P, NP, and NP-Completeness

Siddhartha Sen
Questions: sssix@cs.princeton.edu

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

Polyomial 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 $O(m + n \log n)$ 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:

$$(x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (\neg x_2 \lor \neg x_3)$$
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 $\Sigma = \text{finite set of symbols}$

Language $L$ over $\Sigma$ is any subset of strings in $\Sigma^*$

We’ll focus on $\Sigma = \{0, 1\}$

$L = \{10, 11, 101, 111, 1011, \ldots\}$ 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 = {⟨G, u, v, k⟩ : G = (V, E) is an undirected graph, u, v ∈ V, k ≥ 0 is an integer, and ∃ a path from u to v in G with ≤ k edges}

Encoding of input ⟨G, u, v, k⟩ is important! We express running times as function of input size

Corresponding optimization problem is SHORTEST-PATH
Complexity class P

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

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:

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

$\text{SUBSET-SUM} \in \text{NP}$
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. \( NP = \text{co-NP} \)?

Does \( L \in NP \) imply \( \overline{L} \in NP \)?
P vs. NP

(a) \( P = NP = \text{co-NP} \)

(b) \( \text{NP} = \text{co-NP} \)

(c) \( \text{co-NP} \cap \text{NP} \cap \text{co-NP} \)

(d) \( \text{co-NP} \cap \text{NP} \cap \text{co-NP} \)
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 $L_2 \in \mathsf{P}$, then $L_1 \in \mathsf{P}$
Complexity class NPC

A language $L \subseteq \{0, 1\}^*$ is **NP-complete** if:
1. $L \in \text{NP}$, and
2. $L' \leq_p L$ for every $L' \in \text{NP}$, i.e. $L$ is **NP-hard**

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

**Theorem.** If any NPC problem is p-time solvable, then $P = \text{NP}$. 
P, NP, and NPC
NPC reductions

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

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

1. Prove $L \in \text{NP}$
2. Select $L' \in \text{NPC}$
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

\[ \text{SAT} = \{ \langle \phi \rangle : \phi \text{ is a satisfiable boolean formula} \} \]

Can the variables of \( \phi \) be assigned values in \{0, 1\} s.t. \( \phi \) evaluates to 1?
**Cook-Levin theorem**

**Theorem.** \( \text{SAT} \in \text{NPC}. \)

**Proof.** \( \text{SAT} \in \text{NP} \) since certificate is satisfying assignment of variables. To show \( \text{SAT} \) is \( \text{NP} \)-hard, must show every \( L \in \text{NP} \) is \( p \)-time reducible to it.

Idea: Use \( p \)-time verifier \( A(x,y) \) of \( L \) to construct input \( \phi \) of \( \text{SAT} \) s.t. verifier says “yes” iff \( \phi \) satisfiable.
**Verifier: Turing Machine**

Finite Control

read/write head

input

unbounded tape

-3 -2 -1 0 1 2 3

blank

certificate

**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 $\phi$

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

Symbols: $1, \ldots, z$  // $1 = \text{blank}$, 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 $t$,
$-p(n) \leq i \leq p(n)$, $0 \leq t \leq p(n)$

$s_{jt}$: 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 $\phi$ need to say?

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

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

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

$$\neg c_{ikt} \lor \neg c_{i'k't}, \quad k \neq k', \text{ all } i, \text{ all } t$$
Correct initial state, head position, and tape contents:

\[ h_{00} \land s_{10} \land c_{010} \land c_{1k_10} \land c_{2k_20} \land ... \land c_{nk_n0} \land c_{(n+1)10} \land \]
\[ ... \land c_{p(n)10} \]

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 $j'$:

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

Unread tape cells are unaffected:

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

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

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

$\implies \text{SAT} \in \text{NPC}$

Now that we are bootstrapped, much easier to prove other $L \in \text{NPC}$
Recall recipe for NPC proofs

1. Prove $L \in \text{NP}$
2. Select $L' \in \text{NPC}$
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 ∈ NPC

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

\phi \text{ is 3-CNF if it is AND of clauses, each of which is OR of three literals (variable or negation)}

\quad (x_1 \lor \neg x_1 \lor \neg x_2) \land (x_3 \lor x_2 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4)

Proof. Show \text{SAT} \leq_p 3\text{-CNF-SAT}
Given input of SAT, construct binary parse tree, introduce variable $y_i$ for each internal node

E.g., $\phi = ((x_1 \Rightarrow x_2) \land \neg((\neg x_1 \Leftrightarrow x_3) \lor x_4)) \lor \neg x_2$
Rewrite as AND of root and clauses describing operation of each node:

\[ \phi' = y_1 \land (y_1 \leftrightarrow (y_2 \land \neg x_2)) \land (y_2 \leftrightarrow (y_3 \lor y_4)) \land (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \land (y_4 \leftrightarrow \neg y_5) \land (y_5 \leftrightarrow (y_6 \lor x_4)) \land (y_6 \leftrightarrow (\neg x_1 \rightarrow x_3)) \]

Each clause has at most three literals
Write truth table for each clause, e.g. for
\( \phi_1' = (y_1 \leftrightarrow (y_2 \land \neg x_2)) \):

<table>
<thead>
<tr>
<th>( y_1 )</th>
<th>( y_2 )</th>
<th>( x_2 )</th>
<th>( (y_1 \leftrightarrow (y_2 \land \neg x_2)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Write DNF (OR of ANDs) for \( \neg \phi_1' \):
\[ \neg \phi_1' = (y_1 \land y_2 \land x_2) \lor (y_1 \land \neg y_2 \land x_2) \lor \ldots \]

Use DeMorgan’s laws to convert to CNF:
\[ \phi''_1 = (\neg y_1 \lor \neg y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor \neg x_2) \land \ldots \]
If any clause has < three literals, augment with dummy variables $p$, $q$

$$(l_1 \lor l_2) \iff (l_1 \lor l_2 \lor p) \land (l_1 \lor l_2 \lor \neg p)$$

Resulting 3-CNF formula is satisfiable iff original SAT formula is satisfiable
CLIQUE ∈ NPC

CLIQUE = \{\langle G, k \rangle : \text{graph } G = (V, E) \text{ 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 $\phi = c_1 \land c_2 \land \ldots \land c_k$, construct input of CLIQUE:

For each $c_r = (l_1^r \lor l_2^r \lor l_3^r)$, 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 $\phi$ is satisfiable, at least one literal in each $c_r$ is 1 $\implies$ set of $k$ vertices that are completely connected

If $G$ has clique of size $k$, contains exactly one vertex per clause $\implies \phi$ 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:
$\langle \tilde{G}, |V| - k \rangle$, where $\tilde{G} = (V, \tilde{E})$

If $G$ has clique $V'$, $|V'| = k$, then $V - V'$ is vertex cover of $\tilde{G}$:
$(u, v) \in \tilde{E} \implies$ either $u$ or $v$ not in $V'$, since $(u, v) \not\in E$
$\implies$ at least one of $u$ or $v$ in $V - V'$, so covered

If $\tilde{G}$ has vertex cover $V' \subseteq V$, $|V'| = |V| - k$, then $V - V'$ is clique of $G$ of size $k$
$(u, v) \in \tilde{E} \implies u \in V'$ or $v \in V'$ or both
if $u \not\in V'$ and $v \not\in V'$, then $(u, v) \not\in E$
**SUBSET-SUM ∈ NPC**

SUBSET-SUM = \{ ⟨S, t⟩ : S ⊆ \mathbb{N}, t ∈ \mathbb{N} \text{ and } \exists \text{ a subset } S' \subseteq S \text{ s.t. } t = \sum_{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 \( \leq_p \) SUBSET-SUM
Given formula $\phi$, 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
\( \phi = C_1 \land C_2 \land C_3 \land C_4, \)  

\( C_1 = (x_1 \lor \neg x_2 \lor \neg x_3), \)  

\( C_2 = (\neg x_1 \lor \neg x_2 \lor \neg x_3), \)  

\( C_3 = (\neg x_1 \lor \neg x_2 \lor x_3), \)  

\( \text{and } C_4 = (x_1 \lor x_2 \lor x_3) \)

<table>
<thead>
<tr>
<th></th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( C_3 )</th>
<th>( C_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_1 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( v_1' )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( v_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( v_2' )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( v_3 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( v_3' )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_1' )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_2' )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( s_3' )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( s_4' )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>( t )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Max digit sum is 6, interpret numbers in base \( \geq 7 \)
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 $\phi$ 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