Harvard University Extension School
Computer Science E-121

Problem Set 9

Due December 2, 2016 11:59pm

Problem set by Walter Thornton

Collaboration Statement: I completed this assignment alone and only with the course materials.

When writing proofs, avoid the inclusion of redundant or irrelevant information. Sometimes these details can obscure the argument and we will take off half points when these issues arise.

PROBLEM 1 (4 points, suggested length of 1/3 page)

Nina is making a dessert on Sundae Sunday, and different dining halls have different flavors of ice cream, different toppings, and different cereals. She needs m distinct ingredients in total, and each of the n dining halls on campus has some (possibly empty) subset of the ingredients. Each ingredient can be found in at least one dining hall, and some ingredients can be found in more than one dining hall. Define the problem Make-Dessert as follows: Given an integer k, can Nina get a hold of all the ingredients she needs by visiting at most k of the dining halls? Show that Make-Dessert is NP-complete.

(Hint: Reduce from Vertex Cover.)

Solution.
To be NP-Complete, our problem MAKE DESSERT must be both NP and NP Hard.

MAKE DESSERT $\in$ NP
To show that this in NP, there must be a verifier that verifies a given certificate in polynomial time. The certificate in this case is a list of dining halls visited. Our verifier will check each dining hall visited and check off the ingredients found there.
If all ingredients are found and k is greater than or equal to the number of dining halls visited, then the certificate is verified, the verifier accepts. Otherwise it rejects.
Since the verifier visits each of the dining halls once, the verifier is in P. Therefore MAKE DESSERT is in NP.
MAKE DESSERT $\in$ NP-Hard
To show that MAKE DESSERT is NP Hard, we can reduce to this problem in polynomial time from VERTEXT COVER.
A Vertex Cover is a subset of vertices where each edge of the graph is connected to at least one vertex of the subset. The VERTEX COVER problem takes inputs graph G and integer k. VERTEX COVER accepts iff there is a subset of vertices S of size k such that every edge in G has at least one end in S

Reduction of VERTEX COVER to MAKE DESSERT=" Let each ingredient be an edge in G. Let each dining hall be a vertex in G.
For each ingredient that a dining hall contains, make that edge incident to the dining hall represented by that vertex.
VERTEX COVER ACCEPTS if S of size k exists where each edge is incident to at least one vertex

in S. Otherwise it rejects. If VERTEX COVER accepts then MAKE DESSERT accepts."
This can be seen because each S vertex cover is the set of all dining halls that must be visited to ensure that each ingredient is obtained. This construction of the reduction is clearly in P since you can go through the lists of ingredients m and dining halls n, and which dining halls have which ingredients, m∗n.

PROBLEM 2 (1+2+3+4 points, suggested length of 1 page)

The logical operator " | " is NAND: $p \mid q$ is equivalent to $\neg p \vee \neg q$ or $\neg(p \wedge q)$. A |-formula is a boolean formula that only uses | as a logical symbol, e.g.

$$x_1 \mid x_2 \mid (x_1 \mid x_3)$$

(A) Show that satisfaction of |-formulas is an NP problem.

(B) $\neg p$ is equivalent to $p \mid p$. In the same way, write simple formulas for, $p \vee q$ and $p \wedge q$ using only |.

(C) Therefore any logical formula can be written using just |. Show, however, that using these substitutions *does not* yield a polynomial-time reduction from SAT (and therefore does not prove that satisfiability of |-formulas is NP-complete).

(D) Prove that the set of all |-formulas is NP-complete. (Hint: Find a different way of writing $\neg p$, $p \vee q$, and $p \wedge q$ that do not so drastically increase the length of the formula, by first finding formulas for the constants T = TRUE and F = FALSE.)

Solution.

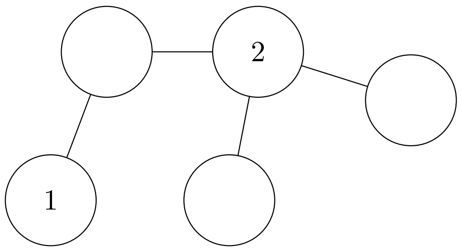
(A)
The satisfaction of |-formulas is an NP problem because a verifier exists to check the solution in polynomial time. In this case you could nondeterministically guess a positive instance that satisfies the boolean formula. Then simply evaluate the boolean formula with the proposed solution.
(B)
$p \vee q \equiv (q \mid q) \mid (p \mid p)$
$p \wedge q \equiv (q \mid p) \mid (q \mid p)$

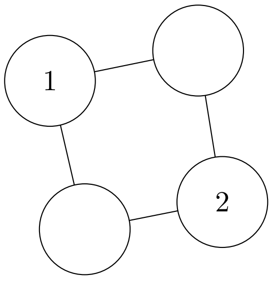
(C)
A reduction from SAT would go through the given boolean expression first replacing NOT operators with the corresponding NAND operator.
$\neg p \equiv p \mid p$
Replace the AND operators
$p \vee q \equiv (q \mid q) \mid (p \mid p)$
Finally, replace the OR operators
$p \wedge q \equiv (q \mid p) \mid (q \mid p)$
Using the unary operator | rather than the binary operators of OR, AND and NOT forces an exponential explosion of the boolean expression.

(D)
We have already shown in (A) that satisfying |-formulas is an NP problem. Now we have to show that it is also NP hard.
From Sipser Theorem 9.33, we know that CIRCUIT-SAT is in NP. Therefore we can reduce CIRCUIT-SAT to |-formulas-SAT to show that |-formulas-SAT is NP complete.
For each Nand gate in Circuit SAT create an equivelant NAND gate for |-formulas SAT. When Circuit SAT is satisfied, likewise |-formulas SAT is satisfied.

PROBLEM 3 (2+4 points, suggested length of 1/2 page)

The game Minesweeper can be played on an undirected graph where each vertex is either blank or labeled with a natural number. Such a board is legal if we can place a set of "mines" on some subset of the blank vertices such that every numbered vertex shares an edge with exactly its number of mines. For example, the board on the left is legal, since we can place mines on the leftmost two blank vertices to satisfy the configuration, while the board on the right does not have a legal set of mines.





Consider the language MS$_{graph}$ $= \{\langle G, g\rangle : G$ is a finite undirected graph and g maps each vertex in G to either "blank" or a natural number such that a valid set of mines exist$\}$.

(A) Show that MS$_{graph}$ is in NP.

(B) Show that MS$_{graph}$ is NP-hard by a reduction from 3SAT. (Note: Don't worry about removing arrows from edges if you use a tool to generate Latex diagrams.)

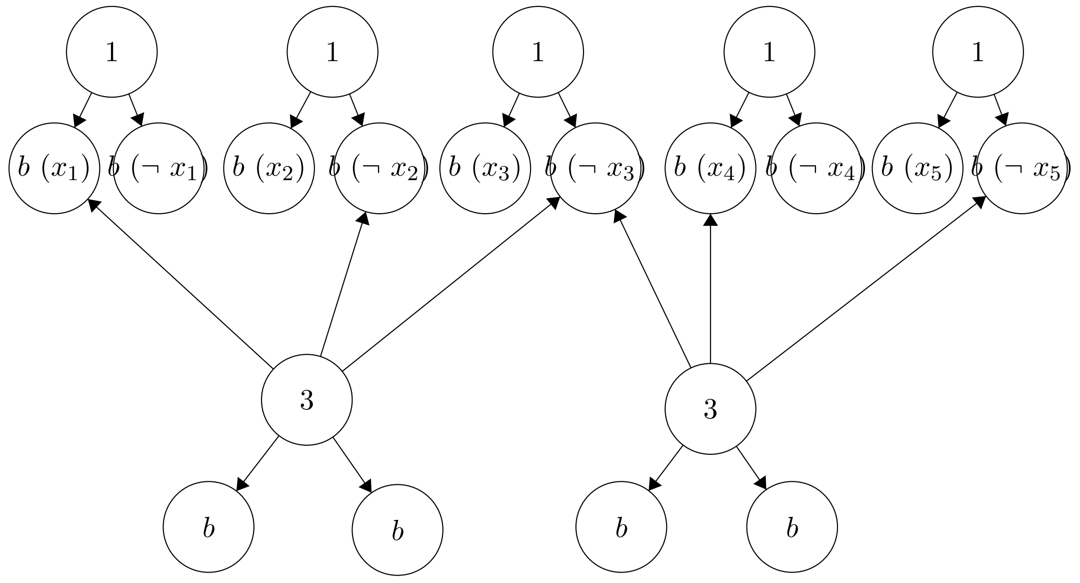
Solution.

(A)
For MS$_{graph}$ to be in NP, there must be a polynomial time verifier for each positive instance. We can nondeterministically guess g's mapping on G to create a valid instance of a legal placement of mines.
Using this certificate, we can simply look at each vertex in G.
At each vertex: Count the number of mines in adjacent nodes. Iff g for each node is equal to the number of adjacent mines, then this is a verified positive instance.
Since, each node must be looked at, as well as each adjacent node, the verifier would take at most $n(n-1)$ steps. This is clearly in polynomial time.

(B)

A reduction from 3SAT shows that MS$_{graph}$ is NP hard.
To accomplish this, we convert a boolean formula in 3CNF to a graph G, an instance of MS$_{graph}$.
For each literal x_i of the boolean formula we create a set of three nodes. $\{x_i^x, x_i^{\neg x}, x_i^1\}$
Each node is mapped to a value by g.
$g(x_i^x) = blank$
$g(x_i^{\neg x}) = blank$
$g(x_i^3) = 1$
Blank value nodes may contain a bomb, natural number valued nodes are nodes that are connected to that number of nodes that contain bombs. Nodes x_i^x and $x_i^{\neg x}$ represent their respective boolean assignment and are both connected to x_i^1. This since only one of x_i and $\neg x_i$ can be true. This is repeated for each variable.

For each clause in the 3CNF we create a set of three nodes $\{c_i^1, c_i^2, c_i^3\}$ where
$g(c_i^1) = blank$
$g(c_i^2) = blank$
$g(c_i^3) = 3$
c_i^3 from each clause has three outgoing edges to each x_i^x or $x_i^{\neg x}$ of c_i of the boolean expression. This is repeated for each clause.
An example for the following boolean string is constructed below.
$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_3 \vee x_4 \vee \neg x_5)$
blank value nodes are labeled b



In this eaxample, the truth assignment for the first clause, $x_1 = T, x_2 = T, x_3 = T$ would result in both the clause being true and a legal bomb assignment at x_1^x, c_1^1 and c_1^2, thus conforming to the rules of both MS$_{graph}$ and 3SAT. Additionally, any true evaluation of the second clause and thus the whole string, results in a legal bomb placement.
Since this can be constructed in polynomial time, MS$_{graph}$ is NP hard.

PROBLEM 4 (Challenge!! Not required; worth up to 1 points, suggested length of 2 pages)

Note: On every problem set we will provide a challenge problem, generally significantly more difficult than the other problems in the set, but worth only a few points. It is recommended that if you attempt these problems, you do so only after completing the rest of the assignment.

Minesweeper is traditionally played on a rectangular grid rather than an arbitrary graph, where edges exist between neighboring cells. That is, most cells have edges to the eight surrounding cells.

Show that an analogously-defined language MS_{grid} can be decided in polynomial time.

Solution.
MS_{graph} can have edges from each node to every other node. MS_{grid} has at most eight neigboring nodes and edges for each node, thus reducing the time required to solve from exponential to polynomial time.

PROBLEM 5 (1+2+2 points, suggested length of roughly 5 lines)

Recall that Co-NP $= \{L : \overline{L} \in \text{NP}\}$. It is unknown whether or not NP = Co-NP. Note that NP = Co-NP if and only if NP is closed under complement.

(A) Prove that if NP $\neq$ Co-NP, then P $\neq$ NP. (Aside: Nonetheless, we can't rule out the possibility that NP = Co-NP and yet P $\neq$ NP.)

(B) To prove that NP = Co-NP, it would suffice to show that for every $L \in$ NP, $\overline{L} \in$ NP. Suppose $L \in$ NP. Then there exists a nondeterministic Turing machine M that decides L in polynomial time. Consider the new Turing machine M', which is identical to M except that its accept and reject states are reversed.

What language does M' decide? Explain briefly.

(C)

Solution.

(A)
To prove that if NP $\neq$ Co-NP, then P $\neq$ NP, we can assume the contrapositive P = NP, and prove that, if this is true, then NP = Co-NP

Prove NP $\subseteq$ Co NP
If P = NP then both L $\in P$ and L $\in NP$
Because P is closed under complementation, $\overline{L} \in P$ and consequently $\overline{L} \in NP$
If $\overline{L} \in$ NP then, by definition, L $\in$ Co NP
therefore NP $\subseteq$ Co NP.

Prove Co NP $\subseteq$ NP
If L $\in$ Co NP then $\overline{L} \in$ NP
If $\overline{L} \in$ NP then $\overline{L} \in P$
Again since p is closed under complement, if $\overline{L} \in P$ then $L \in P$ and $L \in NP$ therefore Co NP $\subseteq$ NP

Since NP $\subseteq$ Co NP and Co NP $\subseteq$ NP, then NP = Co-NP

Thus if NP $\neq$ Co-NP, then P $\neq$ NP

Alternatively, if P = NP, we can build a deterministic polynomial time decider M for L, and switch the accept and reject states. Since P is closed under complement, this would achieve the same effect as the above.

(B)
We cannot assume that L(M') is the complement of L(M)
Imagine some word w:
If w $\in \overline{L}$ and w $\notin L$
then all possible computations of M on input w reject, but there exists a computation of M' on w that accepts.
If w $\notin \overline{L}$ and w $\in L$
Then there exists a computation of M on w that accepts and there exists a computation of M' on w that rejects.
For M' on input w there exists both a computation that accepts and there exists a computation that rejects, but M' does not necessarily reject all words not in L(M')
$L(M')$ = all words for which there exists a computation that rejects M' on w