Homework 10

Aleck Zhao

November 30, 2017

1 Minimum Subgraph (33 points)

Consider the following decision problem, which we will call Minimum Subgraph. The input is a graph $G = (V, E)$, a subset of nodes $M \subseteq V$ known as the *marked* nodes, and an integer k. A YES instance is one in which there is a connected subgraph $H = (V_H, E_H)$ of G so that $M \subseteq V_H$ and $|V_H| \leq k$. Otherwise it is a NO instance. Less formally: can we connect all of the vertices in M using at most k vertices (including M)?

(a) Prove that Minimum Subgraph is in NP.

Proof. Here, the witness X is the subgraph $H = (V_H, E_H)$ and the verifier checks that $|V_H| \leq k$, that $M \subset V_H$, and that H is connected. Checking $|V_H| \leq k$ using a counter takes $O(n)$ time since $V_H \subset V$. Checking $M \subset V_H$ takes $O(n^2)$ by simply iterating over all $v \in M$ and iterating through V_H searching for v. Finally, we can perform a DFS in H to check for connectedness in $O(n+m)$ time. This algorithm takes $O(n^2 + m)$ time.

This algorithm is correct because if (G, M, k) is a YES instance, H is a witness that yields YES from the algorithm, due to correctness of DFS in determining connectedness. If G was a NO instance, then such a subgraph does not exist, and any witness would fail one or more of the checks. □

To prove that Minimum Subgraph is NP-hard, we will do a reduction from Vertex Cover. Suppose that we are given an instance $(G = (V, E), k)$ of Vertex Cover (so this is a YES instance if there is a vertex cover of G of size at most k, and is a NO instance otherwise). We construct a new graph $H = (V', E')$ as follows:

- $V' = V \cup E \cup \{z\}$
- $E' = \{\{v, e\} : v \in V \text{ is an endpoint of } e \in E\} \cup \{\{v, z\} : v \in V\}$

In other words, we construct H by subdividing each edge in G with a new vertex, and then connect all of the original vertices to a new *apex* vertex z.

(b) Prove that if G has a vertex cover of size at most k, then there is a connected subgraph of H which contains all vertices in $E \cup \{z\}$ and has at most $k + |E| + 1$ vertices total.

Proof. If $S \subset V$ is a vertex cover of size at most k, then $v \cap E \neq \varnothing$ for all $v \in S$ and $|S| \leq k$. Then let $V_H = S \cup E \cup \{z\}$, which has $|S| + |E| + 1 \leq k + |E| + 1$ vertices and $E_H = \{\{v, z\} : v \in S\} \cup \{\{v, e\} : v \in S \text{ is an endpoint of } e \in E\}$. The claim is that $H' = (V_H, E_H)$ is connected.

Suppose there was an unreachable vertex in H', which must be of the form $\{u, v\}$ for $u, v \in V$ since all vertices in S are connected to z. Since $\{u, v\}$ can only be connected to u or v in H', this means that neither u nor v is in S, but then S is not a vertex cover of G since $\{u, v\} \in E$ has no incident vertices. Contradiction, so H' is connected. □

(c) Prove that if G does not have a vertex cover of size at most k, then there is no connected subgraph of H which contains all vertices in $E \cup \{z\}$ and has at most $k + |E| + 1$ vertices total. Hint: prove the contrapositive.

Proof. Suppose there was a connected subgraph $H' = (V_H, E_H)$ of H containing all vertices in $E \cup \{z\}$ and has at most $k + |E| + 1$ vertices total. Then V_H contains at most k elements from V, say $S \subset V$. The claim is that S is a vertex cover of G.

Since H' is connected, each $\{u, v\} \in E$ has either $u \in S$ or $v \in SS$. If not, it would be unreachable since the only vertices connected to $\{u, v\}$ in H' are u and v. This is exactly the condition that S is a vertex cover of G. □

(d) Using the previous two parts, prove that Minimum Subgraph is NP-hard.

Proof. We perform a reduction from Vertex Cover to Minimum Subgraph. Given an instance $(G = (V, E), k)$ of Vertex Cover, take $f(G = (V, E), k) = (H, E \cup \{z\}, k + |E| + 1)$ where H is constructed as before.

From part (b), if G has a vertex cover of size at most k, then there is a connected subgraph of H containing all of $M = E \cup \{z\}$ and having at most $k + |E| + 1$ vertices, so $(H, E \cup \{z\}, k + |E| + 1)$ is a YES instance of Minimum Subgraph. From part (c), if G does not have a vertex cover of size at most k, then H does not have the desired connected subgraph. Thus, YES and NO instances are mapped to YES and NO instances, respectively.

Finally, by part (a), we can compute f in polynomial time. Since Vertex Cover is NP-hard, it follows that Minimum Subgraph is also NP-hard. □

2 Graduation Requirements Revisited (34 points)

John Hopskins has switched to a more lenient policy for graduation requirements than it had in Homework 9. As in the previous homework, there is a list of requirements $r_1, r_2, \ldots, r_m$ where each requirement r_i is of the form "you must take at least k_i courses from set S_i". However, under the new policy a student *may* use the same course to fulfill multiple requirements. For example, if there was a requirement that a student must take at least one course from $\{A, B, C\}$, and another required at least one course from $\{C, D, E\}$, and a third required at least one course from $\{A, F, G\}$, then a student would only have to take A and C to graduate.

Now consider an incoming freshman interested in finding the *minimum* number of courses required to graduate. You will prove that the problem faced by this freshman is NP-complete, even if each k_i is equal to 1. More formally, consider the following decision problem: given n items (say $a_1, \ldots a_n$), given m subsets of these items $S_1, S_2, \ldots, S_m$, and given an integer k, does there exist a set S of at most k items such that $|S \cap S_i| \geq 1$ for all $i \in \{1, \ldots, m\}$.

(a) Prove that this problem is in NP.

Proof. Here, the witness X is the set S and the verifier checks that $|S| \leq k$ and $|S \cap S_i| \geq 1$ for all $i = 1, 2, \cdots, m$. Checking $|S| \leq k$ using a counter takes $O(n)$ since $|S| \leq n$. Checking $|S \cap S_i| \geq 1$ takes $O(|S|\,|S_i|) = O(n^2)$ by iterating over all elements of S and checking if S_i contains it, because $|S_i| \leq n$. Then doing this for each i takes a total of $O(mn^2)$ time, so the total time for this algorithm is $O(mn^2)$, which is polynomial time.

This algorithm is correct because if $(\{a_1, \cdots, a_n\}, \{S_1, \cdots, S_m\}, k)$ is a YES instance, S is a witness that yields YES from the algorithm. If it is a NO instance, then such an S does not exist, and any witness will fail one or more of the checks. □

(b) Prove that this problem is NP-hard.

Proof. We perform a reduction from VERTEX COVER. Given an instance $(G = (V, E), k)$ of VERTEX COVER, construct an instance of this problem as follows: Let $\{a_1, \cdots, a_n\} = V$. Label the edges $S_1, \cdots, S_m$. Clearly $S_i \subset \{a_1, \cdots, a_n\}$ since S_i is a subset containing 2 vertices.

If $(G = (V, E), k)$ is a YES instance of VERTEX COVER, then there exists a vertex cover with size at most k. The claim is that this vertex cover is also the set S that satisfies this problem, that is, $|S \cap S_i| \geq 1$ for each i. Suppose not, that for some k the intersection is empty: $|S \cap S_k| = \varnothing$. Then if $S_k = \{u, v\}$, in particular S does not contain either u or v, which means that edge S_k has no incident vertices, so S is not a vertex cover. Contradiction, so since $|S| \leq k$, this vertex cover S is also a witness for this problem.

If $(G = (V, E), k)$ is a NO instance of VERTEX COVER, then there does not exist a vertex cover with size at most k. The claim is that there is no subset $S \subset \{a_1, \cdots, a_n\}$ with $|S| \leq k$ such that $|S \cap S_i| \geq 1$ for all i. Suppose there is such a subset S. Then this same subset of vertices in the graph is a vertex cover of size at most k since S intersects each edge S_i, contradiction.

Since VERTEX COVER is NP-hard, it follows that this problem is NP-hard. □

3 Magic Subroutines (33 points)

(a) Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary graph G, the number of vertices in the largest clique in G. Describe a polynomial-time algorithm that computes, given an arbitrary graph G, a clique of G of maximum size, using this magic black box as a subroutine. Prove polynomial running time and correctness.

Solution. Consider the following algorithm:

(1) Compute the size of the largest clique in G. Suppose it is k.

(2) Pick an arbitrary vertex v, and compute the size of the largest clique in $G - v$.

  (a) If this returns k, then v was not part of the largest clique. Remove v from G.

  (b) If this returns $k - 1$, then v was part of the largest clique.

(3) Repeat the process with each remaining vertex.

(4) Return G.

Running time: Step (2) takes polynomial time. The process of computing largest clique size at each step takes polynomial time, with a total of n iterations, one for each vertex, so the total running time is still polynomial in n and m.

Correctness: If there is a single clique of size k, then during the algorithm the black box will return $k - 1$ on $G - v$ if and only if v is in this clique of size k. To see this, if v is in the clique, then removing it will reduce the size of the largest clique by 1, and if the largest clique in $G - v$ has size $k - 1$, then v must have been part of that max clique. Thus, these vertices will all be added to C, which is the largest clique.

If there are multiple cliques of size k, then during the algorithm the black box will return k on $G - v$ until there is only one remaining clique of size k because at each step we remove a vertex from G. Now, this is the same scenario as when there was one clique of size k.

At termination, the remaining vertices must be the largest clique. If not, then we could remove another and not change the size of the largest clique, since that largest clique would still be intact. Vertices were only removed from G when they were not part of the initial largest clique. □

(b) Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary boolean circuit Φ (with one output and no loops, like in CIRCUIT-SAT), whether Φ is satisfiable. Describe a polynomial-time algorithm that either computes a satisfying input for a given boolean circuit or correctly reports that no such input exists, using the magic black box as a subroutine. Prove polynomial running time and correctness.

Solution. Consider the following algorithm:

(1) Check if Φ is satisfiable. If not, return NO.

(2) Pick an input x_0 and set it to 0.

    (a) Check for satisfiability. If the circuit is not satisfiable, then set x_0 to 1.

(3) Continue the process with every input variable.

(4) Return the input.

Running time: Step (1) takes polynomial time. The process of checking for satisfiability at each input also takes polynomial time, and there are n inputs to check total, so the total running time is still polynomial in n.

Correctness: At step (1), if the circuit is not satisfiable, we return NO immediately. Otherwise, it must be satisfiable, so hard coding either $x_0 = 0$ or $x_0 = 1$, the circuit must still be satisfiable. As a base case, if there is a single input value, then the algorithm will determine if 0 satisfies the circuit, or if not, then 1 must satisfy the circuit.

If Φ is satisfiable with $x_0 = 0$, then we have reduced the problem to finding a satisfying input on the remaining $n-1$ inputs, so by induction the algorithm is correct. If it is not satisfiable with $x_0 = 0$, then Φ must be satisfiable with $x_0 = 1$, and we continue with input of size $n-1$. □