CS 577: Introduction to Algorithms Fall 2020

Homework 11 Solutions

Instructor: Dieter van Melkebeek TA: Ryan Moreno

Problem 1

You are given a directed graph $G = (V, E)$, two vertices $s, t \in V$, and a partition of the set of edges: $E = \cup_{i=1}^{k} E_i$, where the sets E_i are pairwise disjoint. Your goal is to find a path from s to t that contains exactly one occurrence of an edge of each E_i, or report that none exists.

Let n denote the number of vertices of the given digraph G, and m the number of edges. The problem is in NP as candidate witnesses can be described by binary strings of length $O(k \cdot \log m) = O(n \log(n^2)) = O(n \log n)$, namely the concatenation of the binary indices of the edges that constitute a purported path from s to t. Checking the validity of such a candidate solution entails verifying that it represents an actual path from s to t in G, and that the path contains exactly one occurrence of an edge from each partition class E_i for $i \in [k]$. All of this can be done in time polynomial in n.

To argue that the problem is NP-hard, we exhibit a polynomial-time reduction from the Hamiltonian cycle problem on directed graphs to the new problem. Let x denote a given instance of the Hamiltonian cycle problem, consisting of a directed graph $G = (V, E)$. For each $v \in V$, we define E_v as the set of all edges in E that emanate from v, i.e., all edges in E of the form (v, w). Let v^* denote an arbitrary vertex in V. We let x' denote the instance of the given problem described by $G = (V, E)$, $s = t = v^*$, and the partition $E = \cup_{v \in V} E_v$.

Note that the problem statement requires the sets E_v to be nonempty, which may not be the case in our construction. However, if the set E_v for some $v \in V$ is empty, G cannot have a Hamiltonian cycle, so there is no need to make a call to the blackbox for our problem. In the sequel we assume that none of the sets E_v is empty.

Claim 1. *There is a one-to-one and onto correspondence between Hamiltonian cycles in G and solutions to the instance x' of the given problem.*

Proof. By construction, a path from $s = v^*$ to $t = v^*$ that contains exactly one occurrence of an edge of each E_v for $v \in V$ is precisely a cycle that leaves every vertex $v \in V$ exactly once, where we view v^* as the arbitrary start and end vertex of the cycle. As a cycle that leaves every vertex of a directed graph exactly once is precisely a Hamiltonian cycle, the claim follows. □

In particular, G has a Hamiltonian cycle iff the constructed instance x' of the given problem has a solution. Thus, we have obtained a reduction from the Hamiltonian Cycle decision problem on digraphs to the given decision problem. The partition $E = \cup_{v \in V} E_v$ is essentially the adjacency list representation of G; it follows that x' can be computed from x in polynomial time. Thus, our reduction is polynomial-time computable.

Problem 2

Recall the Interval Scheduling problem from class. Consider the variant, Multiple Interval Scheduling, in which each job may require the machine to be reserved for multiple time intervals. Show that this variant is NP-hard.

We show that Multiple Interval Scheduling is NP-hard. In this problem we are given n jobs and associated time intervals for each job, and we are asked to determine the maximum number of jobs we can schedule so that no two scheduled jobs have any overlap in time.

To show that the problem is NP-hard, we demonstrate a reduction from Independent Set. The intuition here is that these two problems have a similar flavor: in both we are seeking to choose a maximum number of items that do not "interfere" with each other. We'll construct our reduction so that adjacent vertices in an instance of Independent Set correspond to overlapping jobs in the instance of Multiple Interval Scheduling we construct, and vice-versa.

Suppose we are given a graph G, with n vertices and m edges, and asked to find the size of the largest independent set in G. We construct an instance of Multiple Interval Scheduling with n jobs (one for each vertex) and m time intervals. That is, we divide our "day" up into m indivisible "hours", $t_1, t_2, \cdots, t_m$. Now, we label the edges of G as $e_1, e_2, \cdots, e_m$, and for each edge $e_i = (u, v)$, we say that jobs u and v both need to use the processor during time interval t_i.

Thus, each edge of G prevents the jobs corresponding to its endpoints from being accepted simultaneously. Conversely, if two jobs in our schedule cannot be accepted simultaneously, then their corresponding vertices must be connected in G, since two jobs will have overlapping time intervals only when there is an edge between their corresponding vertices in G.

Hence, we can accept k jobs in the scheduling problem if and only if there are no edges between the k corresponding vertices in G, so the maximum number of jobs we can schedule is equal to the size of the largest independent set in G. This reduction can be accomplished in polynomial time since we need only check each pair of vertices in G to see if they are connected by an edge. So, we've given a polynomial time reduction from Independent Set to Multiple Interval Scheduling, showing that Multiple Interval Scheduling is NP-hard, as desired.

Problem 3

You're president of the student government and planning a flashmob dance. You can choose k students to tell about the flashmob tonight. Tomorrow, these k students will tell everyone they have a class with. Is there a way to choose k students such that all of the students at your school hear about the flashmob by the end of the school day tomorrow? If so, what is a group of k students you could tell tonight?

The input is a list of m sets C_c, each of which represents the students enrolled in class c. There are n students and m classes, and each student is enrolled in at least one class. Show that the Flashmob problem is NP-complete.

For simplicity, we call S a student cover if S is a set of students such that we can tell the students in S about the event tonight and they will tell all other students about the event by tomorrow.

First we show that Flashmob is in NP. Given a candidate student cover, we first verify that it is size at most k. Next we can cross the students in the student cover off our full list of students and then iterate over the m classes, crossing off all students in the class (at most n) if one of the students in the student cover is enrolled. After iterating over all of the classes, if all of the students are crossed off then the original list of k students is in fact a student cover. Therefore, Flashmob is in NP.

We show that the problem is NP-hard by reducing from Vertex-Cover, which we choose because of its similarities to Flashmob. In Flashmob, we are trying to cover all students using at most k students. In Vertex-Cover, we are trying to cover all edges using at most k vertices. A key difference between these two problems is that Vertex-Cover uses *vertices* to cover *edges* while Flashmob uses *students* to cover *students*. We will solve this problem by transforming both vertices and edges from the Vertex-Cover input into students for the Flashmob input.

To develop the intuition for our reduction visually, let's consider graph G' where vertices are students and edges connect two vertices if the corresponding students take a class together. Note that a set of vertices S is a student cover on graph G' iff every vertex is in S or adjacent to a vertex in S. We want to build a graph G' such that a vertex cover of size k exists on G iff a student cover of size k exists on G'.

If we simply take $G' = G$, we run into problems. First off, if we have a vertex cover on G, we don't necessarily have a student cover as seen in Figure 1. This is because a vertex cover on G does not need to contain any isolated vertices in G (since they are not endpoints of an edge). However, a student cover must contain all isolated vertices (since they have no adjacent vertices to cover them). This issue can be easily solved by simply deleting the isolated vertices in G.

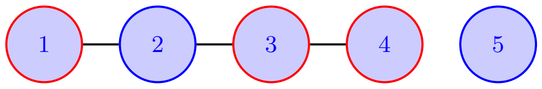

Figure 1: The red nodes form a vertex cover on the graph but not a student cover.

We now consider taking G' as G where G is a connected graph. This still does not work because a student cover on G' is not necessarily a vertex cover on G. As an example, in Figure 2, the highlighted student cover S is not a vertex cover. S covering all vertices does not necessarily mean it covers all edges; it is possible that neither endpoints of an edge $e = (v_a, v_b)$ are in S if both endpoints are covered by neighbors other than v_a and v_b. We can solve this problem by adding an extra vertex v_e that connects to v_a and v_b in G', as seen in Figure 3. We do this for each edge $e = (v_a, v_b)$. This forces a student cover S to select v_e, v_a, or v_b in order to cover v_e. If v_a or v_b are in S, then this solves the problem of covering e in the original graph G. If neither v_a nor v_b is in S, then we know v_e is in S. Since v_a, v_b, and v_e are all connected and v_e is not connected to any other vertices, we can remove v_e from S and replace it by v_a or v_b. This way, S is still a student cover of size k on G'. Further, S is now a vertex cover of size k on G.

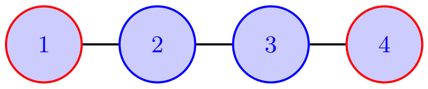

Figure 2: The red nodes form a student cover on the graph but not a vertex cover.

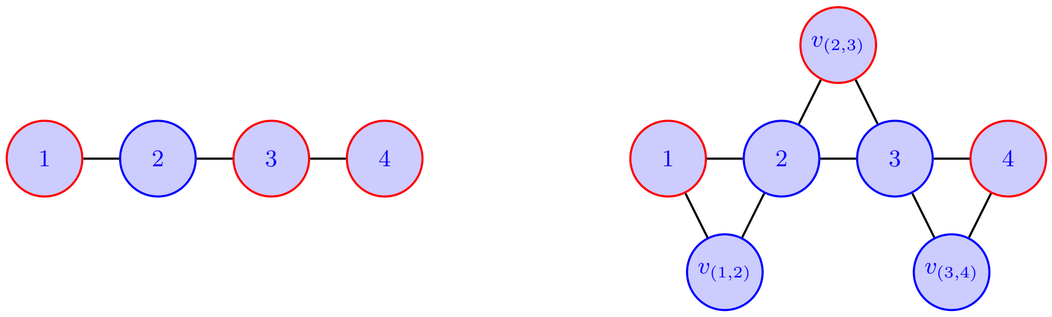

Figure 3: The left graph is G, a connected graph which is input to Vertex-Cover. The right graph is G', our input to Flashmob. A student cover on G' is highlighted, as well as the corresponding vertex cover on G.

Now that we have intuition, we state the reduction in terms of the Flashmob problem. Given the input graph $G = (V, E)$, we need to transform this graph into a set of enrolled students C_j for each class j. We create a student s_{v_i} for each non-isolated vertex in G and a student s_{e_j} for each edge. For each edge e_j in G we create a unique class c_{e_j} with students s_{e_j}, s_{v_a}, and s_{v_b} (this corresponds to having pairwise edges between s_{e_j}, s_{v_a}, and s_{v_b} in G' from the intuition section).

Claim 2. *There exists a vertex cover on G of size k if and only if there exists a student cover of size k on our new input.*

Proof. $\Rightarrow$ Given $I \subseteq V$, a vertex cover of size k on G, we construct a student cover S of size k for our new input by simply setting $S = I$ (meaning $s_{v_i} \in S$ iff $v_i \in I$). By construction, we know that every student s_{e_j} corresponding to edge $e_j = (v_a, v_b)$ is in class c_{e_j} along with s_{v_a} and s_{v_b}. Because I is a vertex cover, we know that either s_{v_a} or s_{v_b} is in $S = I$. This means that every student of the form s_{e_j} is covered. We also know that every student of the form s_{v_i} is covered because we only included non-isolated vertices as students, so student s_{v_a} is in class c_{e_j} which also has student s_{v_b} enrolled, where there is some edge $e_j = (v_a, v_b)$. Because $I = S$ is a vertex cover on G, we know that either s_{v_a} or s_{v_b} is in S, and each will cover the other. So, all students are covered by S.

$\Leftarrow$ Given S, a student cover of size k on the new input, we construct a vertex cover I of size k on G. If S only uses students of the form s_{v_i}, then we know at least one vertex out of v_a and v_b was chosen from class c_{e_j} in order to cover s_{e_j} for every edge in G $e_j = (v_a, v_b)$. So, if we take $I = S$, we know at least one endpoint of each edge is in I, making I a vertex cover of size k for graph

G. If S contains a student of the form s_{e_j}, we cannot simply take $I = S$ because s_{e_j} corresponds to an edge rather than a vertex. Instead of choosing s_{e_j}, we modify S to choose either s_{v_a} or s_{v_b} arbitrarily, where $e_j = (v_a, v_b)$ is an edge in G. Because s_{e_j} only shows up in class c_{e_j} and s_{v_a}, s_{v_b}, and s_{e_j} all show up together in class c_{e_j} we are not leaving any students uncovered by making this switch. Thus, the modified S is still a student covering of size k, but now fits the criteria that all students are of the form s_{v_i}. So we are in the first case and we know $I = S$ is a vertex cover of size k for graph G. □

In this reduction we create $O(|V| + |E|)$ students and $O(|E|)$ classes, each with three students. So, creating the new input to Flashmob takes polynomial time. Thus, we have given a polynomial time reduction from Vertex-Cover (which is known to be NP-hard) to Flashmob, showing that Flashmob is NP-hard. In combination with our earlier discussion showing that Flashmob is in NP, we see that Flashmob is NP-complete.

Problem 4

Your friend plans to achieve a total elevation change of *exactly* k ft every week. She plans to jog from checkpoint s to checkpoint t. She also downloaded the park map including all jogging checkpoints with their elevations, and the roads connecting the checkpoints. Since everyone in the park is jogging, all roads connecting the checkpoints are one-way. All elevations are nonnegative integers.

Show that finding a route from s to t with a total elevation change of exactly k is NP-complete.

The problem is in NP since given any path from s to t, one can follow the path and compute the total elevation realized in polynomial time. It is left to show that the problem is NP-hard. We present a reduction from the Subset-Sum problem.

Let $h(v)$ be the elevation of the vertex v. Given an instance of Subset-Sum consisting of the nonnegative integers $a_1, a_2, \ldots, a_n$ and a target T, we construct an instance of the new problem as follows: Introduce a vertex v_i with $h(v_i) = 0$ for each $i \in \{0\} \cup [n]$, and introduce a vertex u_i with $h(u_i) = a_i$ for each $i \in [n]$. Add a directed edge from v_{i-1} to v_i for each $i \in [n]$; from u_i to v_i for each $i \in [n]$; and from v_{i-1} to u_i for each $i \in [n]$. Set $s = v_0$ and $t = v_n$. Let the resulting graph be G. An example is depicted in Figure 4.

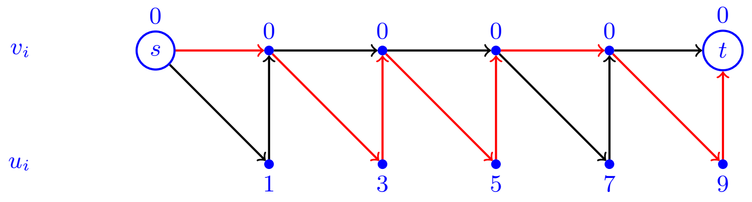

Figure 4: The constructed instance G given $(a_1, a_2, a_3, a_4, a_5) = (1, 3, 5, 7, 9)$ and $T = 17$ as an instance for Subset-Sum. The number beside each vertex denotes the elevation of that vertex. The red route achieves a total elevation change of $2T = 34$.

Claim 3. *There exists a path in G from s to t with total elevation change of exactly $2T$ if and only if there exists a subset $I \subseteq [n]$ such that $\sum_{i \in I} a_i = T$.*

Proof. By construction, all paths from s to t pass through each $i \in [n]$, and either (1) go from v_{i-1} to v_i, or (2) go from v_{i-1} to u_i, and then from u_i from v_i. Any such path P can be characterized by the subset of indices $I \subseteq [n]$ at which P chooses (2). The total elevation change H realized by P is

$$\begin{aligned} H &= \sum_{i \notin I} |h(v_{i-1}) - h(v_i)| + \sum_{i \in I} (|h(v_{i-1}) - h(u_i)| + |h(u_i) - h(v_i)|) \\ &= \sum_{i \notin I} 0 + \sum_{i \in I} (a_i + a_i) = 2 \sum_{i \in I} a_i. \end{aligned}$$

Therefore, there exists a path P achieving a total elevation change H of $2T$ if and only if there exists a subset $I \subseteq [n]$ with $H = 2\sum_{i \in I} a_i = 2T$, or equivalently $\sum_{i \in I} a_i = T$. $\square$

The graph G can be constructed in polynomial time with $n' = n + 1 + n = 2n + 1$ vertices and $m' = 3n$ edges. Since the Subset-Sum problem is NP-hard, the problem of finding a path realizing a total elevation chagne of exactly k is also NP-hard.

The problem is therefore NP-complete.

Problem 5

Let us consider the problem MAX-CUT: Given a network G and a number c, does there exist an st-cut with capacity at least c? Show that MAX-CUT is NP-complete by giving a reduction from NAE-3-SAT.

First we show that MAX-CUT is in NP. Candidate witnesses are simply a partition of the nodes into two sets S and T. We can compute the value of the cut in polynomial time by adding the capacities of all edges from all vertices $u \in S$ to $v \in T$. By comparing this value to c, we can verify a candidate witness in polynomial time.

We now show a reduction from NAE-3-SAT to MAX-CUT. Let φ be an instance for NAE-3-SAT. We will assume that φ contains no clauses with only 1 literal. Otherwise, we would simply return false since there cannot be one true and one false literal in a clause with 1 literal. Now, suppose there are m_3 clauses with 3 literals, m_2 clauses with 2 literals, and n variables in φ. Then we construct an instance for the MAX-CUT problem as follows: Introduce two vertices labeled with x and $\overline{x}$ for each variable x in φ, and add an edge between x and $\overline{x}$ with capacity 1. For each clause C, we connect the vertices corresponding to each literal in the clause to form a clique, and set the capacity of each edge to 1. Direct all added edges in both directions, and replace all repeated edges by a single directed edge with capacity set to the sum of all the repeated edge capacities. Introduce two isolated vertices s and t as the source and the sink respectively. Let the resulting graph be G and $c = n + m_2 + 2m_3$.

We now show the following claim.

Claim 4. *There exists a one-to-one and onto correspondence between: (1) assignments to the n variables of φ such that each of the $m_2 + m_3$ clauses contains at least one true and one false literal, and (2) st-cuts (S, T) in G of capacity $n + m_2 + 2m_3$.*

Proof. For any st cut (S, T), by construction, each edge connecting x and $\overline{x}$ contributes at most 1 to the capacity of cut (S, T), each clause-clique of size 3 contributes at most 2 to the capacity of cut (S, T), and each clause-clique of size 2 contributes at most 1 to the capacity of cut (S, T). Therefore the capacity of any cut (S, T) is at most

$$1 \cdot n + 2 \cdot m_3 + 1 \cdot m_2 = n + m_2 + 2m_3,$$

where the equality holds if and only if all vertices labeled by x and $\overline{x}$ are separated by the cut, and each clause-clique contains at least one vertex in S and one vertex in T.

We now show the claimed one-to-one and onto correspondence in one go. For any assignment X to φ, let S be the set of literals assigned to 1 and T the set of literals assigned to 0 under X respectively. Then by construction, X is a not-all-equal assignment if and only if each clause contains at least one literal assigned to 1 and at least one literal assigned to 0. That is in G, each clause-clique contains at least one vertex in S and at least one vertex in T, and vertices x and $\overline{x}$ are separated by the cut, in which case the capacity of the cut $(S \cup \{s\}, T \cup \{t\})$ is at least $n + m_2 + 2m_3$ by our previous argument. □

The graph G contains $2n + 2$ vertices and at most $2(3m_3 + m_2 + n)$ edges, which comes from connecting all clause-cliques of size 3 with 3 edges in each direction, all clause-cliques of size 2 with 1 edge in each direction, and each pair of variable vertices x and $\bar{x}$ with 1 edge in each direction. Adding vertices and edges to the graph can both be done in polynomimal time, and thus the graph H can be constructed in polynomial time. Since NAE-3-SAT is NP-hard, we have shown that MAX-CUT is NP-hard. Since we showed earlier that MAX-CUT is in NP, we know that MAX-CUT is NP-complete.