NP-Completeness Summary

CSC 140 – Advanced Algorithm Design and Analysis

A **decision problem** is a problem where every instance of the problem results in a yes or no answer. Some examples of decision problems:
– Given undirected graph G, is there a cycle in it that visits every vertex exactly once? (HAM-CYCLE)
– Given boolean formula ϕ, is there an assignment to its variables that makes it evaluate to true? (SAT)
– Given integer $k > 1$, is k composite? (COMPOSITE)

An algorithm is **polynomial time** if its runtime on problems of size n is $O(n^k)$ for some constant k.

A problem is in **P** if there is a polynomial-time algorithm for solving the problem. To show a problem is in P, simply write out an algorithm that solves each instance of the problem in polynomial time. If it's not obvious that your algorithm is correct or polynomial time, you should convince your reader that it is.

A problem is in **NP** if there is a polynomial-time algorithm for verifying "yes" answers of the problem. To show a problem is in NP, define what extra information is useful for verification, and write out an algorithm that takes an instance of the problem and the useful information and verifies that the instance deserves a "yes" answer. If it's not obvious that your algorithm is correct or polynomial time, you should convince your reader that it is.

Example: HAM-CYCLE is in NP. Here is a verifier of yes instances. Extra information helping verification is the vertex sequence forming the cycle.

```
HCVerify(G,extra)
  let v1, v2, ..., vn be the sequence of vertices in extra
  if {v1, v2, ..., vn} is equal to G's vertex set
     and (v1,v2), (v2,v3), (v3,v4), ..., (vn,v1) are in G's edge set return true
  else return false
```

A **reduction** from Problem A to Problem B is an algorithm that shows how to solve instances of Problem A using as a subroutine a solver of Problem B instances. A Problem B solver doesn't have to actually exist, this just shows that if one did, then it would enable a Problem A solver. It is a polynomial-time reduction if a polynomial-time Problem B solver would make the Problem A solver polynomial-time too ($A \leq_p B$).

A problem is **NP-Hard** if every problem in NP is polynomial-time reducible to it. The usual way to show a problem NP-Hard is to polynomial-time reduce some known NP-Hard problem to it.

A problem is **NP-Complete** if it is in NP and it is NP-Hard. The usual way to show a problem is NP-Complete is to show it is in NP (by providing a polynomial-time verifier) and NP-Hard (by reducing a known NP-Hard problem to it).

Example: CLIQUE is NP-Complete. First we show CLIQUE is in NP. The extra information this CLIQUE verifier requires is the list of vertices forming the clique.

```
CliqueVerify(G,k,extra)
  let v1, v2, ..., vn be the vertices in extra
  if v1, v2, ..., vn are k distinct vertices from G's vertex set
     and every pair from from v1, v2, ..., vn forms an edge in G's edge set return true
  else return false
```

Next we show that a known NP-Hard problem (in this case 3CNF-SAT) reduces to CLIQUE.

```
3CNFSolver(phi)
  for each term of phi create three vertices in G, one representing each literal in the term
  Add an edge between each pair of literals that come from different terms and don't contradict
  return CliqueSolver(G,k), where k is the number of terms in phi
```

A k-clique in G represents k literals in ϕ, one per term, that can simultaneously be true. Making all these literals true would make all terms true, thus making ϕ true. Any satisfying assignment of ϕ will have such a clique.