NP-Completeness (cont)

An algorithm accepts $e(i)$ if it answers YES
An algorithm rejects $e(i)$ if it answers NO

An algorithm accepts $L$ in polynomial time if
$\exists c$ such that the algorithm accepts $\forall e(i) \in L$ with $|e(i)| = n$ within time $O(n^c)$ and does not accept any $e(i) \notin L$

An algorithm decides $L$ in polynomial time if
$\exists c$ such that the algorithm accepts all $e(i) \in L$ within time $O(n^c)$ and rejects all $e(i) \notin L$ within time $O(n^c)$

NP-Completeness (cont)

Proof: (cont) Now suppose $\exists$ an algorithm that accepts $L$ in polynomial time, $A$.

Then $\exists c$ such that $A$ accepts $L$ in time $T_A(n) = O(n^c)$.

Modify $A$ by adding a timer which expires after time $T_A(n)$ and make $A$ reject when the timer expires if it hasn’t already accepted by then.

This new algorithm decides $L$ in time $T_A(n) + 1 = O(n^c)$.

So $\exists$ an algorithm that decides $L$ in polynomial time.

QED

NP-Completeness (cont)

$P = \{\text{polynomially solvable concrete decision problems}\}$

$P = \{L \mid \exists \text{an algorithm that decides } L \text{ in polynomial time}\}$

Theorem

$P = \{L \mid \exists \text{an algorithm that accepts } L \text{ in polynomial time}\}$

Proof: If $L \in P$ then $\exists \text{an algorithm that decides } L \text{ in polynomial time}.$

This algorithm also accepts $L$ in polynomial time.

NP-Completeness (cont)

PCP - a language which cannot be decided (undecidable)

Instance: $n$ pairs of binary strings: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

Question: Is there a sequence of indices $i_1, i_2, \ldots, i_m$

such that $x_{i_1} x_{i_2} \cdots x_{i_m} = y_{i_1} y_{i_2} \cdots y_{i_m}$?

Example: $(111, 1), (110, 11), (0, 000), (101, 011111), (010, 101) \ldots$
NP-Completeness (cont)

The class NP (non-deterministic polynomial time)

Traditional model for non-deterministic computation:
Non-deterministic Turing Machine (choice of moves)

Text (CLR): Certificates and verification algorithms

An algorithm $A(x,y)$, verifies $x$ if there exists a $y$ such that $A(x,y) = 1$.

$L = \{ x \mid \exists y \ A(x,y) = 1 \}$ is the language verified by $A(x,y)$

NP-Completeness (cont)

$3\text{SAT} \in \text{NP}$

Input: $\langle \phi, \bar{x} \rangle$ a formula and a truth assignment.
Output: $\phi(\bar{x})$, the formula’s value for the given truth assignment.

$A(\phi, \bar{x})$

for each clause of $\phi$

flag$\leftarrow 0$

for each literal in the clause

if this literal is 1 under $\bar{x}$ then flag$\leftarrow 1$
endfor

if flag=0 then return(0)
endfor

return(1)

NP-Completeness (cont)

$\text{CLIQUE} \in \text{NP}$

Input: $\langle G,k,V' \rangle$ a graph, an integer and a subset of vertices,

Output: 1 if $V'$ is a clique of size $\geq k$ in $G$, and 0 otherwise.

$A(G,k,V')$

if $m<k$ then return(0)
for $i \leftarrow 1$ to $m-1$
for $j \leftarrow i+1$ to $m$

if $(v_i,v_j) \notin E$ then return(0)
endfor
endfor
return(1)
NP-Completeness (cont)

P \subseteq NP
What can be decided in polynomial time, can also be verified in polynomial time.

\text{co – NP} = \{ \overline{L} \mid L \in \text{NP} \}

\text{3SAT} = \{ \phi \mid \phi \text{ is a formula which is not satisfiable} \}
(Assume any binary string which is not a legal encoding of a formula represents the formula \( F \).)

\text{3SAT} \in \text{co – NP}

COMPOSITE = \{ x \mid x \text{ is an integer and not prime} \}
\text{COMPOSITE} \in \text{NP}
(The certificate is a non-trivial factor of \( x \).)

\text{PRIMES} = \text{COMPOSITE} \in \text{co-NP}
But it can also be shown that \text{PRIMES} \in \text{NP}

So \text{PRIMES} \in \text{co-NP} \cap \text{NP}

NP-Completeness (cont)

\text{co – P} = \{ \overline{L} \mid L \in \text{P} \} = \text{P}
If \( P = \text{NP} \), then \text{co-NP} = \text{co-P} = \text{P} = \text{NP}.
If \text{co-NP} \neq \text{NP} \), then \( P \neq \text{NP} \).
4 open possibilities

NP = \text{co-NP} = \text{P}
NP = \text{co-NP}
NP = \text{co-NP}
NP = \text{co-NP}

If \( L \) is \text{NP-complete} if

1) \( L \in \text{NP} \)
2) for all \( L' \in \text{NP} \), \( L' \leq_p L \) \quad \text{(NP-hard)}

If \( L \) is \text{NP-complete}, then \( L \) is “no easier” than anything in \text{NP}.
If \( L \) is \text{NP-complete} and \( L \in \text{P} \) then \( \text{P} = \text{NP} \),

If \( L \) is \text{NP-complete} and \( P \neq \text{NP} \), then \( L \notin \text{P} \).
NP-Completeness (cont)

How to show a problem is NP-complete?

**Cook’s Theorem** \(3\text{SAT}\) is NP-complete.

**Proof:** Already showed \(3\text{SAT} \in \text{NP}\). For second requirement take a non-deterministic Turing Machine which accepts \(L’\) in polynomial time and any input \(x\), and convert it to a 3CNF formula which is satisfiable if and only if the Turing Machine would accept \(x\) within the polynomial time bound.

---

NP-Completeness (cont)

Text: **CIRCUIT-SAT** is NP-complete.

Proof sketch: Show that a verification algorithm and any input \(x\) can be converted it to a boolean circuit which is satisfiable if and only if the verification algorithm would verify \(x\) within a polynomial time bound.

The first proof that a problem is NP-hard was hard.

Subsequent proofs can use the following lemma.

**Lemma**

If \(L’ \leq_p L\) and \(L’\) is NP-hard, then \(L\) is NP-hard.

---

NP-Completeness (cont)

**Lemma**

If \(L’ \leq_p L\) and \(L’\) is NP-hard, then \(L\) is NP-hard.

**Proof:** \(\forall L’ \in \text{NP}, L’ \leq_p L’\) since \(L’\) is NP-hard.

Then \(L’ \leq_p L’ \leq_p L\), so \(L’ \leq_p L\).

So \(L\) is NP-hard.

**QED**

**Corollary**

If \(L \in \text{NP} \) and \(L’ \leq_p L\) and \(L’\) is NP-hard, then \(L\) is NP-complete.

---

NP-Completeness (cont)

To show a problem \(Q\) is NP-complete:

1) show the problem is in \(\text{NP}\).
   
   Come up with a polynomial verification algorithm.

2) show the problem is NP-hard.
   
   Take a known NP-hard problem and reduce it to \(Q\).
NP-Completeness (cont)

3SAT, CLIQUE, VERTEX-COVER, and SUBSET-SUM are NP-complete.

Proof: Already showed 3SAT and CLIQUE are in NP.

VERTEX-COVER is in NP.

\[ A(G,k,V') \quad V' = \{v_1, v_2, \ldots, v_m\} \]

if \( m > k \) then return(0)
for each edge \( e = (v_i,v_j) \in E \)
  if \( v_i \notin V' \) and \( v_j \notin V' \) then return(0)
endfor
return(1)

NP-Completeness (cont)

Proof: (cont)

Using previous reductions,

since CIRCUIT - SAT \( \leq_p \) 3SAT. 3SAT is NP-hard.

since 3SAT \( \leq_p \) CLIQUE. CLIQUE is NP-hard.

since CLIQUE \( \leq_p \) VERTEX - COVER

VERTEX-COVER is NP-hard.

since VERTEX - COVER \( \leq_p \) SUBSET - SUM

SUBSET-SUM is NP-hard.

QED

If anyone of these problems is in \( P \), then \( P = NP \).

0-1 KNAPSACK

Instance: A \( n \) items where item \( i \) has weight \( w_i \) and cost \( c_i \) and two integers \( W \) and \( C \).

Question: Is there a subset \( J \) of \( \{1, 2, \ldots, n\} \) such that

\[ \sum_{j \in J} w_j \leq W \quad \text{and} \quad \sum_{j \in J} c_j \geq C \]?

0-1 KNAPSACK is NP-complete.
NP-Completeness (cont)

**0-1 KNAPSACK** is NP-complete.

**Proof:** First show that **0-1 KNAPSACK** is in **NP**.

\[
A(W[], C[], W, C, J)
\]

\[
A \leftarrow 0 \quad B \leftarrow 0
\]

for each \( j \) in \( J \)

\[
A \leftarrow A + W[j]
\]

\[
B \leftarrow B + C[j]
\]

endfor

if \( A > W \) or \( B < C \) then return(0)
return(1)

---

NP-Completeness (cont)

**Proof:** (cont) Now show that **0-1 KNAPSACK** is NP-hard.

**SUBSET-SUM** is a known NP-hard problem.

**SUBSET-SUM \( \leq_p 0 \text{-}1 \text{ KNAPSACK}**

Step 1: Construct \( f : \{(S,t)\} \rightarrow \{(W[], C[], W, C)\} \)

Given \((S,t)\) where \( S = \{s_1, s_2, \ldots, s_k\} \),

let \( n = k \), \( w_i = c_i = s_i \), and \( W = C = t \).