Homework Assignment 1

Fall 2022

IE 532 Analysis of Network Data

Due on Friday 09/23 (at 11:59 pm)

There are 4 questions in this assignment.

Please answer every question to the best of your knowledge and make sure to show your work for partial credit.

Make sure to cite your sources, if you decide to use more material to help you solve the exercises.

Remember that collaboration between students is allowed and encouraged, but please give your collaborators proper reference by letting me know who you worked with and what they contributed.

Only electronic submissions are accepted for this assignment.

You may either type your answers using LaTeX or other word processing software, or scan your handwritten answers and submit them. If the latter, please make sure the scanned copy is legible.

If submitting multiple files, create a folder and compress it (in either .zip, .rar, or .7z format, among others) before submitting.

Question 1: Eulerian paths (revisited)

Recall the DNA sequencing problem we discussed during class and the Eulerian path problem that was associated with it. A bioinformatics group, motivated by the same lecture material, decided to put this to the test. Alas, the graph they ended up with is **not** Eulerian. They believe the issue to be with their DNA reads, which ended up being noisy and hence they lost exactly one of the subsequences.

(a) First of all, had their graph been Eulerian, how could they solve the Eulerian path problem and obtain a valid DNA sequence? Using networkx and Python, find the Eulerian path using the following set of subsequences of $k = 3$:

$$s = \{CAT, ATT, ATG, TAT, TTG, TGC, TGT, GTA\}.$$

Construct the resultant graph and then identify an Eulerian path on it.

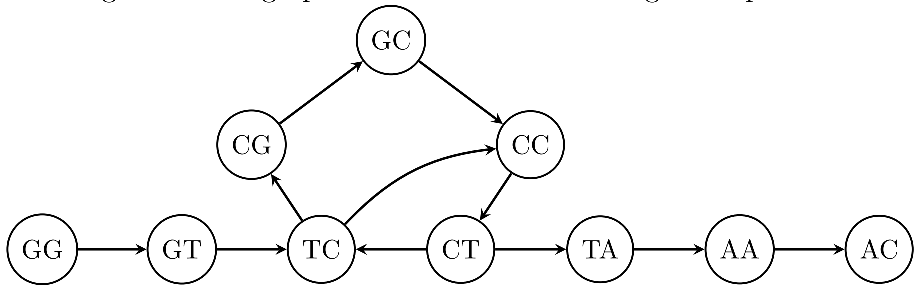
As a hint: use the **nx.eulerian_circuit(graph, source)** function that networkx provides us with to find an Eulerian cycle.

(b) As mentioned originally, the graph the group obtained was not Eulerian. They attributed that to noisy data and they would like to somehow fix it. **Recommend a way for them to render the obtained graph Eulerian by adding the necessary nodes and edges.** Put your algorithm to use with the following set of subsequences of $k = 3$, s:

$$s = \{GGT, GTC, TCC, CCT, CTC, TCG, CGC, GCC, CTA, TAA, AAC\}.$$

The constructed network would be as in Figure 1:

Figure 1: The graph created with one missing subsequence.



Answer (a) A Jupyter notebook containing a possible way to code the eulerian path problem for DNA sequencing is provided in Q1.ipynb.

(b) In Question 1 part (b), we are asked to propose a way to render the graph Eulerian. The pseudocode is pretty straightforward.

First, identify all the imbalances; that is, except for a starting node and an ending node (which should have one more outgoing and one more incoming edge) all the other nodes have to be balanced (outgoing edges equal to the incoming ones). To identify the imbalances, first remove the starting and ending node of the path, and then assign a number α_i to each node equal to the number of incoming edges minus the number of outgoing edges.

If $\sum_i \alpha_i = 0$, then we can first try connecting with an edge the nodes with $\alpha_i > 0$ to the nodes with $\alpha_i < 0$. In the case of the DNA sequencing example, we'd also need to check the condition (that the two nodes can actually be adjacent (i.e., the ending nucleotides of one match the starting nucleotides of the other). After the procedure, if all $\alpha_i = 0$, we are done.

If not done, create a new fake node and connect it to all the imbalances. How? Add a link from some node with $\alpha_i > 0$ and another link towards some node with $\alpha_i < 0$. If need be, create new fake nodes and continue. This process guarantees that all $\alpha_i = 0$ in the end (including of the fake nodes).

In the given example, a possible solution is to add the "missing" subsequence of **ACT**, which would add an edge between nodes AC and CT. Were we to do this, we would obtain at least one Eulerian path starting from node GG and terminating at node CC.

Question 2: Matching problems

(a) Assume you are given an $N \times N$ chessboard with some missing places (call them "holes"). We want to place as many rooks as possible on the chessboard: rooks can be located at any of the $N \times N$ locations that are not "holes". However, we need to make sure that no two rooks can attack each other. Assume that two rooks attack each other if they are placed on the same row or column, regardless of whether there are "holes" between them or not. Figure 2 shows an example of a valid placement of rooks.

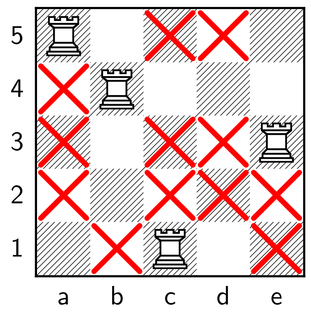


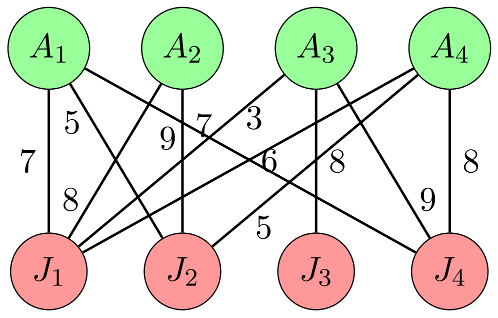
Figure 2: A valid assignment of four rooks in a 5×5 chessboard with 12 "holes" (marked with a red "X". Could we do better?

Set this problem up as a maximum matching problem. Provide a solution approach for solving it (based on the algorithm shown in class for solving maximum matching problems).

(b) Consider an assignment problem of n positions to n applicants. Each assignment comes with a cost $c_{ij} > 0$. We want to create a *stable and minimum cost* matching of positions to applicants with the extra restriction that no assignments is costlier than $\tilde{C}$. Provide a solution approach (could be a heuristic approach) for solving this problem, adapting the algorithm shown in class for solving maximum matching problems as well as Gale-Shapley's algorithm with preferences and the Hungarian algorithm for assignment problems.

To showcase how your algorithm behaves, you may use the following small instance. Consider a 4×4 assignment with costs equal to the values shown below and assume we pick $\tilde{C} = 9$.

	J_1	J_2	J_3	J_4
A_1	7	5	15	9
A_2	8	7	10	12
A_3	3	10	8	9
A_4	6	5	12	8



Answer (a) Assume we have two sets that need to be matched to one another: a set of n rows $I = \{1, 2, \dots\}$ and a set of n columns $J = \{\text{a,b,}\dots\}$. In the case of Figure 2, we would have $I = \{1, 2, 3, 4, 5\}$ and $J = \{a, b, c, d, e\}$. This is a **matching problem**, where we would like to try to have n rooks placed.

We begin by using a greedy placement algorithm. First, scan the first row and assign a rook to the first available position. We continue like that by adhering to one extra restriction; that the rook cannot be placed in the same column as a previously placed one. When we are done, we have a valid assignment; not necessarily maximal.

After that, proceed to find an augmenting path and follow exactly the augmenting path algorithm. When there are no more augmenting paths available, we have a maximum matching.

(b) First, remove all arcs whose costs are higher than $\tilde{C}$. This now leaves us with a sparser graph (instead of the complete graph of the start, where all edges between applicants $i \in I$ and positions $j \in J$ exist).

In this graph, begin by sorting the costs from lowest to highest. Then, take the first applicant and have them be considered for the position of smallest cost (Gale-Shapley algorithm); continue with all applicants. In the end, the positions would pick the best match among the ones being considered at the moment. If the matching is perfect, stop; this is a minimum cost matching.

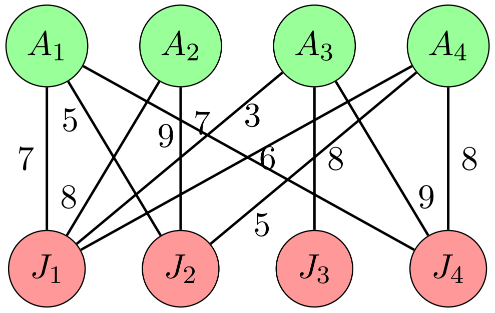
At this point, we quit Gale-Shapley. Now, define modified costs $\hat{c}$ as follows: for every node from I (applicants) find the smallest cost (the one they attempted to connect to during the first part) and subtract it from c. Hence, you will introduce at least one zero cost from each node $i \in I$ towards $j \in J$.

In the remaining modified costs (the ones that are not zero already) pick the lowest one; subtract that from the modified cost. Create a graph with **only the edges** that have a zero modified cost: call this graph G_ℓ. Is there an augmenting path in this graph G_ℓ? If so, do that iteration (increase the matching) and check whether the matching is perfect. If it is, stop. Otherwise, we re-do this step: in the remaining costs pick the lowest one. Subtract it from the modified costs; and continue like that until we have a perfect matching. We can only stop when either no more edges can be added or a perfect matching has been found.

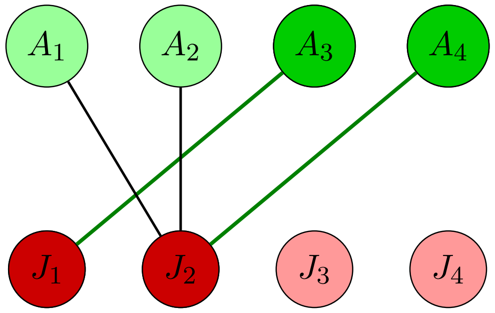
If no perfect matching exists after doing this operation (which will terminate after a finite number of steps as there is a finite number of possible edges to consider for adding in G_ℓ), this implies that there is no valid assignment due to the selection of $\tilde{C}$. An example follows.

Example: Consider a 4×4 assignment with costs equal to the values shown below and assume we pick $\tilde{C} = 9$.

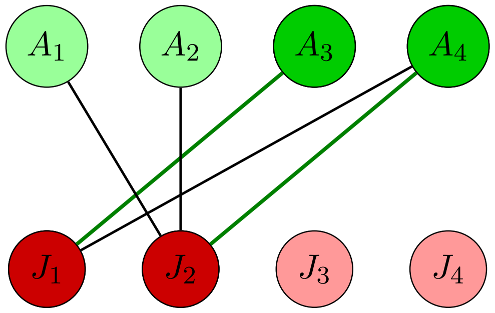
	J_1	J_2	J_3	J_4
A_1	7	5	15	9
A_2	8	7	10	12
A_3	3	10	8	9
A_4	6	5	12	8



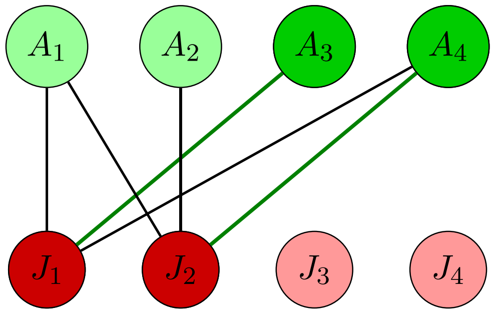
Based on Gale-Shapley, first A_1 tries to connect to J_2; A_2 tries to connect to J_2; A_3 tries to connect to J_1; A_4 tries to connect to J_2. J_2 will pick A_4 (cheapest cost) and J_1 picks A_3 (only option). Then, set the initial labels for all green nodes as the cheapest cost they are assigned to (if assigned) or the cheapest cost they have access to (if not assigned). Here is the first G_ℓ. The darker nodes are assigned; the lighter ones are still unassigned. The green edges are facilitating the assignment; the other ones are not participating in any assignment.



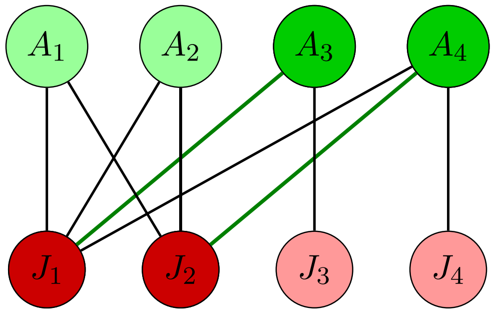
No augmenting path exists, so we try to improve the graph by adding more edges. We do that by finding the smallest cost edge(s) and adding it (them) to the graph. The smallest edges that is not added has a cost of 6. There is only one with this cost: (A_4, J_1).



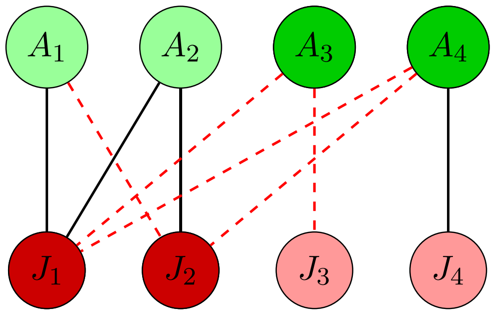
Still no augmenting path. The next smallest cost is equal to 7. Let's add this edge here too: (A_1, J_1).



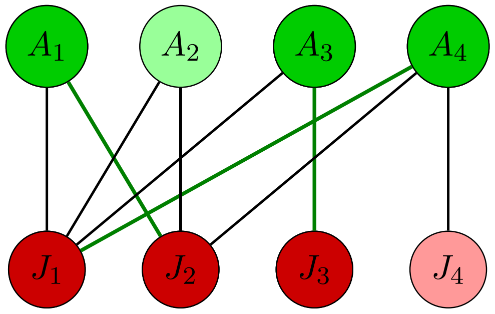
Still no augmenting path. The next smallest costs are equal to 8: $(A_2, J_1), (A_3, J_3), (A_4, J_4)$ are the edges that can be added.



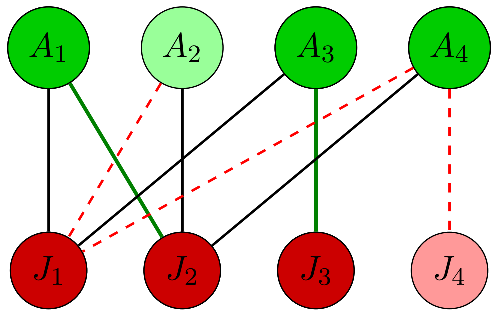
We now have an augmenting path: $A_1 \to J_2 \to A_4 \to J_1 \to A_3 \to J_3$. It is marked in dashed red lines below:



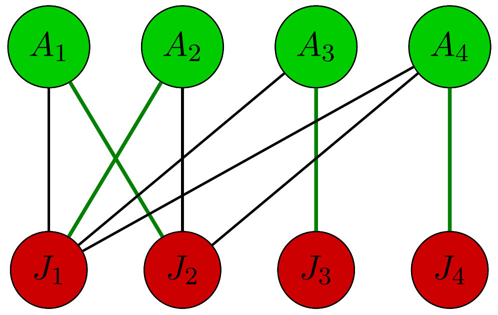
This leads to a new increased cardinality assignment as in:



There is one more augmenting path here:



This increased cardinality matching is also a perfect matching:



This assignment has a total cost of 29 and is the cheapest we can do if we are not allowed any costs above $\tilde{C} = 9$.

Question 3: Fixed cost facility location

In class, we discussed about the facility location problem and provided one of its formulations. However, in reality, opening a facility is associated with a *fixed cost* that needs to be paid before the facility can be used. On top of that, different facilities have different *capacities*; that is, they can only provide so many items to the customers.

Assume that a company is deciding where to open up their new warehouses from a set of 6 potential sites. Their goal is to satisfy all 10 of their customers in this location. The cost of shipping one item from a warehouse to a customer is shown in Table 1. Each facility in order to be "opened" comes with a fixed cost (the cost to open the facility) and a capacity (the amount of products that can be shipped from that facility): these are shown in Table 2. Last, the demands of the customers are known to be 25, 15, 20, 10, 15, 15, 25, 20, 30, and 20, respectively.

	Warehouses					
Customer	1	2	3	4	5	6
1	3	5	8	12	10	11
2	6	7	10	12	8	10
3	5	3	8	10	12	10
4	7	6	7	10	9	5
5	10	8	5	8	6	5
6	12	6	4	7	5	4
7	7	10	6	6	8	3
8	5	6	9	4	4	7
9	8	10	7	5	4	6
10	10	12	8	3	4	6

Table 1: The shipping costs from each facility (warehouses) to each customer.

Warehouse	Fixed Cost	Capacity
1	1750000	75
2	2000000	50
3	2500000	120
4	2250000	100
5	1500000	60
6	1000000	50

Table 2: The fixed cost and capacity of each potential site.

(a) Formulate the problem mathematically. Be careful to define your data, variables, constraints, and objective function, as shown in class. Then, use Gurobi to solve the problem with the data provided.

(b) What if... customers do not want to be satisfied by more than one warehouse? This behavior is called *single-sourcing* and is common with some products that cannot be combined when originating in different sources. What change would you make to your problem? When using Gurobi to solve the problem, do you obtain the same answer?

Answer (a) First of all, let us define the parameters and data we will need. We need two sets $\mathcal{I} = \{1, 2, \dots, 10\}$ for the customers, and $\mathcal{J} = \{1, 2, \dots, 6\}$ for the candidate facility sites. We will then define f_j and u_j to be our fixed costs of opening and the capacity of facility $j \in \mathcal{J}$, respectively. We also define h_i to be the demand of customer $i \in \mathcal{I}$. Finally, let c_{ij} be the cost of shipping one unit to customer $i \in \mathcal{I}$ from facility $j \in \mathcal{J}$.

Now, we proceed to define our variables. There are *numerous* ways to model this problem. I will follow closely on the formulation we discussed in class: hence, I will have $y_j = 1$ signal that facility $j \in \mathcal{J}$ is opened, 0 otherwise. I will also have that $x_{ij} \in [0, 1]$ is a continuous variable between 0 and 1 that represents the fraction of the demand of customer i that facility j is responsible for.

The above definitions lead us to the following constraints:

1. A customer needs to receive 100% of their demand from the facilities that are open.
2. A facility cannot send more items than their capacity.
3. A facility cannot send any items unless they are opened.

Our objective function can be simply to minimize the sum of all costs of the shipped items from each facility j to every customer i. We add to that the cost of opening a facility j. Overall, this leads to the formulation presented in (1).

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij}h_i + \sum_{j\in\mathcal{J}} f_j y_j \tag{1a}$$

$$s.t. \sum_{j\in\mathcal{J}} x_{ij} = 1, \qquad \forall i \in \mathcal{I}, \tag{1b}$$

$$\sum_{i\in\mathcal{I}} x_{ij}\cdot h_i \le u_j y_j, \qquad \forall j \in \mathcal{J}, \tag{1c}$$

$$x_{ij} \ge 0, \qquad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}. \tag{1d}$$

A few notes. First, in the objective function, shown in (1a) we multiply the fractions x_{ij} by the demand h_i, in order to get a correct estimate of the number of items leaving facility j to go to customer i. Then, we also observe that the family of constraints in (1c) capcture both requirement 2 and 3: that is, they both control that a facility j cannot send items unless it is opened, and that it needs to respect its capacity constraint. Last, we do not need to enforce that $x_{ij} \le 1$ in the variable restrictions (1d); this is already enforced by constraint (1b), which also requires that every customer gets 100% of their demand filled.

A Jupyter notebook using Gurobi to solve the problem is provided in Q3a.ipynb.

(b) The only change is a redefinition of x_{ij} from being continuous in $[0, 1]$ to becoming binary (i.e., $x_{ij} \in \{0, 1\}$. Everything else is identical!

Take a look at the Q3b.ipynb notebook.

Question 4: Sudoku!

The game of Sudoku is a very popular, combinatorial, number-placing game. The puzzle maker provides a partially completed 9×9 grid. The goal is then to fill every square in the grid with a number between 1 and 9. The catch?

- Every row needs to have every number between 1 and 9 exactly once.

- Every column needs to have every number between 1 and 9 exactly once.
- The are 9 sub-squares (rows 1-3 and columns 1-3, rows 1-3 and columns 4-6, rows 1-3 and columns 7-9, rows 4-6 and columns 1-3, and so on). Every sub-square also needs to have every number between 1 and 9 exactly once.

Formulate a suitable integer program to solve the sudoku problem. With your formulation, use Gurobi to solve the Sudoku of Figure 3 (left) [1].

Figure 3: A sudoku puzzle (left) and its solution (right).

		6		8		5		
9			3				2	
	4				1			7
					5	6		
	2			4			1	
3								9
		1	9				3	
				2		4		
	5				7			

1	3	6	7	8	2	5	9	4
9	8	7	3	5	4	1	2	6
5	4	2	6	9	1	3	8	7
7	9	8	1	3	5	6	4	2
6	2	5	8	4	9	7	1	3
3	1	4	2	7	6	8	5	9
4	7	1	9	6	8	2	3	5
8	6	9	5	2	3	4	7	1
2	5	3	4	1	7	9	6	8

Answer One way to formulate the problem is by using one set of decision variables, x_{ij}^k defined as:

$$x_{ij}^k = \begin{cases} 1, & \text{if the element in row } i \text{, column } j \text{ has a value of } k = 1, \dots, 9, \\ 0, & \text{otherwise.} \end{cases}$$

Then, the formulation will be as in (2):

[1] An indicative solution is shown in the right side of Figure 3. That said, there could be alternative solutions – so please verify that your obtained solution forms a valid Sudoku!

$$
\begin{aligned}
\min \quad & 0 && && \text{(2a)}\\
s.t. \quad & \sum_{i=1}^{9} x_{ij}^{k} = 1, && \forall j = 1, \ldots, 9, \quad \forall k = 1, \ldots, 9, && \text{(2b)}\\
& \sum_{j=1}^{9} x_{ij}^{k} = 1, && \forall i = 1, \ldots, 9, \quad \forall k = 1, \ldots, 9, && \text{(2c)}\\
& \sum_{i=1}^{3} \sum_{j=1}^{3} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2d)}\\
& \sum_{i=4}^{6} \sum_{j=1}^{3} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2e)}\\
& \sum_{i=7}^{9} \sum_{j=1}^{3} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2f)}\\
& \sum_{i=1}^{3} \sum_{j=4}^{6} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2g)}\\
& \sum_{i=4}^{6} \sum_{j=4}^{6} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2h)}\\
& \sum_{i=7}^{9} \sum_{j=4}^{6} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2i)}\\
& \sum_{i=1}^{3} \sum_{j=7}^{9} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2j)}\\
& \sum_{i=4}^{6} \sum_{j=7}^{9} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2k)}\\
& \sum_{i=7}^{9} \sum_{j=7}^{9} x_{ij}^{k} = 1, && \forall k = 1, \ldots, 9, && \text{(2l)}\\
& \sum_{k=1}^{9} x_{ij}^{k} = 1, && \forall i = 1, \ldots, 9, \quad \forall j = 1, \ldots, 9, && \text{(2m)}\\
& x_{ij}^{k} = 1, && \text{when provided}, \forall i = 1, \ldots, 9, \quad \forall j = 1, \ldots, 9, \quad \forall j = 1, \ldots, 9, && \text{(2n)}\\
& x_{ij}^{k} \in \{0, 1\}, && \forall i = 1, \ldots, 9, \quad \forall j = 1, \ldots, 9, \quad \forall k = 1, \ldots, 9. && \text{(2o)}
\end{aligned}
$$

The "square constraints" in (2d)–(2l) can also be written as the easier:

$$
\sum_{i=k}^{k+2} \sum_{j=\ell}^{\ell+2} x_{ij}^{k} = 1, \qquad \forall k = 1, \ldots, \quad \forall k \in \{1, 4, 7\}, \forall \ell \in \{1, 4, 7\}. \tag{3}
$$

See also Sudoku.ipynb for an indicative code in Gurobi and Python.