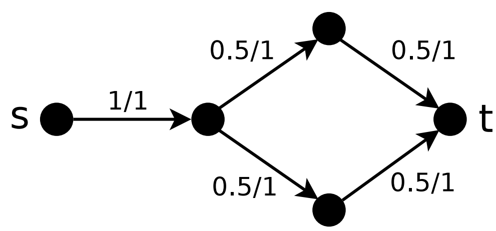
COMP 360 - Fall 2012 - Assignment 1
Solutions

October 9, 2012

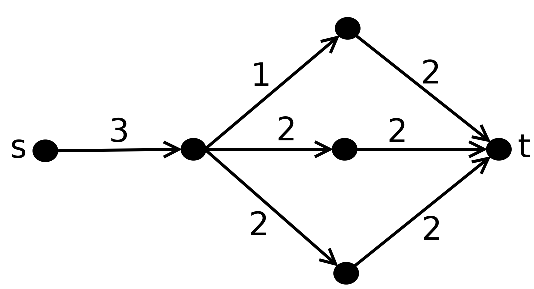
1. (a) Below is an example of a maximum flow with non-integer flow on some edges. Here the notation 0.5/1 means that the capacity of the edge is 1 and we are pushing 0.5 units of flow along it.



Note that the question is not specifically asking about the Ford-Fulkerson algorithm, which does not assign non-integer flow to edges. Also note that the capacities are always integers.

(b) True. It is easy to see that by multiplying every edge capacity by 2, we multiply the capacity of every cut by 2. Therefore a minimum cut in the original graph is also a minimum cut in the new graph with all the capacities doubled. Using max flow = min cut, we conclude that the flow in the new graph is doubled.

(c) The following graph is a counter-example. Note that there is only one minimum cut (A, B) and in this cut $A = \{s\}$. The capacity of this cut is 3 and it does not contain the edge with capacity 1.



(d) The graph above is a counter-example for this as well.

2. This is known as the *flow decomposition theorem*. To prove this, we apply the following algorithm to decompose a maximum s, t-flow f into a set of flow-paths.

(1) Find an s, t-path P with positive flow, i.e., $f(e) > 0$ for all $e \in P$.

(2) Let $\Delta = \min_{e \in P} f(e)$. That is, Δ is the minimum value of the flow f on edges of P. We construct a flow f_P by setting $f_P(e) = \Delta$ for all $e \in P$ and $f_P(e) = 0$ for all other edges. Then we decrease the flow f on each edge $e \in P$ by Δ.

(3) Repeat Step (1) and (2) until there is no such s,t-path.

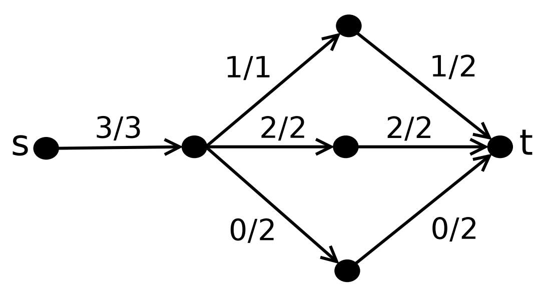
Each time we decrease the flow f on at least one edge to zero. Thus, the algorithm terminates in at most m times. So, we have a set $\mathcal{P}$ of at most m flow-paths. These flow-paths give augmentations that lead to a flow $f' = \sum_{P \in \mathcal{P}} f_P$. Since f' is the sum of flow-paths in $\mathcal{P}$ and $f'(e) \leq f(e) \leq c(e)$ for all edges e, we conclude that f' is a valid flow. It remains to show that f' is the maximum flow.

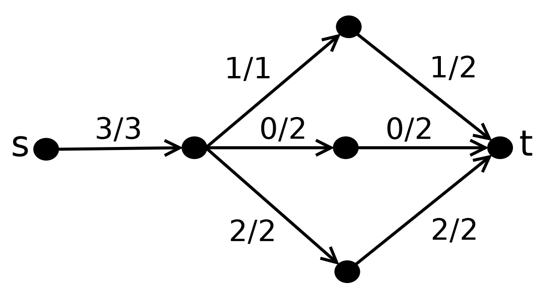
First, observe that the flow f remains valid after Step (2). The flow conservation constraint holds because, for each node $v \in P$, we decrease the flow f on leaving and entering edges by the same amount. The capacity constraint holds because we only decrease the flow on each edge, and by the choice of Δ, we never decrease the flow below zero.

By flow conservation, if $f^{out}(s) > 0$, then there must be an s,t-path with positive flow in Step (1). Suppose there is no such path. Then every path from s with positive flow ends at some vertex $v \neq t$. But, this would imply that $f^{in}(v) > 0$ and $f^{out}(v) = 0$, a contradiction.

Hence, at the termination, $f^{out}(s) = 0$. The value that $f'^{out}(s)$ increases is equal to the value that $f^{out}(s)$ decreases. Thus, f' is the maximum flow as required.

3. In the example given in 1c, we have one min cut but several max flows. Here are two of them:

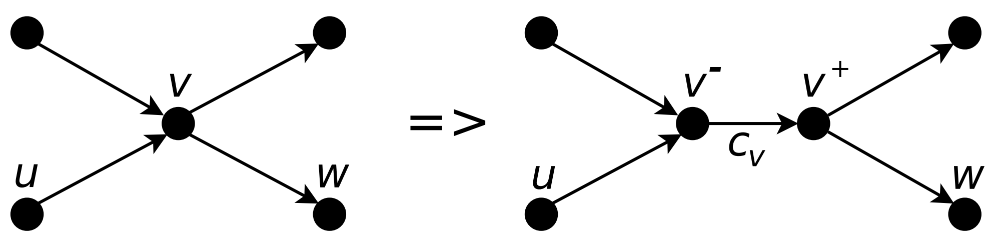




4. Build a flow network where we add a source s and connect it to every vertex in S, we add a sink t and connect every vertex Y to t. We direct the edges between S and Y so that they go from S to Y. All edge capacities are set to 1. Run the Ford-Fulkerson algorithm and find a maximum flow f. This will correspond to a maximum matching between the vertices of S and Y (as seen in class). If the size of this matching is not $|S|$, all the vertices of S are not matched so a solution to our problem does not exist. Otherwise, we take the residual graph corresponding to f and modify it in order to add the vertices in $X - S$. We add an edge

between s and the vertices in $X - S$ and also add the edges between $X - S$ and Y that were already present in the original bipartite graph G. We set the capacities of all the new edges to 1. We now take this modified residual graph and run the Ford-Fulkerson algorithm with it. This will find a maximum matching in which all the vertices in S are matched. To see this we need to make two important observations. First, the flow that is going through the S vertices can never be pushed back and therefore they will always be a part of the maximum matching. Second, the matching we find is maximum: no other matching between X and Y can be bigger. This is because in the Ford-Fulkerson algorithm, it does not matter how (or in what order) we pick the augmenting paths, we will always find a maximum flow. The first part of the algorithm where we find a maximum matching between S and Y actually corresponds to an initial choice of augmenting paths. Then by adding in the vertices in $X - S$, we allow ourselves to pick augmenting paths that go through those vertices as well.

5. Let N be the network of the node-capacitated maximum s, t-flow problem. First, we construct an auxiliary network N' by replacing each node v by v^+ and v^-. Then we add an edge (v^-, v^+) with capacity c_v. We move heads of edges entering v to v^- and move tails of edges leaving v to v^+. That is, we replace each edge (u, v) by (u, v^-) and replace each edge (v, w) by (v^+, w).



Intuitively, this is the same as putting a capacity c_v on each vertex v. Thus, we can apply any efficient maximum s, t-flow algorithm on N' to solve the standard maximum flow problem and transform it to the a maximum flow in the original problem.

To be precise, we have to show that there is a one-to-one mapping between a flow f in N and a flow f' in N'. Observe that every edge of N is also an edge of N' (with heads and tails moved). Thus, it is easy to transform between a flow in N and a flow in N'. Given any flow f in N, we can construct a flow f' in N' by letting $f'(e) = f(e)$ for all edges e of N and letting $f'(v^-, v^+) = f^{in}(v)$ for all nodes v of N. Clearly, f' satisfies both capacity and flow conservation constraints. Thus, f' is a valid flow in N'. Given any flow f' in N', we can construct a flow f in N by letting $f(e) = f'(e)$ for all edges e of N. The edge-capacity and flow conservation constraints remain valid for f. The vertex-capacity constraint holds on each vertex v because $f^{in}(v) = f'^{in}(v^-) = f'^{out}(v^-) = f(v^-, v^+) \leq c_v$. So, f is a valid flow in N. Since there is a one-to-one mapping between flows in N and N', the maximum flow in N' gives the maximum flow in N. This proves the correctness of the algorithm.

Remark: We have to show that there are mappings in both directions. A mapping from f to f' shows that a maximum flow in N is also a flow in N', but N' may have a flow with a larger value. To rule this out, we have to show that any flow f' can be mapped to a valid flow f in N which means that any maximum flow in N' is also a maximum flow in N.