Pre-tutorial questions

Do you know the basic concepts of this week's lecture content? These questions are only to test yourself. They will not be explicitly discussed in the tutorial, and no solutions will be given to them.

1. Reduction.
   (a) What is a polynomial time reduction?
   (b) Is polynomial time reductions transitive?
   (c) What are the three types of standard reductions?
2. Classes
   (a) What's the definition of the class P?
   (b) What's the definition of the class NP?
   (c) Is it known if P=NP?
   (d) How do you prove that a problem is in NP?
   (e) What's the definition of the class NP-complete?
3. How do one prove that a problem is NP-complete?

Tutorial

Problem 1
The input to the *set cover problem* is a collection of subsets $S_1, S_2, \ldots, S_m$ of some universal set U, and a number t. The problem is to determine if there are t sets $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$ whose union equals U; that is, $\cup_{j=1}^{t} S_{i_j} = U$.

1. Prove that the set cover problem is NP-hard.
2. Argue why the Set Cover problem is more general than the Vertex Cover problem.

Solution: We did this in the lecture.

1. We know that the vertex cover problem is NP-complete. We reduce vertex cover (VC) to set cover (SC). Recall that the vertex cover problem is "Given an undirected graph G and a number k, does G have a vertex cover of cardinality k". Given (G, k) we construct the following instance of Set Cover:

   (a) U is the edge set of G

   (b) For each vertex u of G, we create a set S_u containing the edges incident on u.

   (c) Set $t = k$

   There is a one-to-one correspondence between vertex covers in G and set covers in our set system; furthermore, the correspondence preserves the cardinality of the covers. Thus, "yes instances" of VC are mapped to "yes instances" of SC, and "no instances" of VC are mapped to "no instances" of SC. Thus Vertex Cover$\leq_P$ Set Cover.

2. The above reduction shows that vertex cover is just a special case of the set cover problem. Vertex Cover is the special case of Set Cover when every item appears in exactly two subsets (one for each endpoint of an edge). Note that every time we generalize a problem that is already known to NP-hard (in this case VC), we are bound to get a problem (in this case SC) that is at least NP-hard.

Problem 2

Consider a generalization of the interval scheduling problem where requests are not a single interval but a collection of d intervals. The objective is to choose a maximum number of requests that do not create any conflicts. More formally, we have n requests. The ith request has associated intervals $I_1^i, I_2^i, \ldots, I_d^i$. A subset S of requests is feasible if for all $i, j \in S$, and $a, b \in [1, d]$ we have $I_a^i \cap I_b^j = \emptyset$. The objective is to select a maximum size set of feasible requests. Show that this problem is NP-hard.

Solution: We reduce independent set to the decision version of this problem. Recall that an instance (G, k) of the independent set problem is to determine whether there is an independent set of cardinality k. Given (G, k), we label the edges $e_1, \ldots, e_m$ of the graph G. Now we construct an instance of the multiple interval scheduling problem as follows: For each vertex u in G we create a request r_u in our instance; if e_i is incident on u we let $[i, i+1]$ be one of the time intervals associated with r_u.
Let S be a set of vertices in G and J_S the set of requests induced by those vertices. It is easy to see that S is independent if and only if J_S is feasible. Therefore, (G, k) is a "yes instance"if and only if we can find a feasible schedule with k requests. Thus Independent Set$\leq_P$ Interval Scheduling with multiple Intervals.

Problem 3

In the *Degree* Δ *Spanning Tree problem* we are given a graph $G = (V, E)$ and we have to decide whether there is a spanning tree of G whose maximum degree is at most Δ. (Note that Δ is not part of the input, but rather part of the problem definition.) It is known that the Degree 2 Spanning Tree problem (D2ST) is NP-complete. Using this fact, prove that Degree Δ Spanning Tree is NP-complete for all fixed $\Delta > 2$ (DΔST).

Solution: The problem is clearly in NP: The certificate would be the tree itself. It is trivial to check in polynomial time that the maximum degree is at most Δ and that it spans all the vertices.
Given an instance $G = (V, E)$ of D2ST we create an instance $G' = (V', E')$ of DΔST. Keep the vertices and edges of G; add, for every $u \in V$, $\Delta - 2$ dummy nodes $d_1^u, \ldots d_{\Delta-2}^u$ and connect them to u. In other words,

$$V' = V \cup \{d_i^u \mid u \in V \text{ and } 1 \leq i \leq \Delta - 2\}, \text{ and}$$
$$E' = E \cup \{(u, d_i^u) \mid u \in V \text{ and } 1 \leq i \leq \Delta - 2\}.$$

Given a D2ST of G we can attach all the dummy nodes (thus increasing the degree of every vertex in V by $\Delta - 2$) to get a spanning tree of G' having degree Δ. The other way around also holds, we can trim the dummy nodes (thus reducing the degree of every vertex in V by $\Delta - 2$) to get a spanning tree of G having degree 2. Therefore, G is a "yes instance" of D2ST if and only if G' is a "yes instance" of DΔST. Thus D2ST$\leq_P$ DΔST.

Problem 4
Given a graph $G = (V, E)$, a subset of vertices X and a number k, the Steiner Tree problem is to decide whether there is a set $S \subseteq V$ of size at most k such that $G[X \cup S]$ is connected. Consider the following reduction from 3-SAT to the Steiner Tree problem.

Let $\phi = C_1 \wedge \cdots \wedge C_m$ be a boolean formula over variables $x_1, \ldots, x_n$ such that each clause C_i is the disjunction of three literals. We define a graph $G = (V, E)$ and a target k based on ϕ:

1. For each clause C_i we create a vertex $u_i \in X$; for each variable x_j we create two vertices v_j^T and v_j^F that belong to $V \setminus X$. Finally, we add a dummy node $d \in X$.
2. For each clause C_i, if C_i contains the literal x_j then we create the edge (u_i, v_j^T), while if C_i contains the literal $\neg x_j$ then we create the edge (u_i, v_j^F). Finally, we connect d to every v_j^T and v_j^F.
3. We set the target k to be n.

Prove that the reduction is broken. That is, show a 'Yes' instance being mapped to a 'No' instance or vice versa.

Solution: While Yes instances of 3-SAT will all be mapped to Yes instances of the Steiner Tree problem, a No instance of 3-SAT may be mapped to a Yes instance of the Steiner Tree problem because there is nothing stopping us from picking both v_j^T and v_j^F in S. For example, the formula $\phi = (\neg x_1 \vee \neg x_1 \vee \neg x_1) \wedge (x_1 \vee x_1 \vee x_1) \wedge (x_1 \vee \neg x_1 \vee x_2)$ is infeasible but it maps to an instance where $\{v_1^T, v_1^F\}$ connect the terminals.

Problem 5
Given a graph $G = (V, E)$, a distinguished subset of vertices $X \subset V$ and a number k, the Steiner Tree problem is to decide whether there is a set $S \subseteq V$ of size at most k such that $G[X \cup S]$ is connected.

Prove that this problem is NP-complete.

Solution: The problem is clearly in NP. The certificate is the set S of k nodes whose addition to X connects the set, which is trivial to check in polynomial time.
To show that it is NP-hard, we reduce 3-SAT to it. Recall that an instance of 3-SAT consists of a formula $\phi = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, \ldots, x_n$ such that each clause C_i is the disjunction of three literals. The question is whether there is a truth assignment that satisfies all clauses.
We define a graph $G = (V, E)$ and a target k based on ϕ:

1. For each clause C_i we create a vertex $u_i \in X$; for each variable x_j we create a vertex $v_j \in X$; we also create a dummy vertex $d \in X$. Additionally, for each variable x_j we create two vertices v_j^T and v_j^F that belong to $V \setminus X$.

2. For each variable x_j, we create the edges (v_j^T, v_j) and (v_j^F, v_j) to E. For each clause C_i, if C_i contains the literal x_j then we create the edge (u_i, v_j^T), while if C_i contains the literal $\bar{x}_j$ then we create the edge (u_i, v_j^F). Finally, we connect d with every v_j^T and v_j^F.

3. Finally, we set the target k to be n.

Notice that in order to connect v_j with the rest of X we must select either v_j^T or v_j^F into the set S; since $k = n$, exactly one of the them must be chosen for each $j = 1, \ldots, n$. There is a one-to-one correspondence between truth assignments for the variables and a choice between v_j^T and v_j^F for each $j = 1, \ldots, n$, which will define our set S. It is not difficult to show that a truth assignment is satisfying if and only if for the corresponding S, the graph $G[X \cup S]$ is connected. (Notice that if we hadn't added the dummy node d the reduction would not work, why?)

Problem 6
Given a directed graph $G = (V, E)$ a feedback set is a set $X \subseteq V$ with the property that $G - X$ is acyclic. The Feedback Set problem asks: Given G and k, does G have a feedback set of size at most k? Prove Feedback Set $\leq_P$ Set Cover.

Solution: This problem has an easy and correct solution, and a complicated and incorrect solution. The correct solution is to point out that Feedback Set is clearly in NP and that because Set Cover is NP-complete all problems in NP reduce to it, in particular Feedback Set.
The incorrect solution attempts to directly map instances of Feedback Set to Set Cover in the "obvious" way: For each simple cycle in G, we create an element to be covered; for each vertex u in G, we create a set containing all simple cycles going through u. Clearly, every feedback set X induces a set cover of the same cardinality and vice-versa. However, this reduction does not run in polynomial time since there could be an exponential (or worse) number of cycles in the graph!

Problem 7
Decision problems involve answering yes/no questions. For example, the independent set problem is "Given an undirected graph G and a number t, does G have an independent set of size t?". In practice we are usually interested in solving optimization problems. For example, the maximum independent set problem is "Given an undirected graph G, find the largest in independent set in G".

Assuming you have a polynomial time algorithm for the independent set problem, show how to solve the maximum independent set problem.

Solution: Let $\mathcal{A}$ be the algorithm for the decision version of the independent set problem. Let $G = (V, E)$ be our input graph, and let t^* be the cardinality of the largest independent set in G.
First, we find the value of t^* using binary search and $\mathcal{A}$; this takes $\log_2 n$ calls to $\mathcal{A}$. Having computed t^*, we pick an arbitrary vertex u and call $\mathcal{A}$ on $(G - u, t^*)$. If it returns "yes" then we know that there is a maximum independent set that does not use u; otherwise, if it returns "no" we know that any maximum independent set must use u. Therefore, if the answer is "yes" we can ignore u and find a maximum independent set in $G - u$. On the other hand, if the answer is "no" we can find a maximum independent set in $G - u - N(u)$ and then add u to it.
Notice that we perform polynomial amount of computation plus we make a polynomial number of calls to $\mathcal{A}$. Thus if $\mathcal{A}$ runs in polynomial time, so does our algorithm.

Problem 8
Prove that CLIQUE is NP-complete by using a reduction from 3-SAT.

Solution: CLIQUE is clearly in NP. Why?
We will reduce 3-SAT to CLIQUE. Specifically, given a 3-CNF formula F of m clauses over n variables, we construct a graph as follows. First, for each clause c of F we create one node for every assignment to variables in c that satisfies c. E.g., say we have:

$$F = (x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee x_4) \wedge (x_2 \vee x_3) \wedge ...$$

Then in this case we would create nodes like this:

$$\begin{array}{lll} (x_1 = 0, x_2 = 0, x_4 = 0) & (x_3 = 0, x_4 = 0) & (x_2 = 0, x_3 = 0)... \\ (x_1 = 0, x_2 = 1, x_4 = 0) & (x_3 = 0, x_4 = 1) & (x_2 = 0, x_3 = 1) \\ (x_1 = 0, x_2 = 1, x_4 = 1) & (x_3 = 1, x_4 = 1) & (x_2 = 1, x_3 = 0) \\ (x_1 = 1, x_2 = 0, x_4 = 0) & & \\ (x_1 = 1, x_2 = 0, x_4 = 1) & & \\ (x_1 = 1, x_2 = 1, x_4 = 0) & & \\ (x_1 = 1, x_2 = 1, x_4 = 1) & & \end{array}$$

We then put an edge between two nodes if the partial assignments are consistent. Notice that the maximum possible clique size is m because there are no edges between any two nodes that correspond to the same clause c. Moreover, if the 3-SAT problem does have a satisfying assignment, then in fact there is an m-clique (just pick some satisfying assignment and take the m nodes consistent with that assignment). So, to prove that this reduction (with $k = m$) is correct we need to show that if there isnt a satisfying assignment to F then the maximum clique in the graph has size $< m$. We can argue this by looking at the contrapositive. Specifically, if the graph has an m-clique, then this clique must contain one node per clause c. So, just read off the assignment given in the nodes of the clique: this by construction will satisfy all the clauses. So, we have shown this graph has a clique of size m if and only if F was satisfiable.
Also, our reduction is polynomial time since the graph produced has total size at most quadratic in the size of the formula F ($O(m)$ nodes, $O(m^2)$ edges). Therefore CLIQUE is NP-complete.