Section 9: Solutions

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

Solution:

Satisfiable. Many possible solutions (students only need to list one of these):

- $a = 0$, $b = 0$, $c = 0$, $d = 0$
- $a = 0$, $b = 0$, $c = 1$, $d = 0$
- $a = 0$, $b = 1$, $c = 0$, $d = 0$
- $a = 0$, $b = 1$, $c = 1$, $d = 0$
- $a = 0$, $b = 1$, $c = 1$, $d = 1$
- $a = 1$, $b = 0$, $c = 1$, $d = 0$

- $a = 1, b = 1, c = 1, d = 0$
- $a = 1, b = 1, c = 1, d = 1$

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

Solution:

Not satisfiable. Although you might be able to try some ad hoc arguments for why, there is generally no explanation significantly faster than "try everything".

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.

Solution:

An assignment to the variables that makes the formula true.

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

Solution:

Verifier takes as input the original 5SAT input and the certificate (the assignment). Just apply the assignment to every clause, and return whether all clauses are satisfied. Runs in linear 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?

Solution:

We will reduce from A = 3SAT to B = 5SAT. In other words, convert instances of 3SAT into instances of 5SAT.

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

Solution:

Let $C_1, C_2, ..., C_m$ be the clauses of the 3SAT instance.

Create two dummy variables y and z. For each clause C_i, create four clauses:

$$(C_i \vee y \vee z) \qquad (C_i \vee \neg y \vee z) \qquad (C_i \vee y \vee \neg z) \qquad (C_i \vee \neg y \vee \neg z)$$

Our 5SAT instance is the AND of all $4m$ clauses described above.

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

Solution:

We loop through the clauses and create a constant number of new clauses for each, thus linear 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 $f(x)$!)

Solution:

- There is an assignment α that makes the original 3SAT YES-instance true.
- Let us define an assignment β that satisfies our constructed 5SAT instance:

$$\begin{aligned} \beta(x_i) &= \alpha(x_i) && \text{for all } x_i \text{ in the original instance} \\ \beta(y) &= 0 && \text{(or 1, doesn't matter)} \\ \beta(z) &= 0 && \text{(or 1, doesn't matter)} \end{aligned}$$

- Satisfies our constructed 5SAT instance because every clause contains one of the original 3SAT clauses.

(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!)

Solution:

- There is an assignment β that makes the formula we constructed true.

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).

Solution:

On input G, k (for $k \leq n$) for $\mathsf{VC_D}$, run the library for $\mathsf{VC_O}$ on input G. Count the number of vertices in the output. If it is k or less, return `true`, otherwise return `false`.

If there is a vertex cover of size at most k, then there is a vertex cover of size k (just add vertices until you hit k). If there is not a vertex cover of size at most k, then a minimum one is larger, and so the $\mathsf{VC_D}$ algorithm will give a longer list, and the reduction will return false, as required.

(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.

Solution:

If u is in a minimum vertex cover, then delete u and all edges incident to u from the graph G. Call the resulting graph $G-u$. Call the $\mathsf{VC_O}$ library on $G-u$. Return u along with the result of the library call.

Let S be a vertex cover of $G-u$. Observe that adding u gives a vertex cover of G, as every edge not incident to u was covered in $G-u$, and u was added to the vertex cover to cover all remaining edges. Moreover, we find a minimum vertex cover; We know that u is in a minimum vertex cover and removing u from any vertex cover for G gives a cover of $G-u$; a smaller cover of G including u would give us a smaller cover

for $G - u$, but we called the $\mathsf{VC_O}$ library which gives us the minimum.

(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.

Solution:

If v is not in the vertex cover, then all neighbors w of v must be in the cover (otherwise, we would not cover the edge (v, w)). So we delete v and all its neighbors denoted $N(v)$ from the graph. Then we can run VC_0 on the graph $G - v - N(v)$ and we return the result along with all vertices in $N(v)$.

Let S be a minimum vertex cover of G. Since S does not contain v by our assumption, then every neighbor $w \in N(v)$ has an edge (v, w) that needs to be covered, which means every neighbor must be in S. Then all edges coming out of v and its neighbors are covered, so we only need to solve the minimum vertex cover on the graph minus these vertices.

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

Solution:

```
1: function MinVertexCover(G)
2:     Call VC_D library for all values of k until you find the size of the min vertex cover of G.
3:     Pick an arbitrary vertex u.
4:     if VC_D library says YES on G − u, k − 1 then
5:         return {u} ∪ MinVertexCover(G − u)
6:     elsereturn N(u) ∪ MinVertexCover(G − u − N(u))        ▷ N(u) is the neighbors of u
```

We will skip the proof of correctness, as it is mostly combining the prior parts.

For efficiency, observe that we need polynomial work and $n + 1$ library calls in each recursive call, and each recursive call reduces the problem size by at least 1, so we need at most n recursive calls. Thus the reduction is polynomial.

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.

Solution:

A sequence of pebbling moves that leaves exactly one pebble in the graph.

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

Solution:

Given a sequence of pebbling moves, we just have to check if it's valid and if it's long enough to remove enough pebbles. At most this will take polynomial time in terms of the number of pebbles.

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?

Solution:

We will reduce from Hamiltonian Path to Pebble. A is Hamiltonian Path, B is Pebble.

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

Solution:

Let G be the graph given in the Hamiltonian Path Problem. Let's say that G has n vertices.

Observe there are n possible starting vertices, so it's sufficient to check if there's a Hamiltonian Path for each of these n possible starting vertices.

Let that starting vertex be v_1.

Let $p(v)$ be the number of pebbles at vertex v. Define $p(v_1) = 2$ and $p(v) = 1$ otherwise (i.e. all vertices start with one pebble except the starting vertex, which starts with an additional pebble).

We assert that there is a Hamiltonian Path if and only if any of these n PEBBLE problems is true.

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

Solution:

From the reduction, we run the PEBBLE algorithm n times, once for each start vertex, so this only contributes a polynomial factor. Additionally, to modify the graph each time, we simply label each vertex in constant time, which takes linear time to do so. So overall, this runtime is polynomial.

(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 $f(x)$!)

Solution:

Suppose there is a Hamiltonian Path $v_1, ..., v_n$.

Then consider the sequence of pebble moves: $(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)$. Observe for any v_i where $1 \leq i < n$, since v_i, v_{i+1} is in the Hamiltonian Path, then (v_i, v_{i+1}) must be an edge in G. Since (v_i, v_{i+1}) is

the first pebble move that takes away pebbles from v_i, and since v_i starts with at least one pebble, then v_i has all of its starting pebbles when we attempt to do that pebble move. In the case that $i = 1$, then v_i has enough pebbles to do the move since it starts with two pebbles. In the case that $1 < i < n$, then (v_{i-1}, v_i) was the previous pebble move, so v_i just gained a pebble and started with one pebble. So v_i has at least two pebbles and has enough to pebbles to make the pebble move.

After this sequence, observe v_n has not lost any pebbles, so it still has its starting pebble, and it also just gained a pebble from the move (v_{n-1}, v_n), so it has two pebbles. Since the graph started with $n+1$ pebbles and we made $n-1$ moves, there are only two pebbles left, and v_n has both of them. Then, simply add the move (v_n, v_{n-1}), which is valid since v_n has two pebbles and the edge (v_n, v_{n-1}) exists since (v_{n-1}, v_n) was a valid move. Now we have on pebble left.

So there is indeed a sequence of pebble moves removing all but one pebble.

(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!)

Solution:

Suppose we have a sequence of pebble moves removing all but one pebble. We need to show that there also exists a Hamiltonian Path.

Since we start with $n+1$ pebbles, and each move removes one pebble, this sequence must have exactly n moves. Denote the pebbling sequence as $(v_1, w_1), \ldots, (v_n, w_n)$ where v_i sends a pebble to w_i at step i.

Define the preposition $P(k)$ for $0 \leq k < n$ to be true iff after k moves, for $i = 1 \ldots, k$, v_i has zero pebbles, v_{k+1} 2 pebbles and $v_{i+1} = w_i$. Furthermore all vertices $v_1, \ldots, v_{k+1}$ are distinct. We prove by induction on k.

For $k = 0$, we trivially satisfy that v_1 is distinct.

Suppose $P(k-1)$ holds for some $0 < k-1 < n-1$, then we look at the kth move (v_k, w_k). We know that the $k-1$st move was (v_{k-1}, v_k) and v_k has two pebbles, so now consider all possible moves to w_k. We know that since we still have at least another move (v_{k+1}, w_{k+1}) left that w_k can only be v_{k+1}, or otherwise we only have at most 1 pebble at v_{k+1}. Furthermore $w_k = v_{k+1}$ must be a vertex distinct from v_i for $1 \leq i < k$ since these all had 0 pebbles up to move $k-1$ and can only get 1 more pebble from the kth move. Finally we can see that v_k will now have 0 pebbles after the kth move.

Then by induction, the first $n-1$ moves can be written as the sequence $(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)$.

Since this removes $n-1$ pebbles, there are 2 pebbles remaining after this sequence and by induction again the last vertex v_n contains two pebbles. Also all the vertices are distinct. Hence this sequence of vertices $v_1, \ldots, v_n$ form a Hamiltonian path as (v_i, v_{i+1}) are all valid edges for each $i < n$.

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.

Solution:

A vertex cover in G of size at most k.

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

Solution:

A verifier would take in the subset of k vertices that are a vertex cover. Given this set of vertices, a verifier would check that these vertices are actually covering the graph (i.e. are endpoints to each edge in the graph). This will take time $\mathcal{O}(|E| + |V|)$, so it is 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?

Solution:

We will reduce from IND-SET to VER-COVER. A is IND-SET and B is VER-COVER.

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

Solution:

The idea is taking the complement.

For any graph $G = (V, E)$, S is an independent set if and only if $V - S$ is a vertex cover.

And using the format given in the problem definition, there is an vertex cover in G of size k if and only if there is an independent set in G of size $|V| - k$.

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

Solution:

Our algorithm just returns the output of `VER-COVER` but with parameter $|V| - k$, which is 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 $f(x)$!)

Solution:

Let $G = (V, E)$ and positive integer k be given by `IND-SET`.

Suppose G has an independent set of size at least k, we show that our reduction returns true. Let S be the independent set of size at least k, then every vertex in S touches at most one endpoint of every edge in G. So $V - S$ touches at least one endpoint of every edge of G. Hence $V - S$ is a vertex cover of size at most $|V| - k$ and thus our algorithm output true.

(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!)

Solution:

For the other direction suppose that our algorithm returns true, meaning there is a vertex cover of size at least $|V| - k$ in G. Let S be the vertex cover of size at least $|V| - k$, then S touches at least one endpoint of every edge in G. So $V - S$ touches at most one endpoint of ever edge in G. Hence $V - S$ is an independent set of size at least k.