34.5.6 Reduction strategies

From the reductions in this section, you can see that no single strategy applies to all NP-complete problems. Some reductions are straightforward, such as reducing the hamiltonian-cycle problem to the traveling-salesperson problem. Others are considerably more complicated. Here are a few things to keep in mind and some strategies that you can often bring to bear.

Pitfalls

Make sure that you don't get the reduction backward. That is, in trying to show that problem Y is NP-complete, you might take a known NP-complete problem X and give a polynomial-time reduction from Y to X. That is the wrong direction. The reduction should be from X to Y, so that a solution to Y gives a solution to X.

Remember also that reducing a known NP-complete problem X to a problem Y does not in itself prove that Y is NP-complete. It proves that Y is NP-hard. In order to show that Y is NP-complete, you additionally need to prove that it's in NP by showing how to verify a certificate for Y in polynomial time.

Go from general to specific

When reducing problem X to problem Y, you always have to start with an arbitrary input to problem X. But you are allowed to restrict the input to problem Y as much as you like. For example, when reducing 3-CNF satisfiability to the subset-sum problem, the reduction had to be able to handle *any* 3-CNF formula as its input, but the input to the subset-sum problem that it produced had a particular structure: $2n + 2k$ integers in the set, and each integer was formed in a particular way. The reduction did not need to produce *every* possible input to the subset-sum problem. The point is that one way to solve the 3-CNF satisfiability problem transforms the input into an input to the subset-sum problem and then uses the answer to the subset-sum problem as the answer to the 3-CNF satisfiability problem.

Take advantage of structure in the problem you are reducing from

When you are choosing a problem to reduce from, you might consider two problems in the same domain, but one problem has more structure than the other. For example, it's almost always much easier to reduce from 3-CNF satisfiability than to reduce from formula satisfiability. Boolean formulas can be arbitrarily complicated, but you can exploit the structure of 3-CNF formulas when reducing.

Likewise, it is usually more straightforward to reduce from the hamiltonian-cycle problem than from the traveling-salesperson problem, even though they are so similar. That's because you can view the hamiltonian-cycle problem as taking a complete graph but with edge weights of just 0 or 1, as they would appear in the adjacency matrix. In that sense, the hamiltonian-cycle problem has more structure than the traveling-salesperson problem, in which edge weights are unrestricted.

Look for special cases

Several NP-complete problems are just special cases of other NP-complete problems. For example, consider the decision version of the 0-1 knapsack problem: given a set of n items, each with a weight and a value, does there exist a subset of items whose total weight is at most a given weight W and whose total value is at least a given value V? You can view the set-partition problem in Exercise 34.5-5 as a special case of the 0-1 knapsack problem: let the value of each item equal its weight, and set both W and V to half the total weight. If problem X is NP-hard and it is a special case of problem Y, then problem Y must be NP-hard as well. That is because a polynomial-time solution for problem Y automatically gives a polynomial-time solution for problem X. More intuitively, problem Y, being more general than problem X, is at least as hard.

Select an appropriate problem to reduce from

It's often a good strategy to reduce from a problem in a domain that is the same as, or at least related to, the domain of the problem that you're trying to prove NP-complete. For example, we saw that the vertex-cover problem—a graph problem—was NP-hard by reducing from the clique problem—also a graph problem. From the vertex-cover problem, we reduced to the hamiltonian-cycle problem, and from the hamiltonian-cycle problem, we reduced to the traveling-salesperson problem. All of these problems take undirected graphs as inputs.

Sometimes, however, you will find that is it better to cross over from one domain to another, such as when we reduced from 3-CNF satisfiability to the clique problem or to the subset-sum problem. 3-CNF satisfiability often turns out to be a good choice as a problem to reduce from when crossing domains.

Within graph problems, if you need to select a portion of the graph, without regard to ordering, then the vertex-cover problem is often a good place to start. If ordering matters, then consider starting from the hamiltonian-cycle or hamiltonian-path problem (see Exercise 34.5-6).

Make big rewards and big penalties

The strategy for reducing the hamiltonian-cycle problem with a graph G to the traveling-salesperson problem encouraged using edges present in G when choosing edges for the traveling-salesperson tour. The reduction did so by giving these edges a low weight: 0. In other words, we gave a big reward for using these edges.

Alternatively, the reduction could have given the edges in G a finite weight and given edges not in G infinite weight, thereby exacting a hefty penalty for using edges not in G. With this approach, if each edge in G has weight W, then the target weight of the traveling-salesperson tour becomes $W \cdot |V|$. You can sometimes think of the penalties as a way to enforce requirements. For example, if the traveling-salesperson tour includes an edge with infinite weight, then it violates the requirement that the tour should include only edges belonging to G.

Design gadgets

The reduction from the vertex-cover problem to the hamiltonian-cycle problem uses the gadget shown in Figure 34.16. This gadget is a subgraph that is connected to other parts of the constructed graph in order to restrict the ways that a cycle can visit each vertex in the gadget once. More generally, a gadget is a component that enforces certain properties. Gadgets can be complicated, as in the reduction to the hamiltonian-cycle problem. Or they can be simple: in the reduction of 3-CNF satisfiability to the subset-sum problem, you can view the slack variables s_j and s'_j

as gadgets enabling each clause-labeled digit position to achieve the target value of 4.