

## Homework 3: NP-Completeness

Assigned: October 15, 2007

Due Date: October 29, 2007

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. (15 points) Consider the following problem, called Scheduling with Deadlines (SWD). The input is a list of  $n$  events; event  $i$  has an earliest start time  $s_i$ , a latest finishing time  $f_i$ , and a length  $\ell_i$ . Each event must run entirely between  $s_i$  and  $f_i$ , and no two events may overlap. The problem is to determine whether or not there exists a feasible schedule that includes all  $n$  events.
  - (a) Professor Doolittle has given the following reduction from the (known to be hard) PARTITION problem to show that SWD is NP-hard. Recall that the input to PARTITION is a set  $X = \{x_1 \dots x_n\}$  of non-negative integers, and we wish to know whether  $X$  can be divided into subsets  $S$  and  $X - S$  such that  $\sum_{x \in S} x = \sum_{x \in X - S} x$ .  
Given an input  $X$  to partition, let  $B = \sum_{i=1}^n x_i$ . Create a list of  $n + 1$  events as follows. For all events  $1 \leq i \leq n + 1$ , set  $s_i = 0$  and  $f_i = B + 1$ . For  $1 \leq i \leq n$ , set  $\ell_i = x_i$ ; finally, set  $\ell_{n+1} = 1$ . The idea here is that, if  $X$  can be partitioned, then SWD can schedule events  $1 \dots n$  in two groups of equal size, separated by event  $n + 1$ .  
What is wrong with the professor's reduction?
  - (b) Fix the above reduction and provide a complete proof that SWD is in fact NP-complete.
2. (10 points) Consider the following multiprocessor scheduling problem (MP-SCHED). The input is a set  $X$  of jobs, with each job  $x \in X$  having (integer) length  $\ell_x$ ; a number of processors  $m$ ; and an (integer) deadline  $d$ . Each job can run on any machine, but only one job can run at a time on each machine. The problem is to determine whether there exists a partition of the jobs into subsets  $X_1 \dots X_m$ , such that all jobs in set  $X_i$  run on machine  $i$ , and for all  $i$ ,  $\sum_{x \in X_i} \ell_x \leq d$ . (That is, all machines finish by time  $d$ ).  
Prove that MP-SCHED is NP-complete. (*Hint*: reduce from PARTITION.)
3. (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.)

## Advanced Problem

**This problem is *required* only for CSE 541 students.** 441 students may receive extra credit for a correct solution.

4. (15 points) A shortcoming of our notion of decision problems is that they are restricted to reporting that a solution exists, without actually specifying it. For example, an algorithm for SUBSET-SUM reports *whether* some subset of its input sums to a target  $t$ , but it does not say *which* subset yields this sum. This limitation seems rather major, since we normally want to see a solution, not just to know that it exists.

The following two questions ask you to show that, at least for two of the problems we’ve discussed, solving the decision problem is “as good as” having the solution in hand.

- (a) Suppose you have an “oracle” that correctly answers any instance of SUBSET-SUM in time polynomial in its input size. Show that there exists a polynomial-time algorithm using this oracle that, given an instance  $(X, t)$  of SUBSET-SUM, either returns a subset  $X' \subseteq X$  that sums to  $t$ , or determines that no such subset exists.
- (b) Suppose you have an oracle that correctly answers any instance of SAT in time polynomial in its input size. Show that there exists a polynomial-time algorithm using this oracle that, given a boolean formula  $\phi$ , either returns a satisfying assignment for  $\phi$ , or determines that no such assignment exists.