CIS 320

Homework Assignment 5 [100 pts]

Solutions

Please see Piazza and Canvas for submission logistics and LaTeXtemplate.

1. *Suppose you have a flow network G with integer capacities, and an integer maximum flow f . Suppose that, for some edge e, we increase the capacity of e by one. Describe an $O(m)$ time algorithm to find a maximum flow in the modified graph.*

   Solution: Construct the residual graph using the given maximum flow f. That is, subtract f from the capacities of the respective edges in G and add reverse edges as appropriate. Now, increase the capacity of e by one and use BFS to check if there exists a path from the source to target. If so, then the new maximum flow is one higher than the old maximum flow. Otherwise, it is the same.

2. *You are outfitting a spacecraft to perform experiments on Mars and have available n pieces of scientific equipment $s_1, s_2, \ldots, s_n$. For each i, the cost of carrying equipment s_i is c_i and you can choose to take or not take each of the s_i. On Mars, you would like to perform experiments. The j^{th} experiment confers a utility of u_j but in order for this experiment to be performed we must carry a subset S_j of equipment . The goal is to maximize the sum of the utilities of the experiments we are able to perform minus the sum of the costs of the equipment we carry. Note that a piece of equipment can be used for multiple experiments. In other words, it does not get 'used up'. Design a flow network where the min cut provides a solution to this problem and prove your result.*

   Solution: Construct the following flow network: For each equipment i create node $s_i \in L$, and for each experiment j create node $e_j \in R$. Create source node s directed to each equipment node s_i with capacity equal to c_i, the cost of carrying s_i. Create a sink node t and have directed edges from each experiment node e_j with capacity equal to u_j, the utility of experiment j. Direct s_j to e_i if running experiment i requires equipment j, with capacity equal to infinity.

   Find the max flow f in the network and the corresponding min-cut (S, T). Suppose we carry the pieces of equipment corresponding to nodes on the T side of the cut, and perform experiments corresponding to nodes also on the T-side of the cut. Then the min-cut capacity ($= |f|$) is the total cost of all the equipment we would bring and the total utility of the set of experiments we would not run. Let $U = \sum_j e_j$ be the total utility of all the experiments. Therefore the utility of this trip is $U - |f| = U - min - cut - cap$.

   For any valid choice of equipment we want to take and experiments we do not want to perform, consider the cut (S, T) which puts precisely these equipment and experiment nodes on the T-side. The utility of this choice of equipment and experiments is exactly $U - cap(S,T)$. Since the capacity of any (s,t)-cut is at least the capacity of the min cut, this choice cannot be better than the choice in the previous paragraph based on the min cut.

3. *You are given an undirected graph $G(V,E)$, with three vertices u, v, and w. Design an $O(m+n)$ time algorithm to determine whether or not G contains a simple path from u to w that passes through v. You do not need to return the path and do not need to give a full proof of the solution.* ***Hint:*** *Modify the max flow algorithm to handle capacity constraints at vertices and convert the given question into a question about max flows with vertex and edge capacity constraints. (Feel free to come up with your own solution if you do not want to follow the hint.)*

   Solution: Form a directed graph $G^{'}(V^{'},E^{'})$ by letting $V^{'} = V \cup s$ and letting $E^{'}$ be a bidirection of all edges in E (i.e. $\forall (u,v) \in E\ (u,v) \in E^{'}$ and $(v,u) \in E^{'}$). Now, we will construct a flow network on $G^{'}$. We set the target $t = v$. Assign a capacity to the vertices; in particular, we assign a capacity of 1 to every vertex except s and t. (If we have a vertex v with a capacity on it, say $c(v)$, we can convert it into a standard edge-capacity constraint as follows: We replace v by 2 nodes v_{in} and v_{out} and add an edge from v_{in} to v_{out} with capacity $c(v)$. For each incoming edge (u,v) into v in the original graph, we replace it with an edge (u, v_{in}) of the same capacity, and for each outgoing edge (v,u) in the original graph, we replace it with the edge (v_{out}, u). This is the standard procedure for converting vertex capacities to more standard edge capacities.)

   Finally, we add one capacity-one edge from s to u and one from s to w; all other edges have infinite capacity. Find the max flow using Ford-Fulkerson, output such a path exists if and only if the max flow is 2. We could formally prove this algorithm correct by using path decomposition of max flow. In particular, our flow of 2 must be decomposed into two paths, one passing through u and one passing through w. Given the vertex capacities are 1, the paths are vertex disjoint, and thus we can find a simple path from u to v and a vertex disjoint simple path from w to v. Together, these form a simple path passing through v in the original graph. Since the max flow is 2, Ford Fulkerson runs in linear time.

4. *We know that the subset sum problem is NP-Complete. Remember that the subset sum problem begins with an array of positive integers A and a targeted sum k. It then asks whether there exists a subset of A that sums up to this target value k. We would like to show that the following problem called Zero sum is also NP-Complete. Given a set of integers (of course, not necessarily all positive) is there a non-empty subset whose sum is zero. Show that the zero sum problem is in NP-Complete.*

   Solution: **NP**: The certificate for an instance of the Zero Sum problem is the non-empty subset of values. To verify if this is a solution, simply sum the values and check that it is zero.

   NP-Hard: We can show Zero Sum is NP-Hard by reducing Subset Sum to it. An instance of Subset Sum is an array of positive integers A and the target sum k. To reduce this to zero sum, simply append the value $-k$ to the array A and give this value as input into Zero Sum. The Subset Sum answer is YES iff the Zero Sum answer is YES.

   Correctness: We first show that if Zero Sum returns YES, then the answer to the Subset Sum problem is indeed YES. Note that if Zero Sum returns YES, then the solution subset of integers must include the value $-k$ as all other integers in the array are strictly positive, and we know that the subset is nonempty. Thus, in order for the sum of the solution to be zero, the remaining positive integers in the solution must add to k. Thus, this is the solution to the Subset Sum problem, and the answer should be YES.

Now we show that if the answer to the Subset Sum problem is YES, then the Zero Sum problem it reduces to will return YES. Since Subset Sum should return YES, then there exists a set of the positive integers A that add to k. Then, in the Zero Sum problem it reduces to, we can add those integers together along with the added $-k$ value to yield zero, so the answer to Zero Sum must also be YES.

Runtime: The reduction takes only constant time to add the value $-k$ to the array of integers.

5. *A software company is having a holiday party The CEO would like to invite some of the company's n employees to the party. The company has m projects and each project has a team of employees working on it. (One employee could be part of multiple teams.)*

   For $i = 1, 2, \ldots, m$, let S_i be the set of people working on project i.

   The CEO doesn't want to invite all members of any team to the party, since she knows that if she does that the members of the group will just sit and talk about their project. Note that it is totally fine to invite even all but one member of a group... just not all of them. The CEO wants to know if she can invite at least k people to the party.

   Prove that this problem is NP-complete.

   If you are having a tough time with the notation, here is an example instance of this problem:

   A, B, C, D, E and F are the employees.

   A, B, C are working on a project

   D, A, B are working on a project

   D, E are working on a project

   A, F, C are working on a project

   Can the CEO invite 4 people?

   Yes - A, B, E and F is a safe set of people to invite since no subset of them represents the complete set of people doing a project.

Solution: **NP**: The certificate for the problem is the set of people the CEO wants to invite. To verify, simply check that there is no project in which every member is invited.

NP-Hard: We can reduce from INDEPENDENT SET (IS). The input to IS is a graph G and a parameter k, and the question asks if there is an independent set of size $\geq k$. The reduction is as follows: the employees of the companies correspond to the vertices of the graph. For each edge in the graph (u, v), add a project containing employee u and employee v. The parameter k for this problem is the same as for IS. Now, IS returns YES iff the party problem returns YES.

Correctness: First consider if there exists an independent set of size $\geq k$. Consider its vertices, and let the CEO invite the employees corresponding to these vertices. We know there is no project in which all members are invited because all projects contain only two people, corresponding to edges of G, and by the constraints of an independent set, there is no edge where both vertices are chosen. Thus, these people correspond a valid set to invite.

Now consider if there exists a valid set of at least k people to invite. Then we can choose the vertices corresponding to these people as our independent set. We need to ensure that there is no edge in G for which both vertices are selected. Since edges correspond to projects, this is equivalent to there being no project in which both employees, or vertices, are invited, which is exactly the constraint for the party problem.

6. *We have seen how the independent set (decision) problem is NP complete. Suppose we are given a subroutine that will solve this problem. Show how you can use the subroutine to find a maximum independent set in a graph, making only a polynomial number of calls to the subroutine.*

Solution: We can first find the true size K of the independent set by asking if the graph contains an independent set of size k, iterating from $k = 1$ to n. Once this value is found, iterate through the vertices of the graph. For each, temporarily remove the vertex from the graph and use the subroutine to check if there still exists an independent set of size K. If so, then the vertex chosen is not necessary for our solution and can be permanently deleted. If not, then we know the vertex must be part of our maximum independent set. As a result, we should place it back into the graph and instead remove all of its neighbors from the graph. This process takes at most n iterations. Once complete, we are left with the maximum independent set. The total number of calls to the subroutine is at most n to find the true value K and at most n to iterate through the vertices.