

## Homework 3 Solutions

1. (a) The problem with the reduction given here is that it does not sufficiently constrain the SWD instance. In particular, there exist inputs  $X$  to PARTITION for which the SWD instance  $f(X)$  has a feasible schedule, but  $X$  is not partitionable. For example, consider  $X = \{3, 5\}$ . The corresponding SWD problem has 3 jobs, of lengths 3, 5, and 1, that must run in the interval  $[0, 9]$ . A valid schedule starts the three jobs at times 0, 3, and 8 respectively, but  $X$  clearly cannot be partitioned. (Indeed, *every* instance of PARTITION yields a schedulable instance of SWD under this reduction!)
- (b) To correct the professor's reduction, we make the following tweak: the unit-length job  $n + 1$  is assigned  $s_{n+1} = B/2$  and  $f_{n+1} = B/2 + 1$ . All other jobs are constrained as before. This modification to the reduction does not change the fact that it runs in time  $O(|X|)$ .

**Claim 1:** under the new reduction, if  $X$  is partitionable, then the scheduling instance  $f(X)$  is schedulable.

**Pf:** Let  $S, X - S$  be a partition of  $X$ . First, schedule all jobs in  $S$ , one immediately after the other. The total length of this sub-schedule is the total size of  $S$ , which is  $B/2$ . Then, schedule job  $n + 1$  from time  $B/2$  to time  $B/2 + 1$ . Finally, schedule the remaining jobs, one after the other, in the interval  $B/2 + 1, B + 1$ , which also has total size  $B/2$ . The resulting schedule includes all jobs, is non-conflicting, and respects all start and end constraints.

**Claim 2:** under the new reduction, if an SWD instance  $f(X)$  is schedulable, then the PARTITION instance  $X$  is partitionable.

**Pf:** The total length of all jobs in  $f(X)$  is precisely  $B + 1$ , which is also the length of the available scheduling interval (since all jobs start at or after 0 and end at or before  $B + 1$ ). Hence, any valid schedule completely fills the interval. Let  $Y$  be the subset of jobs that end before time  $B/2$ , and let  $Z$  be the subset that begin after  $B/2 + 1$ . The total length of the jobs in  $Y$  is exactly  $B/2$ , as is the total length of the jobs in  $Z$ . Let  $S$  be subset of  $X$  whose elements correspond to the jobs in  $Y$ ; then  $X - S$  corresponds to the jobs in  $Z$ . We have that the total sizes of  $Y$  and  $Z$  are each  $B/2$ , so the total sizes of  $S$  and  $X - S$  are also  $B/2$ . Conclude that  $(S, X - S)$  is a valid partition of  $X$ .

Finally, to show NP-completeness, we need to argue that SWD is in NP. Given any true instance  $x$  of SWD, a certificate  $c$  for  $x$  is a schedule, which is a list of starting times for every job in  $x$ . This list has size  $O(|x|)$ . To check that  $x$  is a true instance given  $c$ , we look at the scheduled start times and confirm that no jobs overlap, and that no job is scheduled outside its start and end constraints. This can be done in time  $O(|x|^2)$ .

2. We first show that MP-SCHED is in NP. For any true MP-SCHED instance  $(X, m, d)$ , a certificate  $c$  is a valid assignment of the jobs in  $X$  to machines. This certificate has size  $O(|X| \log m)$ , which is polynomial in the input size. To check the certificate, we simply add

up all the jobs for each machine and ensure that the sum of their lengths is at most  $d$ . This takes time  $O(|X| + m)$ . (Note that in any nontrivial instance,  $|X| \geq m$ , so it's OK for the verifier to run in time  $O(m)$ , rather than  $O(\log m)$ .)

We now show that  $\text{PARTITION} \leq_p \text{MP-SCHED}$ . Let  $X$  be an instance of partition, and let  $B$  be the total length of all elements of  $X$ . For each  $x \in X$ , let  $j_x$  be a job with length  $|x|$ , and let  $J$  be the set of all such jobs. Then the MP-SCHED instance  $f(X)$  is  $(J, 2, B/2)$ . It takes  $O(|X|)$  time to create  $J$  and to compute  $B$ , so  $f(X)$  can be computed from  $X$  in time polynomial in its size.

**Claim 1:** if  $X$  is partitionable, then  $f(X)$  is schedulable.

**Pf:** Let  $S, X - S$  be a partition of  $X$ . For every  $x \in S$ , assign  $j_x$  to machine 1. Assign the remaining jobs to machine 2. By construction of the job lengths, the total length of jobs for each machine is precisely  $B/2$ , so this schedule satisfies the deadline.

**Claim 2:** if  $f(X)$  is schedulable, then  $X$  is partitionable.

**Pf:** Let  $J_1, J_2$  be the sets of jobs assigned to machines 1 and 2 by a valid schedule for  $f(X)$ . We have that the total length of  $J_1$  and  $J_2$  are each at most  $B/2$ . But the total length of  $J_1 \cup J_2$  is  $B$ , so each of these sets must have length *exactly*  $B/2$ .

Let  $S$  be the subset of  $X$  whose elements correspond to the jobs in  $J_1$ . Then  $S$  has total size  $B/2$ , as does  $X - S$ , and so  $X$  is partitionable.

*Note:* in this and the previous problem, a false instance of PARTITION might have entries that add up to an odd value. If we interpret the divisions above as integer division (i.e. taking the floor), we need not worry about inconsistencies in interpretation across the problems.

3. The canonical decision problem for this 0-1 max flow problem is: "Given a directed, weighted graph  $G$  with source and target vertices  $s$  and  $t$ , does there exist a flow for  $G$  such that each edge is either filled to capacity or empty, and the total flow is *at least*  $k$ ?" Call this decision problem FLOW-01.

We claim that FLOW-01 is NP-complete. We first show that it is in NP. For any true instance  $(G, s, t, k)$ , a certificate is the list of edges with nonzero flow, which has size  $O(|G|)$ . We can check the following properties:

- For every vertex  $v$  except  $s$  and  $t$ , the total flow into  $v$  equals the total flow out.
- The total flow out of  $s$  is at least  $k$ .

if  $G = (V, E)$ , these checks can be done in time  $O(|V||E|)$ .

We now show that  $\text{SUBSET-SUM} \leq_p \text{FLOW-01}$ . let  $(X, t)$  be an instance of subset sum. We define  $G$  as follows.  $G$  contains three designated vertices  $a, b$ , and  $c$ , along with a vertex  $v_i$  for every  $x_i \in X$ . For each  $1 \leq i \leq n$ , there exist edges  $a \rightarrow v_i$  and  $v_i \rightarrow b$  with capacity equal to  $x_i$ . Finally, there exists an edge  $b \rightarrow c$  with capacity  $t$ . The FLOW-01 instance is then  $(G, a, c, t)$ .

**Claim 1:** if  $X$  has a subset that sums to  $t$ , then  $G$  has a valid flow of at least  $t$ .

**Pf:** Let  $X'$  be the subset that sums to  $t$ . For each  $x_i \in X'$ , push  $x_i$  units of flow along the edges  $a \rightarrow v_i$  and  $v_i \rightarrow b$ . The total flow into  $b$  is then exactly  $t$ , so push  $t$  units of flow along the edge  $b \rightarrow c$ . The result is a valid flow that pushes  $t$  units into  $c$ .

**Claim 2:** if  $G$  has a valid flow of at least  $t$ , then  $X$  has a subset that sums to  $t$ .

**Pf:** every valid flow on  $G$  is *at most*  $t$ , since no more than  $t$  units can flow into  $c$ . Consider a flow  $\phi$  of at least  $t$  on  $G$ ; this flow must be *exactly*  $t$ . Because no flow is gained or lost at  $b$ , there must be a collection of edges  $v_i \rightarrow b$  with nonzero flow, such that their total flow is  $t$ . Let  $X'$  be the set of all  $x_i$  such that  $v_i$  is nonzero in  $\phi$ . By construction of  $G$ ,  $X'$  sums to  $t$ .

4. (a) The following algorithm returns the desired subset for an instance  $(X, t)$ . First, use the oracle to check whether  $(X, t)$  is solvable. If not, fail. Otherwise, order the elements of  $X$  arbitrarily, and for each  $i$  from 1 to  $|X|$ , proceed as follows. Check whether  $(X - \{x_i\}, t)$  is solvable. If so, delete  $x_i$  from  $X$  and continue; otherwise, just continue with  $X$  unchanged. After all elements have been processed, return the resulting reduced set  $X$  as the solution.

Suppose  $(X, t)$  is solvable. We first observe that each processing step from 1 to  $|X|$  preserves the solvability of  $(X, t)$ . Indeed, we remove an element from  $X$  only if doing so does not cause the subset-sum oracle to report “false,” so we do so iff the resulting reduced set still has a subset that sums to  $t$ .

After all steps have finished, we are left with a final subset  $X' \subseteq X$ , such that  $X'$  contains a subset that sums to  $t$ . We now argue that  $X'$  itself sums to  $t$ . Suppose not; then  $X'$  has a sum greater than  $t$ , else it could not have a subset that sums to  $t$ . Let  $S$  be the subset of  $X'$  that sums to  $t$ ; then there exists some lowest-numbered  $x_i \in X' - S$ . But  $x_i$  would have been removed in step  $i$ , since doing so would have left the subset  $S$  in  $X$ , and so the oracle would have returned true. Conclude that  $x_i$  cannot exist, and so  $X'$  sums to  $t$  as desired.

This algorithm makes  $O(|X|)$  calls to the oracle and performs only  $O(|X|)$  additional work.

- (b) The following algorithm returns a satisfying assignment  $A$  for a formula  $\phi$  over  $n$  variables  $x_1 \dots x_n$ . First, check whether  $\phi$  is satisfiable; if not, fail. Otherwise, set  $\phi_0 = \phi$ , and for each  $i$  from 1 to  $n$  proceed as follows. Set  $\phi' = \phi_{i-1} \wedge x_i$ , and check whether  $\phi'$  is satisfiable. If so, set  $\phi_i = \phi'$  and add  $x_i = \text{true}$  to  $A$ . Otherwise, set  $\phi_i \rightarrow \phi_{i-1} \wedge \neg x_i$ , and add  $x_i = \text{false}$  to  $A$ . After all  $x_i$  have been processed, return the final assignment.

We claim that  $A$  is a satisfying assignment for  $\phi$ . To show this, we first argue inductively that, after any number of variables have been assigned, the partial assignment  $A$  is *consistent* with some satisfying assignment to  $\phi_i$ ; that is, there exists a satisfying assignment  $B$  that does not contradict the truth values in  $A$ .

**Bas:** Before any variables are assigned,  $A$  is empty and so does not conflict with any satisfying assignment to  $\phi_0 = \phi$ .

**Ind:** We know that there exists some satisfying assignment to  $\phi_{i-1}$  consistent with  $A$ . If  $\phi'$  is satisfiable, then there is a satisfying assignment  $B$  for  $\phi'$  that sets  $x_i = \text{true}$ , since  $B$  must satisfy the subformula  $x_i$  of  $\phi'$ . In this case, we set  $x_i = \text{true}$  in  $A$ , which leaves it consistent with  $B$ . Otherwise, any satisfying assignment  $B'$  for  $\phi_{i-1}$  consistent with  $A$  must set  $x_i = \text{false}$ ; hence,  $B'$  satisfies  $\phi_{i-1} \wedge \neg x_i$ . In this case, we set  $x_i = \text{false}$  in  $A$ , which is consistent with  $B'$ .

To conclude, we have that, after all  $x_i$  have been processed,  $A$  is consistent with a satisfying assignment to  $\phi_n$ . Note that the final  $A$  assigns every variable of  $\phi_n$ , so it is a

complete truth assignment; moreover,  $A$  must satisfy every conjunct of  $\phi_n$ , one of which is the original formula  $\phi$ . Hence,  $A$  is a satisfying assignment to  $\phi$ .

The above algorithm makes  $O(n)$  calls to the SAT oracle and otherwise does  $O(n)$  work.

**Note:** there is more than one way to constrain a formula to make certain variables true or false while leaving it a valid Boolean formula. It would also be OK to substitute “true” or “false” for a variable in  $\phi$ , then partially evaluate  $\phi$  until we are left with a new, smaller formula or a truth value (in the latter case, the remaining assignment does not matter). It would also be OK to make a variable true by replacing it everywhere with a tautology  $(q \vee \neg q)$ .