CS5800 Algorithms *Out: November 1, 2019*

Solutions for Problem Set 6

Name: Instructor's solutions *Due: November 12, 2019*

Understanding Flows and Cuts

Problem 1 (Messing with capacities) 10

Let G be a network flow graph with source s and sink t, and let P be any directed path in G from s to t. Suppose we increase the capacity of every edge in P by one unit. Indicate which one of the following three statements is true. Justify your answer.

1. It is always the case that (i.e., for every choice of G and P) the maximum flow from s to t increases by one.
2. It may be the case that (i.e., there exist G and P such that) the maximum flow from s to t does not increase.
3. It may be the case that (i.e., there exist G and P such that) the maximum flow from s to t increases by more than one unit.

Solution:

The 3rd statement is true. The maximum flow may increase by more than 1.

Consider the network formed by six vertices $\{s, a, b, c, d, t\}$ and edges $s \to a$, $a \to b$, $b \to t$, $s \to c$, $c \to d$, $d \to t$, and $b \to c$. Suppose all the edges coming out of s and going into t have capacity 2, and the other edges have capacity 1. The min-cut has $\{s, a, c\}$ on one side and $\{b, d, t\}$ on the other, with capacity 2. So max-flow is 2.

Let P be the path $s \to a \to b \to c \to d \to t$. Note that this crosses the min-cut twice. If we increase the capacity of every edge on this path by 1, then there is a max-flow of value 4.

■

Problem 2 (Updating Maximum Flow) 10

You are given a graph $G = (V, E)$ on which you have already computed the maximum flow f (and have a residual graph G_f). Suppose that now the capacity of a single edge $(u, v) \in E$ is increased by 1. Prove that you can update your maximum flow by running just a single iteration of Ford-Fulkerson instead of running the whole algorithm from scratch. A single iteration of Ford-Fulkerson includes searching for an augmenting path and pushing more flow through it if such a path is found.

Solution:

Suppose that the edge from u to v increases its capacity by one. From the previous maximum flow F, just make a new iteration of Ford-Fulkerson algorithm with the modified graph: The residual flow graph increases the capacity of edge (u, v) with one. Make a graph search (in time $O(|V| + |E|)$) to see if there is any path in the residual flow graph along which the flow can increase. If there is one, there must be a flow of size one (because all flows are integers). In this case, value of maximum flow

of new graph is increased by one. If there is no flow in the residual flow graph, flow F is still the maximum flow. ■

Problem 3 (Critical Edge) 15

An edge of a flow network G with source s and sink t is called critical if decreasing the capacity of this edge results in a decrease in the maximum flow from s to t in G. Prove that every edge in a minimum $s-t$ cut in G is critical.

Solution:

In the minimum cut, the flow of all the edges is equal to the respective capacities of the edges. Hence if we decrease the capacity of any of these edges then the flow along that edge must necessarily decrease. This would mean that the overall flow must also decrease. The only other possibility is that the flow decrease on this edge gets compensated for along a different $s-t$ path. However, note that all the other edges along this minimum $s-t$ cut are already saturated and cannot take any more flow. So the flow decrease would not be compensated for and the overall flow decreases due to decreasing the capacity of the edge. Hence if an edge belongs to the minimum $s-t$ cut then it is critical. ■

Using Flows and Cuts

Problem 4 (Edge connectivity) 10

The edge connectivity of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how to determine the edge connectivity of an undirected graph $G = (V, E)$ using a maximum-flow minimum-cut algorithm.

Solution:

Let s be an arbitrary vertex of V. Construct the flow network N in the following way: For each edge of the graph G, consider two directed edges of capacity one in both directions. For each vertex $v \in V \setminus \{s\}$, solve the maximum flow/minimum cut problem on network (N, s, v), let c_v: be the value of that. Take the minimum over all c_v.

The algorithm uses $|V|-1$ minimum cut computations in networks, each of which can be solved by a maximum flow computation. Since each network can have a maximum flow of cost at most $|V|-1$ and all capacities are integers, the Ford-Fulkerson algorithm finds each maximum flow in time $O(|E||V|)$ and so the overall running time is $O(|E||V|^2)$.

We claim that the edge connectivity equals to $c^* = min_{v \in V \setminus \{s\}} c_v$. Suppose k is the edge connectivity of the graph and R is the set of edges such that removal of R disconnects the graph and creates cut S and $V - S$. Without loss of generality assume the node $s \in S$. Let u be an arbitrary node in $V - S$. Since $s \neq u$ the value c_u of will be computed by the algorithm. By the max-flow min-cut theorem, c_u equals the min cut size between the pair s and u, which is at most k. Since R disconnects s and u, we have $c^* \leq c_u \leq k$. But c^* cannot be strictly smaller than k since that would imply a cut set of size smaller than k, contradicting the fact that k is the edge connectivity. Therefore $c^* = k$ and the algorithm returns the edge connectivity of the graph correctly. ■

Problem 5 (Are you a bottleneck?) 15

Let $G = (V, E)$ be a flow network with source s and sink t. We say that an edge e is a bottleneck if it crosses every minimum-capacity cut separating s from t. Give an efficient algorithm to determine if a given edge e is a bottleneck in G.

Solution:

Since a bottleneck edge belongs to every min-cut, if we increase the capacity of the edge by 1, every current min-cut will increase by 1. The value of other cuts (which do not include e) remain the same, but the capacity of these cuts already exceeded the current value of the maximum flow. Hence, the minimum-cut of the network (with the capacity of e increased by 1) increases by 1. Hence the max-flow will increase by 1. So the algorithm is to increase the capacity of e by 1, and recompute the maximum flow. If the max-flow increases by 1, then e is a bottleneck edge, otherwise it is not. For recomputing the maximum flow, one can simply use any efficient maximum flow algorithm. While this would be polynomial-time, one can do much better. In linear time, one can update the current maximum flow. See problem . ■

Problem 6 (Orwellian viruses) 20

One of the computers in a lab in the university has been infected with an Orwellian virus which makes the computer believe that $2 + 2 = 5$. We need to stop the spread of this virus before the very foundations of mathematics are destroyed! All the other computers are doing *very important work* and they are connected (in a known configuration) to each other via LAN cables so that they can help each other. Thankfully lab is isolated and there is just one computer (named Shangri-La) which is actually connected to the internet. We need to stop the virus before it reaches Shangri-La.

We have two methods we can use while trying to stop the virus. The first is to shut down computers other than the infected one and Shangri-La. But we'll need to save the *very important work* they're doing. The other method is to physically cut the LAN cables. But only some of the cables are actually visible, the rest are all hidden inside walls. Both saving the work on a computer and cutting a LAN cable take the same time and we don't care how many of each method is choosen. We want to finish disconnecting Shangri-La from the infected computer as quickly as possible.

We come to you for help. We'll tell you which computers are connected to which, and we'll tell you which LAN cables are visible and can be cut. As someone who has mastered algorithms, help us save the foundations of mathematics. That is, come up with an efficient algorithm such that the sum of the number of computers shutdown and LAN cables cut is minimized. We'll need proof that the algorithm works and that it works fast so make sure to provide proof of correctness and analyze the running time.

Solution:

We create a graph with the computers as nodes and the LAN cables as edges in the graph. Consider the infected computer to be the source s and Shangri-La to be the sink t. Since LAN cables do not inherently have a direction, we have an undirected graph. We will transform this into a directed graph by making every edge into a pair of anti-parallel edges: the undirected edge $\{u, v\}$ becomes two directed edges (u, v) and (v, u).

For every node, we use a capacity of 1, this can be changed into capacities of edges by transforming every node u into two nodes u_{in}, u_{out}. Every edge coming into u now comes into u_{in}, every edge going out of u now goes out of u_{out} and we make an edge (u_{in}, u_{out}) with capacity equal to what the

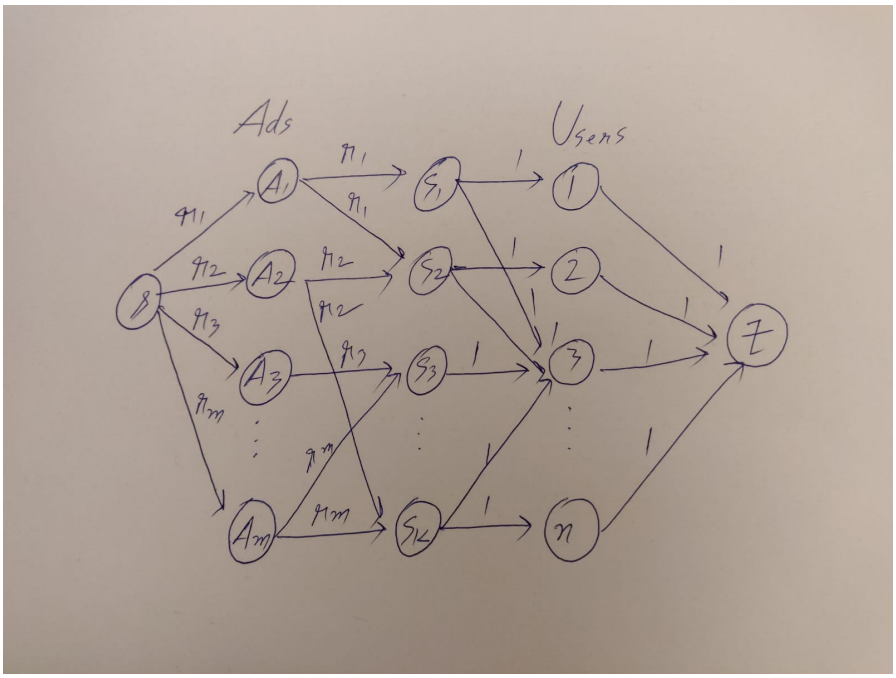
capacity of the node u was, 1 in this case. Every edge that corresponds to a visible LAN cable (that can be cut) also gets a capacity of 1. Every other edge gets a capacity of ∞.

Now we run a max-flow algorithm and find the edges that cross the min $s-t$ cut. By construction, this will not contain any edges corresponding to the invisible LAN cables. This is because we know that the min-cut value is at most n where n is the number of computers not counting the infected computer and Shangri-La (we could always shut down every computer). Hence, we know that no ∞ capacity edge will be used in the min-cut. So only edges corresponding to computer nodes, (u_{in}, u_{out}), and visible LAN cable edges will be a part of the min-cut. Since it is a min-cut, this is the smallest number of edges that will disconnect the infected computer and Shangri-La.

Thus, finding the edges in the min-cut will give us the smallest set of computer to shutdown and LAN cables to cut that will prevent the virus from reaching Shangri-La.

Finding the Min-cut by running Edmonds-Karp takes $O(|V||E|^2)$ time. Here $|V|$ is equal to the number of computers and $|E|$ is the total number of LAN cables. ■

Problem 7 (Advertising) 20



Major online portals like Google and Facebook have considerable information about the individual users based on their past interactions. This allows them to post targeted advertisements to the users. Suppose a set U of n users, labeled 1 through n, visit the portal on a particular day. The portal has a set A of m ads, labeled 1 through m, to choose from. The analysis of the users has revealed k different groups (from a marketing standpoint), the ith group consisting of subset S_i of users from U. A user may be part of several groups; i.e., a user may be an element of several different S_i's. The jth ad is targeted to a subset $G_j \subseteq \{1, \ldots, k\}$ of the groups.

The portal needs to decide whether there exists a way of assigning advertisements to users such that the following conditions hold: (a) each user is shown exactly one ad; (b) ad j is shown to user i only if i is in a group k in G_j; (c) the number of times the ad j is shown is exactly r_j, where r_j is a given integer.

Give a polynomial-time algorithm that takes the above input: U, A, the sets S_i's, the groups G_j's, the r_j's — and determines whether the portal can assign an ad to each user so that the above

three conditions are satisfied, and if so, then returns such an assignment. State the running time of your algorithm.

Solution:

We can solve it by converting this question into a flow diagram and then applying the Ford Fulkerson Algorithm. We can make a few nodes and connect flow to them accordingly. First we take the source node. Connect them to m ads nodes with the capacity of r_j for each respective ad. Then connect the ads to the respective group nodes to which they can be shown. Keep the capacity of all edges equal to the r_j value of the ads they are outgoing from. Then connect the groups to the users contained in them with a capacity of 1 for each edge. Finally connect all the users to the sink with the capacity 1.

Now we just run Ford-Fulkerson on this network flow graph. A assignment is possible iff the max flow is $\sum r_j = n$. Clearly, if the flow is $< \sum r_j$ then there is some ad which cannot be shown exactly r_j times. If $sumr_j \neq n$ then the constraint of each user seeing exactly 1 ad is violated. We only need to show that we can extract a valid assignment if the max flow is $\sum r_j = n$.

This is done by using the edges between ads and groups and between groups and users that have been saturated. If the flow equals capacity then we show those ads to corresponding users. This assignment will satisfy condition (b) by construction of the edges. Due to the max flow being $\sum r_j$ this assignment will also satisfy condition (c). Since $\sum r_j = n$ condition (a) will also be satisfied by the assignment.

If we run Ford-Fulkerson on this, it will take time $O(|E|f)$ where $|E|$ is the number of edges and f is the max-flow. Note that in the constructed graph the number of nodes is $O(nmk)$ and the number of edges is $O(mk + kn)$ (this happens when every ad can be shown to every group and every group contains every user) and the final max-flow is $O(n)$. So the total running time will be $O(n(mk + kn))$. ■