Week 10 Applied Sheet

(Solutions)

Useful advice: The following solutions pertain to the theoretical problems given in the applied classes. You are strongly advised to attempt the problems thoroughly before looking at these solutions. Simply reading the solutions without thinking about the problems will rob you of the practice required to be able to solve complicated problems on your own. You will perform poorly on the exam if you simply attempt to memorise solutions to these problems. Thinking about a problem, even if you do not solve it will greatly increase your understanding of the underlying concepts. Solutions are typically not provided for Python implementation questions. In some cases, pseudocode may be provided where it illustrates a particular useful concept.

Problems

Problem 1. Consider the following circulation with demands and lower bounds problem presented in Figure 1. Does it have a feasible solution?

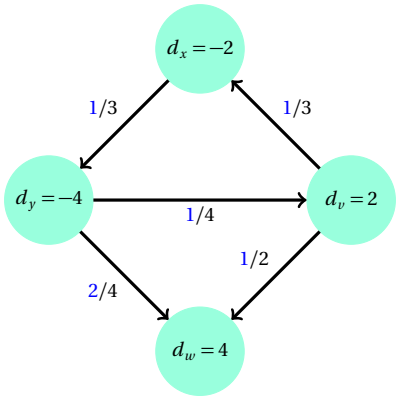


Figure 1: An instance of the circulation with demands and lower bounds problem. The demand is indicated in each vertex, and in each edge its capacity is in black and its lower bound in blue.

Solution

The idea is that we will reduce this problem to the problem of circulation with demands but no lower bounds.

We first define a flow f^ℓ by setting, for each edge, $f^\ell(u,v) = \ell(u,v)$. And then we figure out if there is exists a flow f^* such that, for $f = f^\ell + f^*$, f is a feasible solution to the circulation with demands $\{d_u\}$ and lower bounds problem on G. Note that f^ℓ already takes care that the flow f is at least equal to the lower bound $\ell(u,v)$ on each edge. Now we just have to consider the related graph G^* with adjusted capacities and demands:

- the vertices of G^* are the same as in G, but they now have demand $d^*_u = d_u - \sum_{v\in V} f^\ell(v,u) + \sum_{v\in V} f^\ell(u,v)$;
- the edges of G^* are the same as in G, but they now have capacities $c^*(u,v) = c(u,v) - \ell(u,v)$ and no lower bounds;

and solve the problem of circulation with demands $\{d^*_u\}$ - without lower bounds - in G^*.

For our instance of the problem, we now consider the flow f^ℓ in Figure 2, and the adjusted graph G^* in Figure 3.

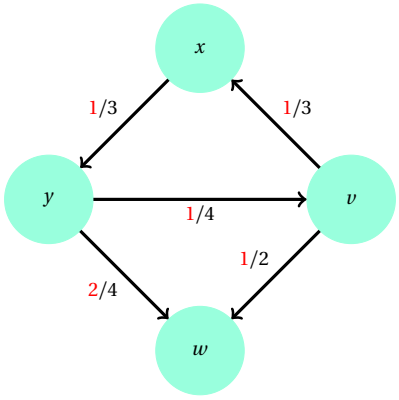


Figure 2: The fixed flow f^{ℓ} (in red) corresponding to the circulation with demands and lower bounds.

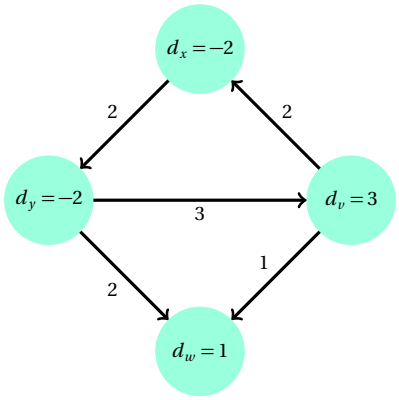


Figure 3: The adjusted graph G^* for the circulation with demands $\{d_u^*\}$ problem (without lower bounds) corresponding to our instance.

A solution of the max-flow problem associated to the circulation with demands $\{d_u^*\}$ problem is presented in Figure 4.

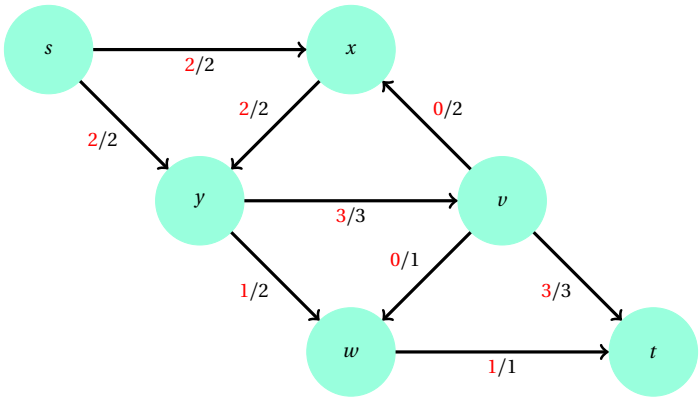


Figure 4: A solution of the max-flow problem associated with the circulation with demands $\{d_u^*\}$ problem.

As all outgoing edges of the source and incoming edges of the sink are saturated, the circulation with

demands $\{d_u^*\}$ problem has a feasible solution f^* that is obtained by just deleting the source, the sink, and their edges. The final solution $f = f^\ell + f^*$ for the circulation with demands $\{d_u\}$ and lower bounds problem in G is depicted in Figure 5.

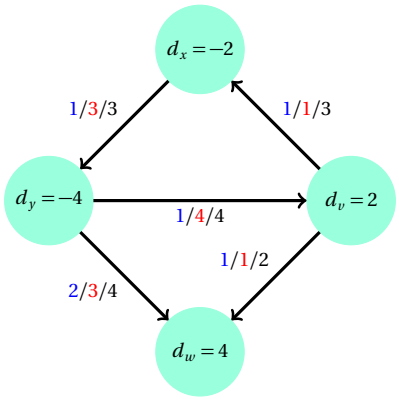


Figure 5: The final solution $f = f^\ell + f^*$ for the circulation with demands $\{d_u\}$ and lower bounds problem in G (lower bounds in blue, flow in red and capacities in black).

Problem 2. Recall the concept of a directed acyclic graph (DAG). A *path cover* of a DAG is a set of paths that include every vertex **exactly once** (multiple paths cannot touch the same vertex). Note that path covers can include paths of length 0 (i.e., paths that contain a single vertex but no edge at all). A minimum path cover is a path cover consisting of as few paths as possible. Devise an efficient algorithm for finding a minimum path cover of a DAG. [Hint: Use bipartite matching]

Solution

Create a bipartite graph where each side has one copy of each vertex in V. For each edge (u, v) in the original graph, add an edge in the bipartite graph (u on the left, v on the right).

Solve for the maximum cardinality matching (using network flow). We want to show that any unmatched vertex on the left is the end of a path, hence the answer is $|V|$ - #successful matches. To see why this is true, note that the matching is essentially choosing for each vertex, where to go to next in the path? Each vertex can be in one path only, hence it can be matched to one vertex, and vice versa (at most one vertex can match to it). The unmatched vertices on the left must therefore be the end of paths, and the unmatched vertices on the right are the beginning of paths.

Problem 3. You are in charge of projects at a large company and need to decide for this year, which projects the company will undertake. Each project has a profit value associated with it. Some projects are worth positive profit, meaning you earn money from completing them. Some projects are worth negative profit, meaning they cost more money than they make. Projects have prerequisites with other projects, meaning that a project can only be completed once all of its prerequisites have been completed. The goal is to determine which projects to complete in order to make the maximum amount of profit, while ensuring that all prerequisite relationships are satisfied. This problem can be solved by reducing it to a minimum cut problem as follows:

- Create a flow network with a source vertex s and a sink vertex t.
- Create a vertex for each project.
- For each project x with positive profit p, add a directed edge from s to x whose weight is p.
- For each project x with negative profit $-p$, add a directed edge from x to t whose weight is p.
- For each project x that has y as a prerequisite, add a directed edge from x to y with weight ∞.

The minimum cut of this network can be used to determine the optimal set of projects to complete.

(a) Explain what the components of the minimum $s-t$ cut correspond to in the optimal solution.

(b) What is the purpose of the infinite capacity edges?

(c) Explain how to compute maximum profit from the capacity of the minimum $s-t$ cut.

Solution

Let's think about the structure of the graph. The source vertex connects to all of the profitable projects, while the unprofitable projects get connected to the sink vertex. Suppose that some particular profitable project x remains in the S component of the minimum cut. Then because of the infinite capacity edges connecting that project to its prerequisites, they too must be in the S component, since disconnecting them would cost ∞, which would not be a minimum cut. This means that we must pay the cost of disconnecting any unprofitable prerequisites from the sink vertex, which is the total cost of undertaking said unprofitable projects. Suppose instead that a profitable project is not in the S component. This means that the edge from the source to it must be disconnected, which costs the value of the project. In this case, it is not required to disconnect the unprofitable prerequisites since they are not connected to s directly, but only via their dependants. From these observations, we can deduce the following answers.

(a) Each project in the S component is a project that we will complete. Each project in the T component are the projects that we will not complete. This makes sense since any profitable project in S forces all of its prerequisites to also be in S.

(b) The infinite capacity edges ensure that the prerequisite relationships are satisfied. You cannot disconnected an infinite capacity edge since that would make the cost of the cut infinity, which will never be a minimum cut!

(c) From the discussion above, when we decide to complete a profitable project, we subsequently pay the cost of its unprofitable prerequisites by disconnecting them from t. Alternatively, when we decide to not complete a profitable project, we pay the value of that project, i.e. we pay the profit that we are choosing to give up. This implies that the capacity of the cut is

$$\begin{aligned} c(S,T) = &\text{ the cost of the unprofitable projects we are forced to do} \\ &+ \text{ the cost of the profitable projects we choose not to do} \end{aligned}$$

From this, it is easy to compute the maximum possible profit by noticing that

$$\text{Max Profit} = \text{Total profit of all profitable projects} - c(S,T),$$

which makes sense since we are seeking the **minimum** value of $c(S,T)$, which will therefore maximise the profit.

Problem 4. You work on a company that provides essential services and needs to have one employee on duty each Saturday and Sunday. You got the results of a Doodle showing, for each employee e_i, which specific Saturdays and Sundays they are available to work during the next couple of weekends. You also know the maximum amount m_i of weekend days that each employee e_i is allowed to work during that time. And for each employee e_i, your team has agreed that it should work at least ℓ_i of those days to achieve a fair distribution of the workload.

(a) How can you figure out if there is an allocation that has exactly one employee on duty each Saturday and each Sunday, and that meets the constraints above?

(b) Suppose that stronger employment regulations are been approved, and for each weekend, each employee will only be allowed to work at most one day. How can you solve this version of the problem?

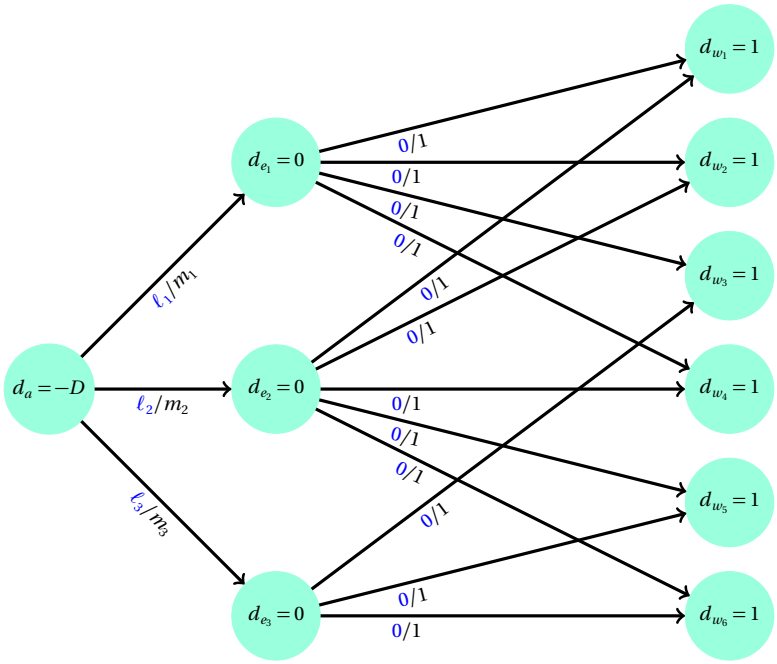
Solution

We will translate this to a circulation with demands and lower bounds problem.

(a) Let D denote the total number of weekend days during the timeframe considered. You shall create a graph in the following way:

- Add one vertex e_i for each employee (with demand 0).
- Add one vertex w_j for each Saturday and each Sunday (with demand 1). Those vertices will only have incoming edges, and therefore the demand of 1 will ensure that exactly one employee is allocated to each day in any feasible solution.
- For each weekend day w_j that an employee e_i is available, add an edge (e_i, w_j) with capacity 1 and lower bound 0.
- Add a vertex a with demand $-D$, which is the origin of all shift allocations.
- For each employee e_i, add an edge (a, e_i) with capacity m_i (the maximum number of weekend days that the employee can work during the timeframe) and lower bound ℓ_i (the minimum workload he should take to achieve a fair distribution).

See an example of a graph below.

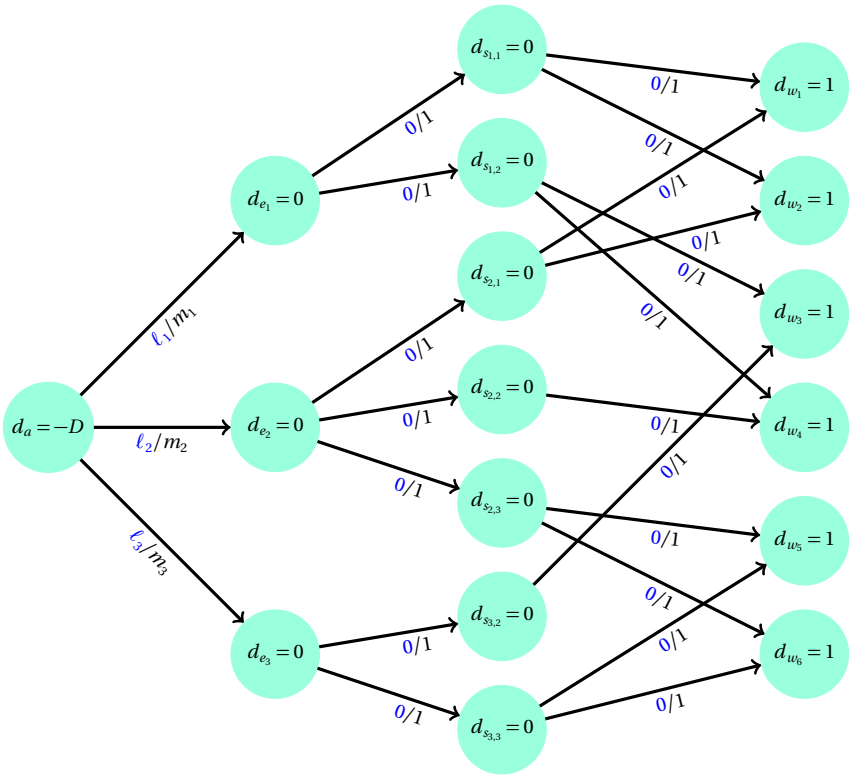


Then you just need to solve this circulation with demands and lower bounds problem to determine if there is a feasible solution or not. If there is a feasible solution, the allocation corresponds to the edges between employees and weekend days that have flow equal to 1.

Note that on one side of the graph we can put the demands of 1 directly into the nodes denoting the weekend days, while on the employee side we need to keep the demand of the nodes representing the employees as 0 and add the additional node a with demand $-D$. The reason for that is the fact that for each weekend day we have a precise number of shifts to be allocated per day (i..e, only 1). On the other hand, on the employee side we only have lower bounds and upper bounds on the number of shifts each employee can be allocated to, and only know that the total amount of shifts to be allocated across

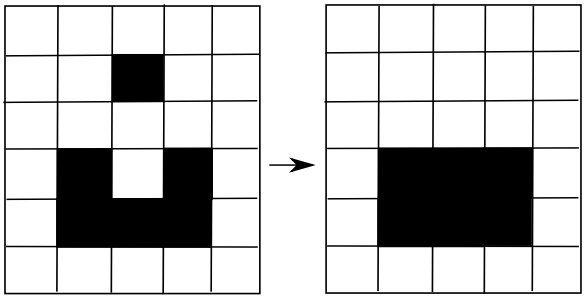
all employees is D. Therefore we need the additional node a with demand $-D$ and whose edges to the employees nodes capture the lower and upper bounds on the number of shifts each employee can take.

(b) In this case, you have to enforce the fact that no employee can work both w_1 and w_2 (the Saturday and Sunday of the first weekend), nor both w_3 and w_4, and so on... so you cannot straightforwardly connect the employee nodes to the weekend days nodes. Instead, the idea is to insert a selector node in between them. For each employee e_i, if he can work at least one day of the k-th weekend, then you create a node $s_{i,k}$ (with demand 0) and add an edge $(e_i, s_{i,k})$ with capacity 1 and lower bound 0 (this will enforce that the employee works at most one day of that weekend). Furthermore, for each day w_j in that k-th weekend that e_i can work, you add an edge $(s_{i,k}, w_j)$ with capacity 1 and lower bound 0. Then you just need to solve this circulation with demands and lower bounds problem to determine if there is a feasible solution or not. The adapted graph corresponding to the same example as before is shown below.



You can check that there is a path between each e_i and w_j in this adapted graph if, only if, there was a path in the previous graph. But now the selector nodes $s_{i,k}$ enforce the rule that no employee should work both days of a weekend.

Problem 5. You are the owner of a plot of land that can be described as an $n \times m$ grid of unit-square-sized cells. Each cell of land is either filled in, or a hole. The government has decided to crack down on safety regulations and requires you to fence off the holes in the ground. Each unit of fencing will cost you $\$c_{\text{fence}}$. In order to reduce the amount of fencing required, you have the option to fill in some of the holes, or even dig some additional holes. It will cost you $\$c_{\text{dig}}$ to dig a new hole, or $\$c_{\text{fill}}$ to fill in an existing hole. For safety, there are no holes on the boundary cells of your land, and you are not permitted to make any. For example, in the following case (where black represents a hole), with $c_{\text{fence}} = 10$, $c_{\text{dig}} = 10$, and $c_{\text{fill}} = 30$, the cost to fence the land initially would be \$160. The optimal solution is to dig out the middle cell and fill in the topmost hole, yielding a total cost of \$140.



Describe how to determine the minimum cost to make your land safe by reducing it to a minimum cut problem.

Solution

In order to model this as a minimum cut problem, we need to decide what the components of the cut will represent, i.e. what two things are we trying to separate? The answer is reasonable obvious, we are paying a cost to separate holes from non-holes, so the components of the cut will be the cells that end up as holes, and the cells that end up as non-holes. Two adjacent cells must pay a cost to be disconnected (the cost $\$c_{\text{fence}}$), or we can pay to change something from a hole to a non-hole or vice versa.

So, let's make a graph where each vertex represents one of the cells of land, and add two special vertices, the sink and source, s and t. Let's say that we want the S component to represent the non-holes, and the T component to represent the holes. First we will add edges from s to all of the boundary cells with capacity ∞ because we are not allowed to change these into holes. For the rest of the non-holes, we add a vertex from s to their respectively vertices with capacity c_{dig}, because it costs $\$c_{\text{dig}}$ to remove this cell from the non-holes component and make it a part of the holes component. Correspondingly for each cell that is currently a hole, we add an edge from its corresponding vertex to t with capacity c_{fill}, since it costs $\$c_{\text{fill}}$ to make this cell part of the non-holes component.

Finally, we must deal with the fencing cost. It costs $\$c_{\text{fence}}$ to have two adjacent cells be in different components, so for every pair of adjacent cells, we add edges in both directions with capacity c_{fence}. This ensures that if the two cells are in different components, we will have to pay $\$c_{\text{fence}}$ to keep them apart, but if they are in the same component, we pay nothing. Note that we must add the edge in both directions since we do not know in advance which cell will be in which component, but the cut will never double count the cost since the capacity of a cut only measures the edges going $S \to T$, but not those in the other direction.

The minimum cost to make your land safe will be the capacity of the minimum cut of this graph, i.e. the value of its maximum flow.

Problem 6. A useful application of maximum flow to people interested in sports is the *baseball elimination* problem. Consider a season of baseball in which some games have already been played, and the schedule for all of the remaining games is known. We wish to determine whether a particular team can possibly end up with the most wins or at least tied for the most wins. For example, consider the following stats.

	Wins	Games Left
Team 1	30	5
Team 2	28	10
Team 3	26	8
Team 4	20	9

It is trivial to determine that Team 4 has no chance of winning, since even if they win all 9 of their remaining games, they will be at least one game behind Team 1. However, things get more interesting if Team 4 can win enough games to reach the current top score.

	Wins	Games Left
Team 1	30	5
Team 2	28	10
Team 3	29	8
Team 4	20	11

In this case, Team 4 can reach 31 wins, but it doesn't matter since the other teams have enough games left that one of them must reach 32 wins. We can determine the answer with certainty if we know not just the number of games remaining, but the exact teams that will play in each of them. A complete schedule consists of the number of wins of each team, the number of games remaining, and for each remaining game, which team they will be playing against. An example schedule might look like the following.

		Games Remaining				
	Wins	Total	vs T1	vs T2	vs T3	vs T4
Team 1	29	5	0	2	1	2
Team 2	28	10	2	0	4	4
Team 3	28	8	1	4	0	3
Team 4	25	9	2	4	3	0

Describe an algorithm for determining whether a given team can possibly end up with the most wins or at least tied for the most wins.

Solution

Assume that Team x in question wins all of its remaining games since this is the best it can do, and let the resulting number of wins of Team x be denoted by y. In the following we consider only the other teams, and the remaining games that do not involve Team x.

Each remaining game will assign a point to one of the two teams that plays in it, so make a graph with games on the left and teams on the right. For efficiency reasons, we can merge duplicate games played between the same pair of teams. Consider any pair of teams, Team i and Team j, with $i < j$. If there are r games to be played between Team i and Team j, we put a single node $G_{i,j}$ in the left to represent those games. This node will have demand $-r$ (as the wins of those r games need to be assigned to the teams involved). $G_{i,j}$ is connected with edges of capacity r to the nodes T_i and T_j that represent Team i and Team j, respectively. So the points from those games (the flow) flow into one of the two teams that played it. The team nodes have demand 0.

We would like to prevent any Team i (with $i \neq x$) from getting more points than Team x, so we create a node A with demand equal to the total of remaining games that do not involve Team x. Each node T_i representing Team i is connected to A with an edge of capacity equal to y minus the current amount of wins of Team i.

If Team x can still finish at least tied for the most wins, then there is a feasible solution for this circulation with demands problem. Otherwise, it was not possible to assign all of the points and hence Team x cannot finish as the leader anymore.

Supplementary Problems

Problem 7. Implement the algorithm based on Ford-Fulkerson to solve the problem of circulation with demands and lower bounds.