Solutions to Homework 5: NP-Completeness

Solution 1:

(i) *All NP-complete problems are solvable in polynomial time*: Yes. Every problem in NP is polynomially reducible to SAT, and SAT is reducible to every NP-hard problem. Therefore, a polynomial time solution to *any* NP-hard problem (such as 3Col) implies that every problem in NP can be solved in polynomial time. Since the set of NP-complete problems is a subset of NP, it follows that they are all solvable in polynomial time.

(ii) *All problems in NP, even those that are not NP-complete, are solvable in polynomial time*: Yes, for the same reason given part (a).

(iii) *All NP-hard problems are solvable in polynomial time*: This does not follow. Indeed, there may NP-hard problems that are much harder than problems in NP.

(iv) *All NP-complete problems are solvable in $O(n^9)$ time*: This does *not* follow. Transformations may increase the size of the input of a problem (by a polynomial amount). Thus, if a reduction from some other NP-complete problem, call it X, to 3Col, expands an input of size n into an input of size n^2, then using this reduction to solve problem X solution could take as much time as $O((n^2)^9) = O(n^{18})$ time. This is still polynomial, but not the same as $O(n^9)$.

(v) *No NP-complete problem can be solved faster than $\Omega(n^7)$ time in the worst case*: This does not follow. For example if 3Col $\leq_P A$, but the reduction runs in $O(n^7)$ time, then it is possible that A could be solved in $O(n)$ time, since this only implies an $O(n^7)$ solution to 3Col, which does not violate the lower bound.

Solution 2:

(a) The transformation of a CNF formula to 3-CNF involves three cases depending on the number of literals in the clause.

One literal: We simply replicate the literal three times. For example,

$$(x_i) \longrightarrow (x_i \vee x_i \vee x_i).$$

Two literals: We replicate the last literal. For example,

$$(x_i \vee x_j) \longrightarrow (x_i \vee x_j \vee x_j).$$

(Let me add that if having duplicate literals in a clause bothers you, it is possible to avoid this. In the 1-literal case you need two new variables and four clauses, and in the 2-literal case you need one new variable two clauses.)

Four or more literals: Suppose that the clause has $k \geq 4$ literals. We will replace this clause with $k-3$ new clauses that are joined with $k-3$ new *bridging variables*, which we denote by z_3 through z_{k-1}. We transform the clause $(x_1 \vee \cdots \vee x_k)$ to

$$(x_1 \vee x_2 \vee z_3)(\overline{z}_3 \vee x_3 \vee z_4)(\overline{z}_4 \vee x_4 \vee z_5) \cdots (\overline{z}_{k-1} \vee x_{k-1} \vee x_k).$$

Note that the bridging variables are not shared between clauses. Every clause will use a distinct set of z-variables.

To show correctness, we claim that if there is an assignment that satisfies the original clause, then we can assign values to the z-variables to satisfy the chain of clauses. Conversely, if a assignment does not satisfy the original clause, then no assignment to the z-variables will satisfy the chain.

First, suppose that there is an assignment that satisfies the original clause. Let x_i be the first variable that is true. Then the unique clause of the chain that contains x_i is immediately satisfied. For all the clauses that come before this, we satisfy them by setting their initial z-literal (z_j for $j \leq i$) to true. For all the clauses that come after this, we satisfy them by setting their final z-literal ($\overline{z}_j$ for $j > i$) to true (that is, we set z_j to false).

Conversely, suppose that an assignment does not satisfy the original clause, meaning that x_i is false for all i. Then since every variable z_j appears in both positive and negated form in every clause of the chain, no matter how we assign values to the z-variables, at least one clause is not satisfied, and hence the overall chain is not satisfied.

(b) If we apply this to formula F we have the following transformations of the individual clauses:

$$\begin{aligned}
(a \vee b \vee \overline{c} \vee \overline{d}) &\longrightarrow (a \vee b \vee z_1)(\overline{z}_1 \vee \overline{c} \vee \overline{d}) \\
(e \vee \overline{b}) &\longrightarrow (e \vee \overline{b} \vee \overline{b}) \\
(d) &\longrightarrow (d \vee d \vee d) \\
(\overline{a} \vee e \vee f \vee \overline{g} \vee \overline{c}) &\longrightarrow (\overline{a} \vee e \vee z'_3)(\overline{z'}_3 \vee f \vee z'_4)(\overline{z'}_4 \vee \overline{g} \vee \overline{c}).
\end{aligned}$$

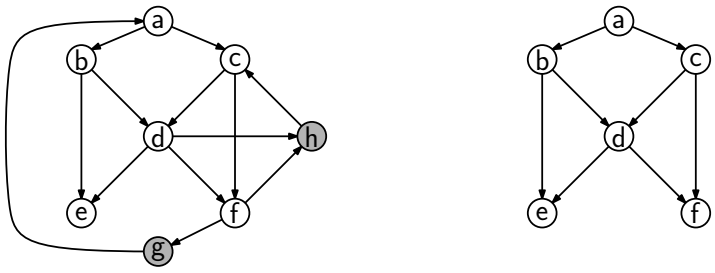
(c) The bridging trick shown above requires that we have at least three literals per clause (one to hold the original variable, one for the bridge to the previous clause, and one for the bridge to the next clause). While this is not a formal proof that no such transformation is possible, it is worth noting that 2SAT is solvable in polynomial time, whereas 3SAT is NP-complete.

Solution 3: This problem is more commonly referred to as the Feedback Vertex Set. A naive approach to showing that it is in NP-would be to guess the k vertices of V' that will constitute the VCC, and then verify that every simple cycle passes through at least one of these vertices. The problem is that there are exponentially many simple cycles, so you cannot enumerate them all.

You might wonder, couldn't we simply guess the cycle as well? If the guessed-at cycle passes through some vertex of V' then we output "no" and otherwise we output "yes." The problem is that this does not work according to the rules of nondeterministic computation. Recall that a nondeterministic computation succeeds of *any* sequence of guesses leads to an answer of "yes". Now, suppose that your graph does not have a VCC. The above program will guess some subset of k vertices and then may guess a simple cycle that does not pass through these vertices. It then

answers "yes" which implies that the global answer is "yes." But the global answer should be "no". (The problem is that it checked only one cycle, but it would need to check them all before answering "yes".)

The trick is to realize that the problem can be restated in a manner that makes the verification process much simpler. We can equivalently define a VCC to be a subset of vertices such that, after removing these vertices, the graph is acyclic. (For example, in Fig. 1(a) the vertices $\{g, h\}$ form a VCC, and in (b) their removal results in a DAG.) To see this, observe that if every simple cycle passes through some vertex of the VCC, then removing these vertices destroys all cycles. Conversely, if the removal of the vertices of the VCC results in an acyclic graph, then every cycle of the original graph must pass through at least one of these vertices. We can check that a digraph is acylic either by (1) running DFS and checking that there is no back-edge or (2) compute the strong components and check that every vertex is in its own strong component.



(a) (b)

Figure 1: Solution to Problem 3.

Solution 4: To prove that the LDC problem is NP-complete, we need to show the following two items.

LDC $\in$ NP: Given a graph G and integer k, we guess k vertices of G. We then compute the degrees of all the vertices of G and compute the median of this set. (This can be done in $O(n + m)$ time by inspecting the adjacency lists of all the vertices and using a fast selection algorithm.) We check that every one of the guessed vertices has degree less than or equal to the median degree, and that each of these vertices is adjacent to all the others. If both of these are satisfied, we output "yes," and otherwise we output "no." If G has an LDC, then one of these guesses will succeed in identifying the LDC and we will output "yes." If not, none of the guesses will work, and we will output "no."

LDC is NP-hard: We will show that the standard clique problem is polynomially reducible to LDC (Clique $\leq_P$ LDC). The idea is to artificially increase the median degree of G so that any valid clique in the original graph is an LDC in the modified graph. We want to do this without accidentally creating an LDC.

Suppose that we are given an input graph G and integer k. Let n be the number of vertices in G, and let us make the (trivial) assumption that $k \geq 3$. (Otherwise, the problem reduces to determining whether G has at least one edge.) We create a new graph G' by making a copy of G and adding to this a complete $n \times n$ bipartite graph. The result is a graph with

$3n$ vertices. The original vertices of G have degree at most $n-1$, and the $2n$ newly added vertices all have degree n, so the median degree of G' is n. We output G' and k. Clearly, this can be done in polynomial time. (An alternative approach is to generate a very large clique in G, but some care is needed to avoid this large clique from providing a spurious LDC.)

To establish correctness, we will show that G has a clique of size k if and only if G' has an LDC of size k.

($\Rightarrow$) Suppose that G has a clique V' of size k. We assert that same vertices form an LDC within G'. The reason is that G' has median degree n and each original vertex of G has degree at most $n-1$, so all the vertices of V' are LDC-eligible.

($\Leftarrow$) Suppose that G' has an LDC V' of size k. The largest clique in any bipartite graph is of size 2, and by our assumption $k \geq 3$. Thus, the bipartite part of G' cannot contribute to the existence of a clique, which implies that V' is a clique within the original graph G.

Solution to Challenge Problem 1: Here is a reduction from Vertex Cover to VCC (VC $\leq_P$ VCC). Given an (undirected) graph G and integer k for the vertex cover problem, we create a directed graph G' by replacing each undirected edge $\{u, v\}$ from G with a pair of directed edges (u, v) and (v, u) (thus replacing each edge with a "mini-cycle"). We output G' and k. Clearly, this can be done in polynomial time.

Here is a sketch of the correctness. If G has a vertex cover V' of size k, then we assert that the same vertices form a VCC in G'. Since every edge of G is incident to a vertex of V', every edge of G' is also incident (either at its head or its tail) to a vertex in V'. Thus, every cycle in G' must pass through some vertex of V'.

Conversely, if G' has a VCC V' of size k, then we assert that the same vertices form a vertex cover in G. Suppose not. If there were an edge (u, v) in G such that neither u nor v is in V', then the cycle $\langle u, v, u \rangle$ is not covered, implying that V' is not a VCC, a contradiction.

Solution to Challenge Problem 2.: The solution to Problem 3 already achieves this (through the use of the bipartite graph).