Homework 1 - Network Dynamics and Learning

Tommaso Mazzarini s334004
Network Dynamics and Learning
November 2024

Exercise 1

This exercise is based on the network depicted in Figure 1, where we analyze flows and capacities to optimize the throughput from the origin node o to the destination node d. The network has the following link capacities:

$$c_1 = c_3 = c_5 = 3, c_6 = c_7 = 1, c_2 = c_4 = 2.$$

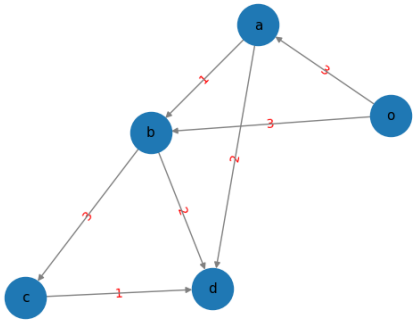

Figure 1: Directed graph where each edge has its own capacity (in red).

a) Cut Capacity Calculation

Given the directed graph in Figure 1, where each edge has a specific capacity, the objective is to compute the capacities of all possible cuts in the network. We focus on the $o-d$ cuts, which are defined as any partition of the set of nodes into two disjoint subsets U and U^c, such that the origin node o belongs to U and the destination node d belongs to U^c. The following is a list of all such cuts:

- **Cut 1:** $U = \{o, a, b, c\}, U^C = \{d\}$
- **Cut 2:** $U = \{o, a, b\}, U^C = \{c, d\}$
- **Cut 3:** $U = \{o, a, c\}, U^C = \{b, d\}$
- **Cut 4:** $U = \{o, b, c\}, U^C = \{a, d\}$

- **Cut 5:** $U = \{o, a\}, U^C = \{b, c, d\}$
- **Cut 6:** $U = \{o, b\}, U^C = \{a, c, d\}$
- **Cut 7:** $U = \{o, c\}, U^C = \{a, b, d\}$
- **Cut 8:** $U = \{o\}, U^C = \{a, b, c, d\}$

The capacity of a cut is defined as the sum of the capacities of all the edges crossing from U to U^c. Formally, for a cut U, the cut capacity is:

$$c_U = \sum_{e \in \partial U} c_e$$

After computing the capacities for each cut, we obtain the following results:

- **Cut 1:** $U = \{o, a, b, c\}, U^C = \{d\}$ – $C_U = 5$
- **Cut 2:** $U = \{o, a, b\}, U^C = \{c, d\}$ – $C_U = 7$
- **Cut 3:** $U = \{o, a, c\}, U^C = \{b, d\}$ – $C_U = 7$
- **Cut 4:** $U = \{o, b, c\}, U^C = \{a, d\}$ – $C_U = 6$
- **Cut 5:** $U = \{o, a\}, U^C = \{b, c, d\}$ – $C_U = 6$
- **Cut 6:** $U = \{o, b\}, U^C = \{a, c, d\}$ – $C_U = 8$
- **Cut 7:** $U = \{o, c\}, U^C = \{a, b, d\}$ – $C_U = 7$
- **Cut 8:** $U = \{o\}, U^C = \{a, b, c, d\}$ – $C_U = 6$

According to the Max-Flow Min-Cut Theorem, the maximum throughput from o to d is equal to the minimum capacity among all $o - d$ cuts. In this case, the minimum cut has a capacity of 5. Therefore, the minimum capacity that must be removed to prevent any feasible flow from o to d coincides with the capacity of the **Cut 1:** $\{U = \{o, a, b, c\}, U^C = \{d\}\}$, which is 5.

b) Extra Capacity Distribution

Given an additional $x > 0$ units of capacity to allocate within the network, the objective is to maximize the throughput from the origin o to the destination d. Our approach focuses on augmenting the capacities of the edges within the minimum cut set, as these edges currently represent the primary bottleneck to flow expansion.

From the prior analysis, **Cut 1**, with an initial capacity of 5, has been identified as the main constraint. Therefore, the first unit of extra capacity is allocated to one of the edges within this cut to relieve the bottleneck. Our solution involves an iterative process that identifies the edges belonging to the current minimum cut set. Among these edges, we select the one with the lowest capacity and increment it by one unit. This allocation process is repeated for each unit of extra capacity until all x units are distributed.

By prioritizing the edges with the most restrictive capacities, each increase provides an optimal improvement in maximum flow from o to d. This method

effectively increases throughput by progressively reducing the impact of bottlenecks.

The maximum throughput from o to d as a function of the additional capacity x is depicted as a stepwise function, with each step representing an increment in the capacity of a bottleneck edge. Figure 2 illustrates this relationship, showing how throughput increases with the allocation of extra capacity units, demonstrating the effectiveness of the proposed allocation strategy.

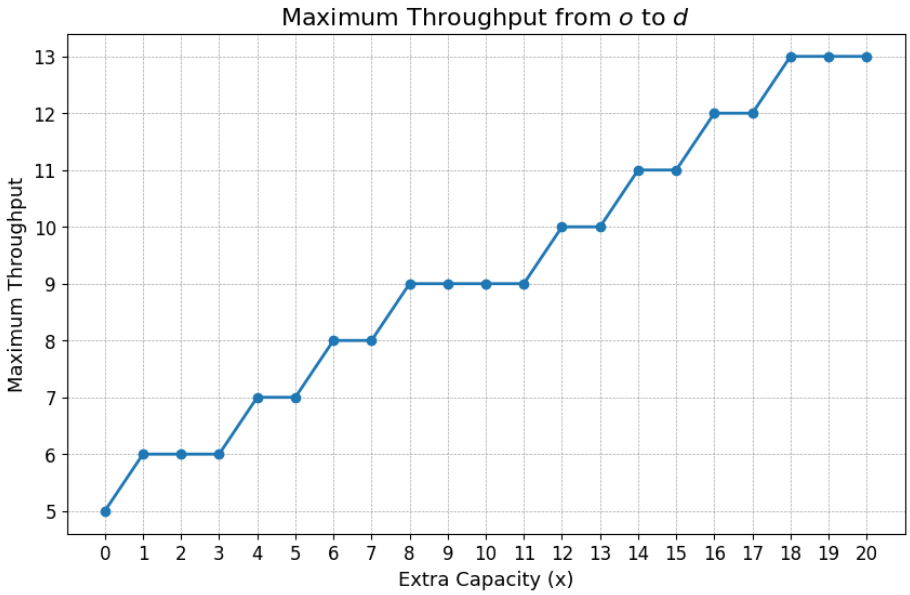

Figure 2: Maximum throughput as a function of extra capacity $x \geq 0$.

c) Adding Link $e8$

Given the opportunity to introduce a directed edge e_8 with an initial capacity of 1, our objective is to identify the optimal location for this edge and determine how to allocate additional capacity in order to maximize the throughput from o to d.

To achieve this goal, we employ an empirical approach where we consider adding the new edge between all possible pairs of nodes, except for those pairs where either the starting or the ending node is d or o, respectively. This preserves the structure of the network, ensuring that the origin node remains o and the destination node is d.

After several iterations, we determine that placing e_8 between nodes o and d increases the maximum throughput by 1, raising it to 6. This results in a new graph, as shown in Figure 3.

From this point forward, we only need to distribute the extra capacity as outlined in the previous section (point b).

Figure 4 shows the maximum throughput as a function of extra capacity $x \geq 0$, starting with a gain from adding e_8, followed by further increases as capacity is allocated to other bottlenecks.

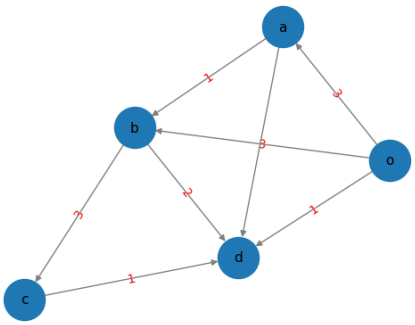

Figure 3: New graph after adding e_8 between nodes o and d.

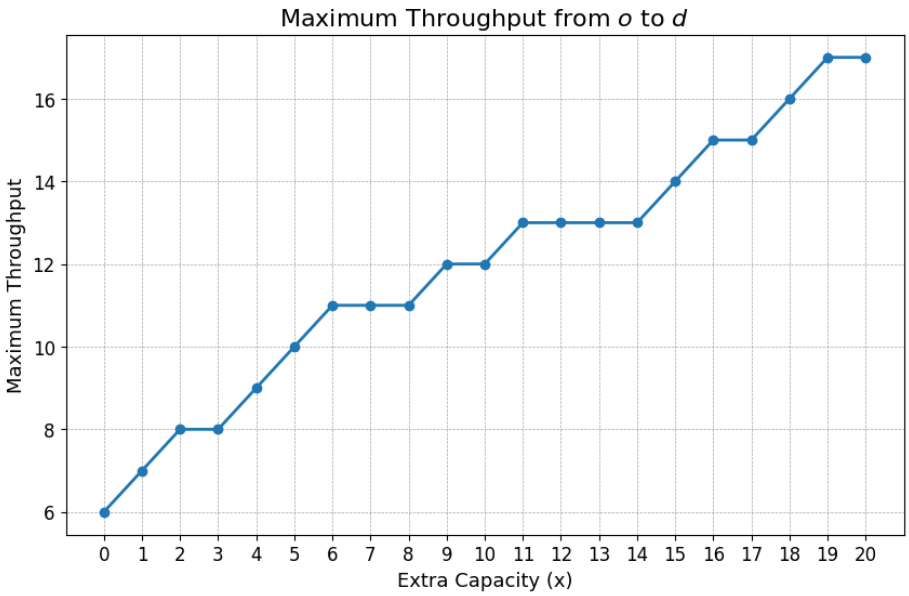

Figure 4: Maximum throughput as a function of extra capacity $x \geq 0$, after introducing the edge e_8.

Exercise 2

This exercise involves a bipartite matching problem where we aim to match individuals with food items based on their preferences. The problem is modeled using a bipartite graph, where one set represents the people $\{a1, a2, a3, a4\}$ and the other set represents the food items $\{b1, b2, b3, b4\}$. Each person has a subset of foods they are interested in, and our goal is to determine the optimal allocation of food items to people under different conditions.

The preferences of the people are as follows:

- $a1$ prefers $\{b1, b2\}$,
- $a2$ prefers $\{b2, b3\}$,

- $a3$ prefers $\{b1, b4\}$,
- $a4$ prefers $\{b1, b2, b4\}$.

a) Perfect Matching

The goal of this first part is to find a perfect matching between people and food items. A perfect matching in a bipartite graph is a set of edges that pairs each person with exactly one food item, respecting their preferences.

We represent this problem using a bipartite graph $G = (P \cup F, E)$, as shown in Figure 5, where:

- $P = \{a_1, a_2, a_3, a_4\}$ is the set of people.
- $F = \{b_1, b_2, b_3, b_4\}$ is the set of food items.
- An edge between person a_i and food item b_j exists if a_i is interested in b_j.

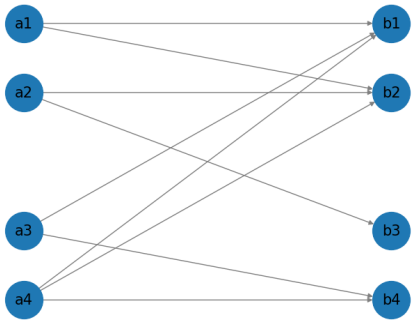

Figure 5: Bipartite graph representation of people and their food preferences.

To find a perfect matching, we can exploit the maximum flow problem by creating a flow network. In this network, the source node o is connected to each person, and each food item is connected to the sink node d. The capacity of each edge between a person and a food item is set to 1, meaning each person can be assigned at most one food item. Similarly, the capacity of edges from people to the source and from food items to the sink is also set to 1. We can visualize this flow network in Figure 6.

The maximum flow algorithm computes the largest possible flow in the network, which corresponds to the number of successful matches. If the maximum flow equals the number of people, a perfect matching exists. In our case, the maximal throughput of the network is 4, which is equal to the number of people, meaning that a perfect matching is indeed possible. An example of such a matching could be $\{a_1 \to b_2, a_2 \to b_3, a_3 \to b_1, a_4 \to b_4\}$, as shown in Figure 7.

b) Multiple Portion Distribution

In this part, we extend the previous model to account for multiple portions of each food item. The distribution of food portions is given as $(2, 3, 2, 2)$,

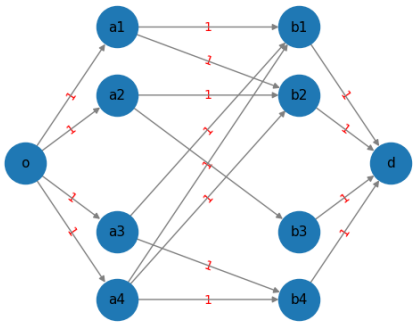

Figure 6: Flow network representation for finding a perfect matching.

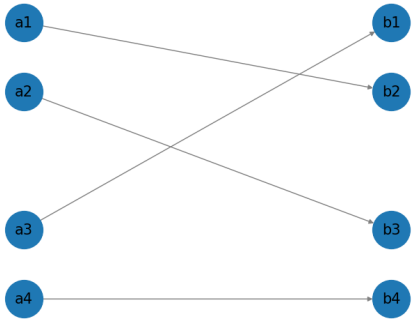

Figure 7: Example of a perfect matching between people and food items.

indicating that there are 2 portions of food b_1, 3 portions of food b_2, 2 portions of food b_3, and 2 portions of food b_4.

We modified the flow network by adjusting the capacity of each food item node to represent the number of available portions, instead of setting it to 1. Additionally, to account for the fact that individuals can choose an arbitrary number of different foods, we set the capacity of the connections between the source and the people to infinity, as illustrated in Figure 8.

In this modified model, the objective is to determine how many food portions can be assigned in total. This is equivalent to computing the maximum flow in the network, where the capacities of the edges from the food items to the sink node d represent the number of available portions.

The maximum flow algorithm calculates the total number of portions that can be allocated while respecting individuals' preferences. In this case, 8 portions are assigned. Although more portions are available, the maximum flow is limited by the number of people and their specific preferences, as not all food items are preferred by everyone. As a result, one food item remains unassigned.

c) Distribution with Individual Requirements

In this part, we modify the model to incorporate individual food portion requirements. Specifically, person a_1 requires 3 portions, while persons a_2, a_3, and a_4

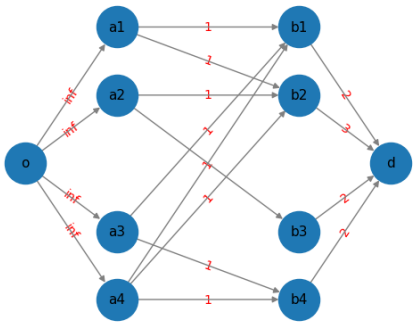

Figure 8: Modified flow network showing food portions and available assignments.

each need 2 portions. The distribution of available food portions remains the same as in the previous section, given by $(2, 3, 2, 2)$, which means there are 2 portions of food b_1, 3 portions of food b_2, 2 portions of food b_3, and 2 portions of food b_4.

To account for these individual demands, we modify the flow network such that each person node now has a capacity equal to the number of portions they require. The edges between people and food items still represent the preferences, but the capacities of the person nodes are updated to reflect the portions each person needs, as shown in Figure 9.

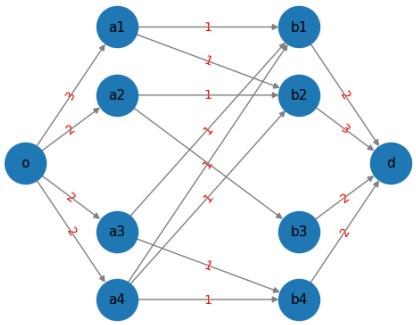

Figure 9: Modified flow network showing food portion allocation with individual requirements.

The objective in this adjusted model is to compute the maximum flow, which determines how many food portions can be assigned while satisfying both the available portions and the specific demands of each individual. The capacity of the edges from the person nodes to the source node o now corresponds to the required portions for each person.

The maximum flow algorithm computes the total number of portions that can be allocated while respecting both the preferences and individual requirements. In this case, the result is 8 portions successfully assigned.

However, not all individuals receive the exact number of portions they requested. Specifically, while person a_1 requested 3 portions, only 2 portions can

be assigned due to the limited availability. Despite this, the portion requirements of the other individuals are fully satisfied.

Exercise 3

This exercise analyzes a simplified model of the Los Angeles highway network, illustrated in Figure 10, which consists of nodes (representing intersections) and links (representing roads). The network is represented using a node-link incidence matrix B, where rows correspond to nodes and columns to links. Each link has a maximum capacity c_e and a minimum travel time l_e, calculated based on a speed limit of 60 miles per hour. The delay function $\tau_e(f_e)$ represents the travel time on link e as a function of the traffic flow f_e, and is given by the formula:

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

where f_e is the flow on link e and c_e is the capacity of the link.

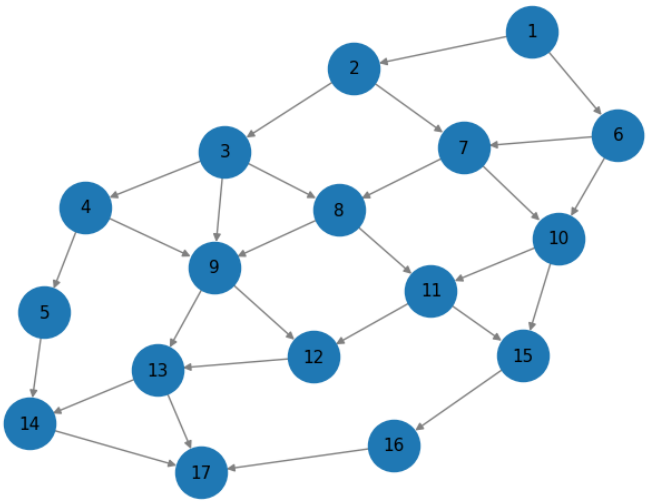

Figure 10: Simplified highway of Los Angeles

a) Shortest Path

The shortest path problem aims to find the path between nodes 1 and 17 that minimizes the total travel time. In an empty network (no traffic), the travel time is simply the sum of the minimum travel times l_e for each link in the path. To compute the shortest path and its corresponding travel time, the network is modeled as a weighted graph, where the weight of each edge represents the travel time l_e. Using the NetworkX library's methods `shortest_path` and `shortest_path_length`, we find that the shortest path is $[1, 2, 3, 9, 13, 17]$. The total travel time for this path, calculated as the sum of the travel times over all the edges, is 0.559833 hours, or approximately 34 minutes.

b) Maximum Flow

The maximum flow problem seeks to determine the maximum amount of traffic that can be transferred between two nodes without exceeding the network's capacity limits. This problem is modeled using a flow network, where each link has a maximum capacity c_e. The solution to the maximum flow problem is obtained using the `max_flow` method from the NetworkX library. The maximum flow value, representing the maximum amount of traffic transferred between nodes 1 and 17, was computed to be 22,448 units.

c) Calculation of the Vector ν

The vector ν represents the exogenous inflow or flow balance at each node in the network. It is computed as the product of the incidence matrix B and the flow vector f. The relationship between the flow vector f and the exogenous balances ν is expressed by the equation:

$$Bf = \nu$$

where f is the flow vector across the links, and ν denotes the exogenous inflow or outflow at the nodes. The values of ν indicate the amount of traffic entering or exiting each node in the network. In this case, the vector ν consists of 17 elements, given by: [16282, 9094, 19448, 4957, −746, 4768, 413, −2, −5671, 1169, −5, −7131, −380, −7412, −7810, −3430, −23544].

d) Social Optimum

The social optimum, or system optimum, seeks to minimize the total network cost, defined as the sum of delays $\tau_e(f_e)$ across all links e. The total cost associated with the flow f is given by:

$$\text{Total cost} = \sum_{e \in \varepsilon} f_e \tau_e(f_e) = \sum_{e \in \varepsilon} \frac{f_e l_e}{1 - \frac{f_e}{c_e}} = \sum_{e \in \varepsilon} \left(\frac{l_e c_e}{1 - \frac{f_e}{c_e}} - l_e c_e \right)$$

The objective is to minimize this cost function while ensuring that flow constraints are satisfied, meaning the inflow and outflow at each node must be balanced. This problem is a convex optimization problem, which can be efficiently solved using convex optimization techniques, particularly useful for traffic flow optimization.

For the current setup, we assume that the exogenous inflow is zero at all nodes except for node 1, where ν_1 is determined as in part (c), and node 17, where $\nu_{17} = -\nu_1$. The objective is to find the social optimum f^*, which minimizes the total cost while satisfying the flow balance constraints.

To solve this, we use the CVXPY library to formulate and solve the convex optimization problem. The resulting optimal flow minimizes the total cost to approximately 23,997 units, representing the most efficient allocation of traffic across the network.

e) Wardrop Equilibrium

The Wardrop equilibrium, or user optimum, occurs when each user selects the path that minimizes their own travel time, ignoring the impact of their choices

on other users in the network.

To compute the Wardrop equilibrium, we need to optimize the following cost function:

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

After solving this integral, we arrive at the following cost function, which is more convenient to optimize using CVXPY:

$$\sum_{e \in \varepsilon} -l_e c_e \ln\left(1 - \frac{f_e}{c_e}\right)$$

This leads to the Wardrop equilibrium flow $f^{(0)}$. The resulting Wardrop cost is approximately 14,927.

In the Wardrop equilibrium, each user minimizes their own cost, which may not result in the most efficient overall flow compared to the social optimum. The difference in efficiency between the Wardrop equilibrium and the social optimum is quantified by the Price of Anarchy (PoA), defined as:

$$\text{PoA} = \frac{\sum_e f_e^{(0)} \tau_e(f_e^{(0)})}{\sum_e f_e^* \tau_e(f_e^*)}$$

The PoA in this case was calculated to be 1.014, indicating that the Wardrop equilibrium is slightly less efficient than the social optimum.

f) Introduction of Tolls

To correct the inefficiency of the Wardrop equilibrium and guide users toward the social optimum, a system of marginal tolls was introduced. The toll on each link e is given by:

$$\omega_e = \psi_e'(f_e^*) - \tau_e(f_e^*) = f_e^* \tau_e'(f_e^*)$$

where f_e^* represents the optimal flow on link e, and $\tau_e'(f_e^*)$ is the derivative of the delay function with respect to the flow.

By introducing these tolls, the effective delay on each link becomes:

$$\tau_e(f_e) + \omega_e$$

This adjustment encourages users to select paths that align more closely with the social optimum, improving the overall efficiency of the network. When the Price of Anarchy (PoA) is recalculated between the optimal and Wardrop costs under these tolls, the result is closer to 1 compared to the previous PoA. This highlights the improved alignment with the social optimum and the effectiveness of marginal tolls in reducing inefficiencies from the original Wardrop equilibrium.

g) Optimization of Total Additional Costs

In this exercise, we aimed to compute the system's social optimum flow by minimizing the total additional travel time compared to the free-flow travel time. The cost for each link e was defined as:

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

where $\tau_e(f_e)$ is the delay function, and l_e is the free-flow travel time.

The social optimum flow, denoted as f^*, was obtained by solving an optimization problem using CVXPY, subject to flow conservation constraints.

Next, we constructed a toll vector ω^* such that the Wardrop equilibrium flow with tolls, $f^{(\omega^*)}$, coincides with the social optimum. The toll for each link was calculated as:

$$\omega_e = f_e^* \tau_e'(f_e^*) - l_e$$

We then solved another optimization problem to determine the Wardrop equilibrium with tolls, ensuring the flow matched f^*. For an epsilon value on the order of 10^{-5}, the two flow vectors were found to be nearly identical.

Finally, we calculated the Price of Anarchy (PoA), which quantifies the efficiency loss between the Wardrop equilibrium with tolls and the social optimum. The PoA was found to be around 1, suggesting that the toll system effectively reduced inefficiencies and brought the system closer to optimal efficiency.