Lecture 23: NP completeness and polynomial time reduction

Our next example is the independent set problem. Given a graph, find the largest subset of nodes of \( V \) s.t. no two nodes in the set are connected by an edge of \( G \). Again, we can convert this to a search problem, and it is easy to see the search problem is in NP. Now we look at the vertex cover problem. Given graph \( G \), does there exist \( k \) nodes in \( G \) s.t. each edge in \( G \) is touched (covered) by one of the \( k \) nodes. This again, is in NP.

Similarly, I showed in class that another problem, Clique, is also in NP. Note: you should know what each problem is about. This can be useful when later you want to use them in the NP-completeness proof.

Now is one of the most important concepts in this chapter: polynomial-time reduction. Suppose we have a black box efficient code that can solve problem \( X \), and we can use it to solve problem \( Y \). Which is harder: \( X \) or \( Y \)? Since \( X \) lets solve \( Y \), so \( X \) is at least as hard as \( Y \). More precisely: can we solve any instance of \( Y \) (say a graph) using poly-time to prepare input and read output for black box for \( X \) and make polynomial number of calls to the black box of \( X \)? If yes, we say \( Y \) is poly-time reducible to \( X \), denoted \( s_{\leq p} Y \Rightarrow X \). Consequence: if \( X \) does have a poly-time algorithm, then \( Y \) also has a polynomial time algorithm. Conversely, if \( Y \leq_{p} X \), and \( Y \) can not be solved in poly-time, then \( X \) can not be solved in poly-time either.

We now show our first reduction for two problems in NP: independent set (IS) problem and vertex cover (VC) problem. Note we will use IS to denote both the IS problem and some independent set for a graph. Same usage of VC. We will show \( IS \leq_{p} VC \) and \( VC \leq_{p} IS \). Here is a useful observation. Given a graph \( G = (V, E) \), a subset \( S \) is an IS of \( G \) iff \( V - S \) is a VC. This can be easily verified by the definition of IS and VC. Check this out if you do not understand it. Thus, given a black box code for the VC problem, we will find solution for the following question on the IS problem: given a graph \( G = (V, E) \), does \( G \) have an IS with \( k \) nodes or more? Based on our previous observation, we note that this question can be easily answered by passing this question to the VC black box: does \( G \) (the same graph) contain a VC with \( n - k \) nodes or smaller? Here \( n = |V| \). Clearly the reduction is done in polynomial-time: we actually just make one call of the black box and the cost of preparing the input is none. I omit the other reduction, which is similar.

NP-complete: the hardest problems in NP. A problem is NP-complete if (1) it is in NP and (2) every problem in NP is polynomial-time reducible to it. In class, we went over the implication of NP completeness (1) if problem \( X \) is NP complete, and \( X \) has a polynomial time algorithm, then every (yes every) problem in NP has polynomial-time algorithm. (2) if any problem in NP has no polynomial time algorithm, the each NP complete problem has no polynomial algorithm.

Does NP complete problem exist? The 1971 Cook-Levin theorem says yes: circuit satisfiability problem (CIRCUIT-SAT) is NP complete. Our textbook has a nice introduction to CIRCUIT-SAT. Read it if you still do not understand it.

From now on, we have one starting problem: CIRCUIT-SAT. Now how to prove problem \( X \) is NP complete? The recipe for NP-completeness: (1) show \( X \) is in NP (2) Find a known NP complete problem \( Y \), (3) describe how to convert instances for \( Y \) into an instance for \( X \) (4) Prove the instance for \( Y \) can be efficient solved iff \( X \) can be solved. (5) Ensure the conversion can be done in poly-time.

Lecture 24: NP completeness

We first went over the proof of 3-CNF-SAT (or simply 3SAT) is NP complete (where each clause has exactly 3 literals). Read the textbook for more details. But here is the important points. We need to first show 3SAT is in NP. Then, we will reduce from CIRCUIT-SAT. The basic idea is to construct
equivalent formula for each of three types of logical gates in the circuits. Read the textbook for more details. Finally, some of reductions will lead to clauses with only two literals. But in class, I showed how to convert such clauses into equivalent clauses with 3 literals each. Again, read the book if you still miss something.

We now move on to graph-theoretic problems. Recall to show problem $X$ is NPC: (1) $X$ is in NP, (2) find a NPC problem that is polynomial time reducible to $X$. Our first example is CLIQUE problem. A clique is a fully connected subgraph. The CLIQUE problem (the decision version) is: does given graph $G$ have a clique of size $k$ or more? Clearly, CLIQUE is in NP: its certificate is a set of nodes and we can easily check whether this set of nodes forms a clique or not. We now reduce the 3SAT problem to CLIQUE. I will not repeat the proof which is well documented in the textbook. Refer to the textbook for more details.

We now show the INDEPENDENT-SET (IS) problem, is NP complete. We will reduce CLIQUE to IS. The basic idea is to complement $G$: when there is an edge between two nodes, remove it; otherwise add an edge. We call the resulting graph $G'$. Claim: $G$ has a clique of size $k$ iff $G'$ has IS of size $k$. First suppose $G$ has clique $S$ of size $k$. Then $S$ is an IS for $G'$. This is because no edges in $S$ are between two nodes of $G'$ (since $S$ is a clique in $G$). The other direction is also easy to see. With this observation, we can easily decide the answer for the CLIQUE by asking the black box of IS.

We will continue with even more reduction in the next lecture.