Handout 3: NP-Hardness Proofs

Note: You're welcome to borrow language from this handout as you write your own NP-hardness proofs. You may even use whole sentences verbatim from this document if you wish.

The purpose of this handout is to provide you with a template for how to write NP-Hardness proofs. Following this template will ensure that your proof is more likely to be correct, and communicated in a way that's easier to comprehend and grade. All components listed must be present in order for a proof to be complete.

This first step of an NP-hardness proof is to choose an NP-hard problem to reduce from. Usually, it is best to choose a problem that is as similar as possible to the problem in question, because this makes for a reduction that is simpler to describe and analyze, and less error-prone. Sometimes we can find a very similar problem, but sometimes we need to rely on something generic, like 3SAT.

Writing NP-hardness proof requires care and precision. Let's demonstrate what's expected by giving the full NP-hardness proof for VertexCover from lecture. In lecture we gave (most of) the proof that 3SAT $\leq_p$ VertexCover. Because 3SAT is NP-hard, this implies that VertexCover is NP-hard. Be careful about the direction of the reduction: showing that VertexCover $\leq_p$ 3SAT is not useful for showing that VertexCover is NP-hard!

Overview

First, a reminder about terminology. Saying that x is a "yes" instance of problem L is equivalent to saying that $x \in L$ ("x is a member of the language L"). Similarly, saying that x is a "no" instance of L is equivalent to saying $x \notin L$. In particular, a 3CNF formula φ is a "yes" instance of 3SAT if it is satisfiable; otherwise it is a "no" instance.

By definition of "$\leq_p$", to prove that 3SAT $\leq_p$ VertexCover, we need to give an efficient algorithm that converts an arbitrary instance of 3SAT—i.e., a 3CNF formula φ—into an instance of VertexCover—i.e., a graph G and a budget k—that has the same yes/no designation as the original 3SAT instance. More formally, our goal is to define a polynomial-time computable function f such that for all 3CNF formulas φ,

$$\varphi \in \text{3SAT} \iff f(\varphi) = (G, k) \in \text{VertexCover} .$$

There are several steps involved in showing that 3SAT $\leq_p$ VertexCover (or in giving any other poly-time mapping reduction):

1. Describe the reduction. That is, show how to efficiently convert an arbitrary instance of 3SAT into a corresponding instance of VertexCover. In other words, define the function f.
2. Explain why the reduction runs in polynomial time.
3. Prove that the given original formula φ is a "yes" instance of 3SAT *if and only if* the constructed instance $(G, k) = f(\varphi)$ is a "yes" instance of VertexCover. In other words, prove that $\varphi \in \text{3SAT} \iff f(\varphi) \in \text{VertexCover}$. To do this we prove each direction separately:
   (a) ($\implies$ direction) If φ is a "yes" instance of 3SAT, then the constructed $(G, k) = f(\varphi)$ is a "yes" instance of VertexCover.
   (b) ($\impliedby$ direction) If the constructed $(G, k) = f(\varphi)$ is a "yes" instance of VertexCover, then the original formula φ is a "yes" instance of of 3SAT.

Step 1: Describe the Reduction

Coming up with a (correct) reduction is often the trickiest step, and may require some creative insight and failed attempts, which can be exposed by gaps or counterexamples in attempted correctness proofs. This is normal! Before reading the following description of our reduction from 3SAT to VertexCover, you may want to remind yourself of the definitions of these problems from lecture.

The reduction is given as input an arbitrary instance φ of 3SAT, and it outputs an instance of VertexCover, which consists of a graph G and a budget k. We will describe the graph in words; you can refer to the lecture slides for pictures of a concrete example.

1. Let n denote the number of variables in φ. For each of these n variables x_i, make a pair of vertices, labeled x_i and $\overline{x_i}$, and connect them by an edge. We call each such vertex pair and its edge a "variable gadget".

2. Let m denote the number of clauses in φ. For each of these m clauses, make three vertices, labeled with the three literals in that clause (one literal per vertex), and connect them with edges to form a triangle. We call each such triangle a "clause gadget."

3. For each vertex in each clause gadget, connect it by an edge to the variable-gadget vertex having the same label. This completes the construction of the graph G.

4. Finally, set the budget to $k = n + 2m$.

To develop some intuition about this reduction, observe that, in order to cover a variable gadget's edge, we must select at least one of the gadget's vertices. Similarly, to cover a clause gadget's three edges, we must select at least two of the gadget's vertices. Therefore, any vertex cover of G must have *at least* $n + 2m$ vertices—and this is not even considering the edges that cross between the variable gadgets and the clause gadgets. So, setting the budget as $k = n + 2m$ is therefore asking whether there is a vertex cover of size *exactly* $n + 2m$, i.e., whether all the edges in G can be covered by selecting exactly one vertex from each variable gadget and two vertices from each clause gadget. As we show below, by the design of the "crossing" edges, it turns out that this is possible *if and only if* the original formula φ is satisfiable. Moreover, every satisfying assignment for φ (if any) is in *exact correspondence with* the variable-gadget vertices that are part of some vertex cover of size k.

An important remark is that, when defining a reduction, *we cannot use the* ***solution*** *of the original problem*, because it may not even exist—and even if it does, in general we do not know how to compute such a solution in polynomial time. For example, in our reduction from 3SAT, we cannot say "construct a vertex for every variable that is set to *true* in a satisfying assignment of the formula"—the reduction is not given a satisfying assignment, and we do not know how to compute one efficiently. Instead, the reduction needs to manipulate *just the input formula* so that if the formula happens to be satisfiable (which it may or may not be), then the constructed graph will have a vertex cover of the desired size, and vice versa. (One can confirm that our reduction above obeys this rule.) By contrast, in the *analysis* of the reduction, we can (and usually do) work with a solution for one instance, and use it to demonstrate a solution for the other instance. This is legal because there we are proving a claim whose hypothesis is that the instance has a solution, and analysis is not an algorithm; it does not have a running time.

Step 2: Show that the Reduction Runs in Polynomial Time

We must show that this reduction takes time polynomial in the size of the given 3SAT instance φ. The 3SAT instance has n variables and m clauses, so its size is $\Theta(n+m)$. (All the variables must be listed, along with the contents of the clauses.) Each of the n variable gadgets takes constant time to construct. Each of the m clause gadgets takes constant time to construct. Thus, constructing the gadgets takes time $\Theta(m+n)$. The total number of edges between the variable gadgets and the clause gadgets is $3m$, since there is one edge for each of the 3 literals in each of the m clauses. Computing the budget $k = n + 2m$ takes logarithmic time. Therefore, the entire reduction takes $\Theta(n+m)$ time which is polynomial (in fact, linear) in the size of the 3SAT instance.

Step 3a: Show that $\varphi \in$ 3SAT $\implies (G,k) \in$ VertexCover

Assume that φ is a "yes" instance, i.e., it has a satisfying assignment. We show that for any satisfying assignment for φ, there is a corresponding vertex cover in G that has exactly k vertices, and hence (G,k) is a "yes" instance of VertexCover.

Fix some satisfying assignment for φ, i.e., an assignment of each variable in φ to true or false that makes φ true—i.e., it satisfies *every* clause of φ simultaneously. We use the assignment to construct a corresponding vertex cover C having k vertices. For each variable gadget, we include in C the one vertex whose label is *true* under the assignment. That is, for each variable x_i in φ, if it is assigned true, then we include the variable-gadget vertex labeled x_i; otherwise, we include the vertex labeled $\overline{x_i}$. So far, we have included exactly n vertices.

Next, we know that in each clause in φ, at least one of its literals is true. For each clause gadget, arbitrarily choose one vertex that is labeled with a true literal, and include in C the *other* two vertices of the gadget. For example, if the clause is $(x_3 \vee \overline{x_7} \vee x_2)$ and x_2 is true, we might include in C the corresponding clause-gadget vertices labeled x_3 and $\overline{x_7}$ (but not the vertex labeled x_2). This uses exactly two vertices per clause gadget, for a total of $2m$ additional vertices. So in total, there are exactly $k = n + 2m$ vertices in C.

It just remains to argue that C is, in fact, a vertex cover of the graph G. We do so using the fact that the assignment satisfies φ. There are three types of edges in the graph: those in the variable gadgets, those in the clause gadgets, and those between a vertex in a clause gadget and a vertex in a variable gadget. The edges in the variable gadgets are all covered since each variable gadget had one of its two endpoints selected. The edges in the clause gadgets are all covered since two vertices in each clause gadget were selected, and any two vertices in a triangle cover all its edges.

Finally, consider an edge e between a variable gadget and a clause gadget. If the clause-gadget endpoint of e is one of the two that were included in C, then e is covered, as needed. Otherwise, this endpoint is labeled by a literal, and it was left out of C specifically because this literal is true under the assignment. The other endpoint of e is the variable-gadget vertex having the same label, and because its literal is true, that vertex was included in C. So, e is covered by that vertex. We conclude that every edge in G is covered by C, so C is a vertex cover, as claimed.

Step 3b: Show that $(G,k) \in$ VertexCover $\implies \varphi \in$ 3SAT

Now suppose that (G,k) is a "yes" instance of VertexCover, i.e., there is a vertex cover of size at most k in G. We show that for any such vertex cover, there is a corresponding satisfying assignment of the original formula φ, and hence φ is a "yes" instance of 3SAT . Importantly, notice that (G,k)

is *not an arbitrary instance* of VertexCover; it was constructed by applying our reduction to φ. Our analysis will rely heavily on the features that the reduction built into G.

As argued above (after the description of the reduction), any vertex cover of G must have *at least* one vertex from each variable gadget, and at least two vertices from each clause gadget. So, any vertex cover of G that has at most $k = n+2m$ vertices must have *exactly* one from each variable gadget and *exactly* two from each clause gadget. Fix any such vertex cover C, which exists because $(G, k) \in$ VertexCover. We construct a corresponding assignment for the variables of φ as follows. For each variable, we set its true/false value so that the label of the variable-gadget vertex *in the cover* C is true. In other words, if the variable-gadget vertex labeled x_i is in the cover C, we set x_i to be true; otherwise (i.e., the vertex labeled $\overline{x_i}$ is in C), we set x_i to be false. Since each variable gadget has exactly one of its two endpoints in C, this is a well-defined assignment of the variables, with no undefined or contradictory assignments of variables.

We now argue that the assignment satisfies the original 3SAT formula φ, using the particular structure of G and the fact that C is a vertex cover of it. We show that for each individual clause in φ, our assignment satisfies that clause (and hence the entire formula). As already noted, the corresponding clause gadget has exactly two of its vertices in the vertex cover C, and its third vertex is *not* in C; call this vertex c. Recall that there is an edge from c to the variable-gadget vertex v that is labeled with the same literal as c is. Since this edge (c, v) is not covered by c (because $c \notin C$), it must instead be covered by v, i.e., $v \in C$. Because v is a variable-gadget vertex and $v \in C$, its label literal is *true* under our assignment, by definition. And since c is labeled by the same (true) literal as v, and that literal appears in the clause we are considering (by the design of our reduction), that clause is indeed satisfied, as claimed. This completes the proof.