Proving NP-completeness

Proof Structure

A language B is *NP-complete* if

1. $B \in \text{NP}$, and
2. B is NP-hard.

A language B is *NP-hard* if every language $A \in \text{NP}$ is polynomial time reducible to B (denoted $A \leq_{\text{P}} B$). A language B is a member of NP if there is a polynomial time verifier that verifies it, or, equivalently, there is a nondeterministic algorithm that decides it in polynomial time.

The Cook-Levin Theorem (p. 304-311) directly proves that any language $A \in \text{NP}$ is polynomial time reducible to a particular problem, *SAT* (the satisfiability problem), by showing that any polynomial time nondeterministic Turing machine can be converted to an instance of *SAT* in polynomial time. (It can also be modified to show that the same holds true for *3SAT*, a variation of the same problem that is more amenable to proving polynomial time reductions.) Since $\leq_{\text{P}}$ is a transitive relation, once we've directly proven that every language in NP is polynomial time reducible to *SAT* or *3SAT*, we only need to show that *SAT* or *3SAT* is polynomial time reducible to another language B to show that the same is true for B. We can then use reductions from B to show that other languages are NP-hard, and so on.

A full proof that a language B is NP-complete contains the following.

1. A proof that $B \in \text{NP}$. To do this, either one of the following is necessary:
   (a) Provide a verifier that verifies B in polynomial time.
   (b) Provide a nondeterminsitic polynomial time algorithm that decides B.

   Either way, you should argue that your solution runs in polynomial time with respect to n, where n is the length of the input string to the algorithm (not including the certificate c).
2. A proof that B is NP-hard. Pick a known NP-hard/NP-complete problem A and show that $A \leq_{\text{P}} B$.
   (a) Describe the polynomial time mapping reduction f.
   (b) Prove that f is correct; that is, $w \in A$ if and only if $f(w) \in B$. Often this is split into two arguments:
      - If $w \in A$, then $f(w) \in B$.
      - If $f(w) \in B$, then $w \in A$.

   (c) Prove that the reduction f is computable in polynomial time.

Example showing that *HAMCYCLE* is NP-complete

A *Hamiltonian cycle* in a graph G is a cyclic path through the nodes of G that visits every node exactly once. We can express the problem of determining whether a graph contains a Hamiltonian cycle as a language.

$$HAMCYCLE = \{\langle G \rangle \mid G \text{ contains a Hamiltonian cycle}\}$$

We can prove that $HAMCYCLE$ is NP-complete as follows.

First, we prove that $HAMCYCLE \in \text{NP}$ by describing a polynomial time verifier V for it.

V = "On input $\langle\langle G \rangle, c\rangle$

1. Test whether the string c encodes a permutation of the nodes in G. If not, *reject*.

2. Let $c = \{v_1, \ldots, v_k\}$. For each $i = 1, \ldots, k-1$, test whether v_i and v_{i+1} are connected by an edge in G. Then, test whether v_k and v_1 are connected by an edge. If all tests pass, *accept*, otherwise *reject*."

Let k be the number of nodes in G. Note that $n = |\langle G \rangle|$ and $k = O(n)$. Stage 1 can be done in polynomial time (it is easy to imagine a $O(k^2)$ algorithm that checks that there are no duplicates in c). Stage 2 consists of k tests of whether two nodes are connected by an edge, so it can be performed in $O(k)$ time. So the whole algorithm runs in polynomial time. (If you want to be extra technical, there is an extra factor of $O(\log k)$, supposing that every node is encoded as a natural number in binary.)

Next, we prove that $HAMCYCLE$ is NP-hard by providing a polynomial time reduction from $HAMPATH$, which is a known NP-complete problem (p. 314).

$$HAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

First, we describe the polynomial time mapping reduction f, which is a polynomial time algorithm that transforms an instance of $HAMPATH$ into an instance of $HAMCYCLE$ in such a way that an algorithm that decides $HAMCYCLE$ would produce the correct accept/reject decision for the original instance of the $HAMPATH$ problem. That is, f is an algorithm that takes a string $\langle G, s, t \rangle$ which may or may not be in $HAMPATH$, and constructs a new string $\langle G' \rangle$ which may or may not be in $HAMCYCLE$, so that $f(\langle G, s, t \rangle) = \langle G' \rangle$. The mapping reduction consists of describing how to construct G' given G, s, t. In order to be correct, G must have a Hamiltonian path from s to t *if and only if* G' has a Hamiltonian cycle.

The construction for f works as follows. Let G' have all the same nodes and edges as G, but add to G' a new vertex v, and add an edge from t to v and an edge from v to s. This is the end of the construction.

Now we prove that f is correct by showing that G has a Hamiltonian path from s to t if and only if G' has a Hamiltonian cycle. We split this up into two arguments.

First, we show that if G has a Hamiltonian path from s to t, then G' must have a Hamiltonian cycle. If G has a Hamiltonian path from s to t, then there is a simple path in G' that starts at s and ends at t and visits all the original nodes from G. If we combine this with the edges (t, v) and (v, s), this forms a Hamiltonian cycle in G', since it is a simple path that visits all nodes.

Second, we show that if G' has a Hamiltonian cycle, then G must have a Hamiltonian path from s to t. If G' has a Hamiltonian cycle, it visits every node exactly once, including v, so it must visit the edges (t, v) and (v, s), since that is the only way to visit v. The rest of the cycle connects s back to t with a simple path that visits all the nodes in G' except for v exactly once. This corresponds to a Hamiltonian path from s to t in G.

Finally, we argue that f is computable in polynomial time. This construction merely involves creating a copy of G that adds one vertex and two edges to G, which is clearly no worse than $O(n)$.

Note that directly connecting t to s with an edge without adding v would *not* be correct. It would still be the case that if there is a Hamiltonian path in G from s to t, there would be a corresponding

Hamiltonian cycle in G' that follows the edge from s to t. However, it would not necessarily be the case that if there is a Hamiltonian cycle in G', there would be a Hamiltonian path beginning at s and ending at t. Counterexample: Consider the case where G has nodes $\{s, a, t, b\}$ and edges $s \rightarrow a \rightarrow t \rightarrow b \rightarrow s$. These edges form a Hamiltonian cycle in G', but there is no way to form a Hamiltonian path in G that specifically begins at s and ends at t.

Changelog

- **Apr 28:** Originally, the end of this document included a note that incorrectly claimed that directly connecting s to t with an edge without adding v would also work. As explained in the updated version, this is not true.