^r, v_2^r, v_3^r in V

Add edge between v_i^r and v_j^s if r $\neq$ s and corresponding literals are consistent

If ϕ is satisfiable, at least one literal in each c_r is 1 $\Rightarrow$ set of k vertices that are completely connected

If G has clique of size k, contains exactly one vertex per clause $\Rightarrow$ ϕ satisfied by assigning 1 to corresponding literals

VERTEX-COVER $\in$ NPC

VERTEX-COVER = $\{\langle G, k\rangle$: graph $G = (V, E)$ has vertex cover of size $k\}$

Vertex cover is $V' \subseteq V$ s.t. if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both

Proof. Show CLIQUE $\leq_p$ VERTEX-COVER

Given input $\langle G, k\rangle$ of CLIQUE, construct input of VERTEX-COVER:

$\langle \bar{G}, |V| - k\rangle$, where $\bar{G} = (V, \bar{E})$

If G has clique V', $|V'| = k$, then $V - V'$ is vertex cover of $\bar{G}$:

$(u, v) \in \bar{E} \Rightarrow$ either u or v not in V', since $(u, v) \notin E$

$\Rightarrow$ at least one of u or v in $V - V'$, so covered

If $\bar{G}$ has vertex cover $V' \subseteq V$, $|V'| = |V| - k$, then $V - V'$ is clique of G of size k

$(u, v) \in \bar{E} \Rightarrow u \in V'$ or $v \in V'$ or both

if $u \notin V'$ and $v \notin V'$, then $(u, v) \notin E$

SUBSET-SUM $\in$ NPC

SUBSET-SUM = $\{\langle S, t\rangle : S \subset \mathbf{N}, t \in \mathbf{N}$ and $\exists$ a subset $S' \subseteq S$ s.t. $t = \Sigma_{s\in S'}\, s\}$

Integers encoded in binary! If t encoded in unary, can solve SUBSET-SUM in p-time, i.e. **weakly NPC** (vs. **strongly NPC**)

Proof. Show 3-CNF-SAT $\leq_{\mathrm{p}}$ SUBSET-SUM

Given formula ϕ, assume w.l.o.g. each variable appears in at least one clause, and variable and negation don't appear in same clause

Construct input of SUBSET-SUM:

- 2 numbers per variable x_i, $1 \leq i \leq n$, indicates if variable or negation is in a clause
- 2 numbers per clause c_j, $1 \leq j \leq k$, slack variables
- Each digit labeled by variable/clause, total $n + k$ digits
- t is 1 for each variable digit, 4 for each clause digit

$\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, $C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$, and $C_4 = (x_1 \vee x_2 \vee x_3)$

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v_1'	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v_2'	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v_3'	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s_1'	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s_2'	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s_3'	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s_4'	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

Max digit sum is 6, interpret numbers in base ≥ 7

Reduction takes p-time: set S has $2n + 2k$ values of $n + k$ digits each; each digit takes O(n + k) time to compute

If ϕ has satisfying assignment

Sum of variable digits is 1, matching t

Each clause digit at least 1 since at least 1 literal satisfied

Fill rest with slack variables s_j, s_j'

If $\exists$ $S' \subseteq S$ that sums to t

Includes either v_i or v_i' for each $i = 1,\ldots, n$; if $v_i \in S'$, set $x_i = 1$

Each clause c_j has at least one v_i or v_i' set to 1 since slacks add up to only 3; by above clause is satisfied

Implications of P = NP

Ability to verify a solution $\Rightarrow$ ability to produce one!

Can automate search of solutions, i.e. creativity!

Can use a p-time algorithm for SAT to find formal proof of any theorem that has a concise proof, because formal proofs can be verified in p-time

$\Rightarrow$ P = NP could very well imply solutions to all the other CMI million-dollar problems!

“If P = NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss...”

— Scott Aaronson, MIT