

Homework 3: NP-Completeness

Assigned: October 15, 2008

Due Date: October 29, 2008

For problems requesting an algorithm, the usual standards for proving correctness and complexity apply. For NP-completeness proofs, I expect you to (1) show that the problem is in NP; (2) give a polynomial-time reduction from one of the problems we discuss in class (or the one in the hint); (3) prove that the reduction is correct *in both directions* (i.e., the iff). You must submit your homework with a signed cover sheet attached to the front.

Core Problems

1. Shorter answer [20 pts, 5 pts each]

- (a) Suppose we know that a problem X is NP-complete. Suppose we discover a polynomial time algorithm for X. Would that imply that the SATISFIABILITY problem can be solved in polynomial time? Explain your answer.
- (b) Suppose we know that a problem X belongs to the class NP. Suppose we discover a polynomial time algorithm for X. Would that imply that the SATISFIABILITY problem can be solved in polynomial time? Explain your answer.
- (c) Suppose we discover an $O(n^3)$ algorithm for SATISFIABILITY. Would that imply that every problem in NP can be solve in $O(n^3)$ time? Why or why not?
- (d) Given an integer $N > 1$ the Factoring problem asks for two integers $p, q > 1$ so that $N = pq$. Consider the following algorithm for Factoring:

```

For all integers i: 2 < i < sqrt{N}
  if i divides N, return i and N=i.
Return prime.

```

Is this a polynomial time algorithm for Factoring? Why or why not?

2. (10 pts) In this problem, you are to prove that the “decision problem” for satisfiability is equivalent in difficulty (up to a polynomial factor) to *actually* finding the assignment of variables that makes the expression true. Specifically, prove that there is a polynomial time algorithm A that solves SAT (i.e. given a boolean formula ϕ , A respond “yes” iff ϕ has a satisfying assignment) if and only if there is some polynomial time algorithm B that when given a boolean formula ϕ it computes a satisfying assignment for ϕ (or reports none exists). *Be sure to show both directions for the “iff”.*
3. (15 pts) Given a set of m linear constraints over n variables, the **integer-programming problem** asks whether there is an *integer* n -vector x giving values for each of the n variables such that all the constraints are satisfied. Prove that integer programming is NP-complete by using a reduction from 3-CNF-SAT (which is the 3-SAT problem we have talked about in class). *Hint: The arithmetic expression $1 - x$ performs a negation when x is 0 or 1.*

4. (15 points) In class, we discussed the *maximum flow problem*. You are given a directed, weighted graph $G = (V, E)$ with a specified source vertex $s \in V$ and a specified sink vertex $t \in V$. Each edge $e \in E$ has a capacity $c(e)$. In class, we formulated the problem of finding a flow f , such that $f(e) \leq c(e)$ for each edge e , no flow is gained or lost at any node between s and t , and the total flow into t (equivalently, the total flow out of s) is maximized.

Consider the variant of the maximum flow problem in which, for every edge e , it must be that either $f(e) = c(e)$ or $f(e) = 0$. That is, each edge is either completely full or completely empty. Prove that this variant is an “NP optimization problem;” that is, formulate its canonical decision problem, and show that this problem is NP-complete. (*Hint*: reduce from SUBSET-SUM.)

The following is the advanced problem, required only for CSE 541 students. 441 students may receive extra credit for a correct solution.

5. (15 points) This week, the extra problem is intended to remind you not all hard problems have to do with NP-completeness.

In Dantes Inferno, the last two circles of Hell punish sins that involve conscious fraud or treachery. The circles can be reached only by descending a vast cliff, which Dante and Virgil do on the back of *Geryon*, a winged monster represented by Dante as having the face of an honest man and a body that ends in a long-tailed stinger. (Canto XVII)

An n -vertex graph is called a *Geryon* if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree $n-2$ (the body) connected to the other $n-3$ (the feet). The connections between the feet may be *arbitrary*, so some or all or none of the feet may be connected to some or all or none of the other feet. Design an $O(n)$ algorithm that decides whether a given adjacency matrix represents a *Geryon*. The picture below shows an example graph *Geryon*, and a depiction of the mythical beast.

