NP-Completeness
Running Time Revisited

Input size, $n$
- To be exact, let $n$ denote the number of bits in a nonunary encoding of the input.

All the polynomial-time algorithms studied so far in this course run in polynomial time using this definition of input size.
- Exception: any pseudo-polynomial time algorithm
Dealing with Hard Problems

What to do when we find a problem that looks hard...

I couldn’t find a polynomial-time algorithm; I guess I’m too dumb.

(cartoon inspired by [Garey-Johnson, 79])
Dealing with Hard Problems

Sometimes we can prove a strong lower bound... (but not usually)

I couldn’t find a polynomial-time algorithm, because no such algorithm exists!
Dealing with Hard Problems

- NP-completeness let’s us show collectively that a problem is hard.

I couldn’t find a polynomial-time algorithm, but neither could all these other smart people.

(cartoon inspired by [Garey-Johnson, 79])
To simplify the notion of “hardness,” we will focus on the following:

- Polynomial-time as the cut-off for efficiency
- Decision problems: output is 1 or 0 (“yes” or “no”)
- Examples:
  - Does a given graph G have a Hamiltonian Euler tour?
  - Does a text T contain a pattern P?
  - Does an instance of 0/1 Knapsack have a solution with benefit at least K?
  - Does a graph G have an MST with weight at most K?
A language $L$ is a set of strings defined over some alphabet $\Sigma$.

Every decision algorithm $A$ defines a language $L$:
- $L$ is the set consisting of every string $x$ such that $A$ outputs “yes” on input $x$.
- We say “$A$ accepts $x$” in this case.

Example:
- If $A$ determines whether or not a given graph $G$ has a cycle, then the language $L$ for $A$ is all graphs with at least a cycle.
The Complexity Class P

- A language represents a decision problem.
- A **complexity class** is a collection of languages.
- P is the complexity class consisting of all languages that are accepted by **polynomial-time** algorithms.
- For each language L in P there is a polynomial-time decision algorithm A for L.
  - If n=|x|, for x in L, then A runs in \(O(n^c)\) time on input x for some constant c.
The Complexity Class NP

We say that an algorithm is non-deterministic if it uses the following operation:

- Choose(n): non-deterministically chooses a value k, 0 ≤ k < n.
- Choose(n) looks like random(n), but it may make wise choices.
- Can be used to choose an entire string y (with |y| choices)

We say that a non-deterministic algorithm A accepts a string x if there exists some sequence of choose operations that causes A to output “yes” on input x.

NP is the complexity class consisting of all languages accepted by polynomial-time non-deterministic algorithms.
NP example

Problem: Decide if TSP has a tour bounded by $K$

**Algorithm getTSP(V, E)**

// Non-deterministically choose a set $T$ of $n-1$ edges:

$T = \{ \}$;

while ($|T| < n-1$) {
  $k = \text{choose}(|E|)$;
  $e_k$ from $E$ to $T$; }

Test that $T$ forms a tour
Test that $T$ has weight at most $K$

Analysis: Testing takes $O(n+m)$ time, so this algorithm runs in polynomial time.
The Complexity Class NP
Alternate Definition

We say that an algorithm B verifies the acceptance of a language \( L \) if and only if, for any \( x \) in \( L \), there exists a certificate \( y \) such that B outputs “yes” on input \((x,y)\).

**Theorem**: NP is the complexity class consisting of all languages verified by **polynomial-time** algorithms.

- We know: P is a subset of NP.
- Major open question: \( P = \text{NP} \)?
- Most researchers believe that P and NP are different.
NP example (2)

Problem: Decide if a graph has an MST of weight K

Verification Algorithm:
1. Use as a certificate, y, a set T of n-1 edges
2. Test that T forms a spanning tree
3. Test that T has weight at most K

Analysis: Verification takes $O(n+m)$ time, so this algorithm runs in polynomial time.
Suppose A is a non-deterministic algorithm
- Let y be a certificate consisting of all the outcomes of the choose steps that A uses.
- We can create a verification algorithm that uses y instead of A’s choose steps
- If A accepts on x, then there is a certificate y that allows us to verify this (namely, the choose steps A made)
- If A runs in polynomial-time, so does this verification algorithm

Suppose B is a verification algorithm
- Non-deterministically choose a certificate y
- Run B on y
- If B runs in polynomial-time, so does this non-deterministic algorithm
Polynomial-Time Reduction

Suppose we could solve $Y$ in polynomial-time. What else could we solve in polynomial time?

Reduction: Problem $X$ polynomial reduces to problem $Y$ if arbitrary instances of problem $X$ can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to the blackbox that solves problem $Y$.

Notation. $X \leq_p Y$. 

Polynomial-Time Reduction

If $X \leq_p Y$, and assume the code of $Y$ is $B$, we may obtain the code $A$ which uses $B$ to solve $X$ (the time spent by $B$ is not cared).

- **Design algorithms.** If $X \leq_p Y$ and $Y$ can be solved in polynomial-time, then $X$ can also be solved in polynomial time. That is, if $Y$ is easy, so is $X$.
- **Establish equivalence.** If $X \leq_p Y$ and $Y \leq_p X$, we use notation $X \equiv_p Y$.
- **Prove Hardness of $Y$:** If $X \leq_p Y$ and $X$ is known to be hard, then $Y$ must be hard as well.
Independent Set

INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in $S$?

Ex. Is there an independent set of size $\geq 6$? Yes.
Ex. Is there an independent set of size $\geq 7$? No.
Clique

Clique: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each pair $(x, y)$ of points in $S$, $(x, y)$ is an edge of $E$?

Claim. $\text{CLIQUE} \equiv_p \text{INDEPENDENT-SET}$.

Proof. We show $S$ is an independent set of $G$ iff $S$ is a clique of $G'$, where $G'$ is the complement of $G$: $G' = (V, V^2 - E)$. 
**Vertex Cover**

- **VERTEX COVER:** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in $S$?

- Ex. Is there a vertex cover of size $\leq 4$? Yes.
- Ex. Is there a vertex cover of size $\leq 3$? No.
Claim. VERTEX-COVER $\equiv_p$ INDEPENDENT-SET.

Proof. We show $S$ is an independent set iff $V - S$ is a vertex cover.
Consequently, $S$ is a maximum independent set iff $V - S$ is a minimum vertex cover. If we have an efficient algorithm to solve one, we will have efficient algorithm to solve the other.
Claim. \( \text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET} \).

Proof. We show \( S \) is an independent set iff \( V - S \) is a vertex cover.

\[ \implies \]
- Let \( S \) be any independent set.
- Consider an arbitrary edge \((u, v)\).
- \( S \) independent \( \implies u \notin S \) or \( v \notin S \) \( \implies u \in V - S \) or \( v \in V - S \).
- Thus, \( V - S \) covers \((u, v)\).

\[ \iff \]
- Let \( V - S \) be any vertex cover.
- Consider two nodes \( u \in S \) and \( v \in S \).
- Observe that \((u, v) \notin E\) since \( V - S \) is a vertex cover.
- Thus, no two nodes in \( S \) are joined by an edge \( \implies S \) independent set. \( \blacksquare \)
Set Cover

SET COVER: Given a set $U$ of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$, does there exist a collection of $\leq k$ of these sets whose union is equal to $U$?

Sample application.

- $m$ available pieces of software.
- Set $U$ of $n$ capabilities that we would like our system to have.
- The $i^{th}$ piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all $n$ capabilities using fewest pieces of software.

Ex:

$$U = \{1, 2, 3, 4, 5, 6, 7\}$$

$$k = 2$$

$S_1 = \{3, 7\}$  $S_4 = \{2, 4\}$

$S_2 = \{3, 4, 5, 6\}$  $S_5 = \{5\}$

$S_3 = \{1\}$  $S_6 = \{1, 2, 6, 7\}$
Vertex Cover Reduces to Set Cover

Claim. $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$.

Proof. Given a VERTEX-COVER instance $G = (V, E)$, $k$, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.

- Create SET-COVER instance:
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
  - Set-cover of size $\leq k$ iff vertex cover of size $\leq k$.

---

**VERTEX COVER**

- $k = 2$
  - $V = \{a, b, c, d, e, f\}$
  - $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

**SET COVER**

- $U = \{1, 2, 3, 4, 5, 6, 7\}$
- $k = 2$
- $S_a = \{3, 7\}$
- $S_b = \{2, 4\}$
- $S_c = \{3, 4, 5, 6\}$
- $S_d = \{5\}$
- $S_e = \{1\}$
- $S_f = \{1, 2, 6, 7\}$
For the clique problem, suppose we have an algorithm $A$ to answer the decision problem: $A(G, k) = \text{yes}$ iff $G$ has a clique of size $k$. How can we find the maximum clique of $G$?

**Step 1:** Decide $k$, the size of the maximum clique:
for $i$ from $n$ downto $1$, if $A(G, i) = \text{yes}$ and $A(G, i+1) = \text{no}$, then return $i$;

**Step 2:** Decide the actual max clique:
for each vertex $v$ in $G$
if $(A(G - v, k) = \text{yes}) G = G - v$;
// $G - v$ means $v$ is deleted from $G$.

Suppose we have an algorithm $B$ which returns the maximum clique of $G$. How can we solve the decision version of the clique problem?
Decision vs Optimization: Vertex Cover

**Decision Problem:**
- Input: a graph $G = (V, E)$, integer $k$.
- Question: Does $G$ have a vertex cover of size $\leq k$?

**Optimization problem.** Find vertex cover of minimum cardinality of $G$.

**Self-reducibility:** Decision and Optimization Problems are all equivalent ($=_{P}$)

Applies to all (NP-complete) problems.
- Justifies our focus on decision problems.
Vertex Cover Problem

Ex: Reduce Optimization to Decision

To find min cardinality vertex cover.

- (Binary) search for cardinality $k^*$ of min vertex cover.
- Find a vertex $v$ such that $G - v$ has a vertex cover of size $\leq k^* - 1$.
  - any vertex in any min vertex cover will have this property
- Include $v$ in the vertex cover.
- Recursively find a min vertex cover in $G - v$. 

\[ \text{delete } v \text{ and all incident edges} \]
An Interesting Problem

A Boolean circuit is a circuit of AND, OR, and NOT gates; the CIRCUIT-SAT problem is to determine if there is an assignment of 0’s and 1’s to a circuit’s inputs so that the circuit outputs 1.
CIRCUIT-SAT is in NP

Non-deterministically choose a set of Boolean values for all inputs of the circuit, then test each gate’s I/O.
NP-Completeness

- A problem (language) $L$ is **NP-hard** if every problem in NP can be reduced to $L$ in polynomial time.
- That is, for each language $M$ in NP, we can take an input $x$ for $M$, **transform** it in polynomial time to an input $x'$ for $L$ such that $x$ is in $M$ if and only if $x'$ is in $L$.
- $L$ is **NP-complete** if it’s in NP and is NP-hard.
Cook-Levin Theorem

CIRCUIT-SAT is NP-complete.

- We already showed it is in NP.

To prove it is NP-hard, we have to show that every language in NP can be reduced to it.

- Let M be in NP, and let x be an input for M.
- Let y be a certificate that allows us to verify membership in M in polynomial time, \( p(n) \), by some algorithm D.
- Let S be a circuit of size at most \( O(n^{2c}) \) that simulates a computer (details omitted...)

\[ \text{NP} \xrightarrow{\text{poly-time}} \text{CIRCUIT-SAT} \]
Cook-Levin Proof

We can build a circuit that simulates the verification of $x$’s membership in $M$ using $y$.

- Let $W$ be the working storage for $D$ (including registers, such as program counter); let $D$ be given in RAM “machine code.”
- Simulate $O(n^c) = p(n)$ steps of $D$ by replicating circuit $S$ for each step of $D$. Only input: $y$.
- Circuit is satisfiable if and only if $x$ is accepted by $D$ with some certificate $y$.
- Total size is still polynomial: $O(n^{3c})$. 
Some Thoughts about P and NP

- **Belief**: P is a proper subset of NP.
- **Implication**: the NP-complete problems are the hardest in NP.
- **Why**: Because if we could solve an NP-complete problem in polynomial time, we could solve every problem in NP in polynomial time.
- That is, if an NP-complete problem is solvable in polynomial time, then P=NP.
- Since so many people have attempted without success to find polynomial-time solutions to NP-complete problems, showing your problem is NP-complete is equivalent to showing that a lot of smart people have worked on your problem and found no polynomial-time algorithm.
Problem Reduction

A language $M$ is polynomial-time reducible to a language $L$ if an instance $x$ for $M$ can be transformed in polynomial time to an instance $x'$ for $L$ such that $x$ is in $M$ if and only if $x'$ is in $L$.

- Denote this by $M \rightarrow L$.

A problem (language) $L$ is **NP-hard** if every problem in NP is polynomial-time reducible to $L$.

A problem (language) is **NP-complete** if it is in NP and it is NP-hard.

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

- CIRCUIT-SAT is in NP
- For every $M$ in NP, $M \rightarrow$ CIRCUIT-SAT.
Transitivity of Reducibility

If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

- An input $x$ for $A$ can be converted to $y$ for $B$, such that $x$ is in $A$ if and only if $y$ is in $B$. Likewise, for $B$ to $C$.
- Convert $y$ into $z$ for $C$ such that $y$ is in $B$ iff $z$ is in $C$.
- Hence, if $x$ is in $A$, $y$ is in $B$, and $z$ is in $C$.
- Likewise, if $z$ is in $C$, $y$ is in $B$, and $x$ is in $A$.
- Thus, $A \rightarrow C$, since polynomials are closed under composition.
A CNF Boolean formula is a conjunction (AND) of clauses; a clause is a disjunction (OR) of literals; a literal is a variable or negation of a variable:

- \( (a+b+\neg d+e)(\neg a+\neg c)(\neg b+c+d+e)(a+\neg c+\neg e) \)
- OR: +, AND: (times), NOT: \( \neg \)

**SAT**: Given a Boolean formula \( S \), is \( S \) satisfiable, that is, can we assign 0’s and 1’s to the variables so that \( S \) is 1 ("true")?

- Easy to see that SAT is in NP:
  - Non-deterministically choose an assignment of 0’s and 1’s to the variables and then evaluate each clause. If they are all 1 ("true"), then the formula is satisfiable.
SAT is NP-complete

Reduce CIRCUIT-SAT to SAT.

- Given a Boolean circuit, make a variable for every input and gate.
- Create a sub-formula for each gate, characterizing its effect. Form the formula as the output variable AND-ed with all these sub-formulas:

   - Example: \( m((a+b)\leftrightarrow e)(c\leftrightarrow \neg f)(d\leftrightarrow \neg g)(e\leftrightarrow \neg h)(ef\leftrightarrow i) \ldots \)

The formula is satisfiable if and only if the Boolean circuit is satisfiable.
**3SAT**

- The SAT problem is still NP-complete even if the formula is a conjunction of disjuncts, that is, it is in conjunctive normal form (CNF).
- The SAT problem is still NP-complete even if it is in CNF and every clause has just 3 literals (a variable or its negation):
  - \((a+b+\neg d)(\neg a+\neg c+e)(\neg b+d+e)(a+\neg c+\neg e)\)
- Reduction from SAT (See § 13.3.1).
Vertex Cover

A vertex cover of graph $G=(V,E)$ is a subset $W$ of $V$, such that, for every edge $(a,b)$ in $E$, $a$ is in $W$ or $b$ is in $W$.

VERTEX-COVER: Given a graph $G$ and an integer $K$, is $G$ have a vertex cover of size at most $K$?

VERTEX-COVER is in NP: Non-deterministically choose a subset $W$ of size $K$ and check that every edge is covered by $W$. 
Vertex-Cover is NP-complete

Reduce 3SAT to VERTEX-COVER.

A CNF is a conjunction of m clauses.

Let $S$ be a Boolean formula in CNF with each clause having 3 literals (i.e., variables or negation of variables).

For each variable $x$, create a node for $x$ and $\neg x$, and connect these two:

For each clause $(a+b+c)$, create a triangle and connect these three nodes.
Vertex-Cover is NP-complete

Completing the construction

- Connect each literal in a clause triangle to its copy in a variable pair.
- E.g., a clause $(x+y+z)$

Let $n=$ # of variables
Let $m=$ # of clauses
Set $K = n + 2m$
Vertex-Cover is NP-complete

Example: \((a+b+c)(\neg a+b+\neg c)(\neg b+\neg c+\neg d)\)

Graph has vertex cover of size \(K=4+6=10\) iff formula is satisfiable.
Clique

A **clique** of a graph $G=(V,E)$ is a subgraph $C$ that is fully-connected (every pair in $C$ has an edge).

**CLIQUE**: Given a graph $G$ and an integer $K$, is there a clique in $G$ of size at least $K$?

**CLIQUE** is in NP: non-deterministically choose a subset $C$ of size $K$ and check that every pair in $C$ has an edge in $G$.

This graph has a clique of size 5

![Graph with a clique of size 5]
CLIQUE is NP-Complete

- Reduction from VERTEX-COVER.
- A graph $G$ has a vertex cover of size $K$ if and only if it’s complement has a clique of size $n-K$. 
Some Other NP-Complete Problems

- **SET-COVER**: Given a collection of m sets, are there K of these sets whose union is the same as the whole collection of m sets?
  - NP-complete by reduction from VERTEX-COVER

- **SUBSET-SUM**: Given a set of integers and a distinguished integer K, is there a subset of the integers that sums to K?
  - NP-complete by reduction from VERTEX-COVER
Some Other NP-Complete Problems

- **0/1 Knapsack**: Given a collection of items with weights and benefits, is there a subset of weight at most W and benefit at least K?
  - NP-complete by reduction from SUBSET-SUM

- **Hamiltonian-Cycle**: Given a graph G, is there a cycle in G that visits each vertex exactly once?
  - NP-complete by reduction from VERTEX-COVER

- **Traveling Salesperson Tour**: Given a complete weighted graph G, is there a cycle that visits each vertex and has total cost at most K?
  - NP-complete by reduction from Hamiltonian-Cycle.