Assignment 25

NP-completeness

a. SAT_2 is the language $\{< \phi > | \phi \text{ is a boolean formula with at least 2 satisfying assignments}\}$.

1. Provide an element of the language.
   A: $a \vee b \vee c$
2. Provide an element of the same type but that is not in the language.
   A: $a \wedge b \wedge c$
3. Show that SAT_2 is in NP.
   A: given a certificate containing the expression and variable assignments, it takes polynomial time to plug in the assignments and evaluate the truth of the resulting expression.
4. Show that SAT is polynomial-time reducible to SAT_2. **Hint:** Given a boolean expression, tack on another expression at the end that can be satisfied in 2 ways, such that the overall expression is satisfiable in at least 2 ways if and only if the original expression was satisfiable, otherwise the final result should not satisfiable at all.
   A: assume SAT_2 is polynomial-time-decidable. let M_{SAT_2} be the Turing Machine that decides SAT_2 in poly time. you can implement the machine M_{SAT} which decides SAT as follows: $M_{SAT}(E) = M_{SAT_2}(E \wedge (a \vee b))$ which will return true in polynomial time IFF E is satisfiable by some assignment. essentially, to solve the SAT_2 problem, you need to determine whether the expression is satisfiable by *any* assignment, and so the ability to solve SAT_2 implies the ability to solve SAT.
5. Given that SAT can be reduced in polynomial time to SAT_2 and that SAT_2 is in NP, what does this say about SAT_2?
   A: SAT_2 is NP-complete

b. Given that L is NP-complete, why is $\overline{L}$ not also necessarily NP-complete?
A: because $\overline{L}$ does not reduce to L. rejecting a word, unlike accepting a word, is much harder than verifying a certificate a lot of the time. for example, with SAT, it may take polynomial time to plug in variable assignments, evaluate the expression and determine the validity of the assignment, but one invalid certificate does not instantly prove that NO valid certificate exists for a given word, not deciding its unsatisfiability. therefore, polynomial time verifiability for L does not mean polynomial time verifiability for $\overline{L}$, and so $\overline{L}$ is not necessarily even in NP to begin with.

Assignment 26

NP-completeness pt. 2

Recall the G_{ISO} language from a few homeworks ago:
$\{< G_1, G_2 > | \exists \text{ an isomorphism between the two graphs}\}$. An isomorphism between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a bijection $f : V_1 \to V_2$ such that
$(u, v) \in E_1 \iff (f(u), f(v)) \in E_2$.

A similar, related language is the G_{SUB_ISO} language, which is defined as follows:
$\{< G_{small}, G_{big} > | \exists \text{ an isomorphism between } G_{small} \text{ and a subgraph of } G_{big}\}$.

Terminology: A *subgraph* of a graph is any subset of its vertices and edges, but note that the vertex subset must include all endpoints of the edge subset (it may also include additional vertices).

Show that G_{SUB_ISO} is NP-complete by showing it is in NP and reducing the clique problem to it.

A: given the certificate of a bijection function f, one easily may vaildate the isomorphism by checking that $f(u)$ exists for each vertex in G_{small} , and further checking that the set of all $f^{-1}(v)$ for each vertex v produced in G_{big} is equivalent. however, equivalence also requries correspondence of edges, so in the worst case verification would be $O(n^2)$.

$CLIQUE$ reduces to $G_{SUB-ISO}$ if, given a Turing Machine $T_{SI}(G_S, G_B)$ which decides $G_{SUB-ISO}$, there exists a Turing Machine T_{CLIQUE} which decides $CLIQUE$.

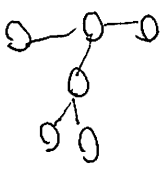
the Turing Machine $T_{CLIQUE}(G_0, n)$ may be implemented as follows:

1. construct a graph G_n such that there are n vertices in the graph and there exists an edge beetween every vertex and every other vertex.
2. return the result of $T_{SI}(G_n, G_0)$. this will determine whether there exists a $CLIQUE$ of size n (the constructed graph G_n) as a subgraph of G_0.

$G_{SUB-ISO}$ is in NP and an NP-complete language reduces to it, so therefore $G_{SUB-ISO}$ is itself NP-complete.

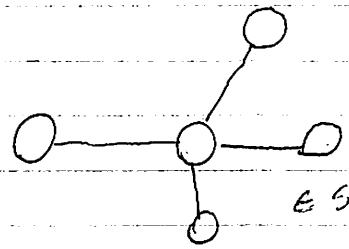
Auren Amster

CS HW 4-3



4. b) let every V in V_G map to the set of edges that contain it,
repeatedly run SUB-ENV $\langle U, S, K \rangle$ where $K=0$, $K++$; $U = E_G$ and $S = map(V_G, \lambda(V \to V.edges))$ until SUB-ENV returns a set $T \subseteq S$ s.t. the union of all edges in T (vertices) equals E_G (T covers the graph)

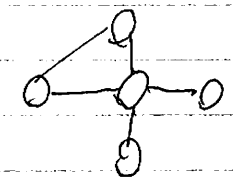
1. a)



$G \in$ STANDALONE$_4$

b) $U = \{1,2,3\}$ $S = \{\{1,2\},\{1,3\},\{1\}\}$ $K=2$

2. a)



$G \notin$ STANDALONE$_4$

b) $U = \{a,b,c,d\}$ $S = \{\{a,b\},\{c\},\{b,c\},\{d\}\}$ $K=2$

3. a) given a certificate of a graph G and a list of vertices $S \subseteq V_G$, check if an edge exists between any two vertices in S. This is $O(|V_G|^2)$ in the worst case.

b) given a set of subsets $S_K \subseteq S$, verify $|S_K| \leq K$ and check if U = the union of all elements of S_K. $O(|U|)$ in worst case (combining $|V|$ subsets, comparing $|V|$ items)

4. a) to compute the minimum vertex cover of a graph G, you may use repeated calls to TM$_{\text{STANDALONE},K}$. First, you can run STANDALONE$(G, |V_G|)$. If this accepts, no nodes in G are connected, and the minimum vertex cover is empty. If this rejects, run STANDALONE$(G, |V_G|-1)$. Keep subtracting from the K of the STANDALONE requirement until there exists a set T of mutually disconnected vertices where $|T|=K$. The minimum vertex cover is $V_G - T$, or all non-mutually disconnected vertices.