NP-Completeness

Problems and Instances
Polynomial Time Verification
NP-completeness and Reducibility
NP-completeness Proofs
NP-complete Problems
Mostly, we have studied problems with \textit{polynomial-time algorithms}: running time is $O(n^k)$, for input size $n$ and some constant $k$.

Many problems have no poly-time-alg, e.g., halting problem.

For a third set of problems, it is unknown whether they have poly-time-algs or not.

This third set includes the class of \textit{NP-complete} problems.
Examples of P and NPC Problems

P = class of problems solvable in polynomial-time.
NPC = class of NP-complete problems.
NP = class of NP problems (includes P and NPC).

Single-source shortest paths is in P.
Finding the longest simple path is in NPC.$^a$

Finding a Euler tour (each edge once) is in P.
Finding a hamiltonian cycle (each vertex once) is in NPC.$^a$

Finding a solution to 2-CNF formulas is in P.
Finding a solution to 3-CNF formulas is in NPC.$^a$

$^a$Technically, this problem is NP-hard.
A decision problem is a problem with yes/no answers.

We can show that decision problem $A$ is in $P$ if there is a poly-time-alg that reduces $A$ to a decision problem $B$ which is known to be in $P$.

That is, each instance $\alpha \in A$ is converted to an instance $\beta \in B$ such that $B$’s algorithm delivers the answer to $\alpha$. 
An abstract problem is a relation from instances to solutions.

E.g., integer addition: instance $2 + 2$, solution 4. E.g., equations: instance $x^2 = 9$, sols. $3$ and $-3$.

A decision problem has solutions $= \{\text{yes, no}\}$.

Abstract problems have related decision problems. Integer addition DP: instance $2 + 2 \geq 3$, sol. yes. Equation DP: instance $x^2 = 9, x \geq 4$, solution no.

An encoding maps instances to (binary) strings. E.g., a graph can be an adjacency list or matrix. Encodings map abstract to concrete problems.
A problem \( Q \) is \textit{polynomial-time solvable} if there exists an algorithm \( A \) and a constant \( k \) such that \( A \) solves all instances in \( O(n^k) \), where \( n \) is the length of the encoded instance.

The \textit{language} of a decision problem is the set of instances with yes solutions.

The polynomial complexity class \( P \) is defined:

\[
P = \{ L : L \text{ is a language decided by a polynomial-time algorithm} \}
\]
Example P Problems

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 
there exists a path from u to v in G 
consisting of k edges or less. \}

MEDIAN = \{ \langle S, m \rangle : 
S is a set of integers, 
m is an integer, and 
m \leq \text{the median of } S \}
A *hamiltonian cycle* of an undirected graph is a simple cycle that contains all vertices.

\[
\text{HAM-CYCLE} = \{ \langle G \rangle : G \text{ has a hamiltonian cycle} \} 
\]
Verification Algorithms

No \( P \) algorithm is known for HAM-CYCLE.

But given a path \( p \) in a graph \( G \), a \( P \) algorithm can verify whether \( p \) is a hamiltonian cycle.

A verification algorithm \( A(x, y) \) has two args.: the input string \( x \) and the certificate \( y \).

The language verified by \( A(x, y) \) is:

\[
L = \{ x : \text{there exists } y \text{ s.t. } A(x, y) = \text{yes} \}
\]

For HAM-CYCLE, \( x \) would be a graph, and the certificate \( y \) would be a path.
NP is the class of problem with \( P \) verification algorithms with poly-size certificates.

\[ L \in \text{NP} \text{ iff there exists a } P \text{ algorithm } A(x, y) \text{ and a constant } c \text{ such that:} \]

\[ L = \{ x : \text{ there exists a } y \text{ such that } |y| \text{ is } O(|x|^c), \text{ and } A(x, y) = \text{yes} \} \]

HAM-CYCLE \( \in \text{NP} \) because if \( G \in \text{HAM-CYCLE} \), then the certificate is the hamiltonian path \( p \).

Consider:

\( SAT = \{ \langle \phi \rangle : \phi \text{ is a satisfiable boolean formula} \} \)

\( \text{COMPOSITE} = \{ \langle n \rangle : n \text{ is a composite integer} \} \)
A problem $Q$ can be reduced to a problem $Q'$ if each instance $x$ of $Q$ can be transformed to an instance $x'$ of $Q'$ such that the solution to $x'$ can be transformed back to a solution for $x$.

Finding the median reduces to sorting. Topological sort reduces to depth-first search. Difference constraints reduces to shortest-paths.

In a larger sense, all programming is finding a way to “reduce” (or “transform”) the problem to be solved to the API of a programming language.
A language $L_1$ is *polynomial-time reducible* to $L_2$ ($L_1 \leq_P L_2$) if a polynomial-time function $f$ satisfies $x \in L_1$ if and only if $f(x) \in L_2$. 
Reducibility Examples

**Introduction**

**The Complexity Class** $P$

**The Complexity Class** $NP$

**Polynomial Reducibility**
- reducibility 1
- reducibility 2
- reducibility 3
- reducibility 4
- reducibility 5
- reducibility 6

**The Complexity Class** $NPC$

**NPC Proofs**

---

\[
\text{EVEN} = \{ \langle n \rangle : n \text{ is an even number} \}
\]

\[
\text{ODD} = \{ \langle n \rangle : n \text{ is an odd number} \}
\]

\[
\text{EVEN} \leq_P \text{ODD} \text{ (and vice versa)}
\]

\[
\text{MEDIAN} = \{ \langle S, n \rangle : n \leq S\text{'s median} \}
\]

\[
\text{ORDER-STATISTIC} = \{ \langle S, n, i \rangle : n \leq S\text{'s } i\text{th order statistic} \}
\]

\[
\text{MEDIAN} \leq_P \text{ORDER-STATISTIC}
\]

\[
\text{CONNECTED} = \{ \langle G \rangle : G \text{ is a connected, undirected graph} \}
\]

\[
\text{STRONGLY-CONNECTED} = \{ \langle G \rangle : G \text{ is a strongly connected, directed graph} \}
\]

\[
\text{CONNECTED} \leq_P \text{STRONGLY-CONNECTED}
\]
Reducibility Properties

If $L_1 \leq_P L_2$ and $L_2 \in P$, then $L_1 \in P$.
If $L_1 \leq_P L_2$ and $L_1 \notin P$, then $L_2 \notin P$.

Proof:
Let $F$ be a $O(n^k)$ reduction alg. for $L_1 \leq_P L_2$.
Let $A_2$ be a $O(n^l)$ algorithm for $L_2$.
We can define an algorithm $A_1$ for $L_1$ by

$$A_1(x) = A_2(F(x))$$

$F$ is $O(n^k)$ implies the size of $F(x)$ is $O(|x|^k)$.
$A_2$ is $O(n^l)$ implies $A_2$ is $O(|x|^{kl})$ on an input of size $O(|x|^k)$.
It follows that $A_1 \in P$ because $A_1$ is $O(n^{kl})$. 
Another Reducibility Example

HAMPATH = \{⟨G, u, v⟩ : there is a simple path from u to v including all vertices\}

SIMPLE = \{⟨G, u, v, l⟩ : there is a simple path from u to v with length l\}

HAMPATH \leq_P SIMPLE by

f(G, u, v) = ⟨G, u, v, |G.V| − 1⟩

Now need to show x = ⟨G, u, v⟩ ∈ HAMPATH if and only if x' = ⟨G, u, v, |G.V| − 1⟩ ∈ SIMPLE.

If x ∈ HAMPATH, then G has a simple path including all vertices, which has length |G.V| − 1.

If x' ∈ SIMPLE, then a simple path of length |G.V| − 1 must include all of G’s vertices.
HAM-CYCLE = \{ \langle G \rangle : G \text{ has a hamiltonian cycle} \}
HAM-PATH = \{ \langle G, u, v \rangle : \text{there is a simple path from } u \text{ to } v \text{ including all vertices} \}

HAM-CYCLE \leq_P HAM-PATH by

\[ f(G) = \langle G', u, u' \rangle \]
where \(G')\) is constructed by copying \(G\) and duplicating any vertex \(u\).

Now need to show \(x = \langle G \rangle \in HAM-CYCLE\) if and only if \(x' = \langle G, u, u' \rangle \in HAM-PATH\).

If \(x \in HAM-CYCLE\), then \(G'\) has a simple path from \(u\): use \(G'\) ham. cycle, but end with \(u'\).

If \(x' \in HAM-PATH\), then \(G\) has a ham. cycle: start from \(u\), use \(G'\) simple path, but end with \(u\).
Recall NP and $\leq_P$.

$L \in \text{NP}$ iff there exists a $\text{P}$ algorithm $A(x, y)$ and a constant $c$ such that:

$L = \{x : \text{there exists a certificate } y \text{ such that } |y| \text{ is } O(|x|^c), \text{ and } A(x, y) = \text{yes}\}$

A language $L_1$ is polynomial-time reducible to $L_2$ ($L_1 \leq_P L_2$) if a polynomial-time function $f$ satisfies $x \in L_1$ if and only if $f(x) \in L_2$. 
The Definition of NP-Complete

$L$ is \textit{NP-hard} if $L' \leq_P L$ for every $L' \in \text{NP}$.

$L \in \text{NPC}$ if $L \in \text{NP}$ and $L$ is NP-hard.

If $L \in \text{NPC}$ and $L \in \text{P}$, then $\text{NP} = \text{P}$.

No known $L$ is both \text{NPC} and \text{P}.

\text{CIRCUIT-SAT} = \text{set of satisfiable boolean circuits}

\text{CIRCUIT-SAT} \in \text{NPC} \text{ because it is in } \text{NP} \text{ and all NP problems } \leq_P \text{CIRCUIT-SAT}.$
Sketch of reduction of any $L \in \text{NP}$ to CIRCUIT-SAT.

Simulate a computer cycle of $L$’s verification algorithm by a boolean circuit plus boolean inputs/outputs.

Simulate a sequence of cycles by a sequence of circuits.

A poly-time verification algorithm and a poly-size certificate implies a poly-number of cycles, which implies a poly-size circuit.
To show that $L$ is NP-complete ($L \in NPC$)

1. Show $L \in NP$.

2. Show $L' \leq_P L$ for a known language $L' \in NPC$.

**SAT = set of satisfiable Boolean formulas**

1. SAT $\in$ NP because if $\phi \in SAT$, then the certificate is the assignment making $\phi$ true.

2. CIRCUIT-SAT $\leq_P$ SAT as follows.
   Map each gate’s output to a variable.
   Map each gate to a formula.
   AND all the formulas.
Illustration for SAT Reduction

$x_5 \leftrightarrow (x_1 \lor x_2) = (x_5 \rightarrow (x_1 \lor x_2)) \land ((x_1 \lor x_2) \rightarrow x_5) = (\neg x_5 \lor x_1 \lor x_2) \land (\neg (x_1 \lor x_2) \lor x_5)$
3-CNF-SAT = satisfiable 3-CNF formula, where each formula is a conjunction of clauses, where each clause is a disjunction of 3 literals, where each literal is a variable or a negated var.

1. 3-CNF-SAT ∈ NP because if \( \phi \in 3\text{-CNF-SAT} \), the certificate is the assignment making \( \phi \) true.

2. SAT ≤_P 3-CNF-SAT as follows.
   Map each operator output to a variable.
   Create a dummy input for each negation.
   Each operator has 2 inputs and 1 output.
   Use truth tables to map each op. to clauses.
   AND all the clauses.
Illustration for 3-CNF-SAT Reduction

\[ y_1 \leftrightarrow (y_2 \land \neg x_2) = \]
\[ (y_1 \rightarrow (y_2 \land \neg x_2)) \land ((y_2 \land \neg x_2) \rightarrow y_1) = \]
\[ (\neg y_1 \lor (y_2 \land \neg x_2)) \land (\neg(y_2 \land \neg x_2) \lor y_1) = \]
\[ (\neg y_1 \lor y_2) \land (\neg y_1 \lor \neg x_2) \land (\neg y_2 \lor x_2 \lor y_1) \]
The Sequence of Reductions

- Introduction
- The Complexity Class \( P \)
- The Complexity Class \( \text{NP} \)
- Polynomial Reducibility
- The Complexity Class \( \text{NPC} \)
- \( \text{NPC} \) Proofs
  - clique 1
  - clique 2
  - vertex cover 1
  - vertex cover 3
  - vertex cover 3
  - subset sum 1
  - subset sum 2
  - subset sum 3

Graph:
- CIRCUIT-SAT → SAT → 3-CNF-SAT
- CLIQUE → VERTEX-COVER → HAM-CYCLE → TSP
- SUBSET-SUM
A clique of an undirected graph is a subset of vertices with edges between all pairs.

\[ \text{CLIQUE} = \{ \langle G, k \rangle : G \text{ has a clique of size } k \} \]

1. \( \text{CLIQUE} \in \text{NP} \) because if \( x \in \text{CLIQUE} \), then the clique of \( k \) vertices is a certificate.

2. \( 3\text{-CNF-SAT} \leq \text{P CLIQUE} \) as follows.
   
   Map \( m \) clauses to \( 3m \) vertices, one per literal per clause.
   
   If a literal in one clause does not conflict with a literal in another clause, add an edge.
   
   Satisfying assignment iff clique of size \( m \).
3-CNF-SAT $\leq_P$ CLIQUE Illustration

Introduction

The Complexity Class $P$

The Complexity Class $NP$

Polynomial Reducibility

The Complexity Class $NPC$

NPC Proofs

NPC Proofs

clique 1

$\triangledown$ clique 2

vertex cover 1

vertex cover 2

vertex cover 3

subset sum 1

subset sum 2

subset sum 3

$\tilde{x}_1 \lor \tilde{x}_2 \lor \tilde{x}_3$

$x_1$

$x_2$

$x_3$

$\tilde{x}_1$

$\tilde{x}_2$

$\tilde{x}_3$

$\tilde{x}_1 \lor \tilde{x}_2 \lor \tilde{x}_3$
A vertex cover of an undirected graph is a subset of vertices that covers all the edges. If edge \((u, v)\), then either \(u\) or \(v\) (or both) are in vertex cover.

\[
\text{VERTEX-COVER} = \{\langle G, k \rangle : G \text{ has a vertex cover of size } k \}
\]

1. \(\text{VERTEX-COVER} \in \text{NP}\) because if \(x \in \text{V.-C.}\), then the vertex cover of size \(k\) is a certificate.

2. \(\text{CLIQUE} \leq_P \text{VERTEX-COVER}\).
   Map clique instance \(x = \langle G, k \rangle\) to vertex cover instance \(x' = \langle \bar{G}, |G.V| - k \rangle\).
   \(\bar{G}\) is the complement of \(G\).
   \(C\) is a clique of \(G\) iff \(G.V - C\) is a v.c. of \(\bar{G}\).
CLIQUE $\leq_{P}$ VERTEX-COVER Illustration

Introduction

The Complexity Class $P$

The Complexity Class $NP$

Polynomial Reducibility

The Complexity Class $NPC$

$NPC$ Proofs

$NPC$ Proofs

clique 1
clique 2
vertex cover 1
$\triangleright$ vertex cover 3
vertex cover 3
subset sum 1
subset sum 2
subset sum 3

$G$

$\overline{G}$
Proof of reduction:

If \( C \) is a clique of \( G \) of size \( k \), then \( \overline{G} \) has no edge in \( C \times C \), so \( G.V - C \) is a vertex cover of \( \overline{G} \) of size \( |G.V| - k \).

If \( G \) has no clique of size \( k \), then for any set of vertices \( C \) of size \( k \), \( C \) is not a clique, which implies \( \overline{G} \) has an edge in \( C \times C \), so \( G.V - C \) is not a vertex cover of \( \overline{G} \).
Subset set is the problem of finding a subset of numbers that sum to a given value.

\[
\text{SUBSET-SUM} = \{ \langle S, t \rangle : \text{A subset } S' \subseteq S \text{ satisfies } t = \sum_{s \in S'} s \}\]

1. \text{SUBSET-SUM} \in \text{NP} because if \( x \in \text{SUBSET-SUM} \), then the subset of numbers that sum to \( t \) is a certificate.
2. 3-SAT-CNFSAT $\leq_P$ SUBSET-SUM.

Map each literal to a decimal number with $v + c$ digits, where $v =$ number of variables and $c =$ number of clauses.

For $i \leq v$, digit $i = 1$ if literal is $x_i$ or $\neg x_i$.
Digit $v + j = 1$ if literal is in clause $j$.
All other digits are 0.

We want a satisfying assignment if and only if some subset sums to $111\ldots333\ldots$.

Need additional numbers to ensure clause columns can sum to 3.
3-CNF-SAT $\leq_P$ SUBSET-SUM Illustration

$$(x_1 \lor x_2 \lor x_3) \lor (\neg x_1 \lor x_2 \lor x_3) \lor (\neg x_1 \lor \neg x_2 \lor \neg x_3) \lor (x_1 \lor \neg x_2 \lor \neg x_3)$$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>0 0 1</th>
<th>1 0 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg x_1$</td>
<td>1 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0 1 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>$\neg x_2$</td>
<td>0 1 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0 0 1</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>$\neg x_3$</td>
<td>0 0 1</td>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>0 0 0</th>
<th>1 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s'_1$</td>
<td>0 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>0 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>$s'_2$</td>
<td>0 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>$s_3$</td>
<td>0 0 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>$s'_3$</td>
<td>0 0 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>$s_4$</td>
<td>0 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>$s'_4$</td>
<td>0 0 0</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

Use $t = 1113333$. 