CPSC 320: Intermediate Algorithm Design and Analysis

Tutorial: Week 13 (NP-completeness notes)

Author: Susanne Bradley

NP-completeness: quick notes

Throughout this unit, we deal with *decision problems*: these are algorithmic problems with a YES or NO answer. The problems typically have the form: *does there exist a solution to this problem that satisfies (some properties...).*

Complexity classes

- **P**: problems that are solvable in polynomial time (i.e., we can come up with an algorithm that correctly solves any instance of the problem in $O(n^c)$ time, for some constant c).
  - Example of a problem in P: given a graph and two nodes u and v, does there exist a path between u and v of length at most k?
- **NP**: problems where a candidate solution (called a *certificate*) can be verified in polynomial time.
  - Example: SAT. We look for a set of truth assignments that will make all clauses evaluate to TRUE. So, our "certificate" is the set of truth assignments, and for a given certificate, we can *verify* whether it actually gives a YES answer to the problem (by satisfying all the clauses) in polynomial time.
- **NP-hard**: all problems that are *at least as hard to solve* as the hardest problems in NP. So, some NP-hard problems are in NP, but some are not (they are *harder* than all problems in NP). As far as we know, none of these problems can be solved in polynomial time.
- **NP-complete**: problems that are in NP and in NP-hard. That means we think we can't solve the problem in polynomial time (because the problem is NP-hard), but we can check whether a particular candidate solution works in polynomial time (because the problem is in NP).
  - Examples: SAT, 3-SAT, independent set, vertex cover, traveling salesperson, subset sum, and many others...

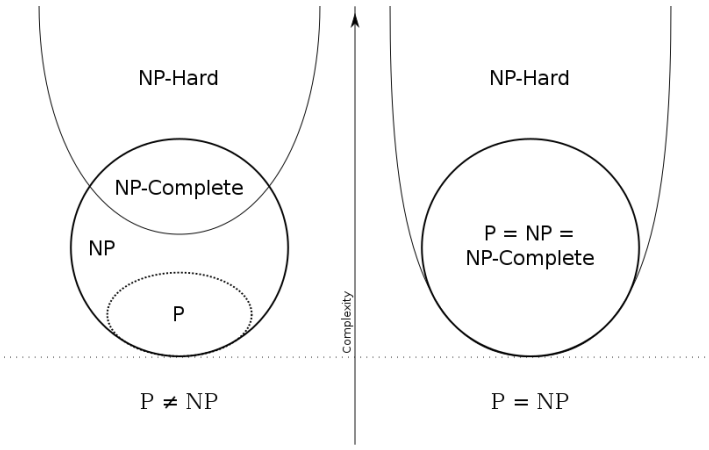


Figure 1: Image by Behnam Esfahbod, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3532181 (retrieved 22 November, 2017).

Proving that a problem is in NP

To prove that a problem is in NP, you have to:

1. Define a certificate for the problem.
2. Describe how the certificate can be verified in polynomial time.

A certificate is a proof of a YES answer. We said earlier that these problems are usually existential – i.e., "Does there exist a solution that satisfies X." Your certificate should be whatever an *actual solution* looks like.

Example: the vertex cover problem asks: given a graph, is there a set of no more than k vertices such that every edge in the graph is adjacent to at least one of these vertices?

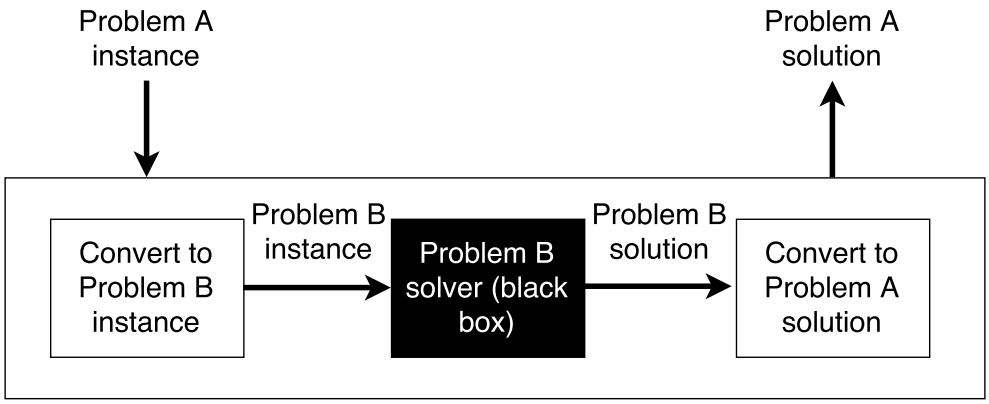
To prove that the problem is in NP:

- *Step 1*: define the certificate. In this problem, we're looking for a set of vertices that form a vertex cover. So, our certificate will be a set of the vertices in the graph.
- *Step 2*: given a certificate, we need to determine, in polynomial time, whether it actually solves our problem – in other words, determine whether a set of vertices is actually a vertex cover of size less than or equal to k. Here are our steps:
  - Check whether the size of the set of vertices is actually less than or equal to k (either constant or linear time in the number of vertices).
  - Now, check whether these vertices are actually a cover. We can do this by iterating over all edges in the graph and checking whether each edge is adjacent to at least one of the vertices in the vertex cover (linear time in the number of edges in the graph).

In Step 1, we defined the certificate for the problem, and in Step 2 we showed that a certificate can be verified in polynomial time. Therefore, vertex cover is in NP.

Proving that a problem is in NP-hard

Suppose we have a new problem called B, which we want to show is NP-hard. We prove this with a polynomial-time reduction *from some problem A that we know to be NP-hard*:



We *imagine* we have a way to solve our new unknown problem, and then use that to solve some NP-hard problem. Intuitively, this means that a solver for B is at least as powerful as a solver for A (since we can use our black box for B to solve A), so B is at least as hard. More formally, if we could solve B in polynomial time, then our reduction would give us an algorithm to solve A in polynomial time; therefore, if A is NP-hard and (we think) not solvable in polynomial time, then B can't be solved in polynomial time, either.

Some important things to remember:

- Remember to reduce in the correct direction! **You reduce *from* a hard problem *to* a new, unknown problem.**

- Both algorithms in your reduction (converting the instance, and converting the solution) must run in polynomial time! Otherwise, you could solve B in polynomial time and still not have a polynomial time algorithm for A, which means you haven't actually proven anything about B.

Proving that a problem is in NP-complete

Just prove that it's in NP, and prove that it's in NP-hard!

Proving correctness of your NP-hardness reduction

For the purposes of this class, if we ask you to give a reduction to prove that a problem is NP-hard, you *do not* need to prove correctness of your reduction unless we specifically ask for it. But, should you need to give such a proof, here's a high-level sketch of what you can do.

In nearly all your NP-hardness reductions, your second algorithm will be: return YES to problem A if and only if the answer to B is YES. In this case, any proof of correctness has to establish: *the answer to reduced problem B is YES if and only if the answer to the original problem A is YES*. This requires proving that YES to A implies YES to B, and proving that NO to A implies NO to B.

But, it's hard to reason about NO instances. So instead, we'll generally prove that YES to A implies YES to B, and that YES to B implies YES to A. Why is that easier? It's because when we deal with YES instances, we can use the handy notion of certificates that we defined earlier!

These are the steps you'll generally want to take to prove correctness of your reduction:

1. Define the certificates for A and B.
2. Prove that YES to A implies YES to B. The most straightforward way to do this: assuming you have a working certificate (i.e., a true YES solution) for A, describe how you would construct the certificate (i.e., the actual YES solution) for the reduced instance of problem B.
3. Prove that YES to B implies YES to A. Proceed as before: assuming you have a working certificate for your reduction's instance of B, describe how to construct the certificate for the original instance of A.