Section 9: NP-completeness

This section reviews the concepts of P, NP, NP-completeness, and reduction.

First, let's review some definitions:

- **Problem**: a set of inputs and the correct outputs
- **Instance**: a single input to a problem
- **Decision problem**: a problem where the output is "yes" or "no"
- **Reduction**: We write $A \leq_p B$, and read "A reduces to B", "A is not harder than B", or "Solve A using B".

  Formally, $A \leq_p B$ if there is an algorithm that solves A using polynomially many calls to a solver for B, running in polynomial time (excluding calls to B). (In this class, such algorithms will almost always use just one call, but generally, it is allowed to use many calls.)
- **P**: ("polynomial") The set of decision problems A that can be solved in polynomial time.
- **NP**: ("nondeterminisitic polynomial") The set of decision problems A for which YES-instances can be verified in polynomial time.

  Formally, there is a polynomial time algorithm VerifyA such that for all inputs x,
  - If x is a YES-instance to A, then there exists a polynomial length string y such that VerifyA$(x, y) =$ YES.
  - If x is a NO-instance to A, then for all polynomial length strings y, VerifyA$(x, y) =$ NO.
- **NP-hard**: A problem B is NP-hard if $A \leq_p B$ for all A in NP.
- **NP-complete**: A problem B is NP-complete if B is in NP and B is NP-hard.
- **Boolean literal**: A Boolean variable x_i or its negation $\neg x_i$
- **Clause**: OR of zero or more literals
- **CNF formula**: AND of zero or more clauses
- **3SAT problem**:

  Input: A CNF formula with exactly 3 literals per clause
  Output: Is there an assignment to the variables that makes the formula true?

  3SAT is a fundamental NP-complete problem.

1. SATisfy This

Determine whether each instance of 3SAT is satisfiable. If it is, list a satisfying variable assignment.

(a) $(\neg a \vee \neg b \vee c) \wedge (a \vee c \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee b \vee c) \wedge (\neg b \vee c \vee \neg d)$

(b) $(\neg a \vee b \vee d) \wedge (\neg b \vee c \vee d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$

2. 5SAT

To prove NP-completeness, we recommend following this 7-step pattern:

First, to show B is in NP:

1. State what the certificate is.
2. Say why the certificate can be checked in polynomial time.

To show B is NP-hard:

3. Identify an NP-hard problem A and say, "We will reduce from A to B".
4. Define a reduction function f, which converts instances of A into instances of B.
5. Say why f is computable in polynomial time.
6. Show that "x is a YES-instance for A" $\implies$ "$f(x)$ is a YES-instance for B".
   - Do this by converting a certificate for x into a certificate for $f(x)$.
7. Show that "$f(x)$ is a YES-instance for B" $\implies$ "x is a YES-instance for A".
   - Do this by converting a certificate for $f(x)$ into a certificate for x.

Let's give it a try.

Define the problem 5SAT to be:

Input: A CNF formula with exactly 5 literals per clause
Output: Is there an assignment to the variables that makes the formula true?

We will show that 5SAT is NP-complete.

First, we will show that 5SAT is in NP.

(a) State what the certificate is.

(b) Say why the certificate can be checked in polynomial time.

Recall 3SAT:

Input: A CNF formula with exactly 3 literals per clause
Output: Is there an assignment to the variables that makes the formula true?

We will now prove that 5SAT is NP-hard with a reduction involving 3SAT.

(c) Fill in the blank: "We will reduce from ___ to ___". Which is A, and which is B?

(d) Define a reduction function f, which converts instances of A into B.

(e) Say why f is computable in polynomial time.

(f) Show that "x is a YES-instance for A" $\implies$ "$f(x)$ is a YES-instance for B".
(Remember: convert a certificate for x into a certificate for $f(x)$!)

(g) Show that "$f(x)$ is a YES-instance for B" $\implies$ "x is a YES-instance for A".
(Remember: convert a certificate for $f(x)$ into a certificate for x!)

3. Reduction with different types

The Integer Linear Programming problem (ILP) is:

Input: An integer matrix A and integer vector b
Output: Is there an integer vector x such that $Ax \leq b$?

In lecture, you saw that 3SAT $\leq_p$ ILP via a long series of reductions. Prove this directly using a single reduction. We will skip proving that ILP is in NP.

(c) Fill in the blank: "We will reduce from ___ to ___". Which is A, and which is B?

(d) Define a reduction function f, which converts instances of A into B.

(e) Say why f is computable in polynomial time.

(f) Show that "x is a YES-instance for A" $\implies$ "$f(x)$ is a YES-instance for B".
(Remember: convert a certificate for x into a certificate for $f(x)$!)

(g) Show that "$f(x)$ is a YES-instance for B" $\implies$ "x is a YES-instance for A".

(Remember: convert a certificate for $f(x)$ into a certificate for x!)

The following problems will not be covered in section, but may be useful to think about.
We recommend trying them by yourself first. Solutions will be posted in the evening.

4. Reduce to decision

NP is a set of decision (yes/no) problems, but in practice we're often interested in optimization problems (instead of "is there a vertex cover of size k?" we usually want to "find the smallest vertex cover"). **Usually**, this isn't a problem, though; we'll see an example in this problem.

Let $\mathsf{VC_D}$ be the problem: Given a graph G and an integer k, return `true` if and only if G has a vertex cover of size k.
Let $\mathsf{VC_O}$ be the problem: Given a graph G, return a list containing the vertices in a minimum size vertex cover.

(a) Show that $\mathsf{VC_D} \leq_P \mathsf{VC_O}$ (this is the easy direction).

(b) We'll now start working on the other reduction. Imagine someone came to you and said "See this vertex u, I promise it is in a minimum vertex cover." Use this promise to solve $\mathsf{VC_O}$ on a graph of size $n-1$ instead of n.

(c) Now imagine the same person said "See this vertex v, I promise it is **not** in any minimum vertex cover." Use this promise to solve $\mathsf{VC_O}$ on a graph of size at most $n-1$ instead of n.

(d) Use the ideas from the last two parts to show $\mathsf{VC_O} \leq_P \mathsf{VC_D}$.

5. Another Reduction

Consider an undirected graph G, where each vertex has a non-negative integer number of pebbles. A single *pebbling move* consists of removing two pebbles from a vertex and adding one pebble to an adjacent vertex, where we can choose which adjacent vertex. A pebbling move can only be done on a vertex that already has at least two pebbles, and it will always decrease the total number of pebbles in the graph by exactly one. Our goal is to remove as many pebbles as we can. Observe that at best, we'll have at least one pebble remaining in the graph.

Define the PEBBLE problem as the following problem:

Input: An undirected graph and the number of pebbles at each vertex
Output: `true` if there is a sequence of pebbling moves that leaves exactly one pebble in the graph, `false` otherwise.

Define the Hamiltonian Path Problem as the following problem:

Input: An undirected graph.
Output: `true` if there exists a path in the graph visiting every vertex exactly once, `false` otherwise.

Given that the Hamiltonian Path Problem is NP-complete, show that PEBBLE is as well. You may assume that the total number of pebbles in a graph is polynomial in terms of the size of the graph.

Hint: A single pebbling move can be represented as an ordered pair of vertices (u, v) where we take two pebbles from u and place one pebble in its neighbor v. A sequence of pebbling moves can be represented by a sequence of these pairs. Is there any way we can order these pairs nicely?

Follow the standard steps. To show that PEBBLE is in NP,

(a) State what the certificate is.

(b) Say why the certificate can be checked in polynomial time.

Now, to show that PEBBLE is NP-hard,

(c) Fill in the blank: "We will reduce from ___ to ___". Which is A, and which is B?

(d) Define a reduction function f, which converts instances of A into B.

(e) Say why f is computable in polynomial time.

(f) Show that "x is a YES-instance for A" $\implies$ "$f(x)$ is a YES-instance for B".
(Remember: convert a certificate for x into a certificate for $f(x)$!)

(g) Show that "$f(x)$ is a YES-instance for B" $\implies$ "x is a YES-instance for A".
(Remember: convert a certificate for $f(x)$ into a certificate for x!)

6. Vertex Cover and Independent Set

Define IND-SET as follows:
Input: An undirected graph G and a positive integer k
Output: `true` if there is an independent set in G of size at least k, `false` otherwise.

Define VER-COVER as follows:
Input: An undirected graph G and a positive integer k
Output: `true` if there is an vertex cover in G of size at most k, `false` otherwise.

Prove that VER-COVER is NP-complete using IND-SET.

Follow the standard steps. To show that VER-COVER is in NP,

(a) State what the certificate is.

(b) Say why the certificate can be checked in polynomial time.

Now, to show that VER-COVER is NP-hard,

(c) Fill in the blank: "We will reduce from ___ to ___". Which is A, and which is B?

(d) Define a reduction function f, which converts instances of A into B.

(e) Say why f is computable in polynomial time.

(f) Show that "x is a YES-instance for A" $\implies$ "$f(x)$ is a YES-instance for B".
(Remember: convert a certificate for x into a certificate for $f(x)$!)

(g) Show that "$f(x)$ is a YES-instance for B" $\implies$ "x is a YES-instance for A".
(Remember: convert a certificate for $f(x)$ into a certificate for x!)