CSE 101 Homework 3 Answers

Note that you wouldn't necessarily be expected to give the same answers as those given here.

1. Verifier A(x, y) below takes instance $x =< G_1, G_2 >$ and certificate y. Assume the graphs are specified by their connectivity matrices E_1 and E_2 (for a graph with n vertices, connectivity matrix E is an n by n matrix with binary entries $E[i, j]$: a 1 indicates the presence of an edge between vertices i and j, and 0 indicates absence of an edge). If the two graphs are isomorphic, then there will be some way to rename the vertices of G_1 so that its connectivity matrix is identical to that of G_2. If the two graphs are not isomorphic, there will be no such renaming. So, if G_1 in instance x has n vertices, we define certificates y for this instance to encode a permutation $S_1, S_2, \ldots, S_n$ of the vertex indices 1 through n; this permutation is used to reorder the vertices in G_1, and the verifier checks if this reordering makes the connectivity matrix identical to that of G_2.

```
A(x =< E_1, E_2 >, y =< permutation S_1, S_2, ... S_n >)
1  if number of vertices in G_1 ≠ number of vertices in G_2
2      then return 0
3  for i ← 1 to n
4      for j ← 1 to n
5          if E_1[S_i, S_j] ≠ E_2[i, j]
6              then return 0
7  return 1
```

Lines 5-6 take constant time, and are executed n^2 times, for a total running time of $O(n^2)$, which satisfies the requirement that the verifier's running time be polynomial.

If the graphs are isomorphic, then they have the same number of vertices, so the verifier reaches line 3. Then, there is some certificate y that contains the renumbering of the vertices in G_1 that makes the connectivity matrices identical, so that line 6 doesn't return and line 7 is reached, returning 1. If the graphs are not isomorphic, then either they have a different number of vertices (causing A to return 0 on line 2), or they have the same number but for every certificate y, the connectivity matrices remain different and the verifier returns 0.

The verifier is correct and runs in polynomial time, so GRAPH-ISOMORPHISM is in NP.

2. To show this problem is NP-complete, we must first show that it is in NP. An instance $X =< A, b >$ consists of the matrix A and the vector b. Let certificates Y consist of a candidate solution: a vector of n binary values giving assignments to the n variables in x (note that lower-case x denotes a vector of n variables here, not an instance). The verifier simply checks that, for the vector x given in Y, $Ax \leq b$. If this is satisfied, the verifier returns 1. If not, it returns 0.

The verifier need only do a matrix multiply, which requires polynomial time. If there is a solution x to the inequality, then the certificate Y containing that x will cause the verifier to output 1. Otherwise, there is no solution so there is no certificate Y that makes the verifier output 1. Since the verifier is correct and takes polynomial time, the problem is in NP.

To show the problem is also NP-hard, we reduce 3-CNF-SAT to it. Given a 3-CNF formula $P = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ with n variables, we build an instance of 0-1 INTEGER PROGRAMMING, also with n variables - the same variables that appear in P. The instance of 0-1 INTEGER programming will have k inequalities (so A is a k by n matrix). Each inequality is satisfied if the corresponding clause in P is satisfied. To achieve this, say we have a clause

$$L_i \vee L_j \vee L_k,$$

where literal L_i is either variable x_i or its negation $\neg x_i$, and similarly for L_j and L_k.

To see how to get a corresponding inequality, say we have the clause $x_1 \wedge \neg x_2 \wedge \neg x_3$. We want to check that at least one of x_1, $1 - x_2$, and $1 - x_3$ (accounting for x_2's and x_3's negation) is equal to

1. So, we check that $x_1 + 1 - x_2 + 1 - x_3 \geq 1$. Rewriting in a form appropriate to 0-1 INTEGER PROGRAMMING, $-x_1 + x_2 + x_3 \leq 1$.

To do this in general, let a_i be $+1$ if variable x_i appears negated, and -1 if it appears in unnegated form (note the sign reversal, which is needed because the inequalities are $\leq$ rather than $\geq$), and similarly for a_j and a_k. Let g be the number of variables in this clause that appear in negated form. Then the corresponding inequality is:

$$a_i x_i + a_j x_j + a_k x_k \leq g - 1.$$

If this inequality is satisfied by a 0-1 assignment to x_i, x_j, and x_k, then the corresponding clause of P has value 1 because at least one of its literals has value 1. Otherwise, the corresponding clause in P has value 0.

The matrix A consists of one row per inequality. The row has a zero entry for each variable not appearing in the inequality, and the coefficients a_i, a_j, and a_k given above for the variables x_i, x_j, and x_k that do appear in the inequality. The corresponding row of the column vector b contains the value $g - 1$ given above.

This reduction is straightforward to compute, requiring only linear time in the number of variables to translate each clause into an inequality, and polynomial time overall to do this for all k clauses.

Given the 0-1 INTEGER PROGRAMMING instance built in this way, if it has a solution x, then each clause's inequality is satisfied, so the value of each clause is 1 and the original instance of 3-CNF-SAT is satisfiable. If the original 3-CNF-SAT instance has a satisfying assignment to its variables, this assignment gives each of its clauses value 1 and also satisfies the corresponding inequalities - so the 0-1 INTEGER PROGRAMMING instance has this same solution.

So, the reduction is correct, showing that 0-1 INTEGER PROGRAMMING is NP-hard. Since it is also in NP, it is NP-complete.

3. We first need to show that LONGEST SIMPLE CYCLE is in NP. An instance x is an undirected graph and an integer k. Let a certificate y be a sequence of at least k vertices from the graph. The verifier is similar to that for traveling salesman: it goes through the vertices in the certificate one by one, marking each as it goes to check that there are no repeats, and verifying that there is an edge from each vertex in the certificate to the next one given in the certificate. Finally, the verifier checks that the cycle is completed by an edge from the last vertex in the certificate back to the first vertex. If these checks are satisfied, the verifier outputs 1. If any fail, the verifier outputs 0.

   This verifier takes linear time in the number of vertices. If there is a simple cycle with at least k vertices, then passing that cycle as a certificate causes the verifier to output 1. Otherwise, no certificate contains a valid cycle with at least k vertices, so the verifier always outputs 0 for this instance. Since the verifier is correct and takes polynomial time, the problem is in NP.

   To show the problem is NP-hard, we reduce HAM-CYCLE to it. Given a graph G that is an instance of HAM-CYCLE, define an instance $< G, k >$ of LONGEST SIMPLE CYCLE containing the same graph $G = (V, E)$, and $k = |V|$, the number of vertices in G.

   This reduction involves only simple copying and counting, so it takes polynomial time.

   Now, if graph G contains a Hamiltonian cycle, then this cycle is a simple cycle with $|V|$ vertices, so the correct decision on the instance of LONGEST SIMPLE CYCLE is 1. If there is no Hamiltonian cycle, then there is no simple cycle with $|V|$ or more vertices (simply by the definition of Hamiltonian cycle), so the correct decision on the instance of LONGEST SIMPLE CYCLE is 0.

   So, the reduction is correct and takes polynomial time, showing that LONGEST SIMPLE CYCLE is NP-hard. Since it is also in NP, it is NP-complete.

4. (a) Define the decision problem as that of deciding whether a given graph G contains an independent set with at least k vertices. So, we want to decide membership in the set $\{< G, k >: G$ contains an independent set with at least k vertices $\}$.

   First show that this problem is in NP, by giving a polynomial-time verifier. This is similar to the verifier for CLIQUE. The verifier takes an instance $< G = (V, E), k >$ and a certificate y consisting of a set of k vertices. For each pair of vertices in y, we check each edge and see that it does not go between that pair of vertices. if this check passes for all pairs in y and all edges, the verifier outputs 1. Otherwise it outputs 0.

The verifier examines at most $|V|^2$ pairs of vertices, and at most E edges for each pair, so total time is polynomial: $O(|V|^2 E)$.

If there is an independent set with at least k vertices, then a certificate containing any k of its vertices will cause the verifier to output 1. Otherwise, if there is no such independent set, no set of k vertices passes the independent-set check, so the verifier outputs 0 for all certificates. Since the verifier is correct and takes polynomial time, the problem is in NP.

To show the problem is NP-hard, we reduce from CLIQUE. Given an instance $< G, k >$ of CLIQUE, we construct an instance $< G', k >$ of INDEPENDENT SET. G' is constructed from G by inverting the connectivity matrix: G' contains the same vertices in G, and for each pair of vertices i, j, G' has the edge (i, j) if G does not have it, and G' does not have the edge (i, j) if G does have it.

Now, if G has a clique with at least k vertices, then there is a subset of k vertices in G such that each pair in this subset has an edge between them. So, in G', each pair of vertices in this subset does *not* have an edge between them: this subset of vertices is an independent set of size k in G'. In the other direction, if G' has an independent set with at least k vertices, there is a subset of k vertices in G' such that each pair in this subset does not have an edge between them. So, each such pair of vertices in this subset *does* have an edge between them in G, so this subset is a clique of size k in G.

This shows that the reduction is correct. Inverting the connectivity matrix takes only linear time in the number of edges, so the reduction takes polynomial time. So, INDEPENDENT SET is NP-hard. Since it is also in NP, it is NP-complete.

(b) Given an instance $< G, k >$ of INDEPENDENT SET, let "black box" B return the correct decision on this instance. The following algorithm finds the maximum-size independent set by first determining how many vertices it contains (by trying each threshold k to find the largest threshold for which there is still an independent set). It then finds the vertices in the independent set by temporarily deleting vertices. If deleting a vertex causes the size of the maximum independent set to decrease, then that vertex must be in the independent set. If it doesn't cause the size of the maximum independent set to decrease, then that vertex can be permanently deleted, yielding a subproblem of reduced size. Below, $MAX - I - SIZE$ returns the size of the maximum independent set, and $MAX - I$ returns the actual maximum independent set.

Let $G - v$ denote the graph obtained by deleting vertex v from G's vertices, and deleting all edges connected to v from G's edges.

MAX-I-SIZE($G = (V, E)$)
for $k \leftarrow 1$ **to** $|V|$
  if $B(G, k) = 0$
    then return $k - 1$
return $|V|$

MAX-I($G = (V, E)$)
$s \leftarrow$ MAX-I-SIZE(G)
$v \leftarrow$ first vertex in G
if MAX-I-SIZE($G - v$) $= s$
  then return MAX-I($G - v$)
else return $\{v\} \cup$ MAX-I($G - v$)

Counting calls to B as a single step, each call to MAX-I-SIZE takes $O(|V|)$ time. A call to MAX-I on a graph with $|V|$ vertices takes $O(|V|)$ time for two calls to MAX-I-SIZE, and also does a call to MAX-I on a graph with $|V| - 1$ vertices. Deleting a vertex from a graph can be done in $O(|V|^2)$, since there are at most $|V|^2$ edges in the graph; the time for MAX-I also includes this time. Letting $n = |V|$, this yields the recurrence:

$$T(n) = O(n^2) + O(n) + T(n-1) = O(n^2) + T(n-1)$$

$$T(n) = \sum_{i=1}^{n} i^2$$

by iteration. So, $T(n) = O(n^3)$, which is polynomial. Note that this could be improved: binary search can cut the time required by MAX-I-SIZE, and a carefully chosen data structure can reduce the time required to delete a vertex. But for purposes of this problem, all that matters is to obtain a polynomial-time algorithm.