Network Dynamics and Learning

Homework 1

Ali Ghorbanpour
Student ID: s309992

Niusha Parsa
Student ID: s312923

Sina Hamedani
Student ID: s315495

Introduction

Graph theory, a fundamental branch of mathematics, plays a crucial role across numerous domains by providing essential tools for analysis and modeling. A prominent application of graph theory is found in social network analysis, where it aids in mapping and examining complex relationships. In these models, vertices symbolize individuals or entities, while edges represent the connections or interactions between them. This approach is vital for understanding social dynamics, identifying key influencers, and exploring network structures. These applications underscore the versatility of graph theory in dissecting and interpreting intricate relational patterns.

Requirements

The Python libraries utilized in this project include NetworkX, NumPy, SciPy, Matplotlib, and CVXPY.

Exercise 1.

Consider the network in Figure 1 with link capacities:

$$c_1 = c_3 = c_5 = 3,\ c_6 = 1,\ c_2 = c_4 = 2$$

Algorithms:

NetworkX provides implementations for numerous network-related algorithms, enabling the execution of complex graph analyses with ease. This project utilizes functions from the modules `nx.algorithms.flow.minimum_cut` to compute the value and node partition of a minimum (Source o, Sink d) cut, and `nx.algorithms.flow.maximum_flow` to maximize the flow from the source to the sink, effectively determining the flow f_{max} with the highest value. The underlying theorem equates two quantities: the maximum flow through a network and the minimum capacity of a network cut.

Main Theorem: In the given scenario, it can be shown that the value of any flow through a network is always less than or equal to the capacity of any o-d cut. Furthermore, it can be proven that there exists a flow with the maximum value and a cut with the minimum capacity. The main theorem establishes a relationship between the maximum flow value and the minimum capacity of a cut in the network. This theorem, known as the *Max-flow Min-cut Theorem*, states that the maximum value of an o-d flow is equal to the minimum capacity among all o-d cuts.

Data preparation: To prepare the data, I first create the graph. Then, in order to determine the maximal flow, the edges of the graph need to be labeled with a 'capacity' label (refer to *Figure 1*).

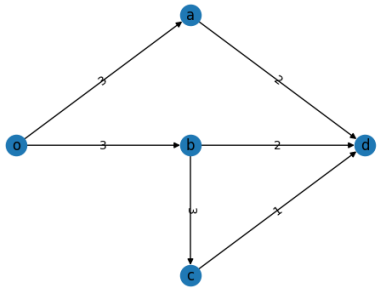

Figure 1: Network with link capacity

Ex1(a): What is the minimum aggregate capacity that needs to be removed for no feasible flow from o to d to exist?

The Minimum Cut algorithm identifies the edges with the lowest combined capacities that can be removed from the graph to create two partitions, where 'o' is in one partition and 'd' is in the other, preventing any possible flow from 'o' to 'd'.

The G graph has 2 cuts which are minimum as below:

$$\{c, o, b, a\}, \{d\},\ Cu = 5$$

$$\{o, a\}, \{c, b, d\},\ Cu = 5$$

The `nx.algorithms.flow.minimum_cut()` method chooses the first one.

The result shows that the minimum capacity that could be removed is 5, which includes two partitions $\{c, o, b, a\}$ and $\{d\}$.

Ex1(b): What is the maximum aggregate capacity that can be removed from the links without affecting the maximum throughput from o to d?

First with using `nx.algorithms.flow.maximum_flow()` I try to find the maximum flow of the graph. The maximum flow problem involves determining the maximum amount of flow that can be sent from a source vertex to a sink vertex in a directed weighted graph, subject to capacity constraints on the edges. But the question contains an ambiguous point that divides it into two possible scenarios:

Scenario 1. If we need to find the total capacity that could be decreased from the edges' capacity without affecting the Maximum flow.

The main objective here is to identify the edges with unused capacity equal to the initial capacity, or more specifically, edges with a flow equal to zero, indicating that the edge is not utilized in the maximum flow algorithm. By iterating over the edges, I identified those with a flow less than the initial capacity to find edges with unused capacity. By summing the unused capacities (total unused capacity), it can be determined which edges could be removed. As a result, I could decrease the capacity by a total of 3 units: 2 units for the edge (b, c) and 1 unit for the edge (o, a).

Scenario 2. If we need to find the edges that could be removed from the graph without affecting the Maximum flow.

Again, the main task here is to iterate over the edges to identify those with no flow, meaning they are not involved in the maximum flow algorithm. These edges can be removed without affecting the maximum flow. After iterating over the edges again, I extracted these specific edges and stored them in a list. The result was that the list was empty, indicating that there are no edges in the graph that can be removed without affecting the maximum flow.

Ex1(c): You are given $x > 0$ extra units of capacity. How should you distribute them in order to maximize the throughput that can be sent from o to d? Plot the maximum throughput from o to d as a function of $x \geq 0$.

Linking edges are the optimal candidates for adding extra capacities. Referring to Figure 2, I utilized a plot to show the correlation between maximum throughput and additional capacity units, up to 10 extra capacity units on the edges. At each step, the code indicates which edge receives the additional capacity and the resulting maximum flow after the increment. Consequently, after adding a total of 10 extra capacity units to the edges of the graph, the capacity of edge (o, a) was increased 4 times, and the capacity of edge (a, d) was increased 6 times. Also at the end, maximum flow value will be 10.

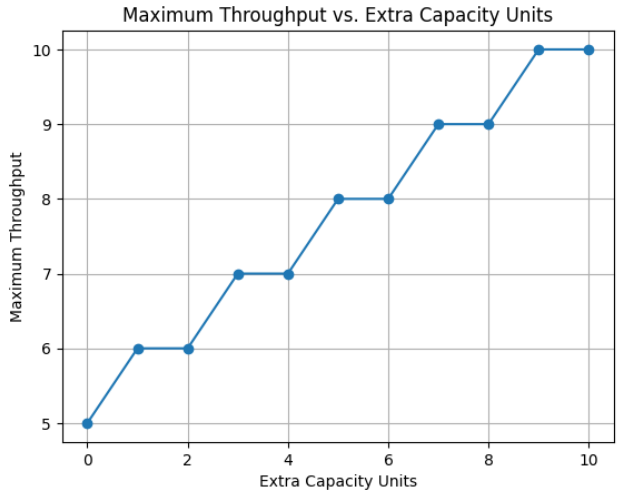

Figure 2: Relation between maximum throughput and extra capacity units

Exercise 2.

Main Theorem: Perfect Matching: A set of edges in a graph $G = (V, E)$ is independent if no two edges share a common incident vertex. Such independent sets of edges are referred to as matchings. M is a matching of $U \cup V$ if every vertex in U is incident with some edge in M. The vertices in U are then called matched (by M), while vertices not incident with any edge of M are unmatched. A perfect matching in $G = (V, E)$ is a matching that includes all vertices in V. My goal is to identify conditions that ensure the existence of large or perfect matchings in arbitrary graphs.

The optimal solution in this scenario is to add two additional source and sink nodes and apply the maximum flow algorithm. Using this approach, I can find a perfect matching where each member of V is linked to a member of E. Each link between these sets represents the interest of one person in one book.

Consider a set of people $\{p1, p2, p3, p4\}$ and a set of books $\{b1, b2, b3, b4\}$. Each person has an interest in a subset of books, specifically:

$$p1 \to \{b1, b2\},\ p2 \to \{b2, b3\},\ p3 \to \{b1, b4\},\ p4 \to \{b1, b2, b4\}$$

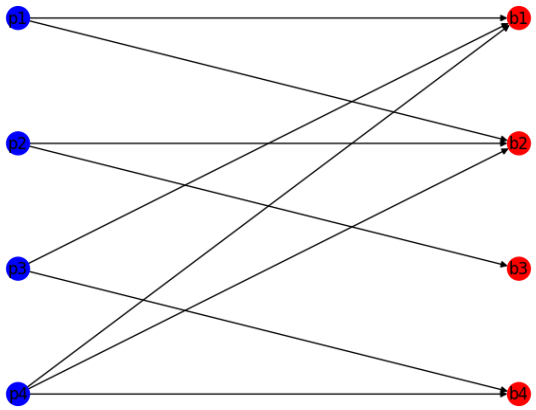

Figure 3: People-Books bipartite graph

Ex2(a): Exploit max-flow problems to find a perfect matching (if any).

For this question, after defining the directed bipartite graph, I set the capacity to 1 for all edges of the graph. This is because we need to match 4 people to 4 books, which is an equal requirement. After running the `nx.algorithms.flow.maximum_flow()` method from the source to the sink, I iterated over the resulting flows and extracted the edges of the matched pairs. As a result, the matched pairs are (p1, b2), (p2, b3), (p3, b1), and (p4, b4), which are shown in the network with red links. Additionally, it is evident that the maximum flow value for this situation is 4.

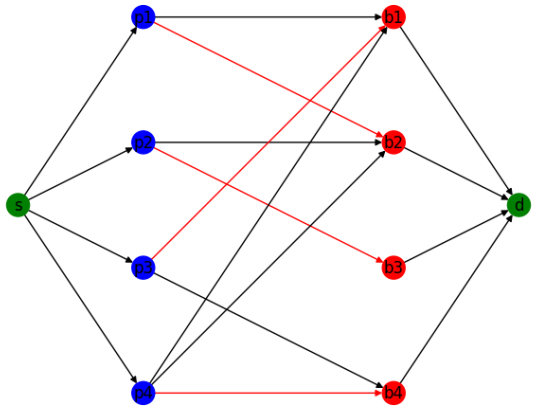

Figure 4: People-Books perfect matching

$$p1 \rightarrow \{b2\},\ p2 \rightarrow \{b3\},\ p3 \rightarrow \{b1\},\ p4 \rightarrow \{b4\}$$

Ex2(b): Assume now that there are multiple copies books, and the distribution of the number of copies is (2, 3, 2, 2). Each person can take an arbitrary number of different books. Exploit the analogy with max-flow problems to establish how many books of interest can be assigned in total.

In this question, after defining the graph, I set the capacities such that the edges from the source s to the persons are infinite, allowing each person to buy any book they want, but only one copy of each book. For example, a person could buy both $b1$ and $b2$, but only one copy of each. Finally, I specified the capacities

as (2, 3, 2, 2) for the edges from the books to the sink d, indicating the number of available copies for each book.

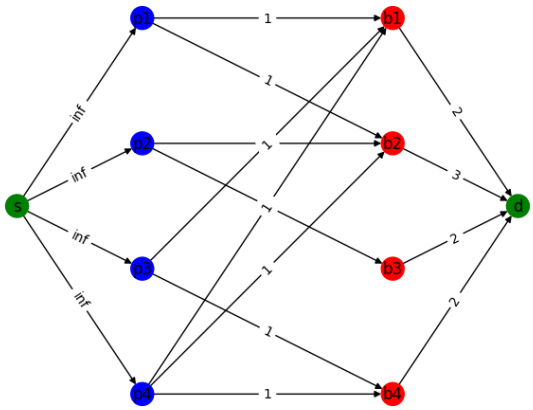

Figure 5: Persons-Book graph with different copies

Afterward, I applied the max-flow algorithm to determine the maximum number of books that could be assigned. The result, considering the individuals' interests, is that 8 books could be assigned.

Ex2(c): Suppose that the library can sell a copy of a book and buy a copy of another book. Which books should be sold and bought to maximize the number of assigned books?

The main goal is to determine the in-flow and out-flow of the books and the sink node d. By examining the in-flow and out-flow for each book, we can see if there are extra copies or if more copies are needed. A higher in-flow means there are extra books to sell, while a higher out-flow means there's a shortage, and more copies are needed. For the sink node d, the in-flow shows the total number of books sold. As a result, we should sell 1 copy of $b1$ and 1 copy of $b3$, with a total of 9 books assigned.

Exercise 3.

We are given the highway network in Los Angeles, see Figure 6. To simplify the problem, an approximate highway map is given in Figure 7, covering part of the real highway network. The node-link incidence matrix B, for this traffic network is given in the file traffic.mat. The rows of B are associated with the nodes of the network and the columns of B with the links.The i-th column of B has 1 in the row corresponding to the tail node of link e_i and (1) in the row corresponding to the head node of link e_i. Each node represents an intersection between highways (and some of the area around). Each link e_i e_1, ..., e_{28}, has a maximum flow capacity c_{e_i}. The capacities are given as a vector c_e in the file capacities.mat. Furthermore, each link has a minimum travelling time l_{e_i}, which the drivers experience when the road is empty. In the same manner as for the capacities, the minimum travelling times are given as a vector l_e in the file traveltime.mat. These values are simply retrieved by dividing the length of the highway segment with the assumed speed limit 60 miles/hour. For each link, I introduce the delay function

$$\tau_e(f_e) = \frac{l_e}{1 - \frac{f_e}{c_e}}, \quad 0 \leq f_e < c_e$$

For $f_e \geq c_e$, the value of $\tau_e(f_e)$ is considered as $+\infty$.

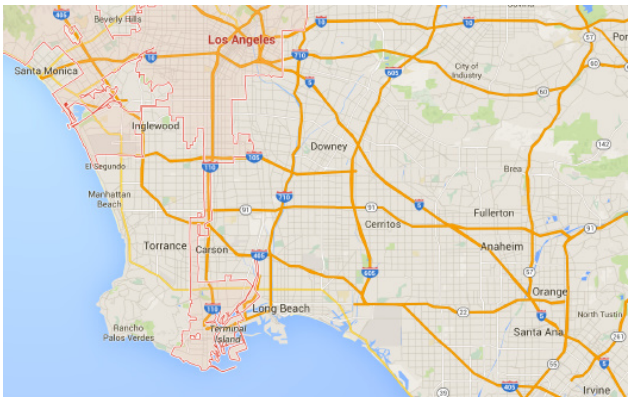

Figure 6: The highway network in Los Angeles.

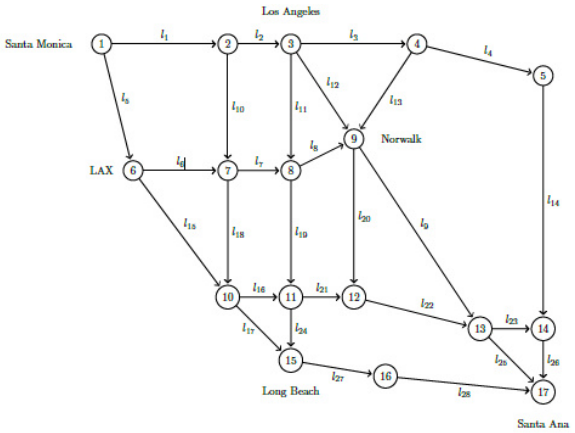

Figure 7: Some possible paths from Santa Monica (node 1) to Santa Ana (node 17).

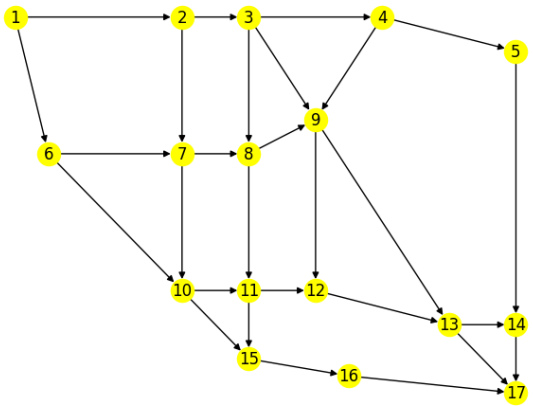

Figure 8: Paths from Santa Monica (node 1) to Santa Ana (node 17).

Ex3(a): Find the shortest path between node 1 and 17. This is equivalent to the fastest path (path with shortest traveling time) in an empty network.

The NetworkX library provides a method called the Shortest Path algorithm, which is used to calculate the shortest path in a graph. This algorithm considers several parameters, including the *Source* (the starting node of the path), the *Target* (the ending node of the path), the *Weight* (the value returned by the traveltime function, representing the weight of an edge), and the *Method* (the algorithm used to compute the path). The default method is 'dijkstra', but other supported options are also available.

The shortest path from node 1 to node 17 is: {1, 2, 3, 9, 13, 17}

The output path includes the source and target nodes, along with a list of nodes that form the shortest path between them.

Ex3(b): Find the maximum flow between node 1 and 17.

I calculated the maximum flow between node 1 and node 17 using the `nx.algorithms.flow.maximum_flow()` method, which shows that this value is 22,448.

Ex3(c): Given the flow vector in flow.mat, compute the external inflow v satisfying Bf = v. In the following, I assume that the exogenous inflow is zero in all the nodes except for node 1, for which v1 has the same value computed in the point (c), and node 17, for which v17 =-v1.

The traffic matrix, previously mentioned, is extracted from the traffic file and stored in a variable. Next, I establish the unitary inflow (exogenous flow vector) from node 1 to node 17, and the value of the traveltime function is stored as an array in another variable. I then formulate a problem that includes an objective and a set of constraints. Using the `solve()` method, the problem is solved by the instance, providing the optimal value and assigning optimal values to the variables. By multiplying the flow and traffic matrices, I calculate the external inflow ν that satisfies the equation $Bf = v$.

lenght

Ex3(d): Find the social optimum f^* with respect to the delays on the different links $\tau_e(f_e)$. For this,minimize the cost function

$$\sum_{e\in E} f_e\tau_e(f_e) = \sum_{e\in E} \frac{l_e}{1-\frac{f_e}{c_e}} = \sum_{e\in E} \frac{l_e}{1-\frac{f_e}{c_e}} - l_e c_e$$

subject to the flow constraints.

To achieve the social optimum f^* for the delays on various links $\tau_e(f_e)$, I use a specific formula and store it in the variable `Func`. By minimizing `Func` and treating it as the objective, I establish the constraints $0 \leq f_e < c_e$ to solve this problem. Subsequently, I invoke the `Solve()` method to attain the optimal objective value.

Ex3(e): Find the Wardrop equilibrium $f^{(0)}$. For this, use the cost function

$$\sum_{e\in E} \int_0^{f_e^0} \tau_e(s)\, ds.$$

To determine the Wardrop equilibrium $f(0)$ using the given formula, the first step is to calculate the integral, which serves as an input parameter for the problem. Afterward, the constraints outlined in the previous section are applied.

Ex3(f): Introduce tolls, such that the toll on link e is $\omega_e = f_e^*\tau_e'(f_e^*)$ where f_e^* is the flow at the system optimum. Now the delay on link e is given by $\tau_e(f_e) + \omega_e$. compute the new Wardrop equilibrium f^ω . What do you observe?

The price of anarchy (PoA) associated to a Wardrop equilibrium f^ω is

$$\text{PoA}(\omega) = \frac{\text{total delay at user optimum}}{\text{total delay at social optimum}}$$

The concept of Wardrop equilibrium, with or without tolls, represents a user-centric approach to optimization. However, it can also be interpreted as an optimal network flow, provided that appropriate costs are assigned to the links.

The System-Optimum Traffic Assignment Problem (SO-TAP) focuses on optimizing network flow by modeling traffic based on the self-interested behaviors of users. Rather than a centralized approach, it considers users choosing routes that minimize their own delays. This concept is further formalized by the price of anarchy, which is defined by the Wardrop equilibrium f^{ω} introduced below.

$$\text{PoA}(\omega) = \frac{\sum_{e \in E} f_e^{(\omega)} \tau_e(f_e^{(\omega)})}{\sum_{e \in E} f_e^* \tau_e(f_e^*)}$$

The Wardrop equilibrium is determined by the ratio of the total delay to the minimum achievable delay. After incorporating tolls into the flow calculations and evaluating the Price of Anarchy (PoA) for both scenarios (with and without tolls), it was observed that the PoA decreases once tolls are implemented.

Ex3(g): Instead of the total travel time, let the cost for the system be the total additional delay compared to the total delay in free flow, given by

$$\psi_e(f_e) = f_e(\tau_e(f_e) - l_e)$$

subject to the flow constraints. Compute the system optimum f^* for the costs above. Construct tolls (ω^*)such that the Wardrop equilibrium $f^{\omega *}$ coincides with f^*. Compute the new Wardrop equilibrium with the constructed tolls $f^{\omega *}$ to verify your result.

The cost function in the free flow delay is used to determine the optimal flow. This optimal flow, obtained from the previous functions is used to compute the constructed tolls. Then the new Wardrop equilibrium and PoA is then calculated based on the optimal flow and constructed tolls.

Part d:
f*: The social optimal flow aims to minimize the total cost across the network. This cost considers the delays on all links, optimizing the overall system performance for the common good. It focuses on minimizing the sum of the delays experienced by all users in the network.

Part e:
f^(0): The Wardrop equilibrium flow represents the state where no individual user can reduce their own travel time by unilaterally changing their route. It is based on user equilibrium where each user acts selfishly to minimize their own travel time, potentially leading to a suboptimal total system performance.

The social optimal flow f* has more balanced and lower flow values on certain links compared to f^(0), indicating a more efficient distribution of traffic that minimizes total delay.
The Wardrop equilibrium flow f^(0) shows higher flow values on certain links, suggesting that users' individual optimizations lead to higher overall congestion and system cost (though specific numbers were not provided for cost, it can be inferred from the flow values).

Part f:
The Price of Anarchy (PoA) decreases when tolls are implemented because the tolls are designed to align individual users' incentives with the overall system efficiency. By adding a cost to each route that reflects its true impact on congestion, users are encouraged to choose routes that minimize their own travel time while also reducing overall system delays. This leads to a more efficient distribution of traffic, lowering the total system cost and bringing the Wardrop equilibrium closer to the social optimal flow, hence reducing the PoA.