Computer Science C63 Winter 2025
Scarborough Campus University of Toronto

Proving **NP**-completeness: where to start?

Vassos Hadzilacos

Often the hardest part about proving **NP**-completeness of a problem B is figuring out where to start: which of the many problems A we already know to be **NP**-complete should we try to polytime-reduce to B? Technically, any **NP**-complete problem is polytime-reducible to any other, so one might think that it does not matter. In practice, it matters a lot: some reductions are much easier and more natural than others, because some pairs of problems are intuitively more similar than others. Unfortunately there are no hard-and-fast rules here; the task is somewhat of an art, which is what makes it interesting and fun. But there are some things to keep in mind to make the search for the "right" source problem A more effective.

First, you need to understand clearly the problem B you are asked to prove is **NP**-complete: What are some small examples of yes-instances and of no-instances? Are there different kinds or interesting instances? "Edge" cases? It is impossible to do a reduction from or to a problem that you haven't really understood. So, playing around with small representative examples is an important starting point.

After you have established a reasonable familiarity with the problem you are trying to prove is **NP**-complete you can go over the list of problems you already know are **NP**-complete and start looking for familiar patterns or analogies. The paragraphs below describe some of these patterns to help narrow down the search for the "right" source problem A.

Is the problem effectively asking to ***minimize*** something or to ***maximize*** something? This is sometimes not explicit in the statement of the problem since, by definition, **NP** consists of decision, not optimization, problems. In many cases, however, an optimization problem is clearly lurking behind the decision problem. For example, in the Vertex Cover problem we ask whether there are k nodes that "touch" all the edges; the bigger the k the easier it is to satisfy this requirement: If there are k nodes that do the job then clearly any superset of these nodes also works. So, Vertex Cover is the decision version of a minimization problem: find a vertex cover with the fewest possible number of nodes. In contrast, consider the Clique problem. Here the question is whether a graph G contains k pairwise connected nodes. Since the existence of a k-clique immediately implies the existence of a clique with fewer nodes, the optimization problem hiding behind the Clique decision problem is a maximization problem: find a clique with the maximum possible number of nodes. By determining if the problem B we want to prove **NP**-complete is the decision version of a minimization or maximization problem we can often focus our search for the right source NP-complete problem A to one that is also a minimization or maximization problem. Minimization? Maybe try a reduction from Vertex Cover or Traveling Salesman. Maximization? Maybe try a reduction from Independent Set or Clique.

Some **NP**-complete problems are not disguised optimization problems, but pure ***search*** problems: we are looking for something, not necessarily an optimal thing. Examples are Satisfiability, the Hamiltonian Circuit Problem, and 3-Dimensional Matching. Some search problems ask for an ***ordering*** of objects; in such cases a reduction from the Hamiltonian Circuit or Path or from the Traveling Salesman Problem might be natural choices to consider. Other search problems ask for a ***partitioning*** of objects into non-overlapping groups; in such cases a reduction from 3-Dimensional Matching, Exact Set Cover, or Partition might be natural choices to consider. Other **NP**-complete problems involve ***numbers***, and in that case Subset Sum and Integer Linear Programming are natural choices to consider.

Some **NP**-complete problems fit under multiple of the categories described above. For example, the Traveling Salesman Problem is both a minimization problem and an ordering problem, so it could be the choice for the source problem A of a reduction to a problem B in either category. This seems to multiply the choices but sometimes it can narrow them down: For example, Colouring is a partitioning problem (different groups of objects classified by a colour) but also a minimization problem: classify the objects in the smallest possible number of groups. So if our problem B has both features, reducing Colouring to B might be a natural choice to consider.

Some problems are just versions of others looked at differently. For example, Independent Set vs. Clique vs. Vertex Cover; or Colouring vs. Covering by Cliques. Some problems are just generalizations of known **NP**-complete problems. For example: Given a propositional formula F and a number k, is there a truth assignment that satisfies at least k clauses of F? If we can do this, we can obviously solve Satisfiability by choosing k to be the number of clauses in F. Or, as we have seen, Exact Cover is (obviously) a generalization of Exact Cover by 3-Sets. Other problems can become generalizations by adding some stuff to a known **NP**-complete problem. For example: Does a graph have two cliques of k nodes each? We can easily reduce the clique problem to this by adding to the graph k new nodes connected into a clique. In some cases we can prove that B is **NP**-complete by adding some information to the known **NP**-complete problem A. For example, we reduced the Subset Sum problem to the Partition problem by adding just one number to the sequence of the given Subset Sum instance.

If nothing looks familiar, 3SAT or other **NP**-complete versions of Satisfiability are good choices to try. This is particularly relevant if the target problem B has the flavour of asking for a selection of objects from various sets; the clauses of a CNF formula could be the sets, the literals in each clause the objects in each set, and the selection of an object from each set could be the job of a truth assignment that satisfies the formula — and therefore satisfies some literal in each clause. Satisfiability is a very adaptable problem; it is not an accident that it was the first to be shown **NP**-complete! But sometimes there are other familiar **NP**-complete problems that are much closer to our target, and then going through Satisfiability may be more circuitous than is needed.

These are some of the tricks of the trade. Much can be gained from experience, and I encourage you to do as many reductions as you can. Your textbook has a large number of exercises as do all good textbooks on algorithms that treat NP-completeness. (This subject is sometimes part of a course on algorithms, rather than a course on computability and complexity like ours.)

Much wasted effort can be avoided by being careful about very basic things: Are we doing the reduction in the right direction? Are we keeping in mind that the reduction need not be onto? That is, we don't need to have a reduction that produces all instances of the problem B we want to prove is **NP**-complete; producing only a special case suffices. Sometimes restricting our attention to a special case of B makes the problem closer to a familiar **NP**-complete problem, thus making the reduction much easier to see. For example consider the so-called "bin-packing problem": We are given the weights $w_1, w_2, \ldots, w_n$ of n items, and we want to know if we can fit all of them into k bins each of which can carry items whose total weight is at most w. This is an **NP**-complete problem. It is much easier to think of a reduction from a familiar **NP**-complete problem if you consider only the special case of $k = 2$. (The fact that bin packing is also a problem about numbers helps to further narrow the search for the "right" source problem A which to reduce to bin-packing. Think about this!) So, sometimes a useful way to approach the question "where to begin" is to see if restricting some of the parameters of the target problem makes it closer to a familiar **NP**-complete problem.

The above advice can often lead to more than one candidate. So part of the skill that you need to develop, which also comes with experience, is to be patient in your efforts and learn when to abandon one approach and try another: if you feel you are making progress, stick to it; if not, after a while of honest effort, it is time to see if a different approach works.