Network Dynamics and Learning

Homework 1

2023-2024

Carena Simone, s309521
Carmone Francesco Paolo, s308126
Mazzucco Ludovica, s315093

This document has been produced jointly by the students mentioned above, who have tightly collaborated throughout the whole homework.

Exercise 1

Given the following graph $\mathcal{G}$

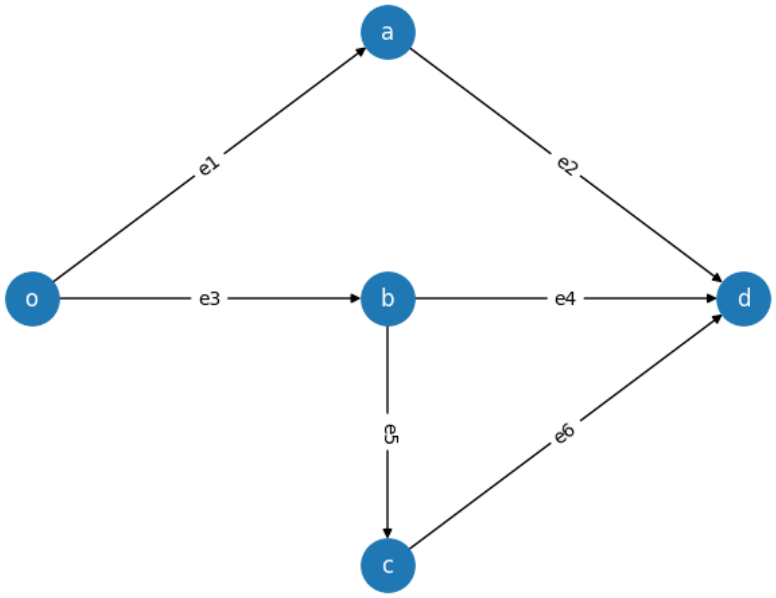

Figure 1: Exercise 1 Graph

With capacities $c_1 = c_3 = c_5$, $c_6 = 1$ and $c_2 = c_4 = 2$, the following points are to be solved:

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

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

(c) Given $x > 0$ extra units of capacity ($x \in \mathbb{Z}$). How should it be distributed in order to maximize the throughput that can be sent from o to d

Point a

A cut with minimal capacity is called a bottleneck because it allows the minimum amount of flow to pass through. As such, the min-cut capacity represents the aggregate capacity that needs to be removed to make d unreachable from o. In the case of the graph (1) the aggregate capacity to remove to get such effect is equal to 5.

Point b

The `networkx.algorithms.flows.max_flow` returns the the maximum admissible flow through the graph, τ^*, and its path. By observing that not all the links are used at their maximum capacity, we can safely say that removing the unused one from the links the maximum throughput would still be preserved. In particular for the graph (1), a flow with maximum throughput would take the path in Figure (2). As it is possible to see, link e_1 only uses 2/3 of capacity, and link e_5 only uses 1/3 of its capacity. By removing the unused capacity from these links, the maximal throughput $\tau^* = 5$ is preserved.

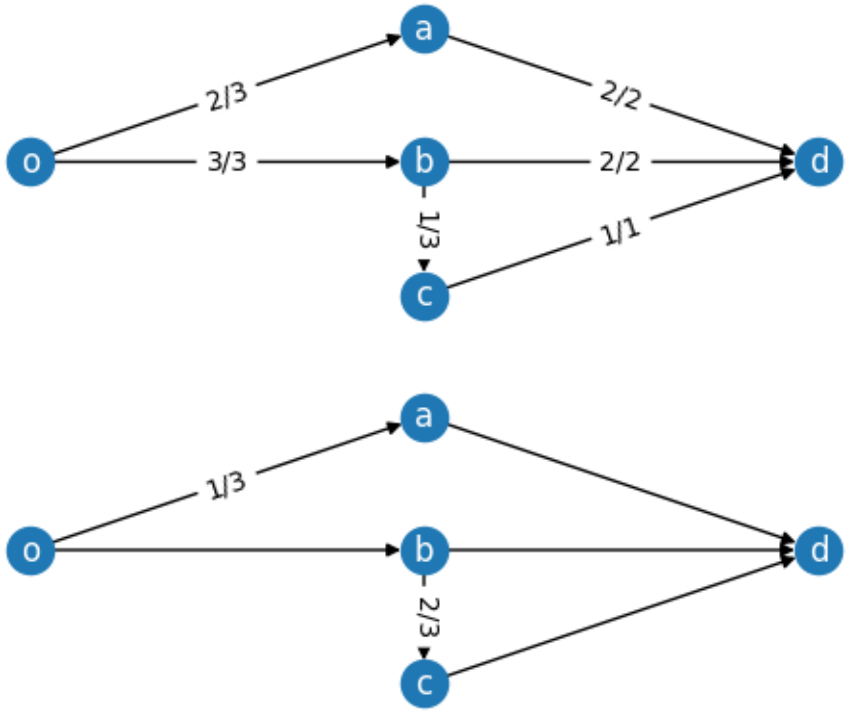

Figure 2: Above the path the flow would take. Below the removed capacity

Point c

Given x units of additional capacity, we can take advantage of the *Max-Flow Min-Cut Theorem* to distribute them in order to increase the maximum throughput. The theorem states that the maximum throughput τ^* equals the min-cut capacity of the graph c^*. We can then increase the min-cut capacity of the graph by adding the additional units of capacity to the least capable links of the cut. To solve the problem we can then use the following iterative algorithm:

1. Compute the current min-cut of the graph, obtaining the two sets of nodes $\mathcal{U}$,$\mathcal{U}^c \subseteq \mathcal{V}$
2. For each node $\theta \in \mathcal{U}$ check, for every node $\kappa \in \mathcal{U}^c$, if κ is a successor of it. In case it is compute the capacity c of the link and add the key-value pair $[(\theta, \kappa) : c]$ to a dictionary
3. For every key-value pair $[(\theta, \kappa) : c]$ in the dictionary add 1 capacity the link (θ, κ) and check if the resulting new min-cut capacity is greater or equal than the current min-cut capacity. If so, save the the link (θ^*, κ^*) that would increase the capacity and set the new highest value to the one just obtained.
4. Update the graph $\mathcal{G}$ by incrementing the capacity of the link (θ^*, κ^*) by 1
5. Compute the new min-cut of the graph and obtain the two new sets of nodes $\mathcal{U}$,$\mathcal{U}^c \subseteq \mathcal{V}$. Repeat from step 2 until the total capacity x has been added

The algorithm above can be described by the following pseudo-code

```
(U,Uc) = min_cut(G)
tau = min_cut_value(G)
taus = []
for i in [1,x]
    d = dictionary()
    for theta in U
```

```
        for kappa in Uc
            if theta.successor_of(kappa) then
                d[(theta, kappa)] = G[theta][kappa]['capacity']
            endif
        endfor
    endfor
    (theta_star, kappa_star) = d.keys[0]
    tau_max = tau
    for edge in d.keys
        G[edge[0]][edge[1]]['capacity'] +=1
        new_tau = min_cut_value(G)
        if new_tau >= tau_max then
            tau_max = new_tau
            (theta_star, kappa_star) = (edge[0],edge[1])
        endif
        G[edge[0]][edge[1]]['capacity'] -=1
    endfor
    G[theta_star][kappa_star]['capacity']+=1
    tau = max_flow_value(G)
    taus.append(tau)
endfor
```

This gives us the following evolution of the maximum flow τ^* in function of the added capacity x

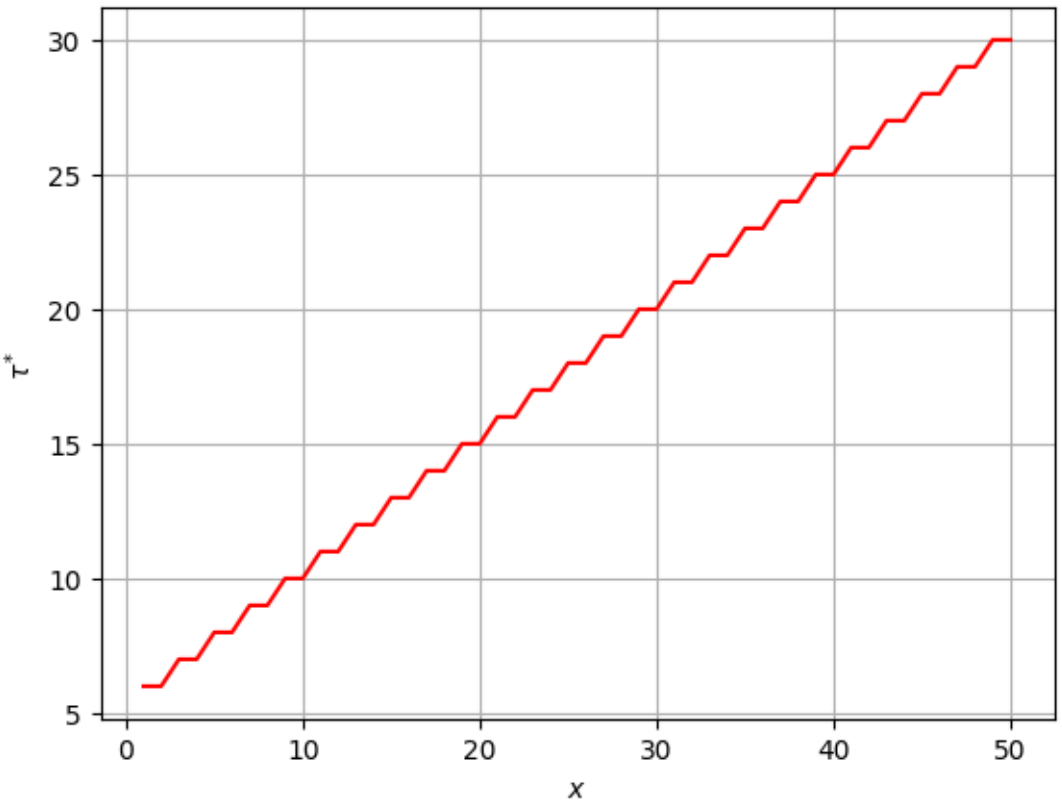

Figure 3: τ^* evolution in function of x

Exercise 2

There are a set of people $\{p_1, p_2, p_3, p_4\}$ and a set of books $\{b_1, b_2, b_3, b_4\}$. Each person is interested in a subset of books, specifically

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

This relation can be represented by the following bipartite graph, with nodes $p_i \in \mathcal{P}$ and $b_j \in \mathcal{B}$

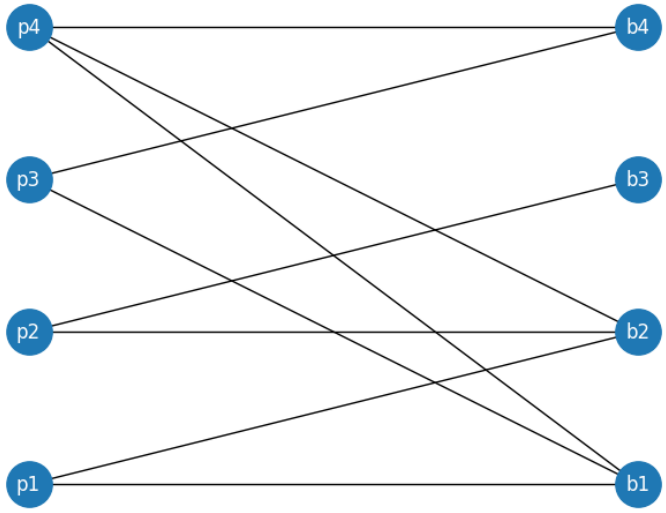

Figure 4: Exercise 2 Graph

The following points are to be solved:

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

(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.

(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?

Point a

In order to solve the matching problem we can modify the graph by adding two nodes o and d, with links coming from o to each node p_i with capacity 1, and links going from each node b_j to d with capacity 1. The links connecting nodes in $\mathcal{P}$ to nodes in $\mathcal{B}$ are assigned infinite capacity. We can now take advantage of the *Ford-Fulkerson Algorithm* to compute a perfect matching between $\mathcal{P}$ and $\mathcal{B}$.

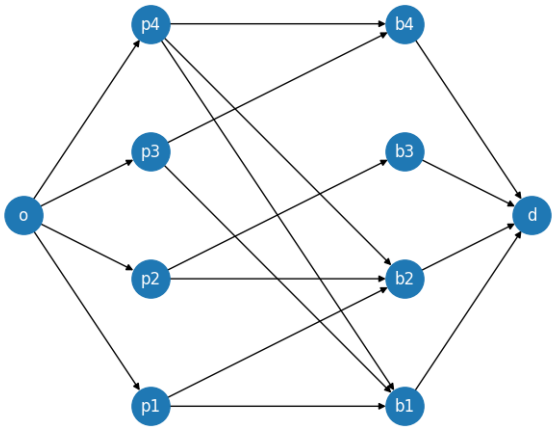

Figure 5: Graph used to compute the matching

The algorithm then gives us the following perfect matching

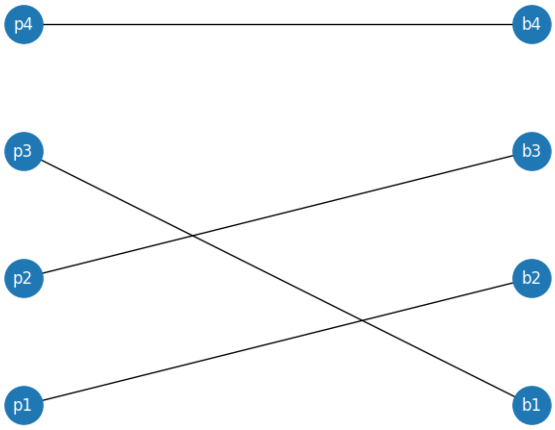

Figure 6: Perfect Matching

Point b

To take into account the presence of multiple copies of the books and that each person cannot take multiple copies of the same book, we can modify the algorithm described in the previous point as follows: the links from o to p_i are assigned a capacity equal to $|\mathcal{N}_i|$, and the nodes going from b_j to d are assigned a capacity equal to the number of books available for each node. We can once again use then the *Ford-Fulkerson Algorithm* of the newly obtained capacitated graph, which gives us the following result

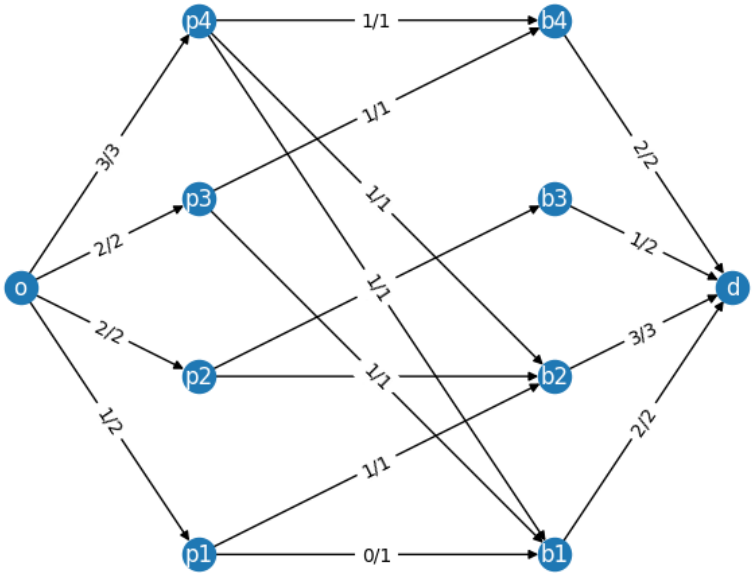

Figure 7: Multiple Books Matching

where each link going from $p_i \in \mathcal{P}$ to $b_j \in \mathcal{B}$ is labeled 1/1 if person p_i takes the book b_j, 0/1 otherwise.

Point c

To choose which books to sell and which to re-buy, we can take advantage of the results got in the previous point and notice that the link (p_1, b_1) is not used, and that only one copy of b_3 can be sold. As such, we can say that to maximize profits we should sell one unit of b_3 and restock it with one unit of b_1.

Exercise 3

The graph below represents the highway network in Los Angeles. Each node represents an intersection between highways. Each link $e_i \in \{e_1, \ldots, e_{28}\}$ has a maximum flow capacity c_{e_i}. Furthermore, each link has a minimum travelling time l_{e_i} which the drivers experience when the road is empty. For each link the corresponding delay function is

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

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

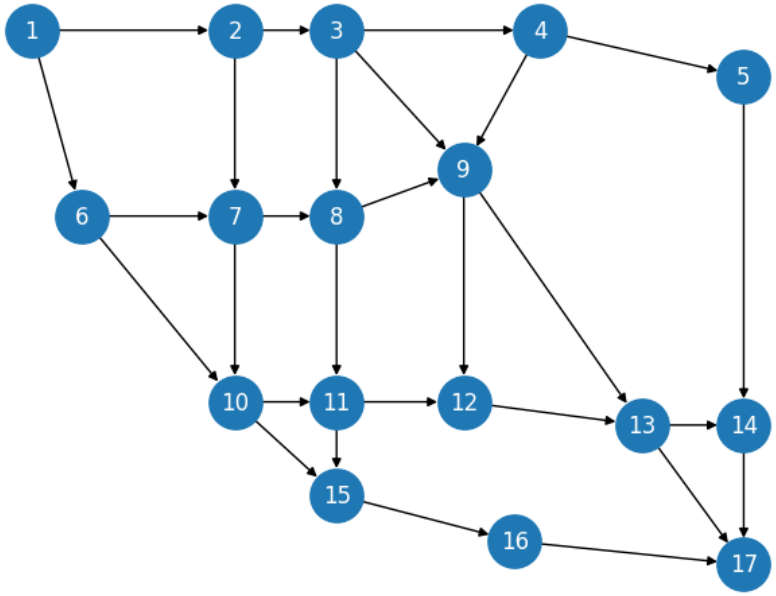

Figure 8: Graph representation of the highway network in Los Angeles

The following points are to be solved:

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

(b) Find maximum flow between node 1 and 17

(c) Given the flow vector, compute the external inflow η satisfying $Bf = \nu$

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

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

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

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

(f) Introduce tolls, such that the toll on link e is $\omega_e = \psi'_e(f^*_e) - \tau_e(f^*_e)$. For the considered $\psi_e(f_e)$, $\omega_e = f^*_e \tau'_e * (f^*_e)$, where f^*_e is the flow at system optimum. Now the delay on link e is given by $\tau_e(f_e) + \omega_e$. Compute the new Wardrop equilibrium and discuss the results

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

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

subject to the flow constraint. Compute the system optimum f^* for the cost above. Then construct a toll vector ω^* such that the Wardrop equilibrium $f^{(\omega^*)}$ coincides with f^*.

Point a

Assuming the throughput is equal to one, the shortest path can be computed by solving the following optimization problem. The final result has been obtained by just considering values close to 1, and discarding the ones close to 0.

$$\min_{f_e} \sum_{e \in \mathcal{E}} f_e \cdot l_e \quad \text{s.t. } Bf = \nu, \;\; f_e \geq 0$$

The optimization problem results in the following set of links

$$e = \{e_1, e_2, e_9, e_{12}, e_{25}\}$$

The correctness of the result is proven by using the `networkx.algorithms.shortest_path` algorithm, as it yields the same result.

Point b

The maximum flow can be computed using the `networkx.algorithms.flows.max_flow` function of `networkx`. The maximum flow possible is 22448.

Point c

The exogenous flow vector can be computed by simply applying the definition of the *flow-balance equation* $Bf = \nu$

$$\begin{bmatrix} 16.282 \\ 9.094 \\ 19.448 \\ 4.957 \\ -0.746 \\ 4.768 \\ 0.413 \\ -0.002 \\ -0.5671 \\ 1.169 \\ -0.005 \\ -7.131 \\ -0.380 \\ -7.412 \\ -7.810 \\ -3.430 \\ -23.544 \end{bmatrix} \cdot 10^3$$

As it is possible to see $\mathbf{1}^{\mathrm{T}} \nu = 0$.

Point d

Using the framework `cvxpy` it is possible to setup and solve a convex optimization problem. In particular we solved the optimization problem for the social optimum traffic assignment,

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

which resulted in the following flow vector:

$$\begin{bmatrix} 6.454 \\ 5.920 \\ 2.995 \\ 2.995 \\ 9.828 \\ 4.497 \\ 2.897 \\ 2.436 \\ 3.047 \\ 0.535 \\ 0.0000233693727 \\ 2.924 \\ 0.0000020768803 \\ 2.995 \\ 5.331 \\ 2.730 \\ 4.734 \\ 2.134 \\ 0.461 \\ 2.313 \\ 3.192 \\ 5.505 \\ 2.311 \\ 0.00000226487352 \\ 6.242 \\ 5.306 \\ 4.734 \\ 4.734 \end{bmatrix} \cdot 10^3$$

Point e

To retrieve the Wardrop equilibrium we first have to integrate the delay function τ_e

$$\int_0^{f_e} \tau_e(s)ds = -c_e l_e \log\left(1-\frac{f_e}{c_e}\right)$$

Which gives us the following cost function to minimize

$$\sum_{e\in\mathcal{E}} -c_e l_e \log\left(1-\frac{f_e}{c_e}\right)$$

We can again use the `cvxpy` framework to solve our optimization problem, which results in the following flow vector:

$$\begin{bmatrix} 6.532 \\ 6.532 \\ 2.207 \\ 2.207 \\ 9.750 \\ 4.493 \\ 2.714 \\ 2.204 \\ 3.343 \\ 0.000263674275 \\ 0.189 \\ 4.137 \\ 0.0000444865186 \\ 2.207 \\ 5.256 \\ 2.247 \\ 4.788 \\ 1.779 \\ 0.699 \\ 2.998 \\ 2.947 \\ 5.945 \\ 2.508 \\ 0.0000619467518 \\ 6.779 \\ 4.714 \\ 4.787 \\ 4.788 \end{bmatrix} \cdot 10^3$$

Point f

To solve the new optimization problem we first need to derive the expression of ω_e as

$$\omega_e = f_e^* \tau_e'(f_e^*) = f_e^* \left. \frac{d\tau_e}{df_e} \right|_{f_e = f_e^*} = f_e^* \frac{c_e l_e}{(c_e - f_e^*)^2}$$

The final cost function for the Wardrop equilibrium is thus

$$\sum_{e \in \mathcal{E}} \int_0^{f_e} (\tau_e(s) + \omega_e)\, ds = \sum_{e \in \mathcal{E}} \left[-c_e l_e \log\left(1 - \frac{f_e}{c_e}\right) + f_e f_e^* \frac{c_e l_e}{(c_e - f_e^*)^2} \right]$$

Which results in the following flow vector f^*:

$$\begin{bmatrix} 6.454 \\ 5.919 \\ 2.995 \\ 2.995 \\ 9.828 \\ 4.497 \\ 2.898 \\ 2.436 \\ 3.047 \\ 0.535 \\ 0.000316772 \\ 2.924 \\ 0.000066526707 \\ 2.995 \\ 5.331 \\ 2.731 \\ 4.734 \\ 2.134 \\ 0.462 \\ 2.312 \\ 3.193 \\ 5.505 \\ 2.310 \\ 0.0000860865914 \\ 6.242 \\ 5.306 \\ 4.734 \\ 4.734 \end{bmatrix} \cdot 10^3$$

Point g

For the last task we first have to compute the system optimum flow f^* corresponding to the new cost function, following the same steps in point d. Now, our goal is to derive a coherent vector of tolls w^* such that the Wardrop Equilibrium resembles the social optimum flow and to do that we take into account the formula here proposed:

$$\omega_e = \psi'_e\left(f_e^*\right) - \tau_e\left(f_e^*\right)$$

All we have to do is to calculate the derivative of the new cost function and iterate the same steps seen in point f, with the toll vector just drawn. For the sake of completeness, this derivative has been used:

$$\omega_e = f_e^* \tau'_e(f_e^*) - l_e = f_e^* \left.\frac{d\tau_e}{df_e}\right|_{f_e = f_e^*} - l_e = f_e^* \frac{c_e l_e}{(c_e - f_e^*)^2} - l_e$$

Eventually, the flow vectors corresponding to the social optimum and to the Wardrop equilibrium with tolls coincide.

$$\begin{bmatrix} 6.468 \\ 5.618 \\ 3.298 \\ 3.298 \\ 9.814 \\ 4.502 \\ 3.003 \\ 2.564 \\ 2.918 \\ 0.850 \\ 0.021 \\ 2.320 \\ 0.012 \\ 3.298 \\ 5.312 \\ 2.932 \\ 4.730 \\ 2.349 \\ 0.439 \\ 1.967 \\ 3.370 \\ 5.337 \\ 2.143 \\ 0.003803678 \\ 6.112 \\ 5.441 \\ 4.730 \\ 4.730 \end{bmatrix} \cdot 10^3, \quad \simeq \quad \begin{bmatrix} 6.468 \\ 5.619 \\ 3.297 \\ 3.297 \\ 9.814 \\ 4.502 \\ 3.003 \\ 2.564 \\ 2.918 \\ 0.849 \\ 0.00150264139 \\ 2.321 \\ 0.00092847672 \\ 3.297 \\ 5.312 \\ 2.931 \\ 4.730 \\ 2.349 \\ 0.430 \\ 1.967 \\ 3.370 \\ 5.337 \\ 2.143 \\ 0.000361410428 \\ 6.112 \\ 5.440 \\ 4.730 \\ 4.730 \end{bmatrix} \cdot 10^3$$