Problem Set #10

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **You do not need to turn these problems in. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Deleting Edges

Consider the following problem. You are given a flow network with unit capacity edges: It consists of a directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c_e = 1$ for all $e \in E$. You are also given a parameter k.

The goal is to delete k edges so as to reduce the maximum s-t flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum s-t flow in $G' = (V, E - F)$ is as small as possible subject to this.

Give a polynomial time algorithm to solve this problem. Argue (prove) that your algorithm does in fact find the graph with the smallest maximum flow.

Solution

Run the Ford-Fulkerson capacity scaled algorithm to find the max flow. Perform a graph search (BFS/DFS) from the source to identify the min cut. Find k edges that cross the cut and delete them. You have reduced the flow in the original graph by k. If k such edges do not exist, just delete all of the edges that cross the cut. You have reduced the flow in the original graph to 0.

Problem 2: Flow Networks and Sensor Failures

An ad hoc sensor network network is made up of low-cost, low-power devices that are distributed in some physical space. One example of these networks in the literature involves tossing a bunch of these sensors out of an airplane over a forest. The sensors are then randomly distributed and expected to monitor for conditions that are indicative of a forest fire.

The sensor nodes, however, are prone to failure for a variety of reasons. A given sensor s can detect that it is about to fail (in practice, this may not be possible, but let's assume it's true). When s determines it is failing, it needs to send a representation of its current state to some other nearby sensor that can take over for it. Each device has a limited transmission range, d; i.e., a node can communicate with exactly the set of nodes within d meters. (Notice that this relationship is symmetric: if sensor s_1 can communicate with s_2, the reverse is also true.) Because some set of the sensor's neighbors may have already failed, we actually want to make sure that we send the backup copy to some live node; therefore when a sensor s is failing, it should send its state to k other sensors. These k sensors constitute the *back up set* for s.

You're given a set of n sensors with known positions represented by an (x, y) coordinate for each sensor. You must design an algorithm that determines whether it is possible to choose a back up set for each of the n devices (i.e., for each device, find k other sensors with d meters), with the added constraint that a single sensor can serve as a back up for at most b other sensors. You must also output the back up sets for each sensor, given that a solution exists.

To be completely clear, ensure you complete the following steps:

- Formulate the problem as a network flow problem. This means define the vertices, edges, and capacities of the network flow graph *and* relate your rationale for the structure of the network flow graph to the original problem.
- Be sure to represent k and b in your network flow graph.
- Given a max-flow or a min-cut on your network flow graph, describe how you can determine whether it is possible to choose a back up set for each sensor.
- Given a max-flow or a min-cut on your network flow graph, describe how to state the back up sets, given that a feasible solution exists.

Solution

For the flow network G, there is a single source vertex s and a single sink vertex t. For each device i, we have two nodes u_i and v_i. There is an edge (s, u_i) from the source to every u vertex with capacity k and an edge (v_i, t) from every v vertex to the sink with capacity b. If devices i and j ($i \neq j$) are within d meters of each other, we have two edges (u_i, v_j) and (u_j, v_i), all of capacity 1. These edges connecting a vertex to s signify the backup requirement of each node, and the edges connecting a vertex to t significy the backup capability ot each node. The pair of edges connecting the u and v vertices for i and j represent each's ability to back up the other. That is, for each sensor i, it has two roles: a sensor that needs back up support (signified by u_i) and a sensor that can provide back up support (signified by v_i). The capacity of (v_i, t) gives the maximum ability of sensor i to provide back up support.

We compute the max flow f of G. If all of the edges leaving the source are saturated (i.e., they all have k capacity), then it is possible to make the back up system. In such a max flow, if $f_{u_i,v_j} = 1$ then device j is one of the backup sensors for device i.

Problem 3: Zero-Weight Cycle

You are given a directed graph $G = (V, E)$ with weights w_e on its edges $e \in E$. The weights can be negative or positive. The *Zero-Weight Cycle* Problem is to decide if there is a simple cycle in G so that that sum of the edge weights on this cycle is exactly 0. Prove that *Zero-Weight Cycle* is NP-Complete by reducing from the subset sum problem.

The Subset Sum Problem. *Given natural numbers $w_1, w_2, \ldots, w_n$ and a target number W, is there a subset of $\{w_1, w_2, \ldots, w_n\}$ that adds up to precisely W?*

Solution

Zero-Weight-Cycle is in NP because we can exhibit a cycle in G and it can be checked that the sum of the edge weights on this cycle are equal to 0.

We now show that *Subset-Sum* $\leq_p$ *Zero-Weight-Cycle*. We are given numbers $w_1, \ldots, w_n$ and we want to know if there is a subset that adds up to exactly W. We construct an instance of *Zero-Weight-Cycle* in which the graph has nodes $0, 1, 2, \ldots, n$ and an edge (i, j) (a directed edge from i to j) for all pairs $i < j$. The weight of an edge (i, j) is equal to w_j. Finally, there is an edge $(j, 0)$ of weight $-W$.

We claim that there is a subset that adds up to exactly W if and only iff G has a zero-weight-cycle. If there is such a subset S, then we define a cycle that starts at 0, goes through the nodes whose indices are in S, and then returns to 0 on an edge $(j, 0)$. The weight of $-W$ on the edges $(j, 0)$ precisely cancels the sum of the other edge weights. Conversely, all cycles in G must use an edge $(j, 0)$, and so if there is a zero-weight-cycle, then the other edges must exactly cancel $-W$—in other words, their indices must give a set that adds up to exactly W.

Problem 4: Efficient Recruiting

Suppose you're helping to organize a summer sports camp, and the following problem comes up. The camp is supposed to have at least one counselor who is skilled at each of the n sports covered by the camp (baseball, volleyball, etc.). They have received job applications from m potential counselors. For each of the n sports, there is some subset of the m applicants qualified in that sport. The question is: For a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified in each of the n sports? We'll call this the *Efficient Recruiting* Problem. Show that *Efficient Recruiting* is NP-Complete.

The Vertex Cover Problem. *Given a graph G and a number k, does G contain a vertex cover of size at most k? (Recall that a vertex cover $V' \subseteq V$ is a set of vertices such that every edge $e \in E$ has at least one of its endpoints in V'.)*

Solution

Efficient Recruiting is in NP, since given a set of k counselors, we can check that they cover all of the sports.

Suppose we had an algorithm A that solves *Efficient Recruiting*; here is how we would solve an instance of *Vertex Cover*. Given a graph $G = (V, E)$ and an integer k, we would define a sport S_e for each edge e and a counselor C_v for each vertex v. C_v is qualified in sport S_e if and only if e has an endpoint equal to v.

Now if there are k counselors that, together, are qualified in all sports, the corresponding vertices in G have the property that each edge has an end in at least one of them; so they define a vertex cover of size k. Conversely, if there is a vertex cover of size k, then this set of counselors has the property that each sport is contained in the list of qualifications of at least one of them.

Thus, G has a vertex cover of size at most k if and only if the instance of *Efficient Recruiting* that we create can be solved with at most k counselors. Moreover, the instance of *Efficient Recruiting* has size polynomial in the size of G. Thus if we could determine the answer to the *Efficient Recruiting* instance in polynomial time, we could also solve the instance of *Vertex Cover* in polynomial time.

Problem 5: Integer Knapsack Problem
Prove that the integer knapsack problem is NP-complete, by a reduction from the subset-sum problem, defined in Problem 3.

Solution

Knapsack problem
Instance: Non-negative weights $a_1, a_2, ..., a_n, b$, and profits $c_1, c_2, ..., c_n, k$. Question: Is there a subset of weights with total weight at most b, such that the corresponding profit is at least k?

Subset Sum problem
Instance: Non-negative integer numbers $s_1, s_2, ..., s_n$ and t. Question: Is there a subset of these numbers with a total sum t? The first step is to prove Knapsack is in the NP class.

Given an input set, it is very easy to check if the total weight is at most b and if the corresponding profit is at least k. It takes only linear time to add the weights and profits of all the goods to find the true/false result.

The second is to prove a certain problem, which is already known to be NP-Complete, can be reduced to Knapsack problem in polynomial time. We can choose any of the NP-Complete problem we have learned. Because we already know all the problems in the NP class can be reduced to the chosen problem, say Subset Sum, we know all these problems can also be reduced to Knapsack problem. It is very easy to reduce an instance of Subset Sum problem to an instance of Knapsack problem. We just create such a Knapsack problem that

$$a_i = c_i = s_i$$
$$b = k = t$$

The Yes/No answer to the new problem corresponds to the same answer to the original problem. Now prove: The following deduction implies the new problem is equivalent to the original problem:

$$\begin{matrix} \sum_{i\epsilon S} a_i \leq b \iff \sum_{i\epsilon S} s_i \leq t \\ \sum_{i\epsilon S} c_i \geq k \iff \sum_{i\epsilon S} s_i \geq t \end{matrix} \iff \sum_{i\epsilon S} s_i = t$$

Suppose we have a Yes answer to the new problem, it means we can find such a subset $S \subseteq [1, 2, ..., n]$ that satisfies the left part of the deduction. Then this subset S is also a solution to the right part. So we must also have a Yes answer to the original problem. Conversely, suppose we have a No answer, it means there is no subset S that satisties the left part. So, of course, the answer to the original problem must also be No.