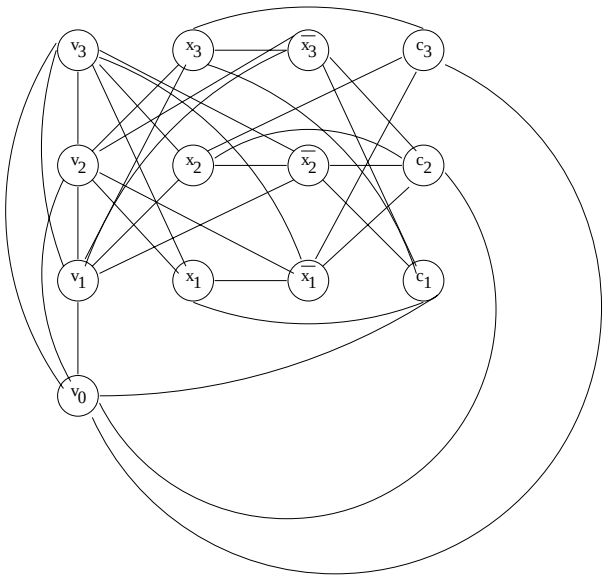
CS–172 Computability & Complexity, Spring 2021

Homework 8 Solutions

Note: These solutions are not necessarily model answers. Rather, they are designed to be tutorial in nature, and sometimes contain more explanation than is required for full points. Also, bear in mind that there may be more than one correct solution. The maximum total number of points available is 36.

1. (a)(i) The graph is as follows: *2pts*



(ii) We need to show that *6pts*

$$\phi \text{ is satisfiable} \iff G \text{ can be colored with } k = (n+1) \text{ colors.} \qquad (*)$$

Suppose first that ϕ is satisfiable. We demonstrate how to color the vertices of G with $n+1$ colors, which for convenience we refer to as $0, 1, 2, \ldots, n$. Consider a particular assignment of truth values to the variables x_i which satisfies ϕ. Then the following is a valid $(n+1)$-coloring of G:

- Each vertex v_i receives color i.
- For each variable x_i which is *true* under the assignment, vertex x_i receives color i and vertex $\overline{x}_i$ receives color 0.
- For each variable x_i which is *false* under the assignment, vertex x_i receives color 0 and vertex $\overline{x}_i$ receives color i.
- From each clause C_k, select a literal, say x_i or $\neg x_i$, which is true under the assignment; then vertex C_k receives color i.

[You should check that the above is indeed a valid $(n+1)$-coloring, i.e., that no pair of adjacent vertices receives the same color.] This deals with the forward implication in $(*)$.

For the reverse implication, suppose that G is $(n+1)$-colorable. Consider any valid $(n+1)$-coloring of the vertices of G. The vertices $v_0, v_1, \ldots, v_n$ form a clique, and so must acquire $n+1$ distinct

colors; let these colors be referred to as $0, 1, \ldots, n$ respectively. In each pair of vertices $\{x_i, \overline{x}_i\}$, one of the vertices must have color i and the other vertex color 0. [Why?] The coloring of G defines an assignment of truth values to the variables of ϕ in the following way: for each variable x_i, set x_i to be true if vertex x_i has color i, and false if vertex x_i has color 0. We complete the analysis by showing that this truth assignment makes each clause of ϕ true (and hence ϕ itself true). Consider any vertex C_k in G. Since vertex C_k is adjacent to vertex v_0, it must receive a color, i, other than 0. Thus, by the construction of the edge set of G, the clause C_k must contain a literal, either x_i or $\neg x_i$, which is true under the proposed assignment.

Note that $(*)$ asserts that the function that maps the formula ϕ to the pair $(G, n+1)$ is a reduction from SAT to COLORABILITY. It is easy to see that the reduction can be computed in polynomial time, since the number of vertices in the graph is equal to $3n + 1 + r$, which is linear in the size of the formula ϕ, and the graph can be simply written down in simple fashion from the structure of ϕ.

(iii) From part (ii) we see that COLORABILITY is NP-hard. We can also easily see that COLORABILITY belongs to NP: given a graph G and an integer k, we simply non-deterministically guess a possible assignment of k colors to the vertices of G, and then check (in linear time) that no two adjacent vertices have received the same color. Hence COLORABILITY is NP-complete. *2pts*

Some students seemed to think that the reduction from the NP-complete problem SAT *already implies that* COLORABILITY *is itself NP-complete. In fact this only shows that* COLORABILITY *is NP-hard. To prove NP-completeness, you also have to show that* COLORABILITY *belongs to NP. This is easy, but it has to be done.*

(b) It should be clear that TIMETABLE is in NP: guess a possible assignment of papers to slots and check it. We therefore just need to show that TIMETABLE is NP-hard. Since COLORABILITY is NP-hard by part (a), it suffices to reduce COLORABILITY to TIMETABLE. *10pts*

Let $G = (V, E)$ be an undirected graph, and k a positive integer. Taken together, G and k form an instance of COLORABILITY. We construct an instance of TIMETABLE as follows. Let $e_1, \ldots, e_m$ be the edge set of G. Then define

$$\begin{aligned} P &= V; \\ C &= \{c_1, \ldots, c_m\}; \\ S &= \{1, 2, \ldots, k\}; \\ P_i &= e_i, \quad \text{for } 1 \leq i \leq m; \end{aligned}$$

where edges are regarded as unordered pairs of vertices. Thus each candidate sits precisely two papers. (It is in this respect that we are using a very restricted special case of the target problem.)

We claim that the function that maps (G, k) to $(P, C, S, \{P_i\})$ is a reduction from COLORABILITY to TIMETABLE. To see this, suppose first that there is a way to color G with k colors. Then we can schedule the papers by placing in timetable slot i all papers whose vertices receive color i in the coloring. Since no two adjacent vertices receive the same color, no candidate will have a clash of papers. Conversely, suppose that there is a way to schedule the papers with no clashes. Then we can color G with k colors by assigning color i to all vertices whose corresponding paper is scheduled in slot i. Now the fact that no candidate has a clash of papers ensures that no two adjacent vertices receive the same color. Thus we have shown that G is colorable with k colors if and only if the corresponding timetable instance can be scheduled, so we do indeed have a reduction. Finally, it is clear that the reduction is polynomial time since the size of the timetable instance is linear in the size of the graph and it can be generated from the graph in simple fashion.

As in Q1(a)(iii) above, some students failed to show that TIMETABLE *is in NP.*

2. Suppose $\mathrm{P} = \mathrm{NP}$ and L is any NP-language other than $\emptyset$ or Σ^*. Thus there are strings w_1 and w_2 such that $w_1 \in L$ and $w_2 \notin L$. We show that L is NP-complete. To see this, let L' be any other NP-language. Since $\mathrm{P} = \mathrm{NP}$, we know that L' can be decided in polynomial time. On input w to L', our reduction f works as follows: Decide in polynomial time if $w \in L$. If yes, set $f(w) = w_1$; otherwise set $f(w) = w_2$. This is clearly a valid reduction, and runs in polynomial time. *6pts*

Another way to say the same thing is to show that every non-trivial $L \in \mathrm{P}$, is P-complete – i.e., every other language $L' \in \mathrm{P}$ is such that $L' \leq_P L$. The proof is exactly the same as above – namely, as L is non-trivial, there exist $w_1 \in L, w_2 \notin L$, and the reduction simply takes the input x, decides in polytime whether $x \in L'$ (can be done as $L' \in \mathrm{P}$), and if so sets $f(x) = w_1$ else sets $f(x) = w_2$. From this fact, and that $\mathrm{P} = \mathrm{NP}$ we can deduce that every language in NP (which is then the same as P) is NP-complete.

Note that we required that $L \neq \emptyset$ and $L \neq \Sigma^*$ since we need the existence of at least one string in L and another string not in L. If, for example, $L = \emptyset$, then we would have no possible value $f(w)$ for strings $w \in L'$. Similarly, if $L = \Sigma^*$ then we would have no possible value $f(w)$ for strings $w \notin L'$.

3. (a) Let (G, k) be an arbitrary instance of VERTEXCOVER. We will construct a set of linear inequalities that has a solution in the integers if and only if G contains a vertex cover of size k. Since VERTEXCOVER is NP-complete (as we saw in class), this will imply that INTEGER PROGRAMMING is NP-hard. *8pts*

For each vertex i of G, we introduce a variable x_i. We also introduce the inequalities $x_i \leq 1$ and $-x_i \leq 0$ for each i. We shall call these the 0-1 inequalities for x_i. (Note that these two constraints force all the variables x_i to take on either the value 0 or the value 1; informally, our interpretation is that $x_i = 1$ corresponds to vertex i belonging to our vertex cover, and $x_i = 0$ corresponds to i not belonging to it.) Next we introduce inequalities that ensure that at least one endpoint of every edge belongs to the vertex cover: thus, for each edge (i, j) of G, we introduce the inequality $-x_i - x_j \leq -1$, which forces at least one of x_i and x_j to be 1 (since the above is equivalent to requiring $x_i + x_j \geq 1$). Finally, we express the fact that the vertex cover contains (at most) k vertices by including the linear inequality $x_1 + x_2 + \cdots + x_n \leq k$, where n is the number of vertices in G.

To check that the above is indeed a reduction, suppose first that G contains a vertex cover U of size k. Then the following is a solution to the above set of inequalities: $x_i = 1$ for $i \in U$, and $x_i = 0$ for $i \notin U$. This is easily seen: the 0-1 inequalities are satisfied as all x_i are either 0 or 1; the edge inequalities are satisfied because U is a vertex cover, so contains at least one endpoint of every edge; and the last inequality is satisfied because U contains k vertices. Conversely, suppose that the above inequalities have an integer solution. Because of the 0-1 inequalities, this solution must have $x_i = 0$ or $x_i = 1$ for each i. Then we claim that the set $U = \{i : x_i = 1\}$ is a vertex cover of size at most k in G. This is because the edge inequalities ensure that U is indeed a vertex cover, and the final inequality ensures that U has size at most k.

Thus we have a valid reduction. Clearly the reduction can be implemented in polynomial time, given G and k: we just need to write out a pair of inequalities for each vertex of G, one inequality for each edge, and one final inequality.

Some students omitted the parameter k in VERTEXCOVER*, which means they also omitted the corresponding constraint in* INTEGER PROGRAMMING.

(b) To show that IP belongs to NP, we need to find a certificate for a set of inequalities that can be checked in polynomial time. In particular, the size of the certificate must be only polynomial in the input size. The obvious choice for a certificate for IP is a set of integer values of the variables that satisfies the inequalities. However, for all we know, it could be that a given set of inequalities has an integer solution, but that any such solution has extremely large values. Then we couldn't use the solution as a certificate. *2pts*

However it turns out that, with some linear algebra, one can show that if the inequalities have a solution then they must have a solution whose values are not too large: specifically, in which for every i, $|x_i| \leq M$ where $M = n(am)^{2m+3}(1+b)$ where n is the number of variables, m the number of constraints, and a and b are the largest (in absolute value) of the coefficients on the left and right respectively. Thus, the total number of bits needed to represent the satisfying x_i's is $n \log_2 M$, which is polynomial in n, m and $\log_2 a$, $\log_2 b$, and hence polynomial in the size of the input. For details, you may look at: Hopcroft and Ullman pp. 337–338, or the book "Combinatorial Optimization" by Papadimitriou and Steiglitz, from which the above fact is taken.