Josh Huff, huffj@oregonstate.edu
Professor Schutfort
CS 325 – Winter 2018
March 4, 2018

Homework 7

---

1. Let X and Y be two decision problems. Suppose we know that X reduces to Y in polynomial time. Which of the following can we infer? Explain

a. If Y is NP-complete then so is X.

False. It is the case that X is in NP-Hard, but it could be that X is not in NP.

b. If X is NP-complete then so is Y.

False. It is the case that Y is in NP-Hard, but it could be that Y is not in NP.

c. If Y is NP-complete and X is in NP then X is NP-complete.

False. X is not necessarily in NP-Hard.

d. If X is NP-complete and Y is in NP then Y is NP-complete.

True. Y is both NP and a NP-Complete problem can be reduced to Y.

e. X and Y can't both be NP-complete.

False. We know X reduces to Y, so if they are both in NP, then they can both be NP-Complete.

f. If X is in P, then Y is in P.

False. X reducing to Y means Y is at least as hard. It could be that Y is more difficult than P.

g. If Y is in P, then X is in P.

True. If X is in P and X reduces to Y, then they are both in P.

2. Consider the problem COMPOSITE: given an integer y, does y have any factors other than one and itself? For this exercise, you may assume that COMPOSITE is in NP, and you will be comparing it to the well-know n NP-complete problem SUBSET-SUM: given a set S of n integers and an integer target t, is there a subset of S whose sum is exactly t? Clearly explain whether or not each of the following statements follows from that fact that COMPOSITE is in NP and SUBSET-SUM is NP-complete:

a. SUBSET-SUM ≤ p COMPOSITE.

False. SUBSET-SUM does not necessarily reduce to COMPOSITE based on COMPOSITE being in NP and SUBSET-SUM being NP-Complete

b. If there is an $O(n^3)$ algorithm for SUBSET-SUM, then there is a polynomial time algorithm for COMPOSITE.

True. The discovery of a polynomial solution for SUBSET-SUM means it is not NP-Complete after all (or that P = NP!), and instead it is a problem in P.

c. If there is a polynomial algorithm for COMPOSITE, then P = NP.

False. We have not proven that SUBSET-SUM reduces to COMPOSITE, so there is no evidence that COMPOSITE is even NP-Complete.

d. If P != NP, then no problem in NP can be solved in polynomial time.

False. P is a proper subset of NP, so some problems in NP can be solved in polynomial time.

3. Two well-known NP-complete problems are 3-SAT and TSP, the traveling salesman problem. The 2-SAT problem is a SAT variant in which each clause contains at most two literals. 2-SAT is known to have a polynomial-time algorithm. Is each of the following statements true or false? Justify your answer.

a. 3-SAT ≤ p TSP.

True. If TSP and 3-SAT are in the same class (NP-Complete), they can reduce to each other.

b. If P != NP, then 3-SAT ≤ p 2-SAT.

False. 2-SAT is known to have a p-time algorithm, so 3-SAT can’t be reduced to it.

c. If P != NP, then no NP-complete problem can be solved in polynomial time.

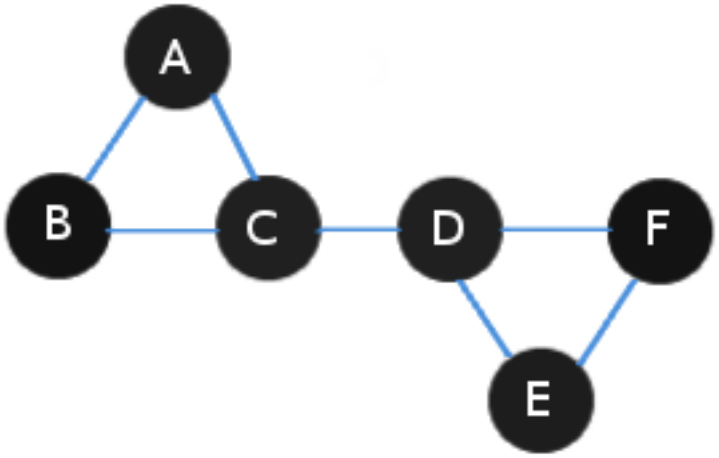
True. It is likely that P != NP, and in that case, NP-complete problems have no p-time solution.

4. A Hamiltonian path in a graph is a simple path that visits every vertex exactly once. Show that HAM-PATH = { (G, u, v): there is a Hamiltonian path from u to v in G} is NP-complete. You may use the fact that HAM-CYCLE is NP-complete

To show that HAM-CYCLE is NP-Complete, we must prove 1) that it is in NP and 2) that HAM-CYCLE, an NP-Complete problem, reduces to HAM-PATH.

1) Prove that HAM-PATH is in NP

HAM-PATH can be proven to be in NP by giving a polynomial-time algorithm that verifies an instance of it. For example, in the graph below, a valid HAM-PATH where u is A and v is F (A to B to C to D to E to F) can be found in p-time using a DFS.



2) Reduce HAM-CYCLE to HAM-PATH (HAM-CYCLE <=p HAM-PATH)

A cycle cannot exist without a valid path (that is, a cycle implies a path). Intuitively speaking, the only difference between a Hamiltonian cycle and a Hamiltonian path is that, in the former, the start and end vertices are adjacent. In the above example graph, an edge from A to F would make the HAM-PATH a HAM-CYCLE, so a HAM-PATH is essentially a HAM-CYCLE with fewer edges.

Furthermore, a HAM-CYCLE can be represented as a HAM-PATH by splitting a vertex such as A into A’ and A’’, where one vertex (A’) receives all incoming edges and the other vertex (A’’) receives all outgoing edges. The effect is a HAM-PATH.

Therefore, a HAM-CYCLE is at least as hard to solve as a HAM-PATH, so they both belong to NP-Hard.

Because HAM-PATH has been demonstrated to be in the classes NP and NP-Hard, we can conclude that HAM-PATH is NP-Complete.

5. LONG-PATH is the problem of, given (G, u, v, k) where G is a graph, u and v vertices and k an integer, determining if there is a simple path in G from u to v of length at least k. Prove that LONG-PATH is NP-complete.

To show that LONG-PATH is NP-Complete, we must prove 1) that it is in NP and 2) that a known NP-Complete problem reduces to HAM-PATH.

1) Prove that LONG-PATH is in NP

LONG-PATH can be proven to be in NP by giving a polynomial-time algorithm that verifies an instance of it. For example, the graph from problem 4 could have edge weights of 1. Given a minimum weight k = 4, and vertices B and F, we see a path from B to C to D to F to E where the cumulative weight of the path satisfies the k requirement and returns a “True” value. (This is, of course, not the shortest path.)

2) Reduce a known NP-Complete problem (for example, HAM-PATH) to LONG-PATH (HAM-PATH <=p LONG-PATH)

If we alter the HAM-PATH, itself a known NP-Complete problem, to include an “accumulator” variable that adds the weight of each edge as it visits its vertex, we will end up with a value to be compared against our minimum weight, k. If this value found using HAM-PATH is smaller than the given k, it will return a “False” value. Otherwise, it will return “True.”

Therefore, a LONG-PATH is at least as hard to solve as a HAM-PATH, so they both belong to NP-Hard.

Because LONG-PATH has been demonstrated to be in the classes NP and NP-Hard, we can conclude that LONG-PATH is NP-Complete.

EXTRA CREDIT. Prove that TPP-D is NP-Complete

To show that TPP-D is NP-Complete, we must prove 1) that it is in NP and 2) that a known NP-Complete problem reduces to TPP-D.

1) Prove that TPP-D is in NP

TPP-D can be proven to be in NP by giving a polynomial-time algorithm that verifies an instance of it. By visiting each vertex in a given solution, and accounting for whether every item in the list of goods is acquired, and tracking the costs of traveling to their respective vertices and the cost of the goods themselves, it is possible to verify whether getting all the required items will not exceed k, the maximum allowable cost. If any one condition is not met, the decision evaluates to “False.”

2) Reduce a known NP-Complete problem (for example, TSP) to TPP-D (TSP <=p TPP-D)

Intuitively speaking, TPP-D is a subset of TSP that has additional complexity; it almost functions like a combination of the Knapsack problem and TSP, both of which are NP-Complete. If we alter the TSP to account for different vertices (marketplaces) being necessary to satisfy a goods requirement and also maintain awareness of their respective costs (i.e. one vertex may have a cheaper price for a good on the “shopping list” than another), as well as the costs for travel (i.e. it may not make sense to travel to a vertex on the other side of the graph to save a little on the cost of a good), we have a specific instance of a TSP. As one academic paper (“Heuristics for the traveling purchaser problem” by Boctor, Laporte, and Renaud) suggests, the TPP becomes the TSP when each vertex represents a store that sells only one item and each item type is available exclusively from one vertex.

Therefore, TSP is at least as hard to solve as a TPP-D, so they both belong to NP-Hard.

Because TPP-D has been demonstrated to be in the classes NP and NP-Hard, we can conclude that TPP-D is NP-Complete.