CMS/CS/EE/IDS 144 – HW 5

Marco Yang

1. **1 Discovering Clusters [40 points]**

   In class, you have seen the power of clustering on many real world applications. The choice of clustering algorithm depends on both the structures of graphs and the metrics used for clustering, among other things. In this problem, you will evaluate various clustering algorithms on a couple of different types of graphs: symmetric stochastic block model (SSBM), as seen in Homework 2, and the "circle graph", to get a sense for the differences between algorithms.

   Your task:

   link to code, also at end of file

   (a) **[20 points]** Generate a SSBM network with $n = 30$, $k = 3$, $A = 0.7$ and $B = 0.1$. Run the spectral clustering and k-means clustering algorithms over the network, for appropriate value(s) of k, and visualize the clusters obtained from each algorithm. You may find Python libraries such as `sklearn.cluster.KMeans` and `sklearn.cluster.SpectralClustering` useful for this question.

   Reminder.

   The stochastic block model (SBM) is a random graph model with planted clusters. The model $\mathrm{SBM}(n, p, W)$ defines an n-vertex random graph with labeled vertices for positive integers n, k, a probability vector p of dimension k, and a symmetric matrix W of dimension $k \times k$ with entries $W_{i,j} \in [0, 1]$. Each vertex is assigned a community label in $\{1, \ldots, k\}$ independently under the community prior p, and pairs of vertices with community labels i and j connect independently with probability $W_{i,j}$. The SBM is called symmetric if p is uniform and if W takes the same value A on the diagonal and the same value B outside the diagonal. We then represent the symmetric SBM as $\mathrm{SSBM}(n, k, A, B)$.

   Turn in a visualization of the results of your algorithms and a discussion of any differences you notice between the two algorithms. You may base this explanation on the visualization or some specific clustering metrics of your choice. Also, discuss how well the clusters were recovered by these algorithms by comparing them to the actual clusters that were used to form the SSBM instance you generated.

   Solution:

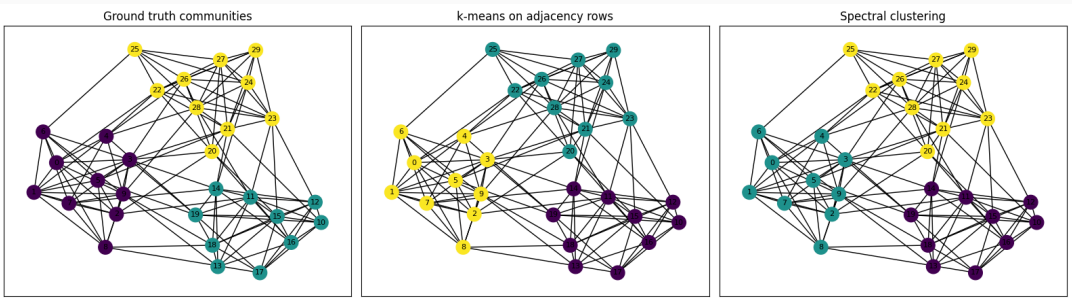
   

   Figure 1: SSBM graph 1 and clusters.

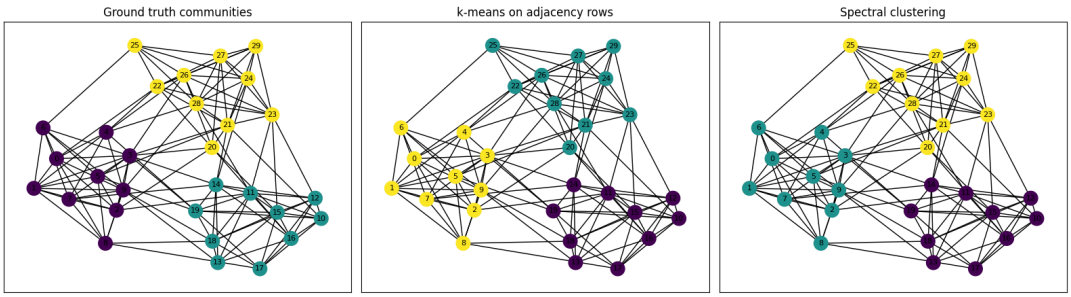


Figure 2: SSBM graph 2 and clusters.

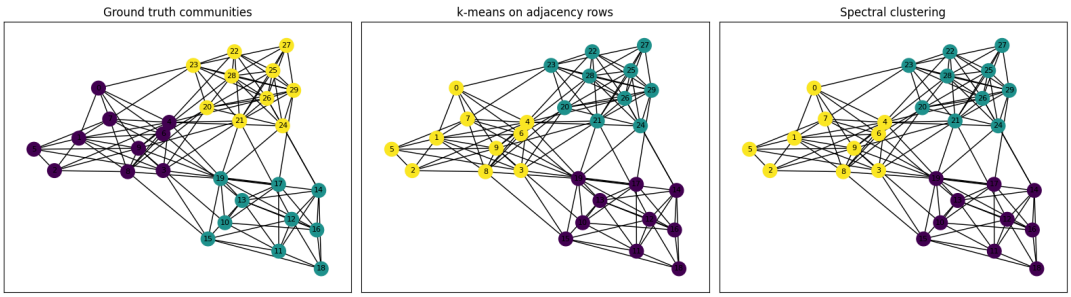


Figure 3: SSBM graph 3 and clusters.

From the above graphs, it's pretty obvious that for the given parameters, both clusters work just fine when given the correct number of clusters, so I decided to change up the number of clusters:

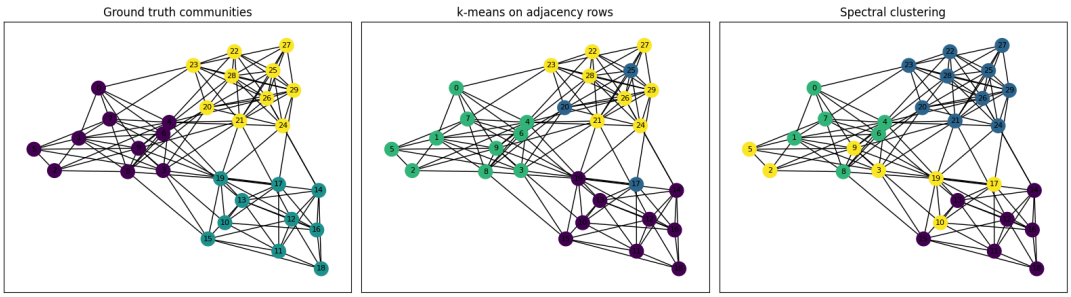


Figure 4: SSBM graph 3 and 4 clusters.

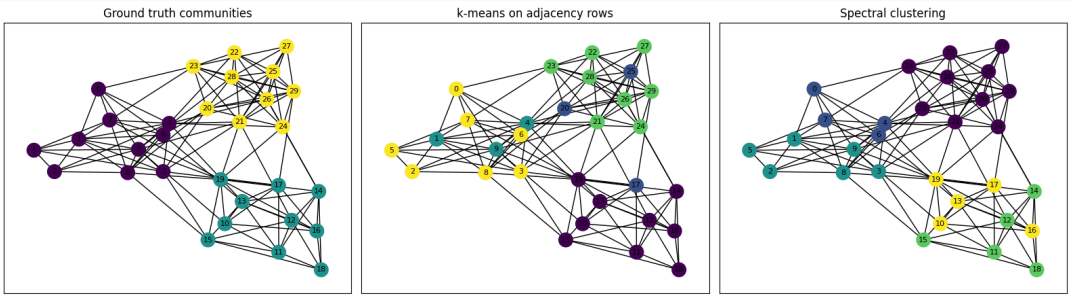


Figure 5: SSBM graph 3 and 4 clusters.

Clearly, the algorithms were doing something different in how they broke apart the correct clusters to form new ones. It kinda looked like K-means was better, but I couldn't give a definitive verdict just based on visual vibes. Google/ChatGPT said that the best metric to judge this was adjusted random index, which "is a robust metric for evaluating

the similarity between two data clusterings by accounting for chance agreement", so I ran it for $k = 2, ...10$ clusters for each algorithm. The output was:

```
k-means ARI: 0.554, Spectral ARI: 0.554 for 2 clusters
k-means ARI: 1.000, Spectral ARI: 1.000 for 3 clusters
k-means ARI: 0.848, Spectral ARI: 0.672 for 4 clusters
k-means ARI: 0.717, Spectral ARI: 0.708 for 5 clusters
k-means ARI: 0.643, Spectral ARI: 0.551 for 6 clusters
k-means ARI: 0.547, Spectral ARI: 0.556 for 7 clusters
k-means ARI: 0.464, Spectral ARI: 0.315 for 8 clusters
k-means ARI: 0.416, Spectral ARI: 0.290 for 9 clusters
k-means ARI: 0.351, Spectral ARI: 0.186 for 10 clusters
Average k-means ARI: 0.615, Average spectral ARI: 0.537
```

So I think K-means does a little better when given the wrong number of clusters.

(b) **[20 points]** Generate an instance of the "circle graph", as seen below in Figure 1, with number of nodes $n = 500$, and noise $\sigma = 0.01$, using the `sklearn.datasets.make_circles` function; half of the n nodes will end up in the inner circle, and the other half in the outer circle. Run the spectral clustering and k-means clustering algorithms over the network, for appropriate value(s) of k, and visualize the clusters obtained from each algorithm. Discuss any differences you notice between the two algorithms, and why any differences might occur. You will again find the aforementioned Python libraries useful for this problem.

Note 1.

The two clustering algorithms differ in that spectral clustering conventionally operates on a graph $G = (V, E)$ represented using an adjacency matrix (which is how the SSBM graph is represented), whereas k-means conventionally operates on a graph whose nodes are points in a vector space (which is how the circle graph is represented: a set of (x, y) coordinate pairs). Therefore, in order to run spectral clustering on the circle graph, in addition to passing in the matrix of data points, you will need to pass in an "affinity" function, which essentially defines a metric by which the algorithm can build an adjacency graph from the points in the vector space; we suggest the `nearest_neighbors` affinity function, which creates an adjacency matrix with edges between nodes that are close to each other in the vector space. When running k-means on the SSBM graph, you can simply pass in the adjacency matrix as is, or transform it as you see fit; regardless of which one you choose, as part of your discussion, talk about how the k-means algorithm is interpreting the data points you pass in to generate its clustering. Running k-means on the circle graph and running spectral clustering on the SSBM graph should follow naturally from the conventional definitions of the two algorithms.

Solution:

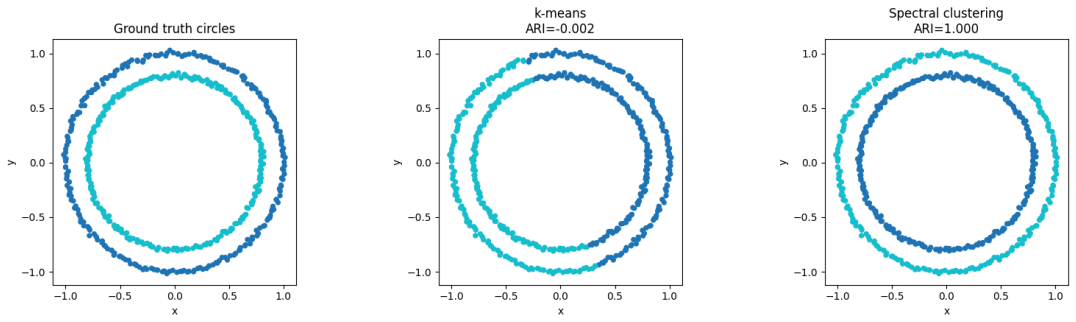


Figure 6: Circle graph and clusters.

One graph was enough here. Spectral clustering correctly recovers the clusters, but K-means doesn't. K-means sucks here because it used the raw x, y values, which are obviously not equipped to handle concentric circles that are close together since raw distance isn't good metric of which circle it's on. Spectral clustering has an unfair advantage since we used the nearest_neighbors affinity, which gave it information on which nodes were closest to which ones. Since the nearest nodes for any node are obviously going to be on the same circle, the spectral clustering on the graph laplacian is able to correctly cluster ndoes on the same circle together.

2. **2 Braess's Paradox Simulation [30 points]**

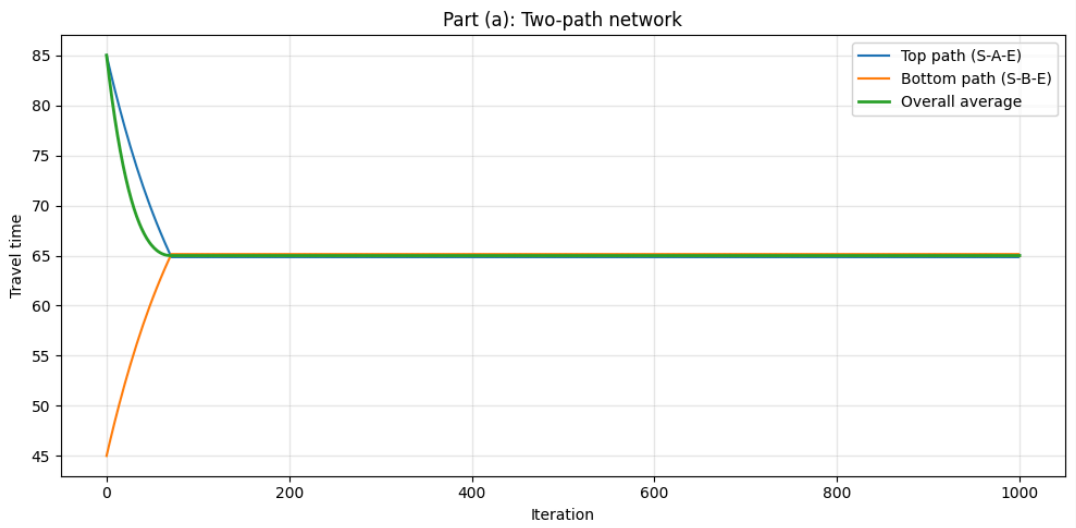
In game theory and traffic assignment, Braess's Paradox is the counter-intuitive observation that adding road capacity to a network can sometimes increase the overall travel time for everyone. This happens because individual drivers act selfishly to minimize their own travel time, reaching a Nash Equilibrium that is not necessarily the social optimum. The new road might offer a shortcut that looks attractive to individuals, but when everyone tries to use it, the resulting congestion slows the whole system down. In this problem, you will simulate this phenomenon to see how selfish routing decisions lead to suboptimal outcomes.

Your task: Simulate 4000 drivers traveling from S to E.

link to code, also at end of file

(a) **[10 points]** Initialize all drivers on the top $(S \to A \to E)$ path. Iteratively move 1% of the drivers currently on the slowest path to the fastest path; do this 1,000 times. Assume drivers act greedily, are rational, and make their decisions in each iteration simultaneously. Generate a plot showing average travel time for each path and overall v. iteration number. Report the final number of drivers on each path and the average travel time. Does this appear to be an equilibrium?

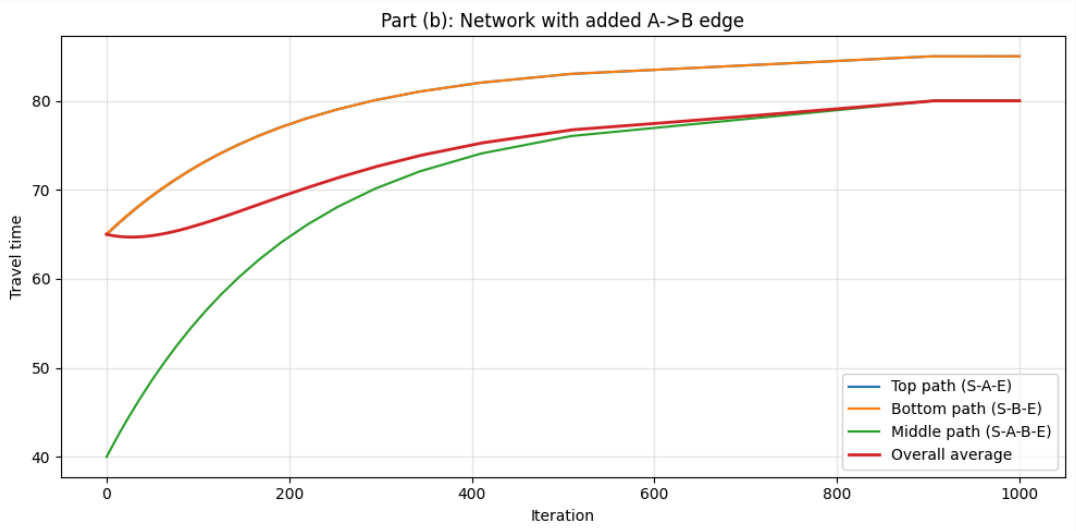
Solution:



The final number of drivers is 2004 and 1996 in the top and bottom, and the average travel times are 65.04 and 54.96. It's not an equilibrium because it is objectively better for the people in the top to switch to the bottom.

(b) **[10 points]** Add the zero-cost edge $A \to B$, creating a third path $(S \to A \to B \to E)$. Continuing from the final distribution of drivers between the top and bottom paths from the previous part, now allow drivers to switch to any of the three paths. Generate a plot showing average travel time for each path and overall v. iteration number. Report the new average travel time and the final number of drivers on each path. Does this appear to be an equilibrium? Did the average travel time increase or decrease?

Solution:



Everyone chose the middle path, and the time is 85 for each graph. It is an equilibrium. The average travel time increased.

(c) **[10 points]** Explain intuitively why rational drivers flocked to the new path despite it worsening the global outcome. Why don't they switch back?

Solution: The drivers go to the new path because they consider their own time selfishly. Their reduction of their own time compounds into an addition of more time for many other drivers because the shared edges proportional to the flow get more traffic, and it worsens the global outcome. They don't switch back because once again, once again, switching doesn't help them individually.

3. **3 Eavesdropping Centrality [15 points]**

For this problem, you will study a new notion of centrality. The idea behind the measure is that the "importance" of a node in a communication network is, in some sense, tied to the desire of an eavesdropper to monitor network communication by "bugging" that node. If the node would be really important for an eavesdropper to bug then the node is "central" in some sense.

Concretely, consider the following setting.

- **Communication Process:** Between time 0 and time 1, each vertex sends a (different) message to each of its neighbors independently at a uniformly random time, i.e., for each edge (i, j), T_{ij} is the time i sends a message to j and is distributed between $[0, 1)$ independently of all other edges.
- **Eavesdropping Process:** At any time between time 0 and time 1 the eavesdropper can "bug" any set of nodes in the network and intercept the incoming messages. But, the probability the eavesdropper is caught by node i at the end of the round is t_i^β where t_i is the total time spent at node i during the round and β is a parameter. The events of being caught at node i and node j are not independent.

The policy of the eavesdropper is $t = (t_1, ..., t_n)$, which specifies how much time each node is monitored and the goal of the eavesdropper is to maximize the expected number of messages intercepted during the round while making sure that the probability of being caught is smaller than γ, where $\gamma \in (0, 1)$.

Intuitively, the importance of a node is proportional to the amount of time it is bugged by a strategic eavesdropper.

(a) **[5 points]** Given policy $t = (t_1, ..., t_n)$, show that the expected number of intercepted messages is $\sum_{i=1}^n t_i d_i$ where d_i is the degree of node i.

Solution: For every node i, the probability that any of its incoming edges had its message while the eavesdropper was on it is the duration of the eavesdropper divided by the total time frame the message on that edge could have been sent, which is $\frac{t_i}{1} = t_i$. That means if a node has d_i neighbors, the expected number of intercepted messages while on that node is $t_i d_i$. Summing over all nodes, the expected number of intercepted messages is

$$\mathbb{E}[m] = \sum_i^n t_i d_i.$$

(b) **[5 points]** Prove that if $\sum_{i=1}^n t_i^\beta \leq \gamma$, then the probability of being caught is at most γ. This means that such a $t = (t_1, ..., t_n)$ is a valid policy.

Solution: Since the probability of being caught at two nodes i and j are not independent, by union bound,

$$p(\text{caught}) \leq \sum_{i=1}^{n} t_i^{\beta} \leq \gamma.$$

(c) **[5 points]** Given parts (a) and (b), we obtain an optimization problem that can be solved to determine the vector of eavesdropper centrality, t^*:

$$t^* = \arg\max \sum_{i=1}^{n} t_i d_i \text{ subject to} \sum_{i=1}^{n} t_i^{\beta} \leq \gamma.$$

Note that this problem uses a bound on the probability of being caught. We will ignore that fact for this homework, but please convince yourself why that is okay! To that end, consider two specific cases: $\beta = 1$ and $\beta = 2$. For each of these cases, solve the optimization problem and interpret the resulting t^* as a centrality measure. (Hint: The Cauchy-Schwarz inequality may be useful.)

Solution: For $\beta = 1$,

$$\sum_{i=1}^{n} t_i \leq \gamma.$$

To maximize $\sum_{i=1}^{n} t_i d_i$, it makes sense to simply allocate all of our time onto the highest degree node. Thus,

$$t^* = (t_1 = 0, t_2 = 0, ..., t_i = \gamma, ..., t_n = 0) \text{ for } i = \text{argmax}_{i \in [n]} d_i$$

For $\beta = 2$, we have the problem

$$\max \sum_{i=1}^{n} t_i d_i \text{ subject to} \sum_{i=}^{n} t_i^2 \leq \gamma.$$

The Cauchy-Schwarz inequality states that

$$\left(\sum_{i=1}^{n} t_i d_i \right)^2 \leq \left(\sum_{i=1}^{n} t_i^2 \right) \left(\sum_{i=1}^{n} d_i^2 \right)$$

$$\sum_{i=1}^{n} t_i d_i \leq \sqrt{\gamma \sum_{i=1}^{n} d_i^2}.$$

The right hand side is just a constant. We know that the dot product of $\boldsymbol{t} = (t_1, ..., t_n)$ and $\boldsymbol{d} = (d_1, ..., d_n)$ is maximized when the vectors are scalar multiples, so

$$t_i = \sqrt{\gamma} \frac{d_i}{\sqrt{\sum_{i=1}^{n} d_i^2}}.$$

4. **4 Game Theory Warm Up [10 points]**

This question is designed to put in use the concepts you learned in the game theory refresher lecture. Remember that for each element in the table, (x, y) represents a payoff of x for Player 1 and y for Player 2.

Payoff matrix:

		Player 2			
		a	b	c	d
Player 1	w	(3, 2)	(4, 1)	(4, 0)	(2, 1)
	x	(2, 4)	(5, 3)	(5, 5)	(0, 3)
	y	(2, 1)	(3, 3)	(3, 0)	(0, 6)
	z	(1, 3)	(4, 6)	(5, 4)	(3, 1)

(a) **[5 points]** Which actions survive the iterated elimination of dominated actions? Explain your claim.

Solution: In our first iteration, we find that w strictly dominates y for player 1 since the payout is always better across every column for row w compared to row y. Thus, we can remove row y from the table:

		Player 2			
		a	b	c	d
Player 1	w	(3, 2)	(4, 1)	(4, 0)	(2, 1)
	x	(2, 4)	(5, 3)	(5, 5)	(0, 3)
	z	(1, 3)	(4, 6)	(5, 4)	(3, 1)

Now, we have that action a strictly dominates d, so we can remove it:

		Player 2		
		a	b	c
Player 1	w	(3, 2)	(4, 1)	(4, 0)
	x	(2, 4)	(5, 3)	(5, 5)
	z	(1, 3)	(4, 6)	(5, 4)

There is no strict domination anymore, so we are left with actions w, x, z for player 1 and a, b, c for player 2.

(b) **[5 points]** Find all Nash Equilibria (Hint: you should find 2 pure and 2 mixed).

Solution: Looking at the payoff matrix, we can see that (w, a) and (x, c) are pure NE since there are no better options for either player given the other plays the given action.

Using the above pure NE, we can guess that one of the mixed NE involves those 4 actions. Let the probability of player 1 picking w be p and picking x be $1 - p$. Then,

$$\mathbb{E}[U_2(a)] = p \cdot 2 + (1-p) \cdot 4$$
$$\mathbb{E}[U_2(c)] = p \cdot 0 + (1-p) \cdot 5$$
$$\mathbb{E}[U_2(a)] = \mathbb{E}[U_2(c)] \Rightarrow 4 - 2p = 5 - 5p$$
$$p = \frac{1}{3}.$$

Thus, player 1 should mix between w, x with probability $\frac{1}{3}$ to ensure player 2's indifference. Let the probability of player 2 choosing a be q and choosing c be $1 - q$. Then,

$$\mathbb{E}[U_1(w)] = q \cdot 3 + (1-q) \cdot 4$$
$$\mathbb{E}[U_1(x)] = q \cdot 2 + (1-q) \cdot 5$$
$$\mathbb{E}[U_1(w)] = \mathbb{E}[U_1(x)] \Rightarrow 4 - q = 5 - 3q$$
$$q = \frac{1}{2}.$$

Thus, player 2 should mix between a and c with probability $\frac{1}{2}$ to ensure player 1's indifference. Together, our first mixed NE is player 1 mixing w, x with probabilities $\left(\frac{1}{3}, \frac{2}{3}\right)$ and player 2 mixing a, c with probabilities $\left(\frac{1}{2}, \frac{1}{2}\right)$.

To guess the second mixed NE, notice that if player 2 picks c, then player 1 is already indifferent between x and z. Now, we can solve for what strategy player 1 should take so that player 2 always chooses c over a and b to maintain the equilibrium. Let the probability of choosing x be p and choosing z be $1 - p$. Then,

$$\begin{aligned} U_2(x, c) &= 5p + 4(1-p) \\ &= p + 4U_2(x, a) = 4p + 3(1-p) \\ &= p + 3U_2(x, b) = 3p + 6(1-p) \\ &= 6 - 3p. \end{aligned}$$

Notice that player 2 always prefers c to a, and to prefer c over b,

$$p + 4 \geq 6 - 3p \Rightarrow p \geq \frac{1}{2}.$$

Thus, there is a mixed equilibrium when player 1 plays x with some probability $p \geq \frac{1}{2}$ and z with probability $q \leq \frac{1}{2}$ and when player 2 picks c always.

5. **5 Routing Games with Tolls [5 points]**

   In our discussion of routing games in class, we talked about using tolls to induce independent selfish players to behave in a manner that is optimal for the system. To this end, we discussed the principle of marginal cost pricing. In this exercise, you will prove that marginal cost pricing really works.

Consider the model of routing games described in class. Assume that each user i has traffic $r_i = 1$. We add a "toll" to the original link cost functions $c_{e(.)}$, setting the new cost function to be

$$c_e^*(x) = c_e(x) + (x-1)(c_e(x) - c_e(x-1)).$$

Notice that the "toll" on link e captures the net latency that each player using the link adds to the other players using it.

Your Task: Prove that any optimal strategy for our original game is a Nash equilibrium for this new game (with link cost functions $c_e^*(.)$). Recall that the objective function in the original game is $\sum_{i \in N} \sum_{e \in E_i} c_e(f_e) = \sum_{e \in E} f_e c_e(f_e)$ where f_e is the flow on e, N is the set of players and E_i contains the edges on the path selected by player i. (Hint: Think about constructing a new potential function with the new cost.)

Solution: