Homework 9 Solutions

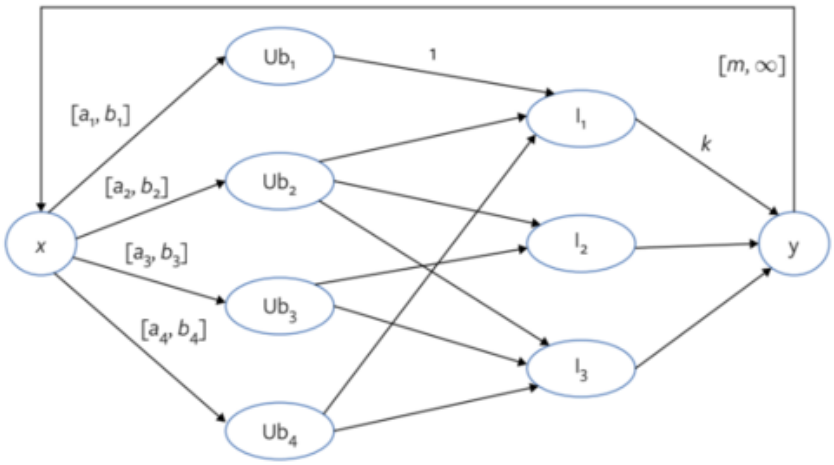
All Ungraded

1) Consider LAX, a notoriously busy airport with many arriving passengers who want to get to their destinations as soon as possible. There is an available fleet of **n** Uber drivers to accommodate all passengers who need to be assigned to various intervals of the day. There is a traffic regulation at the airport that limits the total number of Uber drivers at any given hour-long interval to **k**. Assume that there are **p** time intervals. Each driver provides a subset of the time intervals he or she can work at the airport, with the minimum requirement of driver j being $\mathbf{a_j}$ hour(s) per day and the maximum $\mathbf{b_j}$ hour(s) per day. Lastly, the total number of Uber drivers available per day (summed over all intervals) must be at least **m** to maintain a minimum customer satisfaction and loyalty.

a) Design a network flow based algorithm to determine if there is a valid way to schedule the Uber drivers with respect to these constraints.

b) Prove correctness of your algorithm.

Solution:



We will reduce the Uber driver's problem to a circulation problem. We construct a Network H as follows. We have a node Ub_j for each uber driver j and node I_t for each hour-long time interval t. We insert the edge between driver Ubi and time interval Ij if the driver prefers to work at that hour. The capacity of this edge is 1. There could be many drivers willing to work at that hour, so having flow 0 on that edge is interpreted as a driver not covering that time interval.

Next, we add two new vertices x and y (Note that we don't call these source and sink - a circulation network doesn't have a source/sink). Connect x to all Ub_j, and all I_t to y. The edge (x, Ub_j) has lower bound a_j and upper bound b_j for each driver j. The edge (I_t, y) has capacity k for all t.

Finally, we add the edge (y, x). The flow on this edge represents the total number of Uber drivers serving the airport, and so we set a lower bound of m.

Claim. There is a valid way to schedule the Uber drivers if and only if there is a feasible circulation in network H constructed above.

Proof:
Forward Claim:
Assume that there is a valid way to schedule the Uber drivers subject to all the constraints.
We show there is feasible circulation in H as follows. For every driver j who works during a time interval t, we assign a flow of one unit on edges (x, Ub_j), (Ub_j, I_t), (I_t, y) and (y, x). Thus, doing so sets

- a flow along the edge (s, Ub_j) to the number of time intervals that driver works, so the lower bound and capacity here are satisfied given driver j is correctly assigned to at least a_j and at most b_j hour(s) per day.
- the flow along the edges (I_t, y) is the number of drivers who work during that time interval t, which is at most k, satisfying the capacity constraint on these edges.
- Finally, the flow on edge (y, x) is the total number of Uber drivers serving the airport which is at least m, satisfying the lower bound here.

Thus, the circulation as defined above is feasible.

Backward Claim:
Consider a feasible circulation in H. For each edge (Ub_j, I_t) that carries one unit of flow, we assign driver j to interval t. Then,

- (since there is no demand i.e., imbalance at node Ub_j the flow on the edge (s, Ub_j) represents the total number hours that driver j works. Since the circulation is feasible, the lower bound and capacity here imply that driver j is correctly assigned to at least a_j and at most b_j hour(s) per day.
- Similarly, the flow along the edge (I_t, y) equals the number of drivers who work during interval t, which must be at most k as required, due to the capacity constraint on these edges.
- Finally, by flow conservation at y (or x) again, (since no demand at y), the flow on edge (y, x) gives the total number of Uber drivers through the whole day which must be at least m as required, due to the lower bound on edge (y,x).

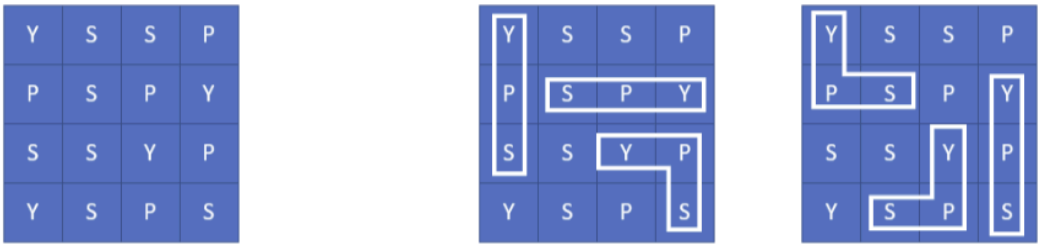
Rubrics:
+12 - Correct network flow diagram
+2 - Correct Iff Claim (combining both with forward and backward claims)
+3 - Justification for Correct Forward claim
+3 - Justification Correct Backward claim

2) Counter Espionage Academy instructors have designed the following problem to see how well trainees can detect SPY's in an nXn grid of letters S, P, and Y. Trainees are instructed to detect as many disjoint copies of the word SPY as possible in the given grid. To form the word SPY in the grid they can start at any S, move to a neighboring P, then move to a neighboring Y. (They can move north, east, south or west to get to a neighbor.) The following figure shows one such problem on the left, along with two possible optimal solutions with three SPY's each on the right.

a) Give an efficient network flow-based algorithm to find the largest number of SPY's. (20 pts)

b) Prove correctness of your solution.

Note: We are only looking for the largest **number** of SPYs not the actual location of the words. No proof is necessary.



Solution:

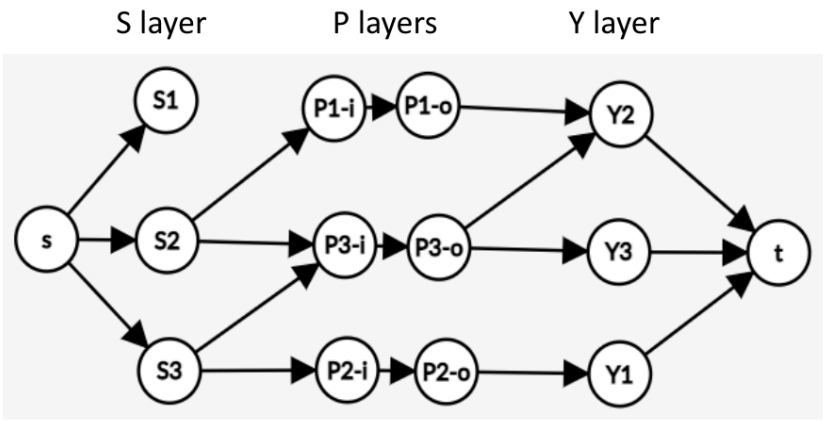
1. Create one layer of nodes for all the Ss in the grid. Connect this layer to a source vertex s, directing edges from s to S.

2. Create two layers of nodes for all the Ps in the grid. Connect the first layer to the Ss based on whether or not they are adjacent in the grid, directing edges from S to P. Also, connect the first layer to the second layer by connecting the nodes that correspond to the same location. In other words, we are representing P's as an edge with capacity 1. This is similar to how we solved the node disjoint paths problem.

3. Create a layer of nodes for all the Y s in the grid. Connect these to a sink vertex t, directing edges from Y to t. Also, connect them to the second layer of Ps based if the P and Y are adjacent in the grid, directing edges from P to Y .

Note that we don't need to represent nodes S or P with edges of capacity 1 since there are edges of capacity 1 going into S's and edges of capacity 1 leaving Y's which will force these letters to be picked only once.

Y_1	S_1	S_2	P_1
P_2	S_3	P_3	Y_2
S_4	S_5	Y_3	P_4
Y_4	S_6	P_5	S_7



The network is demonstrated for the red-bordered section of the grid above.

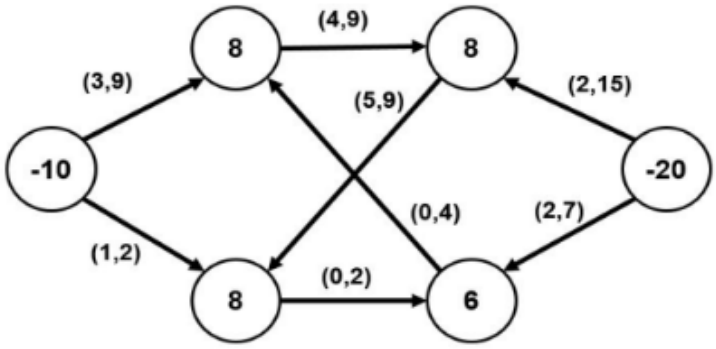
Claim: Value of Max Flow in this Flow Network equals the maximum number of disjoint SPYs.

Proof:

Forward claim: Suppose M copies of SPY are found. For any copy that uses the letter S_i, P_j, Y_k, we add a unit of flow along the path s→ S_i→ P_j-i → P_j-o → Y_k → t. By construction, such a path exists for a valid copy of SPY. Doing so for each copy gives a flow of value M. Since the copies of SPY do not share the letters, each letter is on at most one flow path, and thus, the capacity constraints on the edges are satisfied.

Backward claim: Suppose we have s-t flow of value M in the network. By construction, each unit of s-t flow is on a path s→ S_i→ P_j-i → P_j-o → Y_k → t for some i,j,k. By construction, S_i, P_j, Y_k can form a valid copy of SPY. Doing so for each unit of s-t flow gives M copies. the capacity constraints on s→ S_i, P_j-i → P_j-o, Y_k → t ensure each i,j,k triplet is different, i.e., the resultant copies of SPY are non-overlapping.

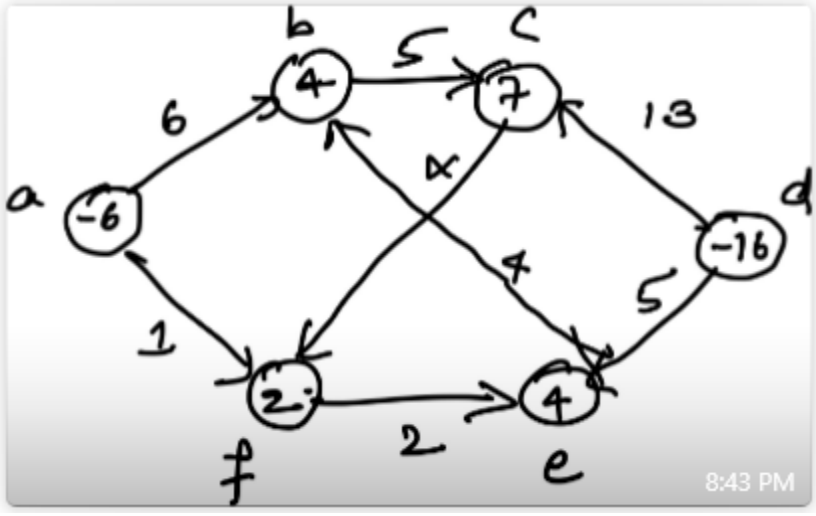
3) In the network below, the demand values are shown on vertices. Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if a feasible circulation exists or not. If it does, show the feasible circulation. If it does not, show the cut in the network that is the bottleneck. Show all your steps



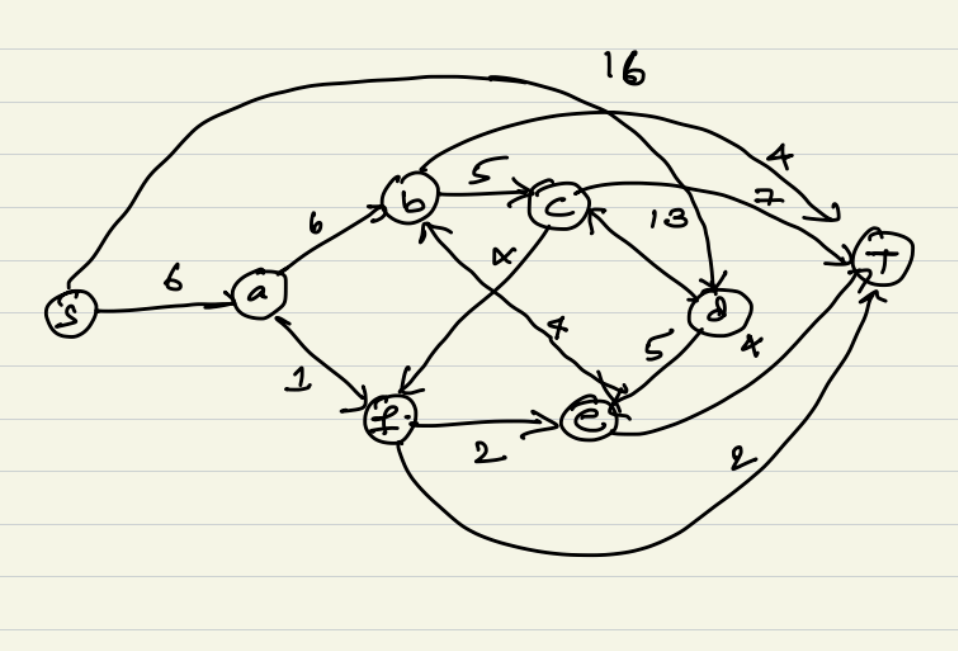
Solution:

First, we eliminate the lower bound from the edges. Then, we attach a super-source s* to each node with negative demand, and a super-sink t* to each node with positive demand. The capacities of the edges attached accordingly correspond to the demand of the nodes.

(A) Turn it into a circulation problem without lower bound.



Describe the node and edge values in the new graph.
(B) Turn it into a max flow problem.



Describe the edge flow values.

Capacity of edges into T is only 17 while that out of S is 22. Thus, Max-flow cannot saturate the latter edges. Hence, there is no circulation. Running any man-flow algorithm gives us ({S,a,b,c,d,e,f}, {T}) as a min-cut.

4) USC Admissions Center needs your help in planning paths for Campus tours given to prospective students or interested groups. Let USC campus be modeled as a weighted, directed graph G containing locations V connected by one-way roads E. On a busy day, let k be the number of campus tours that have to be done at the same time. It is required that the paths of campus tours do not use the same roads. Let the tour have k starting locations $A = \{a1, a2, ..., ak\} \subseteq V$. From the starting locations, the groups are taken by a guide on a path through G to some ending location in $B = \{b1, b2, ..., bk\} \subseteq V$. Your goal is to find a path for each group i from the starting location, ai, to any ending location bj such that no two paths share any edges, and no two groups end in the same location bj .

(a) Design an algorithm to find k paths ai → bj that start and end at different vertices, such that they do not share any edges.

(b) Modify your algorithm to find k paths ai → bj that start and end in different locations, such that they do not share any edges or vertices.

Solution:

(a) The complete algorithm is as follows:

1. Create a flow network G′ containing all vertices in V , all directed edges in E with capacity 1, and additionally a source vertex s and a sink vertex t. Connect the source to each starting location with a directed edge (s, ai) and each ending location to the sink with a directed edge (bi t), all with capacity 1.

2. Run Ford-Fulkerson on this network to get a maximum flow f on this network. If |f| = k, then there is a solution; if |f| < k, then there is no solution, so we return FALSE.

3. To extract the paths from ai to bj (as well as which starting location ultimately connects to which ending location), run a depth-first search on the returned max flow f starting from s, tracing a path to t. Remove these edges and repeat k times until we have the k edge disjoint paths. To justify correctness, any argument conveying that there is a flow of size k if and only if there are k edge disjoint paths from A to B is enough.

(b) Duplicate each vertex v into two vertices vin and vout, with a directed edge between them. All edges (u, v) now become (u, vin); all edges (v, w) now become (vout, w). Assign the edge (vin, vout) capacity 1. With this transformation, we now have a graph in which there is a single edge corresponding to each vertex, and thus any paths that formerly shared vertices would be forced to share this edge. Now, we can use the same algorithm as in part (a) on the modified graph to find k edge disjoint paths sharing neither edges nor vertices,if they exist.

5) Suppose you are on vacation visiting a new city. You will be spending d days there which you want to plan in advance. You have identified k different neighborhoods of the city that you must visit at least once during your trip. Each neighborhood i has R_i different restaurants. Each restaurant offers one of the 5 cuisines you enjoy: Italian, Mexican, Indian, Mediterranean, Chinese. You want to make sure you try at least one restaurant from each cuisine during your vacation, but you don't want to eat at the same exact restaurant more than twice. Your plan for each day consists of visiting a neighborhood and picking one restaurant for dinner within that neighborhood.

a) Devise a network flow based efficient algorithm to determine if such a vacation plan is feasible.
Note: you need to either describe the meaning of all nodes and edges and flow constraints on all edges, etc., or you can draw your network and clearly mark all of that information on the network.

b) Prove the correctness of your solution.

Solution:
The most natural approach is to construct a feasible-circulation based solution.
Construct the network as follows:

S, T with demands -d and +d.
Construct 5 cuisine nodes and connect them to T.
Construct k neighborhood nodes and connect S to them.
Connect a middle layer of restaurant nodes, connect each restaurant to the corresponding cuisine node and from the corresponding neighborhood node (thus, exactly one incoming and outgoing edge for each restaurant).
cuisine -> T edges must have LB 1, capacity large enough (d or infty).
restaurant -> cuisine and Neighborhood -> restaurant edges have lb=0, cap=2.
S -> Neighborhood edges have Lb=1, cap large enough (infty or 2Ri or d etc. works)

If this network has feasible circulation, output 'vacation plan feasible'

Claim: vacation plan satisfying the constraints exists if and only if feasible circulation in the constructed network.

Forward claim: Given a valid vacation plan, for each day construct circulation by assigning a 1 unit of flow S - Nb(i) - Rest(j) - Cuisine(rest j) - T where neighborhood Nb(i) was visited on the particular day, and rest(j) was chosen with the corresponding cuisine. The validity of the vacation plan ensures the lower bounds, demands and capacities of the circulation are met (details left as exercise.)

Backward claim: Given the feasible circulation, for each unit of flow on edge Nb(i) - Rest(j) Assign neighborhood i and restaurant j to a particular day. The feasibility of circulation (lower bounds, demands and capacities being satisfied) will ensure that the vacation plan is valid as per constraints (details left as exercise.)

6) There is a precious diamond on display in a museum. After a recent museum robbery, it was decided to tighten the security arrangements. There are *n* security guards available who can be deployed for the task. The diamond is to be guarded at *m* disjoint time intervals of the day, and must have either one or two guards deployed in each interval. Each guard provides a list of intervals for which they are available to be deployed. Work regulations require that the deployment for each guard must be for at most *M* and at least *L* time slots.

a) Design a network flow algorithm to determine a feasible assignment of the guards given the constraints above, and output "no feasible assignment" if it does not exist.
b) Prove correctness of the algorithm.

Solution:
We create a circulation network as follows.
For the i-th guard, introduce a vertex gi & for the j-th time interval, introduce a vertex tj.
If the i-th guard is available for the jth interval, then connect gi to tj with capacity 1.
Add nodes S and T.
add an edge from S to every guard vertex gi of capacity M and lower bound L.
From every interval vertex tj, add an edge to T of capacity 2 and lower bound 1.
Add an edge from t to s of infinite capacity.

Claim: there exists a valid deployment if and only if the constructed network has feasible circulation.

Forward claim: Given guard deployment, for every guard i deployed at interval j, assign 1 unit flow to edges (S, gi), (gi, tj), (tj, T) and (T,S). The resultant circulation in the end can be shown to be feasible given the guard deployment satisfied all the constraints (details left as exercise.)

Backward claim: Given feasible circulation, for every edge (gi, tj) which has unit flow, assign guard i to interval j. We can show that this deployment satisfies all the constraints given that the circulation was feasible (details left as exercise.)