NP-Completeness

Problems and Instances
Polynomial Time Verification
NP-completeness and Reducibility
NP-completeness Proofs
NP-complete Problems

Introduction
Introduction .............................................. 2
Examples of P and NPC Problems ..................... 3
Reductions ................................................ 4

The Complexity Class P
Problems and Instances ................................. 5
The P Complexity Class ................................. 6
Example P Problems ................................... 7

The Complexity Class NP
Hamiltonian Cycles ....................................... 8
Verification Algorithms .................................. 9
The NP Complexity Class .............................. 10

Polynomial Reducibility
Reducibility (Informal) ................................. 11
Reducibility (Formal) .................................... 12
Reducibility Examples ................................... 13
Reducibility Properties ................................. 14
Another Reducibility Example ......................... 15
And Another Reducibility Example ..................... 16

The Complexity Class NPC
Elements for Defining NPC ............................ 17
The Definition of NP-Complete ......................... 18

CIRCUIT-SAT ........................................... 19
NP-Completeness Proofs and SAT ..................... 20
Illustration for SAT Reduction ......................... 21
3-CNF-SAT ............................................ 22
Illustration for 3-CNFSAT Reduction ..................... 23

NPC Proofs
The Sequence of Reductions ............................ 24
CLIQUE .............................................. 25
3-CNF-SAT \(\leq_P\) CLIQUE Illustration ................. 26
VERTEX COVER ....................................... 27
CLIQUE \(\leq_P\) VERTEX-COVER Illustration .............. 28
CLIQUE \(\leq_P\) VERTEX-COVER Proof ..................... 29
SUBSET SUM ......................................... 30
3-CNF-SAT \(\leq_P\) SUBSET-SUM ......................... 31
3-CNF-SAT \(\leq_P\) SUBSET-SUM Illustration .............. 32
Introduction

Mostly, we have studied problems with 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 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.
- Finding a Euler tour (each edge once) is in P.
- Finding a hamiltonian cycle (each vertex once) is in NPC.
- Finding a solution to 2-CNF formulas is in P.
- Finding a solution to 3-CNF formulas is in NPC.

Technically, this problem is NP-hard.

Reductions

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 $\alpha$.

The Complexity Class P

Problems and Instances

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 = \{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.
**The P Complexity Class**

A problem $Q$ is 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 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** = \{ $(G, u, v, k)$ :  
- $G = (V, E)$ 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** = \{ $(S, m)$ :  
- $S$ is a set of integers, 
- $m$ is an integer, and 
- $m \leq$ the median of $S$\}

**The Complexity Class NP**

**Hamiltonian Cycles**

A *hamiltonian cycle* of an undirected graph is a simple cycle that contains all vertices.

$$\text{HAM-CYCLE} = \{(G) : G \text{ has a hamiltonian cycle}\}$$

**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.
The NP Complexity Class
NP is the class of problem with P verification algorithms with poly-size certificates.

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

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

HAM-CYCLE $\in$ NP because if $G \in$ HAM-CYCLE, then the certificate is the hamiltonian path $p$.

Consider:
$\text{SAT} = \{\langle \phi \rangle : \phi \text{ is a satisfiable boolean formula}\}$ $\text{COMPOSITE} = \{\langle n \rangle : n \text{ is a composite integer}\}$

Polynomial Reducibility

Reducibility (Informal)
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.

Reducibility (Formal)
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
$\text{EVEN} = \{\langle n \rangle : n \text{ is an even number}\}$
$\text{ODD} = \{\langle n \rangle : n \text{ is an odd number}\}$
$\text{EVEN} \leq_P \text{ODD}$ (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 \( L_1 \not\in P \), then \( L_2 \not\in P \).

Proof:
Let \( F \) be a \( O(n^k) \) reduction alg. for \( L_1 \leq_P L_2 \).
Let \( A_2 \) be a \( O(n^k) \) 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 \( O(n^k) \) implies \( A_2 \) is \( O(|x|^k) \) on an input of size \( O(|x|^k) \).
It follows that \( A_1 \in P \) because \( A_1 \) is \( O(n^k) \).

And Another Reducibility Example

HAM-CYCLE = \{\langle G \rangle : G has a hamiltonian cycle\}
HAMPATH = \{\langle G, u, v \rangle : there is a simple path from \( u \) to \( v \) including all vertices\}

HAM-CYCLE \( \leq_P \) HAMPATH by
\[
f(G) = \langle G', u, w \rangle \text{ where } G' \text{ is constructed by copying } G \text{ and duplicating any vertex } u.
\]
Now need to show \( x = \langle G \rangle \in \text{HAM-CYCLE} \) if and only if \( x' = \langle G, u, w \rangle \in \text{HAMPATH} \).
If \( x \in \text{HAM-CYCLE} \), then \( G' \) has a simple path from \( u \) : use \( G' \) ham. cycle,
but end with \( u' \).
If \( x' \in \text{HAMPATH} \), then \( G \) has a ham. cycle: start from \( u \), use \( G' \) simple path,
but end with \( u \).

The Complexity Class NPC

Recall NP and \( \leq_P \).
\( L \in \text{NP} \) iff there exists a P algorithm \( A(x, y) \) and a constant \( c \) such that:
\( L = \{x : \text{there exists a certificate } y \text{ such that } |y| = O(|x|^c), \text{ and } A(x, y) = \text{yes}\} \)
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 NP-hard if \( L' \leq_P L \) for every \( L' \in \text{NP} \).

\( L \in \text{NP} \) 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 NPC and P.

CIRCUIT-SAT = set of satisfiable boolean circuits

CIRCUIT-SAT \( \in \text{NPC} \) because it is in NP and all NP problems \( \leq_P \) CIRCUIT-SAT.

NP-Completeness Proofs and SAT

To show that \( L \) is NP-complete \( (L \in \text{NPC}) \)
1. Show \( L \in \text{NP} \).
2. Show \( L' \leq_P L \) for a known language \( L' \in \text{NPC} \).

SAT = set of satisfiable Boolean formulas

1. SAT \( \in \text{NP} \) because if \( \phi \in \text{SAT} \), 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

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 φ ∈ 3-CNF-SAT, the certificate is the assignment making φ 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 ↔ (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)

NPC Proofs

The Sequence of Reductions

CLIQUE

A clique of an undirected graph is a subset of vertices with edges between all pairs.

CLIQUE = \{⟨G, k⟩ : G has a clique of size k\}

1. CLIQUE ∈ NP because if x ∈ CLIQUE, then the clique of k vertices is a certificate.
2. 3-CNF-SAT ≤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.
**Verdict Cover**

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.

**Verdict-Cover**

\[
\{(G, k) : G \text{ has a vertex cover of size } k \}\]

1. **Verdict-Cover ∈ NP** because if \(x \in V\)-C.,
   then the vertex cover of size \(k\) is a certificate.

2. **Clique ≤p Verdict-Cover.**

   Map clique instance \(x = \langle G, k \rangle\) to
   vertex cover instance \(x' = \langle \overline{G}, |G.V| - k \rangle\).
   
   \(G\) is the complement of \(G\).
   
   \(C\) is a clique of \(G\) iff \(G.V - C\) is a v.c. of \(\overline{G}\).
**SUBSET SUM**

Subset set is the problem of finding a subset of numbers that sum to a given value.

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

1. **SUBSET-SUM \(\in\) NP** because
   
   if \(x \in\) SUBSET-SUM,
   then the subset of numbers that sum to \(t\)
   is a certificate.

---

**3-CNF-SAT \(\leq_p\) SUBSET-SUM Illustration**

\[
\begin{align*}
(x_1 \lor x_2 \lor x_3) \lor (\neg x_1 \lor x_2 \lor x_3) \lor \\
(\neg x_1 \lor \neg x_2 \lor \neg x_3) \lor (x_1 \lor \neg x_2 \lor \neg x_3)
\end{align*}
\]

\[
\begin{array}{cccccccccc}
\hline
& x_1 & x_2 & x_3 & \neg x_1 & \neg x_2 & \neg x_3 \\
\hline
s_1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
s_1' & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
s_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
s_2' & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
s_3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
s_3' & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
s_4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
s_4' & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
\]

Use \(t = 1113333\).

---

**3-CNF-SAT \(\leq_p\) SUBSET-SUM**

2. **3-SAT-CNF \(\leq_p\) 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…333…

   Need additional numbers to ensure clause
   columns can sum to 3.