NP-completeness is a useful concept for showing the difficulty of a computational problem, by showing that the existence of a polynomial-time algorithm for the problem would imply that $P = NP$. This handout reviews the key steps in constructing a proof of NP-completeness for a problem. Specific examples of such proofs may be found in the practice problems.

Recall that an NP-complete decision problem must satisfy two criteria: it must be in NP, and it must be NP-hard, i.e. all problems in NP must be reducible to it. A proof of NP-completeness must prove both of these properties, while a proof of NP-hardness requires only the second.

1 Showing that a Problem is in NP

According to our definitions, a decision problem $P$ is in NP if there exists a certificate scheme that associates every true instance $z \in P$ with a certificate $c(z)$ and supplies a two-argument verification algorithm $V(z, c)$ for checking certificates. The following properties must hold:

1. $V(z, c)$ must return “true” iff $z$ is a true instance of problem $P$ and $c$ is a certificate proving that $z \in P$; otherwise, it must return false.

2. $V$ runs in time polynomial in both $|z|$ and $|c|$.

3. For every $z \in P$, there exists a certificate $c$ of size at most polynomial in $|z|$ such that $V(z, c)$ returns true.

A typical proof of “NP-ness” for a problem $P$ starts by defining the certificate scheme and the verifier algorithm, then checks the three requirements above hold.

Most of the decision problems we will encounter are easily proven to be in NP. This is not, however, guaranteed to be the case for all problems in NP. For example, the decision version of integer factoring was proven to be in NP only after the development of the AKS algorithm – the first deterministic polynomial-time algorithm for primality testing – in 2002.

2 Showing that a Problem is NP-hard by Reduction

Although Cook’s Theorem directly proves that SAT is NP-hard by definition, most NP-completeness proofs argue NP-hardness using a simpler, two-problem reduction strategy.

Let $P'$ be a known NP-hard problem, and let $P$ be a problem of unknown hardness. Our goal is to reduce $P'$ to $P$, i.e. to show that $P' \leq_p P$. Why is this useful? Since $P'$ is NP-hard, we know that for all problems $Q \in NP$, $Q \leq_p P'$. Hence, if we can show that $P' \leq_p P$, then by transitivity of reduction, we have that $Q \leq_p P$ for all $Q \in NP$, and so $P$ is also NP-hard.

The structure of a reduction argument has several pieces, which flow from the definition of polynomial-time reduction.

1. Choose the problem $P'$ to reduce from. $P'$ must be known to be NP-hard. For class purposes, it should be one of the NP-hard problems we studied in class or a problem that was asserted or proven to be
NP-hard in the homework, practice problems, etc. I will often give you a list of NP-hard problems to choose from.

2. Devise a transducer function \( f \) that maps instances \( z' \) of problem \( P' \) to instances \( z \) of problem \( P \). More informally, describe a construction that makes an instance of \( P \) from an instance of \( P' \).

The goal in designing the mapping from \( P' \) to \( P \) is that true instances of \( P' \) should map to true instances of \( P \), and vice versa. You'll prove this in the steps below.

3. Argue that \( f \) runs in time polynomial in \( |z'| \) (and hence produces an output of size polynomial in \( |z'| \)).

4. Show that, if \( z' \in P' \), then \( f(z') \in P \).

5. Show that, if \( f(z') \in P \), then \( z' \in P' \).

The construction of the transducer \( f \), along with proofs of properties 4 and 5, are the real meat of an NP-hardness proof. It is very important that you prove both properties. You may find it natural and intuitive to devise a transducer \( f \) that maps true instances of \( P' \) to true instances of \( P \). However, the converse assertion is often somewhat challenging to prove, especially if you weren’t thinking about it when you designed \( f \).

2.1 Are All “Challenging” Problems in NP Known to Be NP-complete?

There are relatively few problems formulated by computer scientists that are known to be in NP but not known to be either in P or NP-complete. A famous problem in NP whose hardness remains unsettled is graph isomorphism: given two (undirected) graphs \( G \) and \( H \), is there a function \( f \) mapping nodes of \( G \) to notes of \( H \) such that \((u,v) \in G \) iff \((f(u),f(v)) \in H\)?

The aforementioned decision version of integer factoring is also not known to be NP-complete, though most people believe that it is. Other examples of natural problems whose NP-completeness is open can be found online by searching with the keywords “turnpike problem,” “Boolean formula dualization,” and “precedence-constrained 3-processor scheduling.”

A well-known problem whose completeness for NP was open for over 25 years, minimum-weight triangulation, was finally proven NP-complete by Mulzer and Rote only in 2006. The proof (in its 2008 version) used a Python program to help verify the correctness of the authors’ transducer function.

3 Relationship to NP-Optimization Problems

Remember that P and NP are classes of decision problems, whose answer is either “yes” or “no.” If you want to argue that an optimization problem is hard, you need a bridge to decision problems.

Let \( Q \) be an optimization problem, defined by a set of feasibility constraints and an objective function \( V \) on feasible solutions that we wish (WLOG) to maximize. The canonical decision problem \( \text{DEC}_Q \) for \( Q \) is the following decision problem:

\[
\text{Given an instance } x \text{ of problem } Q, \text{ does } x \text{ have a feasible solution } \pi \text{ with } V(\pi) \geq y?
\]

Here, \( y \) as well as \( x \) is an input to the decision problem. There is a similar canonical problem for minimizations.

We say that \( Q \) is an \textit{NP-optimization problem} iff \( \text{DEC}_Q \) is NP-complete. We showed in class that, if \( Q \) is an NP-optimization problem, then there exists a polynomial-time algorithm to find the value of an optimal solution for an instance of \( Q \) iff \( P = NP \).

It may be tempting to show that \( Q \) is hard by translating it to some non-canonical decision problem that is more easily shown to be NP-complete. However, our definition of “NP-optimization” requires the
use of the canonical decision problem. If you want to reduce to any other decision problem, it is up to you
to prove, using methods similar to those we discussed in class, that the decision problem’s hardness implies
that $Q$ is hard.