Homework assignment 1

Jiahao Zhang

November 13, 2022

Abstract

This is a proposed solution for homework assignment 1 of the course Network Dynamics and Learning 2022/23

Exercise1

Consider the network in Figure 1 with link capacities

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

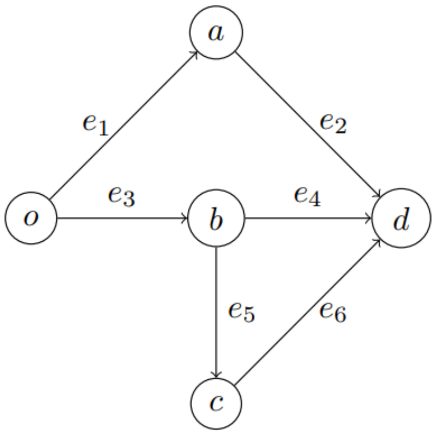

Figure 1

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

- The minimum aggregate capacity that needs to be removed equals the min-cut of the graph. Recall that an $o-d$ cut is a partition of the node set V into two subsets U and $V \setminus U$. The capacity of an $o-d$ cut U is the aggregate capacity of the links from U to $V \setminus U$:

$$c_U = \sum_{\substack{e:\tau(e)\in U \\ \kappa(e)\in V\setminus U}} c_e$$

and the min-cut capacity is the minimum among all $o-d$ cuts. For this specific case, all that is required to do is to import the graph using the networkx library, and then apply the nx.algorithms.flow.minimum_cut algorithm on the graph specifying the nodes that we want to cut. For our case, the algorithm returns 3 which is equal to the maximal aggregate capacity we need to remove in order to disconnect node d from node o.

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

- In order to compute the maximum aggregate capacity, firstly we need to compute the maximum throughput from o to d. To do so, we could use the max-flow min-cut theorem which states that in a graph the max-flow from two specific nodes equals the min-cut of these 2 nodes. From the previous point, we already obtained the min-cut which is equal to 3. After this observation, we would like to understand the flow that goes in each node (for simplicity, we assume that the flow can only have integer values). To do so, we could exploit the output of the nx.algorithms.flow.maximum_flow function which returns the max-throughput of the graph and a dictionary containing for each node the neighbor nodes and for each of the neighbor nodes the respective flow. After handling properly this output we could obtain the flow vector

  $$f = [1, 1, 2, 1, 1, 1]$$

  comparing the flow vector with the capacity vector given in the exercise

  $$c = [2, 1, 2, 1, 2, 1]$$

  we could simply say that we can remove 1 capacity from link e_1 and 1 capacity from link e_5. Hence, the maximum aggregate capacity that we could remove from the graph without affecting the throughput of the graph is equal to 2

(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

- This point requires more attention with respect to the previous 2 points of the exercise. In order to visualize the result first of all we need to compute all the possible $o - dcuts\ U$. To do so, we could use the *combination* function that creates all the possible subsets of nodes given the number of nodes we want for each subset. Since we are looking at the possible subset U, those subsets must contain the node o but not the node d. It is trivial to then obtain the complementary subset U^c. After computing those 2 subsets, all we need to do is to compute the capacity of each $o - d$ cut U. We could exploit the adjacent matrix: for each node i in U, we check if the adjacent node j is in U^c. If this is the case, we increment the capacity of the $o - d$ cut adding the capacity of the link that connects i with j. Otherwise, we just continue our search.

  In the following table, we show all the possible $o - d$ cuts and their respective capacity after one iteration of the proposed approach:

n	U	U^c	c_U
U_1	$\{o, a, b, c\}$	$\{d\}$	3
U_2	$\{o, b, c\}$	$\{c, d\}$	4
U_3	$\{o, a, c\}$	$\{b, d\}$	4
U_4	$\{o, a, b\}$	$\{a, d\}$	4
U_5	$\{o, c\}$	$\{a, b, d\}$	5
U_6	$\{o, b\}$	$\{a, c, d\}$	5
U_7	$\{o, a\}$	$\{b, c, d\}$	3
U_8	$\{o\}$	$\{a, b, c, d\}$	4

  Since the problem requests to maximize the throughput of the graph, our goal is to increment the capacity of the minimum $o - d$ cuts. To do so, we can first retrieve the cuts with the minimum capacity, and then secondly find which edges have been used for the calculation of their capacities. For instance, in our graph, the minimum cut capacities for this step are $U_1 = \{o, a, b, c\}$ and $U_7 = \{o, a\}$. Let's now retrieve their capacity formula.

  $$c_{U_1} = c_2 + c_4 + c_6$$

  $$c_{U_7} = c_2 + c_3$$

  In general, we would like to add the extra capacity (suppose that we are adding integer values of capacity) to the edges that are in common when retrieving the capacity formula for each cut, so that when we add the extra capacity to it, we increase the capacity of all the cuts that have the specific edge in their capacity formula. In the example, since the edge c_2 is the most common edge between those 2 cuts, we add extra capacity to it. To obtain the plot of the maximum throughput as a function of x (extra capacity) added to the graph, we should just repeat the algorithm described above for fixed x. For understanding purposes, the x has been set to 8. The obtained plot is as follows:

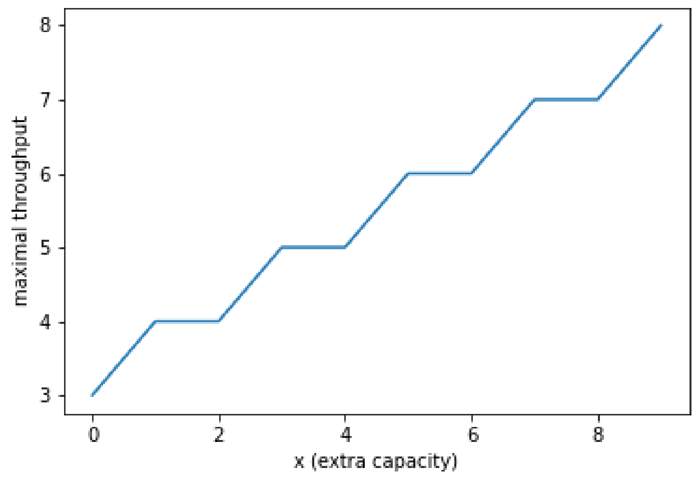

Figure 2

In our special case, we notice from the output of the algorithm and from the graph some peculiar aspects of the graph that we are analyzing:

1 Due to the intrinsic property of the graph, we increase the throughput of the graph by 1 when we add 2 extra capacities on the graph (excluding the 1st case when we add just 1 extra capacity to increase the max throughput by 1).

2 After running the algorithm several times and printing out the edge we would like to add the extra capacity, we notice that the capacity is all concentrated in the edges e_1 and e_2, in particular, firstly we add capacity to e_1, then in the following step we add it to e_2 and then the process repeats, making it like a stationary process where after 2 cycles all the cuts' capacity have the same distribution but only with value increased by one

.

1 Exercise2

There are a set of people $p1, p2, p3, p4$ and a set of books $b1, b2, b3, b4$. Each person is interested in a subset of books, specifically :

$$p_1 \to \{b_1, b_2\} \quad p_2 \to \{b_2, b_3\} \quad p_3 \to \{b_1, b_4\} \quad p_4 \to \{b_1, b_2, b_4\}$$

Before going directly into the exercises and their solution, we should have a visualized graph in order to have a clear view of what we are analyzing :

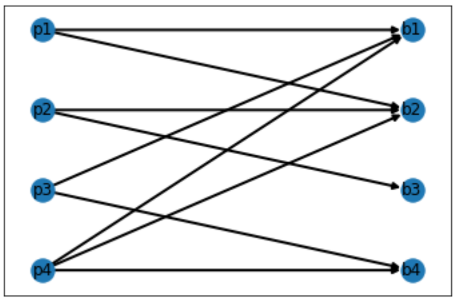

Figure 3

it is clear to see that the graph we are analyzing now is bipartite

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

- In order to exploit the max-flow problem, firstly we need to add 2 additional nodes that represent respectively the starting node o and the ending node d. Since we are in a directed graph, we first create some directed edges that start from node o and go to the nodes $p = \{p_1, p_2, p_3, p_4\}$ and then we do the same on the other side but starting the edges from $b = \{b_1, b_2, b_3, b_4\}$ to node d. Following the process, we will obtain the following graph : For simplicity, we can set the capacity of all the edges to 1, and then we apply the

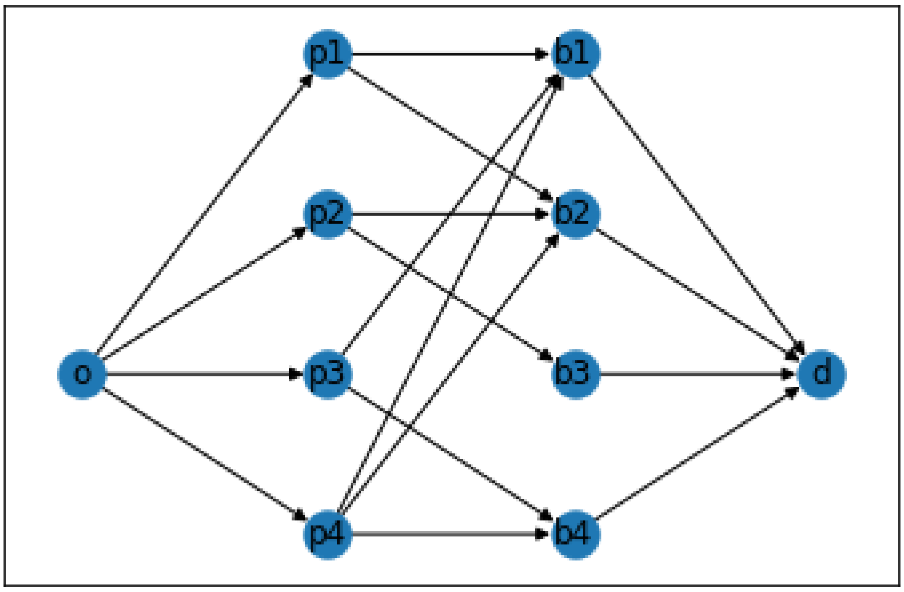

Figure 4

nx.algorithms.flow.maximum_flow function on the graph setting starting node o and ending node d. From the output of the function, we obtain that the maximum throughput of the graph is 4 which means that there exists a perfect matching. In particular, the matching is

$$\{p_1, b_2\} \quad \{p_2, b_3\} \quad \{p_3, b_1\} \quad \{p_4, b_4\}$$

b Assume now that there are multiple copies of 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

- To solve this part of the exercise, we still use the graph we have created for part(a) but we do some manipulation on the capacities of the graph. Since the text says that we have different copies of each book, we should change the capacity of the edges accordingly. in particular:
  - each person can pick more than one book: this means that we should change the capacity of nodes from o to vector p. A safe solution would be to set each of the edges as the total sum of books (since we are looking for a possible perfect match)
  - the rule that one person can pick only one book from a specific "category" is still valid. So we assume that the edges connecting set p to set b maintain their capacity to 1
  - since one "copy" of the book can be picked by different persons, we increase the capacity of the edges accordingly to the copies of each book, which means :

$$G[b_1][d][capacity] = 2$$

$$G[b_2][d][capacity] = 3$$

$$G[b_3][d][capacity] = 2$$

$$G[b_4][d][capacity] = 2$$

Applying those changes on the graph, and applying nx.algorithms.flow.maximum_flow function over the graph from o to d, we obtain as a result that the max throughput is 8, and the matching is:

$$\{p_1, b_2\} \quad \{p_2, b_2\} \quad \{p_2, b_3\} \quad \{p_3, b_1\} \quad \{p_3, b_4\} \quad \{p_4, b_1\} \quad \{p_4, b_2\} \quad \{p_4, b_4\}$$

Notice that in this result, we don't have a perfect matching because the maximum throughput returned by the function is 8, and instead it should return 9 in case of perfect matching. In fact, only one copy of b_3 has been taken while the copies of the other books have been all taken.

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?

- From the last result of point (b) we could say definitely that a copy of b_3 can be sold. But in general, using the code provided together with the report, it's not necessary to have the result of point (b) to compute the result. Doing iteration of all the possible solutions we may have, in order to maximize the number of assigned books, we obtain at the end that we should sell a copy of b_3 and buy a copy of b_1 to maximize the number of assigned books.

2 Exercise3

We are given the highway network in Los Angeles, see Figure 5. To simplify the problem, an approximate highway map is given in Figure 6, 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 $e_i \in \{e_1, ..., e_{28}\}$, has a maximum flow capacity c_ei . The capacities are given as a vector c_e in the file $capacities.mat$. Furthermore, each link has a minimum traveling time l_ei, which the drivers experience when the road is empty. In the same manner, as for the capacities, the minimum traveling 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 by the assumed speed limit of 60 miles/hour. For each link, we introduce the delay function

$$\tau(f_e) = \frac{l_e}{1 - f_e/c_e}, \quad 0 \geq f_e < c_e$$

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

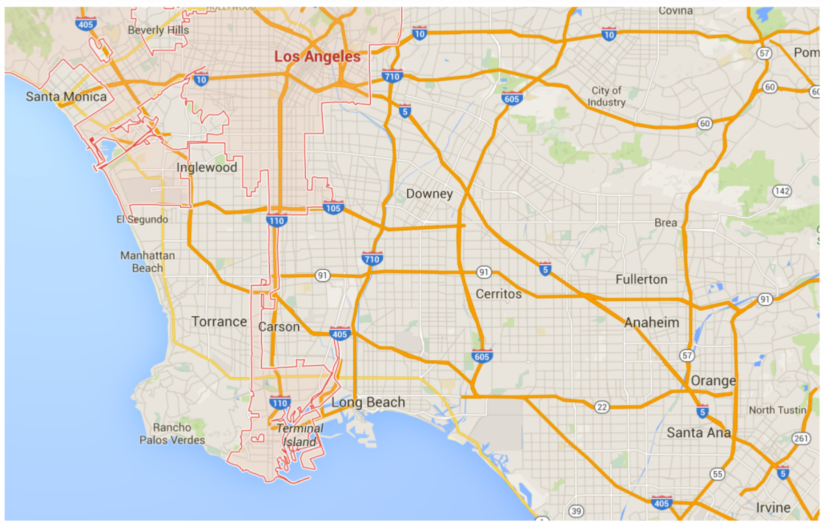

Figure 5

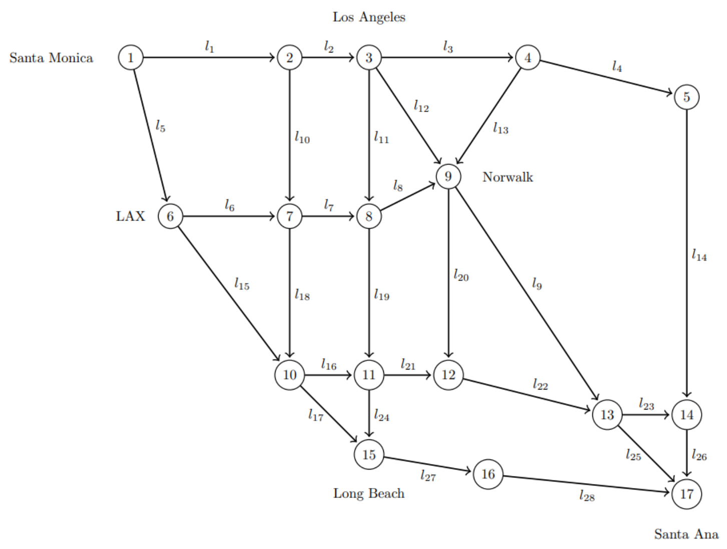
Los Angeles
Santa Monica
Norwalk
LAX
Long Beach
Santa Ana
l_1 l_2 l_3 l_4 l_5 l_6 l_7 l_8 l_9 l_{10} l_{11} l_{12} l_{13} l_{14} l_{15} l_{16} l_{17} l_{18} l_{19} l_{20} l_{21} l_{22} l_{23} l_{24} l_{25} l_{26} l_{27} l_{28}
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Figure 6

Before the presentation of the questions and their relative solutions, some preliminary elements should be presented for the understanding of the solution. In order to write the optimization problems and obtain the solution of the problem (if any), the cvxpy library has been used.
Given a multigraph (V, E), an exogenous network flow is a vector $\nu \in \mathrm{R}^V$ such that

$$\sum_{i\in V} \nu_i = 0. \tag{1}$$

A network flow is a vector $f \in \mathrm{R}^E$ satisfying a positivity constraint and a mass conservation constraint, i.e.,

$$f \geq \mathbf{0}, \quad Bf = \nu. \tag{2}$$

Every edge is endowed with a separable non-decreasing convex cost function $\psi_e(f_e)$ such that $\psi_e(0) = 0$.

Given an exogenous flow ν and a network with node-edge matrix B, we study the following optimization problem:

$$f^* \in \underset{\substack{f\in\mathrm{R}^E_+ \\ Bf=\nu}}{\arg\min} \sum_{e\in\mathcal{E}} \psi_e(f_e). \tag{3}$$

For a simpler visualization of the flow vector, a table format has been used. The elements of the flow vector should be read from left to right, from up to down.

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

- In this problem we are looking for the shortest path between nodes1 and node17. this means that we should set the exogenous network flow vector as follows :

  $$\nu = (1, 0, ..., -1)$$

  which corresponds to setting 1 to node1 and -1 to node17. After this simple setting, we should write down and solve the optimization problem for the shortest path which can be written as

  $$f^* \in \underset{\substack{f\in\mathrm{R}^E_+ \\ Bf=\nu}}{\arg\min} \sum_{e\in\mathcal{E}} \psi_e(f_e).$$

  where the cost function $\psi_e(f_e)$ in this case is equal to

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

  Solving the problem, we obtain the network flow vector. After some manipulation of the output, we obtain the list of nodes of the shortest path γ:

  $$\gamma = \{1, 2, 3, 9, 13, 17\}$$

b Find the maximum flow between nodes 1 and 17.

- In order to tackle this problem, the easiest approach would be to create a graph in networkx and use the nx.algorithms.flow.maximum_flow function. To create the graph, we can use the node-edge B .In fact, from each row of B we can extract the edges of the graph and after that we notice that we could also extract the capacity of each node from the vector c which contains the capacity of all the edges. Applying the algorithm described above we obtain that the maximum throughput of the graph is 22448. A more detailed output and the related flow vector can be found in the output of the code

c Given the flow vector in $flow.mat$, compute the external inflow ν satisfying $Bf = \nu$

- Recall that in this case, the vector f is not the flow vector that has been calculated in the previous points but given by the problem. To solve this point all we need to do is to apply matrix multiplication between B and f using the @ notation and we obtain the following vector

16806	8570	19448	4957	-746	4768	413	-2	-5671
1169	-5	-7131	-380	-7412	-7810	-3430	-23544	

Table 1: external inflow

Notice that the sum of all the elements satisfies the condition

$$\sum_{i \in V} \nu_i = 0. \tag{4}$$

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

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

subject to flow constraints.

- In order to compute the social optimum, we should set

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

The text gives us already an easier way to write the cost function using the cvxpy library. After those considerations, the optimization problem can be written as

$$f^* \in \underset{\substack{f \in \mathbb{R}_+^E \\ Bf = \nu}}{\arg\min} \sum_{e \in \mathcal{E}} (\frac{l_e c_e}{1 - f_e/c_e} - l_e c_e).$$

Solving this problem, we obtain the flow vector f^* can be shown in the following format (numbers have been rounded to the first decimal to not getting confused by the scientific notation as shown by default in the output of the solution) :

6642.2	6058.9	3132.3	3132.3	10163.8	4638.3	3006.3
2542.6	3131.5	583.3	0	2926.6	0	3132.3
5525.5	2854.3	4886.4	2215.2	463.7	2337.7	3318.
5655.7	2373.1	0	6414.1	5505.4	4886.5	4886.5

Table 2: optimal flow

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

$$\psi_e(f_e) = \int_0^{f_e} \tau_e(s) \, ds.$$

- In order to insert this cost function into our optimization problem solver, we need to explicitly calculate the integral the cvxpy does not support integral expression. An intuitive approach is shown here to obtain the explicit formula:

$$\psi_e(f_e) = \int_0^{f_e} \tau_e(s)\ \mathrm{d}s = \int_0^{f_e} \frac{l_e}{1 - s/c_e}\ \mathrm{d}s$$

We can see that it is very similar to the format

$$\int \frac{1}{1-s}\ \mathrm{d}s = -\log(|s|)$$

So in the end we will get

$$\int_0^{f_e} \frac{l_e}{1 - s/c_e}\ \mathrm{d}s = -l_e c_e \log(|x - c_e|)|_0^{f_e} = -l_e c_e(\log(c_e - f_e) - \log(c_e))$$

(Recall that c_e is always greater or equal than f_e, so the abs value can be taken off)

Now, let's put this expression to solve our minimization problem :

$$f^{(0)} \in \underset{\substack{f \in \mathrm{R}_+^{\mathcal{E}} \\ Bf=\nu}}{\arg\min} \sum_{e \in \mathcal{E}} -l_e c_e(\log(c_e - f_e) - \log(c_e)) \tag{5}$$

Now let's show the $f^{(0)}$ Wardrop equilibrium vector: (same as before, the numbers have been rounded to the 1st decimal for better visualization):

6715.6	6715.6	2367.4	2367.4	10090.4	4645.4	2803.8
2283.6	3418.5	0.	176.8	4171.4	0.	2367.4
5445.	2353.2	4933.3	1841.6	697.1	3036.5	3050.3
6086.8	2586.5	0.	6918.7	4953.9	4933.3	4933.3

Table 3: Wardrop equilibrium flow

After obtaining the Wardrop equilibrium, it is straightforward to think how correlated are those 2 vectors. One approach would be to compute the Price of Anarchy(PoA) of the Wardrop equilibrium $f^{(0)}$ which is equivalent to the total delay at Wardrop equilibrium/total delay at system optimum :

$$PoA = \frac{\sum_e f_e^{(0)} d_e(f_e^{(0)})}{\sum_e f_e^* d_e(f_e^*)} = \frac{15729.6}{25943.6} = 1.65$$

It measures how inefficient is to let every possible "user" decide its own best path.

f Introduce tolls, such that the toll on link e is $\omega_e = f_e^* \tau_e'(f_e^*)$ 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?

- First of all, we would like to compute explicitly the toll ω_e value for each edge in the graph. To do so, firstly we compute explicitly $\tau_e'(f_e^*)$ as follows :

$$\tau_e'(f_e^*) = (\frac{l_e}{1 - f_e^*/c_e})' = \frac{l_e c_e}{((f_e^* - c_e)^2}$$

Then the toll for each edge is computed as :

$$\omega_e = f_e^* \frac{l_e c_e}{((f_e^* - c_e)^2}$$

Adding this term to the delay function, we obtain (we now use a different notation for the delay with tolls so that we don't get confused with the delay function without the toll)

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

Now repeat the same procedure to compute the new Wardrop equilibrium $f^{(w)}$ with the new delay function. Firstly, let's find an explicit formulation for the cost function

$$\psi_e(f_e) = \int_0^{f_e} d_e(s)\ \mathrm{d}s = \int_0^{f_e} \frac{l_e}{1 - s/c_e} + \omega_e\ \mathrm{d}s$$

since integration is an associative function, we could do separate integration

$$\psi_e(f_e) = \int_0^{f_e} \frac{l_e}{1 - s/c_e}\ \mathrm{d}s + \int_0^{f_e} \omega_e\ \mathrm{d}s$$

The first term is the same computed in the point (e), and the second term is a simple derivation of a constant function, so in the end

$$\psi_e(f_e) = -l_e c_e(\log(c_e - f_e) - \log(c_e)) + w_e f_e$$

Now, let's put this expression to solve our minimization problem :

$$f^{(w)} \in \underset{\substack{f \in \mathrm{R}_+^{\mathcal{E}} \\ Bf = \nu}}{\arg\min} \sum_{e \in \mathcal{E}} -l_e c_e(\log(c_e - f_e) - \log(c_e)) + w_e f_e \tag{6}$$

And the result of the new Wardrop vector $f^{(w)}$ is as follows :

6643.	6059.1	3132.5	3132.5	10163.	4638.3	3006.3
2542.3	3131.5	583.9	0.	2926.6	0.	3132.5
5524.8	2854.2	4886.4	2215.8	464.	2337.5	3318.2
5655.7	2373.	0	6414.1	5505.5	4886.4	4886.4

Table 4: Wardrop equilibrium flow with tolls

An easy way to compare the Wardrop equilibrium with tolls $f^{(w)}$ with the social optimum f^* is to subtract them, and analyze the output that is shown here in the table:

-0.8	-0.1	-0.1	-0.1	0.8	0.1	0.
0.3	0.1	-0.6	0.	-0	0.	-0.1
0.7	0.	0.1	-0.6	-0.3	0.2	-0.2
0.	0.1,	0.	-0.	-0.1	0.1	0.1

Table 5: pair wise difference between $f^{(w)}$ and f^*

since both values of $f^{(w)}$ and f^* are of the order of 10^3, despite some very small errors due to machine calculation we can say that

$$f^{(w)} = f^*$$

as proof of the theorem. In fact, ω_e has been constructed specially to satisfy this condition because

$$(f_e \tau_e(f_e))' = \tau_e + f_e \tau_e'(f_e)$$

So, when computing the Wardrop equilibrium with the toll ω_e, we are basically computing the social optimum :

$$\psi_e(f_e) = \int_0^{f_e} \tau_e(s) + \omega_e\ \mathrm{d}s = \int_0^{f_e} (s\tau_e(s))'\ \mathrm{d}s = f_e \tau_e(f_e)$$

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

- Basically, in this last part of the exercise we need to repeat all the processes we have done so far from point (d) to point (f) but with a different cost function. Now, let's derive the explicit formula of the cost function to be inserted in the cvxpy solver for the social optimum. in this case

$$\psi_e(f_e) = f_e(\tau_e(f_e) - l_e) = f_e\tau_e(f_e) - f_e l_e = \frac{l_e c_e}{1 - f_e/c_e} - l_e c_e - f_e l_e$$

So, the social optimum f^* can be found by solving the following optimization problem

$$f^* \in \underset{\substack{f \in \mathbb{R}_+^E \\ Bf=\nu}}{\arg\min} \sum_{e\in\mathcal{E}} \left(\frac{l_e c_e}{1 - f_e/c_e} - l_e c_e - f_e l_e\right).$$

And the following table shows the new social optimum vector f^* :

6653.3	5774.7	3419.7	3419.7	10152.7	4642.8	3105.8
2662.2	3009.1	878.6	0.	2354.9	0.	3419.7
5509.9	3043.7	4881.8	2415.6	443.7	2008.	3487.4
5495.4	2203.8	0.	6300.7	5623.5	4881.8	4881.8

Table 6: new social optimum

We can easily see that by doing the difference between this optimal and the social optimal calculated in point (c), there are some components with a significative difference (the result is shown as a print in the code in order to avoid collapse of not very useful tables in the report).

We would like to construct the tolls so that the Wardrop equilibrium calculated with the tolls coincides with the social optimum. To do so, we can use the result cited in point (f) which states that the toll for each edge has to be constructed in this way :

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

Since the cost function ϕ_e has been changed, also the relative τ_e has been changed. In particular, from now on, we will consider as τ_e the new value $\tau_e - l_e$ First of all, let's compute the explicit formulation for the first derivative of τ_e. Due to the associative property of the derivation operator, we could ignore the second term since the derivative of a constant is zero, so in the end, the formula for the toll does not change. In particular :

$$\omega_e = f_e^* \frac{l_e c_e}{((f_e^* - c_e)^2}$$

Adding this term to $\tau_e - l_e$ we obtain the new delay function

$$d(f_e) = \tau_e(f_e) - l_e + \omega_e$$

Now repeat the same procedure to compute the new Wardrop equilibrium $f^{(w)}$ with the new delay function. Firstly, let's find an explicit formulation for the cost function

$$\psi_e(f_e) = \int_0^{f_e} d_e(s)\ \mathrm{d}s = \int_0^{f_e} \frac{l_e}{1 - s/c_e} - l_e + \omega_e\ \mathrm{d}s$$

since integration is an associative function, we could do separate integration

$$\psi_e(f_e) = \int_0^{f_e} \frac{l_e}{1 - s/c_e}\ \mathrm{d}s - \int_0^{f_e} l_e\ \mathrm{d}s + \int_0^{f_e} \omega_e\ \mathrm{d}s$$

The first term is the same computed in the point (e), and both the second term and third term are simple integration of a constant function, so in the end

$$\psi_e(f_e) = -l_e c_e(\log(c_e - f_e) - \log(c_e)) - l_e f_e + w_e f_e$$

Now, let's put this expression to solve our minimization problem :

$$f^{(w)} \in \underset{\substack{f \in \mathbb{R}_+^{\mathcal{E}} \\ Bf=\nu}}{\arg\min} \sum_{e\in\mathcal{E}} -l_e c_e(\log(c_e - f_e) - \log(c_e)) - l_e f_e + w_e f_e \tag{7}$$

And the result of the new Wardrop vector $f^{(w)}$ is as follows :

6653.1	5775.4	3419.5	3419.5	10152.9	4642.4	3105.5
2661.7	3009.2	877.7	0.	2355.9	0.	3419.5
5510.4	3043.4	4881.7	2414.6	443.8	2008.5	3487.1
5495.6	2204.1	0.	6300.7	5623.5	4881.7	4881.7

Table 7: new Wardrop equilibrium flow with tolls

An easy way to compare the Wardrop equilibrium with tolls $f^{(w)}$ with the social optimum f^* is to subtract them, and analyze the output that is shown here in the table:

0.2	-0.8	0.2	0.2	-0.2	0.4	0.3
0.5	-0.1	0.9	0.	-1.	0.	0.2
-0.5	0.3	0.1	0.9	-0.1	-0.5	0.2
-0.2	-0.3	0.	-0.	-0.1	0.1	0.1

Table 8: new pairwise difference between $f^{(w)}$ and f^*

Again, since the components of $f^{(w)}$ and f^* are order of 10^3, we can say that we have proven again the equivalence

$$f^{(w)} = f^*$$

As it should be.