6.045J/18.400J: Automata, Computability and Complexity Prof. Nancy Lynch, Nati Srebro

Quiz 3: Solutions

April 28, 2004 *Susan Hohenberger*

Please write your name in the upper corner of each page.

Name: ______________________________

Problem 1: **True, False, or Unknown (18 points).** In each case, say whether the given statement is known to be TRUE, known to be FALSE, or currently not known either way. Full credit will be given for correct answers. If you include justification for your answers, you may obtain partial credit for incorrect answers.

1. True, False, or Unknown: P $\neq$ NP.

   Unknown; this is generally believed to be true, but it might not be.

2. True, False, or Unknown: The Hamiltonian path problem for undirected graphs is in P (i.e., UHAMPATH=$\{\langle G, s, t\rangle \mid G$ is an *undirected* graph with a Hamiltonian path from s to $t\}$).

   Unknown; UHAMPATH is NP-complete. If P=NP, then UHAMPATH is in P.
   If P$\neq$NP, then UHAMPATH is not in P.

3. True, False, or Unknown: NP $\cap$ coNP = P.

   Unknown; we know that P is contained within NP $\cap$ coNP, but these sets might not be equal.

4. True, False, or Unknown: If SAT $\in$ P, then coNP $\neq$ P.

   False; SAT $\in$ P implies that $\overline{SAT}$ $\in$ P. (Recall that P is closed under complement.)
   Since $\overline{SAT}$ is coNP-complete, this would mean that coNP=P.

Name: ______________________________

5. True, False, or Unknown: There exists a language that is not decidable in time 3^n but is decidable in time 10^n (where n is the length of the input).

True; by the Time Hierarchy Theorem (see page 311 of Sipser).

6. True, False, or Unknown: *Both* of the following exist:

   (a) A language A such that $P^A = coNP^A$.

   (b) A language B such that $P^B \neq coNP^B$.

True; see the discussion of this in Sipser section 9.2. Recall that P is closed under complement, and in fact, so is the P^C for any language C.

Name: ______________________________

Problem 2: **(12 points)** The proof that SAT is NP-complete appears in Sipser's book, p. 254-259. Part of the main construction involves constructing a formula ϕ_{move}, which is expressed as the conjunction of formulas saying that 2 × 3 windows of the tableau are "legal". For each of the following, state whether they represent legal windows.
Assume that the underlying polynomial-time NTM is of the form $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, where $\Sigma = \{0, 1, 2, 3\}$ and $\Gamma = \{0, 1, 2, 3, \sqcup\}$.

1. Legal or not: $\begin{array}{|c|c|c|}\hline 0 & q_1 & 1 \\ \hline 0 & 2 & q_2 \\ \hline \end{array}$, where $(q_2, 2, R) \in \delta(q_1, 1)$.

Legal.

2. Legal or not: $\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline q_2 & 1 & 2 \\ \hline \end{array}$, where $(q_2, 3, R) \in \delta(q_1, 0)$ for some q_1.

Legal.

3. Legal or not: $\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 1 & 1 & 2 \\ \hline \end{array}$, where $(q_2, 1, L) \in \delta(q_1, 0)$ for some q_1 and q_2.

Legal; the TM head could have been on the left and moved left.

4. Legal or not: $\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 0 & 1 & 3 \\ \hline \end{array}$, where $(q_2, 3, R) \in \delta(q_1, 2)$ for some q_1 and q_2.

Not legal; the TM head needs to be to the right of the tape position it writes.

Name: ______________________________

Problem 3: **(10 points)** Suppose that L_1, L_2, and L_3 are nontrivial languages over $\Sigma = \{0, 1\}$. Prove that if:

1. $L_1 \leq_P L_2 \cap L_3$,
2. $L_2 \in$ NP, and
3. $L_3 \in$ P,

then $L_1 \in$ NP. You may invoke theorems proved in class and in the book, but if you do this, cite them explicitly.

Since $L_3 \in$ P, we know that it is also in NP.
Since NP is closed under intersection, we also know that $L_2 \cap L_3$ is in NP.
Finally, since L_1 is reducible to a language in NP in polynomial time, then we have a polynomial-time NTM that decides L_1 – it can go through the reduction to use the polynomial-time NTM for $L_2 \cap L_3$.

Name: ____________________

Problem 4: **(20 points)** For any k, the language k-COLOR is defined to be the set of (undirected) graphs whose vertices can be colored with at most k distinct colors, in such a way that no two adjacent vertices are colored the same color. In class, we learned that 2-COLOR $\in$ P and 3-COLOR is NP-complete.

1. Prove that 4-COLOR is NP-complete.

Part 1. 4-COLOR is in NP. The coloring is the certificate (i.e., a list of nodes and colors). The following is a verifier V for 4-COLOR.
V="On input $\langle G, c\rangle$,

1. Check that c includes ≤ 4 colors.
2. Color each node of G as specified by c.
3. For each node, check that it has a unique color from each of its neighbors.
4. If all checks pass, *accept*; otherwise, *reject*."

Part 2. 4-COLOR is NP-hard. We give a polynomial-time reduction from 3-COLOR to 4-COLOR. The reduction maps a graph G into a new graph G' such that $G \in$ 3-COLOR if and only if G' 4-COLOR. We do so by setting G' to G, and then adding a new node y and connecting y to each node in G'.

If G is 3-colorable, then G' can be 4-colored exactly as G with y being the only node colored with the additional color. Similarly, if G' is 4-colorable, then we know that node y must be the only node of its color – this is because it is connected to every other node in G'. Thus, we know that G must be 3-colorable.

This reduction takes linear time to add a single node and G edges.

Since 4-COLOR is in NP and NP-hard, we know it is NP-complete.

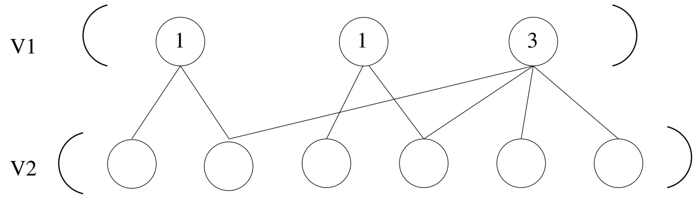
Name:

2. Assuming $P \neq NP$, for which values of $k \in \mathbb{N}$ is k-COLOR NP-complete? Briefly explain.

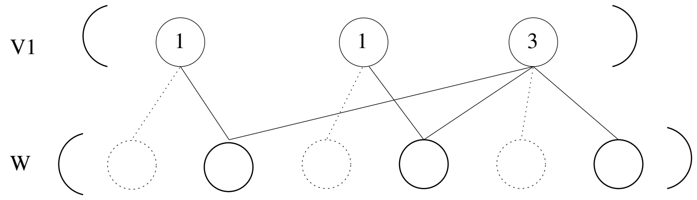
$k \geq 3$**.** We know that 1-COLOR and 2-COLOR are in P, and that 3-COLOR and 4-COLOR are NP-complete. In fact, we can reduce n-COLOR to $(n+1)$-COLOR, for $n \geq 3$, by the same way as above.

Name:

Problem 5: **(20 points)** Let $G = (V_1, V_2, E)$ be a "bipartite" undirected graph, that is, a graph whose nodes are divided into two sets, V_1 and V_2, such that every edge in E connects a node in V_1 to a node in V_2. The nodes in V_1 are all labeled with non-negative integers. For example, G might look like the following:



Some of the nodes in V_2 can be removed to leave a subset $W \subseteq V_2$. A subset W is called *consistent* with G if every node in V_1 which has been assigned a number m is connected to exactly m nodes in W. The problem is to determine whether a consistent W exists. For example, one consistent W for the graph above would be:



We formulate this problem as the language:

$MATCH = \{\langle G, N\rangle |\ G$ is a bipartite graph (V_1, V_2, E), N is an assignment of nonnegative integers to V_1, and there exists $W \subseteq V_2$ that is consistent with G and $N\}$.

(a) Show that $MATCH$ is in NP, using the certificate and verifier method.

The subset $W \subseteq V_1$ is the certificate.

The following is a verifier V for $MATCH$.

V="On input $\langle\langle G, N\rangle, c\rangle$,

1. Test whether c is a set of nodes in V_2.
2. For each node $v \in V_1$, test that it is connected to exactly $N(v)$ nodes in W.
3. If all tests pass, accept; otherwise, reject."

Name: ______________________________

(b) Show that $MATCH$ is NP-hard, using a reduction from 3SAT.

We will show 3SAT $\leq_P$ $MATCH$. The reduction maps a Boolean formula ϕ to a graph G and a numbering scheme N. Our variable gadget will be as follows: for each variable x_i, add one node to V_1 with a label of 1 and add two nodes to V_2 that correspond to x_i and $\overline{x_i}$, then add two edges connecting the 1 node with both the nodes in V_2. Setting a variable x_i to TRUE corresponds to selecting the node x_i to be in W.

Our clause gadget will be as follows: for each clause c_j, add one node to V_1 with a label of 3 and add two "dummy" nodes to V_2, then add edges connecting the 3 node to the two dummy nodes and to the three nodes in V_2 that correspond to the literals in c_j.

We want to argue that this reduction is correct; that is, $\langle \phi \rangle \in$ 3SAT if and only if $\langle G, N \rangle \in MATCH$. First, if $\langle \phi \rangle \in$ 3SAT, then we can always find a consistent W as follows. Place into W each node corresponding to the truth of a satisfying assignment for ϕ (i.e., if $x_1 =$TRUE, add the node x_i; else add $\overline{x_i}$); this makes W consistent with all the '1' nodes in V_1. Next, for each '3' node, check to see how many neighbors it has already in W (this cannot be more than 3, since each clause can have at most 3 true literals). If a '3' node has less than 3 neighbors in W, supplement this amount to 3 by adding the necessary dummy nodes. This makes W consistent with all the '3' nodes, and thus with all of V_1.

Secondly, we argue that if $\langle G, N \rangle \in MATCH$, then ϕ must be satisfiable. Our variable gadget construction restricts only one of the two $x_i, \overline{x_i}$ nodes to be in W (i.e., TRUE); and our clause gadget requires that at least one node corresponding to a literal in c_j is in W for each clause c_j (i.e., all clauses are TRUE).

So far, we have just argued that there exists a mapping from 3SAT to $MATCH$. Now we need to argue that this mapping can be done in polynomial-time. If ϕ is a Boolean formula with m variables and ℓ clauses. Then, our variable gadget adds $3m$ nodes and $2m$ edges to G, while our clause gadget adds 3ℓ nodes and 5ℓ edges to G. (Note: that the size of N is linear in m.) Thus, the size of G and N, constructed by trivial operations, is $O(m) + O(\ell)$, which is polynomial in $|\phi|$.

Name: ____

Problem 6: **(20 points)** Suppose that we are given a polynomial time algorithm M (formally, a basic deterministic TM) that decides membership in the language VERTEX-COVER = $\{\langle G, k\rangle |\ G$ is an undirected graph and G has a vertex cover of size $\leq k\}$.

1. Describe a polynomial time algorithm that, given the representation of an undirected graph G, finds the size of the smallest vertex cover of G. Your algorithm may use M as a "subroutine". Explain why your algorithm takes polynomial time.

For $i = 1, 2, \ldots, |G|$, run M on $\langle G, i\rangle$. Output the smallest value of i for which M accepts.

Let $p(|G|)$ be M's running time on $\langle G, k\rangle$ (note that $k \leq |G|$). Then our algorithm takes $|G| * p(|G|)$ time, which is also polynomial.

Name: ____

2. Describe a polynomial time algorithm that, given the representation of an undirected graph $G = (V, E)$, finds a smallest-size vertex cover of G (that is, a subset $V' \subseteq V$ such that for each edge $\{u, v\} \in E$ at least one of u and v belongs to V').

Use the algorithm from part (1) to determine the minimum size k.

Start with $H = G$. Then repeat until $k = 0$: Select any edge and for each of the two endpoints, see if removing that vertex and all its incident edges leaves a smaller graph G' that has a $k-1$ cover. (You can test for this by again running M on $\langle G', k-1\rangle$.) One of the two tests must succeed. When you find out which, put that vertex into the final cover, and set H to be the remaining graph with that vertex and its incident edges removed. Set $k = k - 1$.

One endpoint of each edge *must* be in the smallest vertex cover. Thus, if $v \in V$ is in the smallest vertex cover of size k (note, there may be many smallest vertex covers, but this algorithm will find one of them), then it must take $k-1$ nodes to cover all the edges not touched by v. This means that M will accept $\langle G', k-1\rangle$ where G' is the graph H with v and all its incident edges removed. Once v is found to be in the smallest vertex cover, we are only concerned with the edges of H that v does not touch, so we removed v and its incident edges from the graph, decrement k, and repeat.

If $v \in V$ is not in a smallest vertex cover of size k, then, by definition, this means that all edges of H which are not incident to v cannot be covered with $k-1$ remaining nodes. Thus, the M will not accept.

Since the algorithm in part 1 and M run in polynomial-time, then this algorithm does so as well, with a multiplicative increase of $O(|G|)$ in the running time.