Homework 6

Frederick Robinson and Mykell Miller

25 February 2010

1 Problem 8.1

1. *True.* We know already that interval scheduling is polynomial and that vertex cover is NP complete. Even if $P = NP$ this is still true: they would be equal.

2. *We don't know.* If $P = NP$ then this would be true. If not, then it is false.

2 Problem 8.4

2.1 Part A

Since a special case of the problem (Part D) is NP-Hard the entire problem must be NP-Hard. Moreover since a set of k or more processes which are compatible is a poly-sized certificate whose validity can be checked in poly-time (just check to make sure that no two processes in the certificate request the same resource). The problem is NP-Complete.

2.2 Part B

Given a set of two processes we can check to see whether or not they are compatible in polynomial time. We need only check to see whether they have any resources in common. So, since there are only polynomial combinations of two processes we can do this special case by brute force.

2.2.1 Algorithm

1. For i in 1 to n:
   (a) For j in i to n:
      i. If checkcompatibility(i,j) == True: Return True
   (b) End For
2. End For

3. Return False

Checkcompatibility(i,j):

1. For k in processes i requests:
   (a) If j requests it too: Return False
2. End For
3. Return True

2.2.2 Runtime

Checkcompatibility runs $O(n^2)$ times. Checkcompatibility takes $O(m)$ time to run. Therefore,the total is just $O(mn^2)$

2.2.3 Correctness

We check each possible combination for compatibility. Thus, if there is a compatible pair we will find it and return "compatible." Conversely if we find no compatible pair in our search of all combinations there cannot exist a compatible pair since we search all of them. Thus, our algorithm is returns True if and only if there exists a compatible pair.

2.3 Part C

We will turn this into a network flow problem and use a network flow algorithm to compute our answer. Then, we convert back.

2.3.1 Algorithm

1. For each process p, create an edge from the person requested by p to p with capacity 1.
2. For each process p, create an edge from p to the equipment requested by p with capacity 1.
3. For each person requested by any process q, create an edge from s to q with capacity 1.
4. For each equipmntt requested by any process e, create an edge from e to t with capacity 1.
5. Find the maximum network flow from s to t. Feel free to stop when the flow reaches k
6. If the maximum network flow is $\geq k$, return True.
7. Else, return False.

2.3.2 Runtime

Set up takes $O(n)$ because you have to create at most 4 edges per process. Max flow takes $O(M^2 \log C)$, where M is the number of edges in the graph and $C = k$. $M = O(n)$ so max flow takes $O(n^2 \log k)$. Returning takes $O(1)$. Therefore, the total run time is $O(n^2 \log k)$.

2.3.3 Correctness

An allowable integral flow of k corresponds to a 'legal' setup with k active processes. Conversely, we can construct a flow of k given a setup with k active processes. If there is such a flow then it goes through k processes and their corresponding resources with no duplicate resources. If a resource were to correspond to more than one process it would violate flow conservation. The converse is equally easy to see as we just construct the flow that goes through each person, process, and piece of equipment in the resource solution. This clearly conserves flow.

Now we prove that maximal network flows correspond to maximal solutions to this problem.

Proof. Suppose that the maximum network flow is $< k$, but there can be at least k active processes. Then there are at least k $s - t$ paths with flow of 1 on them, one for each active process and the person and equipment it requests. This flow is at least k. This is a contradiction.

Suppose that the maximum network flow is $\geq k$, but there can be at most $k - 1$ active processes. Then there are $\geq k$ processes for which they can reach unique people and equipment, and so they can all be active at the same time. There are $\geq k$ active processes. This is a contradiction. □

2.4 Part D

We claim that this problem is NP-Complete.

First we prove lemmas 2.1 through 2.3, as they together are used to prove that the problem is NP-hard.

Lemma 2.1. *A solution to this special case $\mathcal{O}$ leads to an algorithm for the independent set problem in polynomial calls to $\mathcal{O}$.*

Proof. Take a graph with n vertices and m edges for which we want to do independent set. Declare each vertex to be a process. For each edge, declare it to be a resource and make it requested by the nodes adjacent to it. Call $\mathcal{O}$. The vertices corresponding to the active processes are the independent set. It takes only $O(n + m)$ to convert the independent set instance to a resource allocation instance, so if $\mathcal{O}$ successfully solves independent set, then INDEP-SET $\leq_P$ Resource Allocation. □

Now we prove our claim of correspondence:

Lemma 2.2. *The solution returned by $\mathcal{O}$ is an independent set.*

Proof. Assume it is not an independent set. Then 2 adjacent nodes are selected. But these 2 nodes are 2 processes requesting the same resource, so they will not be selected. This is a contradiction. □

Lemma 2.3. *The solution returned by $\mathcal{O}$ is a largest independent set.*

Proof. Assume the largest independent set S is size $|S| = k$. Then there are k nodes, none of which are adjacent to each other. In other words, there are k processes, none of which request the same resource. Therefore, the set S can be chosen. We claim the Resource Reservation Problem will choose S unless it finds another solution of size k. If it finds another solution of size k, then it is still a largest independent set. If it finds a solution of size greater than k, then S was not the largest independent set which is a contradiction. If it finds a solution of size less than k, it will also find S and throw away the smaller solution. □

Theorem 2.4. *Our problem is NP-complete*

Proof. Lemmas 2.1, 2.2, and 2.3 show that our problem is NP-hard, since independent set is a special case of it. Since we proved in Part A that there exist poly-sized certificates whose accuracy can be verified in poly-time there are such certificates in this special case. Thus, the problem is NP-Complete. □

3 Problem 8.12

Theorem 3.1. *Evasive path is in NP.*

Proof. Given an $s - t$ path, one can traverse the path, and at each vertex that is in a zone, we can mark the zone as "visited." If we ever re-visit a node, we will try to mark it as "visited" when it already has been marked, and therefore know the path is not evasive. If we never re-visit a node, we will never try to mark an already-"visited" node as "visited." Since there are $O(n)$ vertices in the path, we can check the entire path in polynomial time. □

We reduce 3-SAT to Evasive Path using the following algorithm.

3.1 Algorithm

1. Order each of the m clauses arbitrarily.
2. For each of the $3m$ literals, create a vertex.
3. For each literal in the first clause, create an edge from s to that literal.
4. For i from 2 to m:
   (a) For j from 1 to 3:
      i. For k from 1 to 3:

   A. Create an edge from the jth literal in the (i-1)th clause to the kth literal in the ith clause

5. For each literal in the last clause, create an edge from that literal to t.
6. For each pair of inconsistent literals z, $\neg z$, create a zone containing that pair.
7. If the created graph contains an evasive path, the 3-SAT function is satisfiable.
8. Else, it is unsatisfiable.

Lemma 3.2. *3-SAT can be reduced to Evasive Path in polynomial time*

Proof. There are $O(m)$ vertices, and $O(m)$ edges in the graph, so it creates the graph in $O(m)$ time. Since a variable or its negation may only occur once in each clause, and it can only be inconsistent with its negation, each literal can only be inconsistent with $O(m)$ other literals. There are $O(m)$ literals, so there are $O(m^2)$ inconsistencies. Thus, the reduction occurs in $O(m^2)$ time. □

Lemma 3.3. *If the 3-SAT instance f is satisfied, then the reduction has an evasive path.*

Proof. Assume f is satisfied. Then there is a way to assign variables such that one or more literals in each clause is true. Then we can select the corresponding vertices and no two vertices are in the same zone. Since every vertex in each clause is connected to every vertex in the next, and there is one or more selectable vertex in each clause, there exists an $s - t$ path consisting only of selectable vertices. This path is an evasive path. □

Lemma 3.4. *If the reduction has an evasive path, then the 3-SAT instance f is satisfied.*

Proof. Assume the reduction has an evasive path. Then for each clause, there is some vertex on the path that does not share a zone with any other vertex on the path. Since inconsistency between two literals implies the corresponding two vertices are in the same zone, and we know that there are no two vertices on the path are in the same zone, no two corresponding literals are inconsistent. Therefore, we can choose the corresponding literals and know that they are all consistent. Since the $s - t$ path includes all clauses, we know the corresponding literals include all clauses, and therefore f is satisfied. □

Theorem 3.5. *Evasive Paths is NP-complete*

Proof. Lemma 3.1 shows that there exists a polynomial-time certificate for Evasive Path and therefore Evasive Paths is in NP. Lemma 3.2 shows that 3-SAT can be reduced to Evasive Paths in polynomial time. Lemmas 3.3 and 3.4 show that a 3-SAT instance f is satisfied if and only if its reduction to Evasive Paths is satisfied. Thus, 3-SAT $\leq_P$ Evasive Paths. Therefore, Evasive Paths is NP-hard. Since Evasive Paths is in NP and it is NP-hard, it is NP-complete. □

4 Problem 8.26

I will show that a solution to the Number Partitioning problem in polynomial time can be converted to a polynomial time solution to the Subset Sum problem.

Assume that we have a set of positive integers $W = \{w_1, w_2, \ldots, w_n\}$ along with a positive integer k and we wish to determine whether there exists some subset, say $S = \{s_1, s_2, \ldots, s_m\}$ with $S \subseteq W$ and $\sum_{i=1}^{m} s_i = k$. This is the Subset Sum problem.

Furthermore, suppose that we have an algorithm $\mathcal{O}$ which outputs the solution to the number partitioning problem. I claim that we can construct an algorithm $\mathcal{A}$ which outputs the answer to Subset Sum in polynomial calls to $\mathcal{O}$ (in particular 1).

Proof. Say for notational convenience that $\sum_{i=1}^{m} s_i = |S|$ and similarly $\sum_{i=1}^{n} w_i = |W|$. There are three cases.

Case 1: $2k = |W|$

Just apply $\mathcal{O}$ to the set W. If there exists a set S such that $|S| = k$ then $|W \backslash S| = k$ since each $w_i \in W$ is a positive integer. Thus, there is a solution to Subset Sum. In particular both W and $W \backslash S$ are solutions.

Conversely if $\mathcal{O}$ does not output a solution then there is no subset S such that $|S| = k$ since, if there were then there would exist a corresponding partition of W into a disjoint union of sets with sums k. This disjoint union would be a solution to the number partitioning problem and would be output by $\mathcal{O}$. So in this case we can compute $\mathcal{A}$ in polynomial calls to $\mathcal{O}$.

Case 2: $2k > |W|$

Form a new set as $N = W \cup \{2k - |W|\}$. This is a set of positive integers since by assumption each $s_i \in S$ is a positive integer and so since $2k$ is also a positive integer we know that $(2k - |W|)$ is an integer. In particular by the case assumption we know that it is positive.

Now again just run $\mathcal{O}$ on N. By the first case we know that $\mathcal{O}$ outputs a solution if and only if there exists a partition of N into a disjoint union of two subsets each which has sum equal to $\frac{1}{2}|N| = \frac{1}{2}(|W| + 2k - |W|) = k$. Since we added only one member to W in order to form N one of these subsets (if they exist) must be comprised wholly of members from W.

Moreover, if such a partition does not exist then there is no subset of N (and therefore no subset of W since $W \subset N$) whose sum is k. For, if such existed then it would have been output by $\mathcal{O}$.

So in this case we can compute $\mathcal{A}$ in polynomial calls to $\mathcal{O}$.

Case 3: $2k < |W|$

There exists some subset S of W with $|S| = k$ if and only if there exists a subset $S' \subseteq W$ with $|S'| = |W| - k$. If there exists such a subset then $W \backslash S' = S$. But, if $2k < |W|$ then we know that $2|S'| = 2(|W| - k) > |W|$ since $|W| > 2k \Rightarrow |W| > k \Rightarrow 2(|W| - k) > |W|$.

Now, by Case 2 we can determine whether or not there exists an S' in polynomial calls to $\mathcal{O}$ since S' falls into that case and again we can compute $\mathcal{A}$ in polynomial calls to $\mathcal{O}$. □

This reduction shows that our problem is NP-Hard.

Since a subset of S of W with $2|S| = |W|$ is a polysized certificate whose validity can be checked in polytime (just compute the sum $|S|$) it is NP complete.