Lecture 24: NP completeness

We now move on to graph-theoretic problems. Recall to show problem \( X \) is in NP, (1) \( X \) is in NP, (2) find a 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.

Our second example is VERTEX-COVER (VC). A vertex cover is a subset of nodes of a graph \( G \) such that all edges in \( G \) have at least one node in the subset. The VC problem is: does \( G \) have a VC of size \( k \) or fewer? It is again easy to see VC is in NP. We can directly reduce CLIQUE to VC, as done in the textbook. But it is slightly easier to first show a related problem, INDEPENDENT-SET (IS), is NP complete. Since we have seen before VC and IS can be reduced to each other, we will then know VC is also NP complete. 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 VC of size \( n - k \). First suppose \( G \) has clique \( S \) of size \( k \). Then \( V - S \) is a VC 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.

Now HAM-CYCLE. This reduction is more complex. In class, I briefly explained the most important part of the reduction. Read the textbook for more details.

Finally, I showed the traveling salesman problem (TSP) is NP complete. Given a complete weighted graph (cost function \( c \)), we want a tour of all cities (each city visited exactly once and return to the starting city) with cost at most \( k \). Now give the NP-complete proof. First TSP is in NP: check a proposed tour can be easily done in polynomial-time (make sure you understand what it means). We now show HAM-CYCLE \( \leq \) \( p \) TSP. Given a graph \( G \) (may not be complete), we form a complete weighted graph \( G'(V,E') \): \( E' \) contain all \((i,j)\), where we assign \( c(i,j) = 0 \) if \((i,j)\) in \( G \) an 1 otherwise. Claim: \( G \) has a Hamiltonian cycle iff \( G' \) has TSP tour of cost 0. First, if \( G \) has a Hamiltonian cycle, then all the edge cost is 0, and so the tour costs zero. On the other hand, if \( G' \) has TSP cost of 0, then each edge will have cost 0 (and thus the edge is in \( G \)). So this tour is Hamiltonian cycle. Therefore, we simply ask the black box of TSP one question: does the input \( G' \) has TSP tour of cost 0 or less? And we can then determine the solution for the HAM-CYCLE problem.

Lecture 25: NP-completeness.

We now move to numerical problems (recall Knapsack problem). We start with the SUBSET-SUM problem. Given finite set \( S \) of positive integers, and an integer \( t \), we ask whether there exists a subset of integers of \( S \) that sum to \( t \). Clearly, the SUBSET-SUM is in NP. We now show VERTEX-COVER is polynomial-time reducible to SUBSET-SUM.

In VC, we have a graph \( G \) and an integer \( k \), and the question is: does \( G \) has a vertex cover of size \( k \) or smaller? To use a black box of SUBSET-SUM, we will construct integers \( S \), and \( t \) from \( G \) s.t. \( G \) has a VC of size \( k \) or smaller iff subset of \( S \) add up to \( t \). Let nodes in graph labeled by 1, 2, 3, . . . \( n \). We will create one integer \( a_i \) per node \( v_i \) and one integer \( b_{i,j} \) for edge \((v_i, v_j)\). Then selected \( a_i \) correspond to a vertex cover (denoted \( C \)), and selected \( b_{i,j} \) are those with precisely one node in the vertex cover. We define integers in a matrix, where a row corresponds to an integer. Integers are treated as base-4. We add a special column as the first column (the most significant bit), which is 1 if the row is for \( a_i \), and 0 for \( b_{i,j} \). Then we have one column for each edge \((u,v)\). For integer \( a_i \), it has value 1 if node \( v_i \) is one of the two nodes for the edge corresponding to the column. \( b_{i,j} \) has precisely 1 at the column corresponding to the edge \((v_i, v_j)\).

Finally, \( t = k * 4^{|E|} + 2 \sum_{i=0}^{4^{|E|}-1} 4^i \).
To see why it works, we show two things. First, given a vertex cover \( C \), we select the following integers. We pick all corresponding \( a_i \) for \( v_i \in C \), and \( b_{i,j} \) if exactly one of \( v_i, v_j \) is in \( C \). The selected \( k \) integers (corresponding to \( a_i \)) have MSB of 1, which add to \( k|E| \). For the rest of edges, we have exactly two 1 selected for that column. Check this out if you do not understand it yet. On the other hand, suppose we have a subset of integers summing to \( t \). We note that there is carry in all columns (except the MSB). It follows easily that the corresponding nodes (for the selected \( a_i \) integers) must form a vertex cover. Check it if you do not yet understand.

Now recall the KNAPSACK problem. Can we have a polynomial time algorithm? Note \( O(nW) \) (from the dynamic programming) is not a polynomial-time algorithm since \( W \) can be very large. Suppose KNAPSACK has a black box polynomial-time algorithm. Now we can solve SUBSET-SUM using this KNAPSACK black box. How? Given a list of \( n \) integers (denoted as \( x_i \)) from the SUBSET-SUM, we create items from integers. We have \( (v_i, w_i) = (x_i, x_i) \). Note we want to find out whether a subset of \( x_i \) sum to \( t \). That is, value/weight are the same as the given integer \( x_i \) for the item corresponding to \( x_i \). Now it is easy to see the SUBSET-SUM problem has a solution if KNAPSACK problem has a solution for the constructed items where the total selected weights is no more than \( t \) and the value is at least \( t \). Check to understand why this is the case.

**Lecture 26: How to deal with NP complete problems**

Often we must work on NP-complete problems, even when polynomial-time algorithms are unlikely obtainable. We can take several approaches.

- Efficient algorithm for special case of the problem.
- Approximation algorithm: we may not find optimal solutions, but we can find a solution not very far from it.

First, some NP complete problems become easier when we restrict the problem instances on special cases. As an example, we consider the problem of finding independent set on a tree. We first note that if there is an isolated node \( u \) (i.e. has no edge incident on \( u \)), we can add \( u \) to the independent set. A key property of a tree is that there exists a leaf \( u \) (only connects to a node \( v \)). For INDEPENDENT-SET problem, we prefer \( u \). To see this, we consider an independent set \( C \). if neither \( u \) nor \( v \) is in \( C \), we can simply add \( u \) and this will lead to a (larger) independent set (why the enlarged set remains an independent set?) On the other hand, if \( v \in C \), we can have another independent set \( C' = (C - \{v\}) \cup \{u\} \) which remains an independent set (why?). Thus, we know for a given tree \( T(V, E) \) and \( u \) is a leaf, then there exists an IS includes \( u \). This gives a greedy algorithm: each time, we look for a leaf \( u \) (which must exist when there is no cycle), include \( u \) in the independent set, and remove \( u \) and \( v \) (where there is an edge \( (u, v) \)). We repeat until the graph is empty. Note: what we got becomes a forest (not a single tree but several disjoint trees). But what we proved still work: there still exists a leaf in a set of trees.

Now a quick tour of approximation algorithm which can help when exact optimal solution (say min) is hard to find. and we can find a solution that is not very far from optimal. Approximation ratio is: for a solution \( C \), and optimal solution \( C^* \), if we show \( C/C^* \leq p \), we say approximation ratio is \( p \). Our example is the VERTEX-COVER problem, where we give a 2-approximation algorithm. We omit the details here. See the textbook (chapter 35) for details on correctness proof.