CS 170 Homework 9

Due **Friday 11/1/2024, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Applications of Max-Flow Min-Cut

Review the statement of max-flow min-cut theorem and prove the following two statements.

(a) In class, we saw that the max-flow algorithm will sometimes cancel existing flow through certain edges; in other words, it is possible that sending a flow along some path may "undo" some existing flow along an edge.

Prove that, if BFS is used to find augmenting paths, this will never happen.

(b) Let $G = (L \cup R, E)$ be a unweighted bipartite graph[1]. Then G has a L-perfect matching (a matching[2] with size $|L|$) if and only if, for every set $X \subseteq L$, X is connected to at least $|X|$ vertices in R. You must prove both directions.

Hint: Use the max-flow min-cut theorem on the cut that forms X and $L \setminus X$.

Solution:

(a) This problem asks you to prove an incorrect statement, so we ended up cancelling it.

(b) The proposition is known as Hall's theorem. On one direction, assume G has a perfect matching, and consider a subset $X \subseteq L$. Every vertex in X is matched to distinct vertices in R, so in particular the neighborhood of X is of size at least $|X|$, since it contains the vertices matched to vertices in X.

On the other direction, assume that every subset $X \subseteq L$ is connected to at least $|X|$ vertices in R. Add two vertices s and t, and connect s to every vertex in L, and t to every vertex in R. Let each edge have capacity one. We will lower bound the size of any cut separating s and t. Let C be any cut, and let $L = X \cup Y$, where X is on the same side of the cut as s, and Y is on the other side. There is an edge from s to each vertex in Y, contributing at least $|Y|$ to the value of the cut. Now there are at least $|X|$ vertices in R that are connected to vertices in X. Each of these vertices is also connected to t, so regardless of which side of the cut they fall on, each vertex contributes one edge cut (either the edge to t, or the edge to a vertex in X, which is on the same side as s). Thus the cut has value at least $|X| + |Y| = |L|$, and by the max-flow min-cut theorem, this implies that the max-flow has value at least $|L|$, which implies that there must be a perfect matching.

[1] A bipartite graph $G = (L \cup R, E)$ is defined as a graph that can be partitioned into two disjoint sets of vertices (i.e. L and R) such that no two vertices within the same set are adjacent.

[2] A matching is defined as a set of edges that share no common vertices.

3 A Cohort of Secret Agents

A cohort of k secret agents residing in a certain country needs escape routes in case of an emergency. They will be travelling using the railway system which we can think of as a directed graph $G = (V, E)$ with V being the cities and E being the railways. Each secret agent i has a starting point $s_i \in V$, and all s_i's are distinct. Every secret agent needs to reach the consulate of a friendly nation; these consulates are in a known set of cities $T \subseteq V$. In order to move undetected, the secret agents agree that at most c of them should ever pass through any one city. Our goal is to find a set of paths, one for each of the secret agents (or detect that the requirements cannot be met).

Model this problem as a flow network. Specify the vertices, edges, and capacities; show that a maximum flow in your network can be transformed into an optimal solution for the original problem. You do not need to explain how to solve the max-flow instance itself.

Solution: We can think of each secret agent i as a unit of flow that we want to move from s_i to any vertex $\in T$. To do so, we can model the graph as a flow network by setting the capacity of each edge to ∞, adding a new vertex t and adding an edge (t', t) for each $t' \in T$. We can add a source s and edges of capacity 1 from s to s_i. By doing so, we restrict the maximum flow to be k.

Lastly, we need to ensure that no more than c secret agents are in a city. We want to add vertex capacities of c to each vertex. To implement it, we do the following: for each vertex v that we want add constraint to, we create 2 vertices v_{in} and v_{out}. v_{in} has all the incoming edges of v and v_{out} has all the outcoming edges. We also put a directed edge from v_{in} to v_{out} with edge capacity constraint c.

If the max flow is indeed k, then as every capacity is an integer, the Ford-Fulkerson algorithm for computing max flow will output an integral flow (one with all flows being integers). Therefore, we can incrementally starting at each secret agent i follow a path in the flow from s_i to any vertex in T. We iterate through all of secret agents to reach a solution.

4 Office Hour Scheduling

You're the new CS 170 head TA and need to assign TAs to host office hours. Unfortunately for you, the department has bizarre rules about when TAs can hold office hours:

- There are N office hours slots in total. You can assume that all N slots are consecutive.
- There are T TAs in total. The i^{th} TA is only available to hold office hours during the contiguous block from slot a_i to slot b_i. The numbers $\{a_i, b_i\}$ for $i \in \{1 \dots T\}$ are given as input.
- Each office hours slot must have exactly k TAs (no more, and no less).
- Not every TA needs to hold office hours.
- If a TA holds office hours, department rules require them to hold office hours for their entire availability, e.g. the i^{th} TA either holds office hours for every slot from a_i to b_i, or does not hold office hours at all.

Your goal is to determine which TAs to assign office hours so that all the above constraints are met.

Devise an efficient algorithm that solves the problem by constructing a directed graph G with a source s and sink t, and computing the maximum flow from s to t.

(a) Describe the nodes of the graph G. How many nodes does G have?

(b) Describe the edges of the graph G (draw a picture if needed). How many edges does G have?

(c) Given a maximum flow in the graph G, how can you determine which TAs to assign to hold office hours?

Solution:

(a) **Solution 1:** We create one vertex for each OH slot from $n = 1 \dots N$. We also create a start node s and an end node t, which we will respectively index as 0 and $N + 1$ for simplicity of the arguments below.

There are $N + 2$ nodes in this graph.

Solution 2: We create one vertex for each TA from $i = 1 \dots T$, plus s and t.

There are $T + 2$ nodes in this graph.

(b) **Solution 1:** Create an edge $u \to v$ for $u, v \in \{0, \dots, N\}$ if there is some TA who is available from slot $u + 1$ to slot v, and set its capacity to the number of TAs with this availability range. Note that we take vertex s to have index 0 for the purpose of these calculations. Additionally, add an edge from vertex N to the end node t with capacity k.

There are $T + 1$ edges in this graph.

Solution 2: Create an edge $i \to j$ with unit capacity if $b_i + 1 = a_j$. Connect s to all vertices i with $a_i = 0$, and those with $b_i = N$ to t.

There are $O(T^2)$ edges in this graph.

(c) **Solution 1:** first, check if the value of the max flow is k; if it's not, then it's impossible to assign TAs that satisfies all the constraints.

Otherwise, we can look at the residual graph and the flow (i.e. residual) along each edge (u, v) will correspond to the number of TAs we want to assign to the OH slots $u + 1$ to v. Based on this, we can assign TAs to office hour slots accordingly.

Solution 2: first, check if the value of the max flow is at least k; if it's not, then it's impossible to assign TAs that satisfies all the constraints.

Otherwise, let f^* be the value of the max-flow. We can decompose our maximum flow into a set of exactly f^* disjoint $s - t$ paths, each with flow exactly 1. We can choose k of these paths, and assign each vertex (i.e. TA) along each of the paths to hold OH.

Note: Solution 2 is preferable to the other solution because N is only given as a parameter (whereas we are given a list of T TAs), so the solution 1 is actually pseudo-polynomial time. Both will be given full points, however.

5 Weighted Rock-Paper-Scissors

You and your friend used to play rock-paper-scissors, and have the loser pay the winner 1 dollar. However, you then learned in CS170 that the best strategy is to pick each move uniformly at random, which took all the fun out of the game.

Your friend, trying to make the game interesting again, suggests playing the following variant: If you win by beating rock with paper, you get 2 dollars from your opponent. If you win by beating scissors with rock, you get 1 dollars. If you win by beating paper with scissors, you get 4 dollar.

For this problem, you are allowed to use code or online tools to automatically solve your LPs. The coding problem (Q6) walks you through how to express your LPs in `Python` and solve them using a `Python` library. However, if you prefer, feel free to use a different LP solver such as: `https://online-optimizer.appspot.com/`.

(a) Draw the payoff matrix for this game. Assume that you are the maximizer, and your friend is the minimizer.

(b) Write an LP to find the optimal strategy in your perspective. You do not need to solve the LP.

The following subparts are independent of the previous subparts.

Your friend now wants to make the game even more interesting and suggests that you assign points based on the following payoff matrix:

		Your friend: rock	paper	scissors
You:	rock	-10	3	3
	paper	4	-1	-3
	scissors	6	-9	2

(c) Write an LP to find the optimal strategy for yourself. What is the optimal strategy and expected payoff?

Feel free to use your code from Q6 or an online LP solver.

(d) Now do the same for your friend. What is the optimal strategy and expected payoff? How does the expected payoff compare to the answer you get in part (c)?

Solution:

(a)

		Your Friend: rock	paper	scissors
You:	rock	0	-2	1
	paper	2	0	-4
	scissors	-1	4	0

(b) Let r, p, s be the probabilities that you play rock, paper, scissors respectively. Let z stand for the expected payoff, if your opponent plays optimally as well.

$$\begin{aligned} \max \quad & z & \\ 2p - s &\geq z & \text{(Opponent chooses rock)} \\ 4s - 2r &\geq z & \text{(Opponent chooses paper)} \\ r - 4p &\geq z & \text{(Opponent chooses scissors)} \\ r + p + s &= 1 & \\ r, p, s &\geq 0 & \end{aligned}$$

(c)

$$\begin{aligned} \max \quad & z & \\ -10r + 4p + 6s &\geq z & \text{(Opponent chooses rock)} \\ 3r - p - 9s &\geq z & \text{(Opponent chooses paper)} \\ 3r - 3p + 2s &\geq z & \text{(Opponent chooses scissors)} \\ r + p + s &= 1 & \\ r, p, s &\geq 0 & \end{aligned}$$

The optimal strategy is $r = 0.3346,\ p = 0.5630,\ s = 0.1024$ for an optimal payoff of -0.48.

See the coding solution for how to express and solve this LP in Python.

(d)

$$\begin{aligned} \min \quad & z & \\ -10r + 3p + 3s &\leq z & \text{(You choose rock)} \\ 4r - p - 3s &\leq z & \text{(You choose paper)} \\ 6r - 9p + 2s &\leq z & \text{(You choose scissors)} \\ r + p + s &= 1 & \\ r, p, s &\geq 0 & \end{aligned}$$

Your friend's optimal strategy is $r = 0.2677,\ p = 0.3228,\ s = 0.4094$. The value for this is -0.48, which is the payoff for you. The payoff for your friend is the negative of your payoff, i.e. 0.48, since the game is zero-sum.

See the coding solution for how to express and solve this LP in Python.

(Note for grading: Equivalent LPs are of course fine. It is fine for part (d) to maximize your friend's payoff instead of minimizing yours. For the strategies, fractions or decimals close to the solutions are fine, as long as the LP is correct.)

6 [Coding] LP / Max Flow / Min Cut

For this week's homework, we'll walk you through solving LPs in Python, and then you'll implement an algorithm to compute the maximum flow in a network and then compute the minimum cut from the flow. There are two ways that you can access the notebook and complete the problems:

1. **On Datahub**: click here and navigate to the `hw09` folder.
2. **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   `https://github.com/Berkeley-CS170/cs170-fa24-coding`

   and navigate to the `hw09` folder. Refer to the `README.md` for local setup instructions.

Notes:

- *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "Homework 9 Coding Portion".
- *Getting Help:* Conceptual questions are always welcome on Edstem and office hours; *note that support for debugging help during OH will be limited.* If you need debugging help first try asking on the public Edstem threads. To ensure others can help you, make sure to:

  1. Describe the steps you've taken to debug the issue prior to posting on Ed.
  2. Describe the specific error you're running into.
  3. Include a few small but nontrivial test cases, alongside both the output you expected to receive and your function's actual output.

  If staff tells you to make a private Ed post, make sure to include *all of the above items* plus your full function implementation. If you don't provide them, we will ask you to provide them.
- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.

