Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
David Karger

Handout 9
Wednesday, October 3, 2012

Problem Set 5

Due: Wednesday, October 10, 2012.

Collaboration policy: collaboration is *strongly encouraged*. However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator.
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 or 4 people in a given week.
4. **No bibles. This includes solutions posted to problems in previous years.**

Problem 1. **(a)** Not every network has an upward-critical edge. Consider the network $G = (\{s, v, t\}, \{(s, v), (v, t)\})$, with the capacity of (s, v) and (v, t) equal to 1. Increasing the capacity of either edge does not increase the maximum flow.

To find all the upward-critical edges in a network, consider the following algorithm:

- Compute a max flow f in the network
- Let U be the set of all vertices reachable from s in the residual graph G_f. (We can find such vertices using DFS)
- Let V be the set of all vertices that can reach t in G_f
- Any edge (u, v) such that $u \in U$ and $v \in V$ is an upward-critical edge.

This algorithm is correct because increasing the capacity of any edge (u, v), s.t. $u \in U, v \in V$ results in an augmenting path, so it is upward-critical. Furthermore, every upward-critical edge has this property, so the set of edges found by the algorithm is equal to the set of upward-critical edges. The running time of the algorithm is dominated by the one calculation of the maximum flow.

(b) The set of downward-critical edges is not the same as the set of upward-critical edges. In the example in part (a), each of the edges is downward-critical, whereas none are upward-critical.

To compute the set of downward-critical edges, consider the following lemma.

Lemma 1 *Let f be a max-flow for G, and let (u, v) be any edge in G. Consider the residual network G_f. Then the set of edges (u, v) such that there is no path from u to v in G_f is exactly the set of downward-critical edges.*

Proof. If there is no path from u to v, then the edge (u, v) is saturated. Therefore, $f((u, v)) = c((u, v))$. Say we decrement the capacity by δ. Then if we are to keep the max flow at its current value, the δ units of flow over capacity must be rerouted. But note that we cannot reroute the flow because there is no path from u to v. Therefore, (u, v) is a downward-critical edge.

Now suppose there is a path from u to v, with minimum capacity δ. If we increase the capacity of (u, v) by δ, we can reroute the flow through the path from u to v, so that the max-flow would remain the same. So (u, v) is not a downward-critical edge. ■

The algorithm then is to compute all pairs (u, v) such that (u, v) is an edge in the graph, but there is no path from u to v in the residual graph G_f. This can be done using the Bellman-Ford algorithm, for example. The running time of this algorithm is just the time to compute the max flow and then the running time of Bellman-Ford.

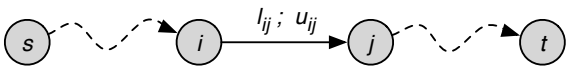
Problem 2. To solve this problem, construct a new graph G' where each node i with a node capacity is replaced by two nodes i_1 and i_2. All edges that previously went into the node should now point to i_1, and all edges that previously exited the node should now exit from i_2. Add an edge from v_1 to v_2 with the capacity $w(i)$. Finding a max flow on this graph will correctly limit the amount of flow that can pass through each node while still finding a max flow allowed by the arc capacities. In terms of worst-case complexity, this problem is only a constant factor worse since n increases by at most a factor of 2 and m increases by at most n, which we can usually assume is less than m. (If $m < n$, not all of the graph is connected, so we don't even care about the new edges that we can't get to.)

(a) As suggested by the hint, we first construct some feasible flow on the graph G, and then construct a minimum flow using the feasible flow.

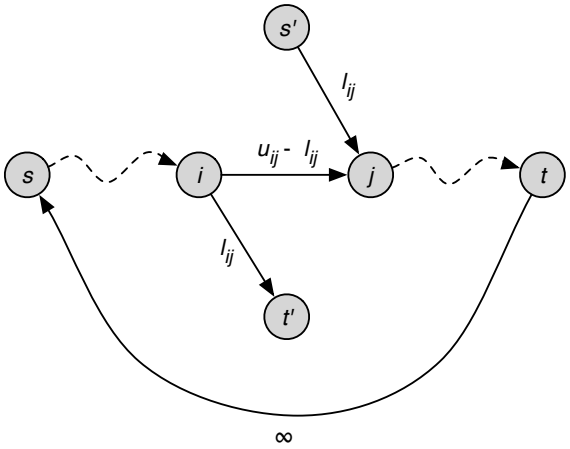
We find a feasible as follows. Note that just finding a max flow from s to t is not sufficient as that does not guarantee that the flow flows along the correct edges (to satisfy the minimums). Instead, we modify the original graph G to get a graph G' as follows. For each edge (i, j) in G, we change the capacity $u'_{ij} = u_{ij} - l_{ij}$. Then we drop the minimum capacities on all edges. Thus, in essence, we are forcing a flow of l_{ij} across each of the edges, but we have a surplus and deficit of l_{ij} at i and j respectively. What ways could we get rid of this deficit? Well, we could find a path from a surplus to a deficit node in G'. Or, we could find a path from a surplus node j to the sink t and the source s to the surplus node i. We want to solve this problem using max flows. So, we add to G' a new source s' and a new sink t'. We add edges (i, t') and (s', j) with capacities $u'_{it'} = u'_{s'j} = l_{ij}$[1]. Finally, we add an edge (t, s) (between the original source and

[1]We may add more than one edge (i, t') for example, or we can add a single edge with the sums of the relevant capacities. More formally, we could say that $u'_{it'} = \sum_j l_{ij}$ and $u'_{s'j} = \sum_i l_{ij}$.

sink) with unbounded capacity $u'_{ts} = \infty$. If we start with the graph:



We would end with the graph



Then all we need to do is find a max flow from s' to t'. We saturate all the edges (s', j) if and only if for every node j with a surplus, there are path(s) to t and/or some other deficit node(s) with enough capacity. Thus, we just check, are all these edges saturated? If no, there is no feasible flow on the graph G'. If yes, then there is a feasible flow, AND we know what it is. All we need to do is drop all the extra edges we added, look at the flow f' we found on G', and add the original edge minimums. For example, if we find flow of f'_{ij} across an edge (i, j), then the flow in the original graph is just $f = f'_{ij} + l_{ij}$. Another way of thinking about this is that going from s' to t' in the graph G' is equivalent to going from s to t in the original graph, because we move the flow from (s', j) and (i, t') to the edge (i, j).

Next, we convert the feasbile flow into a minimum flow. Let our feasible flow on any edge from i to j be f_{ij} (in gross flow formulation). We construct a "reverse residual graph" by taking every edge from i to j in the original graph G, and replacing it with two edges in G': one edge from i to j with capacity $u_{ij} - f_{ij}$, and one edge from j to i with capacity $f_{ij} - l_{ij}$. We might get two edges by this construction going from i to j, so we combine the edges by adding the capacities. So the reverse residual graph has an edge from i to j with capacity $u_{ij} - f_{ij} + f_{ji} - l_{ji}$.

Then we compute a normal maximum flow g_{ij} from t to s in G', and add the result back into the flow in G to get a flow f'. When we add the flow back, we decompose g into $g_{ij} = g'_{ij} + h_{ij}$, where $0 \leq g'_{ij} \leq u_{ij} - f_{ij}$ and $0 \leq h_{ij} \leq f_{ji} - l_{ji}$. We add the flow back by adding g' in the forward direction and subtracting out

h_{ji} in the backward direction. So the new flow from i to j in our original graph G is

$$f'_{ij} = f_{ij} + g'_{ij} - h_{ji}.$$

The inequality relationships we have are

$$\begin{aligned} l_{ij} &\leq f_{ij} \leq u_{ij} \\ 0 &\leq g'_{ij} \leq u_{ij} - f_{ij} \\ 0 &\leq h_{ij} \leq f_{ji} - l_{ji} \end{aligned}$$

Adding f_{ij} to both sides of $g'_{ij} \leq u_{ij}$ and using the fact that $h_{ji} \geq 0$ tells us that $f'_{ij} \leq u_{ij}$. Adding f_{ij} to the inequality $-h_{ji} \geq l_{ij} - f_{ij}$ and using the fact that $g_{ij} \geq 0$ gives us $f'_{ij} \geq l_{ij}$. Therefore, when we add our flow back, we still get a feasible flow.

This flow f' we compute must be a minimum flow. Otherwise, we could reduce the flow further by finding some path from t to s in G. This corresponds to finding some augmenting path in G' from t to s, suggesting that the flow we computed in wasn't a maximum flow for G'. This is a contradiction, so f' must be a minimum flow.

(b) **Claim 2** *Let the lower bound on the cut capacity of an s-t cut S be defined as $L(S) = \sum_{(i,j)\in S\times T} l_{ij} - \sum_{(i,j)\in T\times S} u_{ij}$. Then the minimum value of all feasible flows from node s to t equals the* $\max_S L(S)$.

Proof. We show that for any flow, $f >= \max_S L(S)$ and that min flow $f \not> \max_S L(S)$, which implies that minimum value of all feasible flows from s to t equals $\max_S L(S)$.

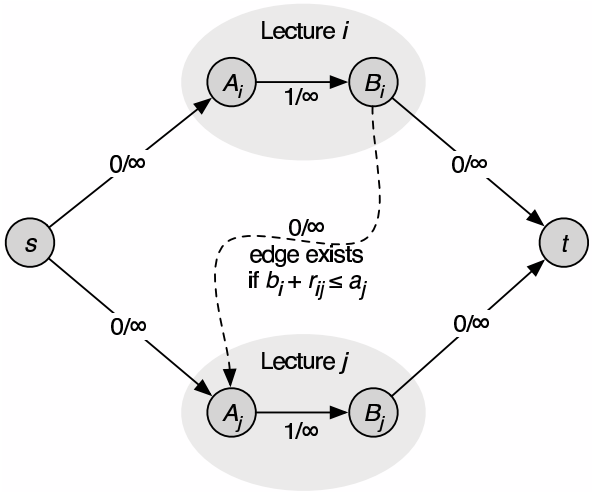
Suppose we have a flow f. Then we want to show that $f \geq \max L(S)$. Instead, we show that $f \geq L(S)$ for any S. Consider any cut S. Since the flow is feasible, we satisfy all the minimum capacities. So, the flow from S to T is at least $\sum_{(i,j)\in S\times T} l_{ij}$. Since the flow is feasible, the most that can be flowing from T to S is bounded by the maximum capacities $\sum_{(i,j)\in T\times S} u_{ij}$. Thus, the net flow across the cut is $\geq \sum_{(i,j)\in S\times T} l_{ij} - \sum_{(i,j)\in T\times S} u_{ij}$. Therefore $f \geq L(S)$.

Suppose that we have a min flow f. Assume for the sake of contradiction that $f > \max L(S)$. Then for all cuts $f > L(S)$. $f > \sum_{(i,j)\in S\times T} l_{ij} - \sum_{(i,j)\in T\times S} u_{ij}$ implies that either we are sending more flow than the minimum from S to T, or we are not sending the maximum amount back from T to S. Consider the first case. Then our modified residual graph G'_f from part (a) has extra capacity available from T to S on the residual (reverse) edges. In the second case, we also have capacity available from T to S. Since there is capacity available across ALL CUTS going from T to S, there is an augmenting path in G'_f from t to s,[2] and

[2]The max-flow min-cut theorem tells us that $|f| \neq u(S)$ for any S implies that there is an augmenting path in the residual graph.

we don't have a min flow. ■

(c) For lecture i, we create two vertices in our graph: A_i to represent the start of the lecture and B_i to represent the end. We draw and edge (A_i, B_i) for every lecture i, with a minimum flow $l(A_i, B_i) = 1$ and a max flow $f(A_i, B_i) = \infty$. We create a source s and a sink t. We create edges (s, A_i) from the source to the start of every class, with unconstrained capacities $l(s, A_i) = 0$ and $f(s, A_i) = \infty$. Similarly, we add edges (B_i, t) from the end of every lecture to the sink, with unconstrained capacities $l(B_i, t) = 0$ and $f(B_i, t) = \infty$. Finally, we introduce edges from the end of lecture i to the start of lecture j if it is possible to commute from lecture i to lecture j. That is, (B_i, A_j) exists if $b_i + r_{ij} \leq a_j$. Again, these edges have unconstrained capacity. The graph looks like:



Now we just solve for the min flow. We claim that the min flow is the minimum number of students necessary to attend all the lectures. Basically, each unit of flow represents a student. We argue correctness by showing that our graph has the same constraints as the students. In particular, a student can only attend two lectures i and j if the commute time allows it. Similarly, we only have an edge in the graph allowing flow between lectures i and j if the commute times allow it. A student can choose any lecture to be the first one he goes to (modeled by edges from s to all lecture starts), and he can choose any lecture to be his last (modeled by edges from lecture ends to t). Finally, the only other constraint we have is that at least one student must be in attendance at each lecture. By creating two vertices for each lecture and having an edge with min capacity of 1 between lecture start and end, we guarantee at least one unit of flow across the lecture—that is, we guarantee that at least one student sits through the entire lecture.

Problem 3. **(a)** Define the desired-meetings graph G to be a bipartite graph with left vertices L for students, right vertices R for faculty members, and an edge (x, y) whenever student x wishes to meet faculty y. There are d edges for each student, and d edges for each faculty member, so that $d\,|L| = d\,|R|$, and by cancellation $|L| = |R|$.

We wish to find a maximum matching in G, so we augment it with a source s connected to all the students, and a sink t that all the faculty are connected to, obtaining a graph G' on which we seek a max-flow. We will find the value of the max-flow by lower-bounding the value of any (s, t)-cut.

It is clear that the minimum (s, t)-cut has value at most n, because the cut $\{s\}$ achieves this. Consider an arbitrary (s, t)-cut $S, \bar{S}$. S consists of the source s, some students $L' \subseteq L$ and some faculty members $R' \subseteq R$. Then the cut consists of

- $n - |L'|$ edges from s to $L \setminus L'$.
- Edges from L' to $R \setminus R'$, and from $L \setminus L'$ to R'. There are at least $d\big||L'| - |R'|\big|$.
- $|R'|$ edges from L' to t.

This yields a total of $n + (d \pm 1)\big||L'| - |R'|\big| \geq n$, so we are guaranteed that all students and faculty can be matched up for a single slot.

(b) Each time we apply the argument from (a), we reduce the degree of each student and faculty node by 1, but keep the regularity property. So we can do this repeatedly and schedule d time slots in which everyone can meet everyone they want to meet.

(c) Let d be the maximum number of students who want to meet any one faculty member, or faculty members any one student wants to meet. Then we can reduce the general problem to the previous case by adding dummy students or faculty to make the numbers equal, and then adding dummy edges between students and faculty whose degree is less than d. In essence, we set up extra meetings that won't actually happen. We can always add these edges, because as long as some student has fewer than d faculty members to meet, some faculty member must have fewer than d students to meet. Note that this might create multiple edges for a student-professor pair. However, this does not affect the minimum cut, so the argument from (a) still holds.

Problem 4. Let $i = 0$ represent the Red Sox. Then in the best case, the Red Sox win all their remaining $\sum_j q_{0j}$ games for a total of $W = w_0 + \sum_j q_{0j}$ season wins. We wish to determine if it possible to decide the remaining games so that any other team wins less than W games in all. We can formulate this as a max-flow problem as follows:

Let T be the set of teams other than the Sox. Define the network $G = (V, E, u)$ where

$V = \{\binom{T}{2} \cup T \cup \{s,t\}\}$ and

$$E = \{s\} \times \binom{T}{2} \cup \{(\{i,j\},i) \mid i,j \in T\} \cup T \times \{t\}$$

with capacities

$$u(x,y) = \begin{cases} q_{ij} & x = s \text{ and } y \in \binom{T}{2} \\ q_{ij} & x \in \binom{T}{2} \text{ and } y \in x \\ W - w_i - 1 & x \in T \text{ and } y = t. \end{cases}$$

(Here, $\binom{T}{2} = \{\{i,j\} \subset T \mid i \neq j\}$). Consider an (integer) max-flow of this network that saturates all the edges leaving s. We can interpret the flow on an edge of the form $(\{i,j\},i)$ as the number of wins team i scores against team j in the rest of the season. The capacity constraints on edges entering t ensure that every team wins fewer games than the Sox at the end of the season. Conversely, if the Sox can win at the end of the season, the record of wins and losses will give us an integer max-flow of this graph.

Therefore, we can check if the Sox can still win by computing the max-flow of this graph and seeing if it saturates all the edges leaving s.

Problem 5. Define the following sets:

$$\begin{aligned} P &= \text{positions on the results page} \\ C &= \text{clicked old results} \\ U &= \text{unclicked old results} \\ N &= \text{new results} \end{aligned}$$

and let $\pi\colon (C \cup U) \to P$ map old results to their positions in the old list. Then define the network $G = (V,E)$ with edge capacities and costs as follows:

$$\begin{aligned} V &= \{s,t,x,y,z\} \cup C \cup U \cup N \cup P \\ E &= \{(s,x),(s,y),(s,z),(z,y)\} \ \cup \ \{x\} \times C \ \cup \ \{y\} \times N \cup \\ &\quad \{z\} \times U \ \cup \ (C \cup N \cup U) \times P \ \cup \ P \times \{t\} \\ u(i,j) &= \begin{cases} |C| & (i,j) = (s,x) \\ k & (i,j) = (s,y) \\ n - |C| - k & (i,j) \in \{(s,z),(z,y)\} \\ 1 & \text{otherwise} \end{cases} \\ c(i,j) &= \begin{cases} v_i - p_{\pi(i),j} & (i,j) \in (C \cup U) \times P \\ v_i & (i,j) \in N \times P \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

It is easy to check that an integer min-cost max-flow of this graph corresponds to a solution to our problem. Such a solution can be computed in polynomial time using one of the algorithms from class.

Problem 6. **(a)** Consider the admissible graph. Let L_i denote the vertices located in the layer i (with distance i from the source). We note that $L_0 = s$ and $L_d = t$. Then we take the summation of number of elements in all pairs of layers

$$\sum_{i=1}^{d} |L_i| + |L_{i+1}| \leq 2n \ .$$

We conclude the inequality because we count each layer at most twice. By pigeonholing, when we add d elements to get $2n$, there must be at least one element of value $\leq 2n/d$. Thus, we conclude that $|L_i| + |L_{i+1}| \leq 2n/d$ for some i, or $(|L_i| + |L_{i+1}|)/2 \leq n/d$.

The number of edges between layers i and $i+1$ is at most[3]

$$|L_i|\,|L_{i+1}| \leq \left(\frac{|L_i| + |L_{i+1}|}{2}\right)^2 \leq \left(\frac{n}{d}\right)^2 \ .$$

Since the full graph has no edges jumping layers that are in the admissible graph (otherwise, the layers are incorrect in the admissible graph), we conclude that the cut between layers L_i and L_{i+1} has a capacity $\leq (n/d)^2$, as each edge has unit capacity. Thus, the max flow is at most $(n/d)^2$.

After k blocking flows, s and t are at a distance of k from each other. The max flow of the residual graph is $(n/k)^2$ from above, thus we need to do at most $(n/k)^2$ more blocking flows. Setting $k = n^{2/3}$, we get the total number of blocking flows is $O(k + (n/k)^2) = O(n^{2/3} + (n/n^{2/3})^2) = O(n^{2/3} + (n^{1/3})^2) = O(n^{2/3})$.

[3]First step of inequality follows from $\sqrt{ab} \leq (a+b)/2$, or geometric mean is at most arithmetic mean.