NP-completeness

To really understand NP-completeness, you will need to read Chapter 36. This handout summarizes the basic concepts we discussed in the first class on NP-completeness.

For the last few months, we have seen many different problems, all of which could be solved fairly efficiently (we were able to design fast algorithms for them). We are going to define an *efficient algorithm* as one that runs in worst case polynomial time – $O(n^c)$ for an input of size n, where c is a fixed constant. However, there are many problems for which the best known algorithms take exponential time in the worst case – these problems arise in various contexts, graphs, layout, scheduling jobs, logic, compiler optimization etc. to name a few.

To make life simple, we will concentrate on rather simple versions of these problems, which are the decision versions of these problems. For example, rather than asking for the shortest path between a pair of vertices in an unweighted graph, we could ask the question: given a graph G and an integer K, is there a path from s to v of length at most K? This question has a simple YES/NO answer. Note that we do not even ask for the path itself! (This was just to illustrate the concept of decision problems, this particular problem is not NP-complete! In fact, we could run BFS, find out the shortest path length from s to v and then output YES/NO appropriately.) It should be clear that if we wanted to determine the length of the shortest path, we could ask "Is there a path of length 1", if the answer was "NO" we could ask "Is there a path of length 2" and so on until we determine the length of the shortest path. Since the path length varies from 0 to $n-1$, asking n questions is sufficient. (Notice that we could also use binary search to find the length of the shortest path.)

Let $\mathcal{P}$ be the class of problems that can be solved in polynomial time. In other words, given an instance of a decision problem in this class, there is a polynomial time algorithm for deciding if the answer is YES/NO.

Let $\mathcal{NP}$ denote the class of problems for which we can *check* a proposed YES solution in polynomial time. In other words, if I claim that there is a path of length at most K between s and v and I can give you a proof that you can check in polynomial time, we say that the problem belongs to $\mathcal{NP}$. Clearly, this problem is in $\mathcal{NP}$, but its also in $\mathcal{P}$. There are many other problems in $\mathcal{NP}$, that are not known to be in $\mathcal{P}$. One example is LONGEST PATH: given a graph G, and an integer L, is there a simple path of length *at least* L from s to v? Clearly, this problem is in $\mathcal{NP}$, but not known to be in $\mathcal{P}$. (To show that it is in $\mathcal{NP}$, I can give you a path of length at least L and in polynomial time it can be checked for validity – namely, that it has length at least L and is a simple path.) Usually we do need to argue that problem is in $\mathcal{NP}$ since there are problems that are not in $\mathcal{NP}$ (more on this later).

Reductions: We say problem $X \propto Y$ (X reduces to Y) if we can show the following: if there is a polynomial time algorithm for problem Y and we can use it to design a polynomial time algorithm for problem X, then X can be reduced to solving Y.

NP-completeness: A problem Q is said to be NP-complete, if we can prove (a) $Q \in \mathcal{NP}$ and (b) $\forall X \in \mathcal{NP} \quad X \propto Q$.

The notion of NP-completeness tries to capture the "hardest" problems in the class $\mathcal{NP}$. In other words, the way we have defined the class of NP-complete problems, if *ever* we were able to find a polynomial time algorithm for Q, an NP-complete problem, we can then find a polynomial time algorithm for *any* problem X that is in the class $\mathcal{NP}$ (and hence for every NP-complete problem). This follows by the definition of NP-completeness.

The next question is: do NP-complete problems even exist? At first glance it appears pretty hard to show that there is a general way to reduce ANY problem in $\mathcal{NP}$ to problem Q. We need to do this to prove that Q is NP-complete. Cook did precisely that in 1971. He showed that the following problem is NP-complete.

SATISFIABILITY PROBLEM: Given n boolean variables $X_1, \ldots, X_n$ (each can be TRUE or FALSE), is there a way to assign values to the variables so that a given boolean formula evaluates to TRUE? The boolean formula could take the form:
$(X_i \cup \overline{X_j} \cup X_k) \cap (\overline{X_p} \cup X_m \cup \overline{X_k}) \cap (X_j) \cap (X_j \cup X_i) \ldots \cap (X_p \cup X_k)$.

Cook's theorem: COOK proved that the SATISFIABILITY problem is NP-complete.

It is easy to see that the problem belongs to the class $\mathcal{NP}$. Understanding the proof requires knowledge of Turing Machines and a more formal description of what an algorithm is.

Now that we have one problem Q that is NP-complete, our task is much easier. If we can prove that $Q \propto Z$ AND $Z \in \mathcal{NP}$ then we claim that Z is NP-complete. If we show that $Q \propto Z$ then we have shown that if Z can be solved in polynomial time then so can Q. But if Q can be solved in polynomial time and since it is NP-complete, we can solve any problem $X \in \mathcal{NP}$ in polynomial time! Thus we have shown $\forall X \in \mathcal{NP} \quad X \propto Z$, and thus established that Z is NP-complete.

We now discuss a few specific NP-complete problems. All of these problems can be proven to be NP-complete by reducing SATISFIABILITY (a known NP-complete problem) to each of them. The proofs for some of them are complex. For fun, I will outline the reduction from SATISFIABILITY to CLIQUE in class.

HAMILTONIAN CYCLE PROBLEM: Given a graph $G = (V, E)$ is there a cycle that visits each vertex exactly once? (Notice that the closely related Euler tour problem, where we desire a tour (or cycle) that visits each edge exactly once can be solved in polynomial time.)

CLIQUE PROBLEM: Given a graph $G = (V, E)$ and an integer K, is there a set $S \subset V$ such that $|S| = K$ and there is an edge between each pair of nodes in S.

TRAVELING SALESMAN PROBLEM: Given a matrix D of distances between n cities (assume entries in D are integers for convenience), and an integer L – Is there a traveling salesman tour of length at most L? (A traveling salesman tour visits each city exactly once.)

BIN PACKING: Given n items, each with a size s_i, and an integer K – can we pack the items into K bins of unit size?

Notice that there are problems that may not be in $\mathcal{NP}$. Given a graph G, and an integer L, is there exactly one path of length L from s to v? Even if i claim that the answer is YES, and show you a path of length L – how do I convince you that there is no other path of length L? So, we dont know if this problem is in $\mathcal{NP}$. I just wanted to point out that there are problems for which we cannot even check a proposed solution in polynomial time (at least for now). Maybe some day, we will be able to figure out a polynomial time algorithm (someone from this class might be able to!).