Homework 1

Network Dynamics and Learning
Politecnico di Torino

Hajali Bayramov - s288874

November 13, 2021

Acknowledgement

In this homework, collaboration with colleagues for changing ideas is done in Exercise 2 point b with H. Ghaemi, and in Exercise 3 points e, f with M. Aghalarov, F. Eterno. No extra help is used by other parties.

Exercise 1.

Consider unitary $o-d$ network flows on the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in Figure 1., and assume that each link l has integer capacity C_l.

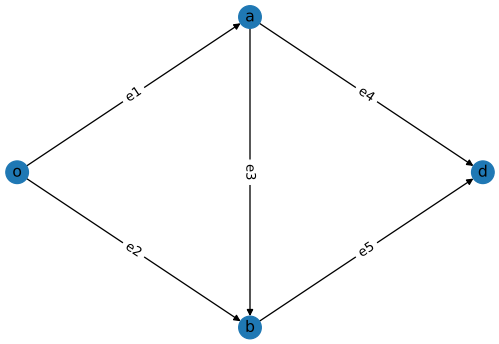

Figure 1:

(a) What is infimum of the total capacity that needs to be removed for no feasible unitary flows from o to d to exist?

"No feasible unitary flow" means there is no path exist between two nodes. Menger's theorem [1, Theorem 3.1] proves that edge-connectivity and node-connectivity, which represent the maximum number of walks between two nodes that do not share edges or nodes respectively, can tell us minimum number of edges or nodes that if removed causes disconnection between two nodes.

In the task we are asked to remove the capacity, hence the edge. Intuitively, we can see that there are two edge disjoint paths in this graph. (e_1, e_4) and (e_2, e_5). It is proved by *networkx.edge_disjoint_paths(G,'o','d')*[2] function of networkx library of python that there are 2 disjoint paths, thus minimum number of cuts needed to disconnect the origin from destination is 2.

(b) Assume that the link capacities are

$$C_1 = C_4 = 3, \quad C_2 = C_3 = C_5 = 2. \tag{1}$$

Where should 1 unit of additional capacity be allocated in order to maximize the feasible throughput from o to d? What is the maximal throughput?

For this exercise it is enough to implement *networkx.maximum_flow(G, 'o', 'd')* [2] function in order to find maximum throughput. First, maximum flow is found with the current values, which is 5. Then with the help of loop capacity in each node increased and maximum flow is found for updated capacity. At the end it is found that increasing any capacity by one does not change the overall result and it is still 5.

(c) Consider link capacities (1). Where should 2 units of additional capacity be allocated in order to maximize the feasible throughput from o to d? Compute all the optimal capacity allocations for this case and the optimal throughput.

Again, with the help of loop every possible combination of capacities is set and maximum flow is found. In this case there are 3 cases that we can improve the maximum flow by a unit (maximum flow = 6). Cases are following:

increase

- e_1, e_4
- e_1, e_5
- e_2, e_5

(d) Consider link capacities (1). Where should 4 units of additional capacity be allocated in order to maximize the feasible throughput from o to d? Compute all the optimal capacity allocations for this case. Among the optimal allocations, select the allocation that maximizes the sum of the cut capacities.

In this case, 4 inside loops are used to build all the combinations of capacities. At the end, maximum 2 units of increase in maximum throughput is achieved (maximum flow = 7) by 6 following changes:

increase

- 2 times e_1, e_4
- 2 times e_1 and once e_5
- 2 times e_1, e_4
- once e_1, e_2, e_4, e_5
- 2 times e_5 and once e_1, e_2
- 2 times e_2, e_5

Exercise 2.

Consider the following problem. 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

$$p_1 \to (b_1, b_2), p_2 \to (b_2, b_3), p_3 \to (b_1, b_4), p_4 \to (b_1, b_2, b_4)$$

(a) Represent the interest pattern by using a simple bipartite graph.

Interest pattern can be represented as simple bipartite graph in Figure 2. Here I represent persons with initials 1 (11, 12, 13, 14), books with initials 2 (21, 22, 23, 24) for simplicity. In general, V_0 is the set of people and V_1 and the set of books, and an edge (n_0, n_1) (with $n_0 \in V_0$, $n_1 \in V_1$) exists if the person n_0 is interested in the book n_1.

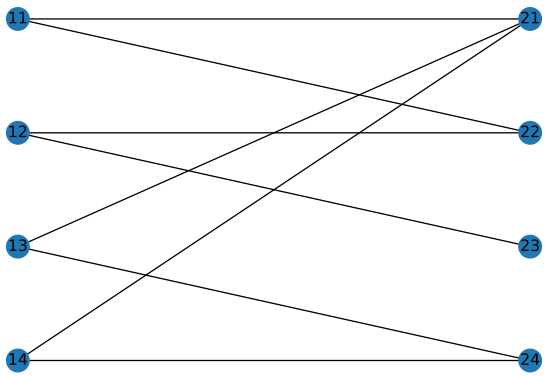

Figure 2: Interest pattern

(b) Exploit max-flow problems to establish whether there exists a perfect matching that assigns to every person a book of interest. If a perfect matching exists, find at least a perfect matching.

To find if the perfect matching exists sufficient and necessary condition in Hall's theorem[2, Theorem 1.1] can be verified. Also, in our case perfect matching can exist because there are equal number of people and books. To solve the problem however analogy is used between maximum flow and perfect matching to find a perfect matching by max-flow min-cut theorem[2].

Here making graph capacitated and directed and adding source (0) and target (3) nodes will help us to solve the problem. There is an edge added from node 0 to every node $n \in V_0$ with capacity 1, and from every node $n \in V_1$ to node 3. At the end we get the graph in Figure 3. If there exist a flow with throughput $|V_0|$ in new directed graph it means there exist a perfect matching, every person can get the book he/she wants. It can be explained by the fact that if every node n has 1 as incoming and outgoing flow 1, and maximum flow is equal to the number of books/persons Remains only finding maximum flow in python.

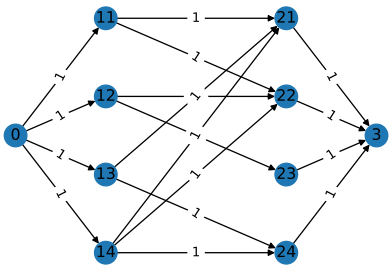

Figure 3: Directed capacitated graph G1

After implementing maximum flow function it becomes clear that perfect matching exists: (11, 22), (12, 23), (13, 21), (14, 24). It can be shown more clearly in Figure 4

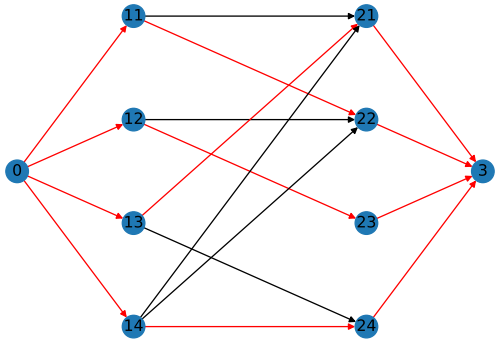

Figure 4: Flow in G1

(c) Assume now that there are multiple copies of book, specifically the distribution of the number of copies is (2, 3, 2, 2), and there is no constraint on the number of books that each person can take. The only constraint is that each person can not take more copies of the same book. Use the analogy with max-flow problems to establish how many books of interest can be assigned in total.

Here the problem is similar and analogy to the previous case is used. Since there are multiple copies of each book, outgoing flow should be equal to the copies respectively. However, since we have a constraint that each person can only take one copy of each book flow in edges between $n \in V_0$ and $n \in V_1$ should be 1, so no change. Lastly, each person can take more than one book, thus we do not care about capacity in incoming edges to $n \in V_0$. Practically it could be maximum 3 in this example, because there is just one person who is interested in 3 different books, it can be put $|V_0|$ however.

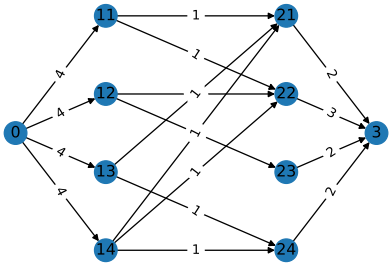

Figure 5: G1 with new capacities

Maximum flow is found 8 with the persons taking 1 (22), 2 (22, 23), 2 (21, 24), 3 (21, 22, 24) books respectively, also shown better in Figure 6.

(d) Starting from point (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?

As can be seen in Figure 6, person 11 also wants book 21 but since there are 2 copies of book 21 he/she does not get it. But book 23 is wanted by just one person and have 2 copies. Intuitively it is guessed that if bookstore sells one copy of 23 and buys 21 all the copies of books can be sold/rented.

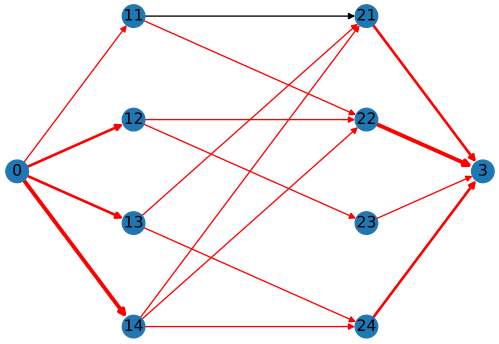

Figure 6: Flow in G1 after fresh copies of book

It can be proved by using a loop similar to previous exercise by changing capacities and applying maximum flow algorithm. As found intuitively, new capacities are [3, 3, 1, 2], which means selling 3rd book and buying 1st.

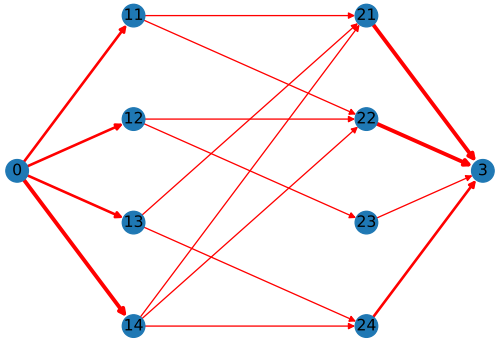

Figure 7: Flow in G1 after buy and sell

Exercise 3

We are given the highway network in Los Angeles. To simplify the problem, an approximate highway map is given in Figure 8, 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 $e_i \in \{e_1, \ldots, e_{28}\}$, 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, we introduce the delay function

$$d_e(f_e) = \frac{l_e}{1 - f_e/C_e}, \quad 0 \leq f_e < C_e \tag{2}$$

More detailed graphs are drawn below:

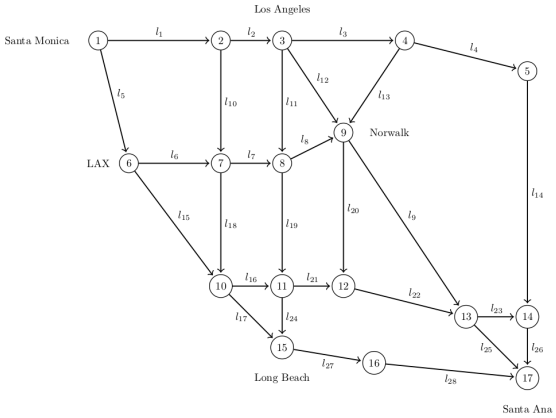

Figure 8: Some possible paths from Santa Monica (node 1) to Santa Ana (node 17).

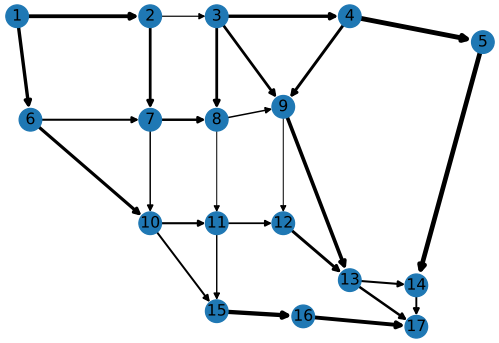

Figure 9: More travel time is shown with bolder lines

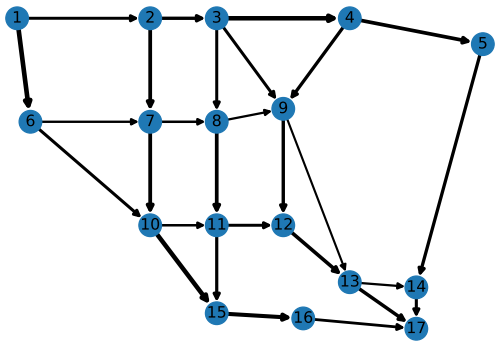

Figure 10: More capacity is shown with bolder lines

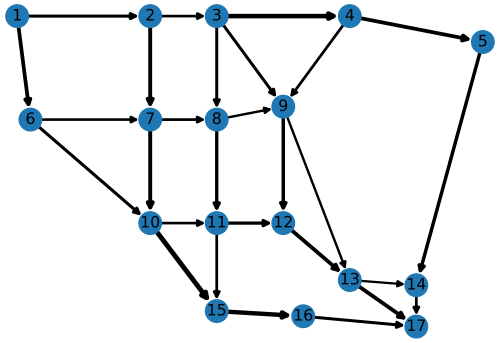

Figure 11: More flow is shown with bolder lines

(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.

In order to find the shortest path in an empty network we only need to consider travel time as weight of edges, because empty network means there is no congestion, hence no flow. *networkx.shortest_path(G,1,17, weight='weight')* returns path [1, 2, 3, 9, 13, 17], and *networkx.shortest_path_length(G,1,17, weight='weight')* gives us *0.532996* (where 'weight' is an edge attribute which represents travel time array).

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

Maximum flow is calculated the same way and based on same principles as in the previous example, with the help of *networkx.maximum_flow(G, 1, 17)*=22448

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

$\nu = [16806, 8570, 19448, 4957, -746, 4768, 413, -2, -5671, 1169, -5, -7131, -380, -7412, -7810, -3430, -23544]$

But for the following, we assume that the exogenous inflow is zero in all the nodes except for node 1, for which ν_1 has the same value computed in the point (c), and node 17, for which $\nu_1 7 = -\nu_1$. So $\nu = [16806, 0, \ldots, 0, -16806]$.

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

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

subject to the flow constraints.

For this task, python library CVXPY [3] is used. This library is built in order to solve convex optimization problems. Minimization problem is put in terms of simple mathematical forms inside the function with constraints $[Bf^* = \nu,\ f^* \geq 0,\ f^* \leq C_e]$ ($f^* \leq C_e$ and not $f^* < C_e$ because of library constraints). After solving

$$min \sum_{e\in\mathcal{E}} \left(\frac{l_e C_e}{1 - f_e/C_e} - l_e C_e \right)$$

minimal cost and f^* is found for Social Optimum (SO): Cost: 25943.623 Flow: [6642.199, 6058.938, 3132.328, 3132.326, 10163.801, 4638.317, 3006.341, 2542.635, 3131.544, 583.261, 0.015, 2926.596, 0.002, 3132.326, 5525.484, 2854.273, 4886.449, 2215.237, 463.721, 2337.688, 3317.991, 5655.679,

2373.107, 0.002, 6414.116, 5505.433, 4886.451, 4886.451]. Figure 12 shows flow in SO, where bolder edges represent more flow.

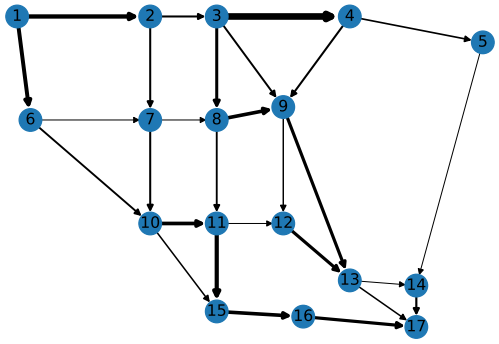

Figure 12: Flow in Social Optimum

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

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

First definite integral of the function is found by;

$$\sum_{e\in\mathcal{E}} \int_0^{f_e} d_e(s)ds = -l_e C_e \left(ln(C_e - s)\right) \Big|_0^f = \sum_{e\in\mathcal{E}} l_e C_e \left(ln(C_e) - ln(C_e - f_e)\right)$$

With the help of CVXPY library

$$min \sum_{e\in\mathcal{E}} l_e C_e \left(ln(C_e) - ln(C_e - f^{(0)})\right)$$

is solved and following results are found; Total delay: 26292.96330601271 Flow: [6715.649, 6715.645, 2367.409, 2367.409, 10090.351, 4645.394, 2803.846, 2283.557, 3418.48, 0.005, 176.816, 4171.42, 0., 2367.409, 5444.956, 2353.173, 4933.337, 1841.554, 697.104, 3036.497, 3050.276, 6086.773, 2586.513, 0.001, 6918.741, 4953.922, 4933.337, 4933.337], where Total delay = $\sum_{e\in\mathcal{E}} f_e^{(0)} d_e(f_e^{(0)})$. Figure 13 shows flow in Wardrop equilibrium, where bolder edges represent more flow.

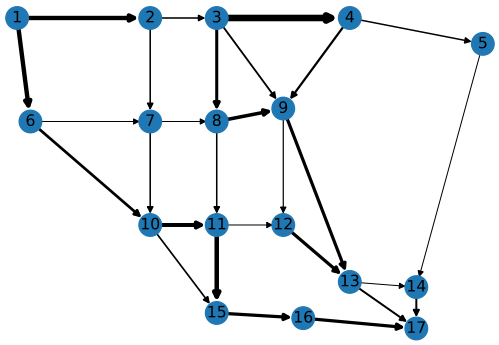

Figure 13: Flow in User Optimum (Wardrop equilibrium)

To make things more clear and compare both flows, PoA(0) (Price of Anarchy) associated to $f^{(0)}$. It shows us the total difference all the users make with their selfish decision to move away from social optimum. $PoA \geq 1$ and closer to 1 is closer the UO to SO.

$$PoA(0) = \frac{\sum_{e\in\mathcal{E}} f_e^{(0)} d_e(f_e^{(0)})}{min \sum_{e\in\mathcal{E}} f_e d_e(f_e)} = \frac{total\ delay\ in\ Wardrop\ equilibrium}{Cost\ in\ social\ optimum} \approx 1.0135$$

This number tells us that the flow in Wardrop equilibrium is a bit deviated from SO.

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

$$d_e'(f_e^*) = \frac{C_e l_e}{(f_e^* - C_e)^2}$$

Then ω_e is found as

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

With these on hand we can find new problem to optimize with tolls:

$$min \sum_{e\in\mathcal{E}} \left[l_e C_e \left(ln(C_e) - ln(C_e - f_e^{(\omega)}) \right) + f_e^{(\omega)} \omega_e \right]$$

After solving the convex problem with the same library that was used before the results are as: Total delay: 25943.622350312617 Flow: [6642.97446922, 6059.07643864, 3132.47228908, 3132.47200297, 10163.02541378, 4638.25874653, 3006.32655066, 2542.33807888, 3131.48796558, 583.89803058, 0.00114255, 2926.60300701, 0.00028611, 3132.47200297, 5524.76666724, 2854.22629974, 4886.37059396, 2215.83022646, 463.98961434, 2337.45340641, 3318.21555075, 5655.66895717, 2373.03536544, 0.00036332, 6414.1215573, 5505.50736841, 4886.37095728, 4886.37095728]

Figure 14 shows flow in Wardrop equilibrium with tolls, where bolder edges represent more flow.

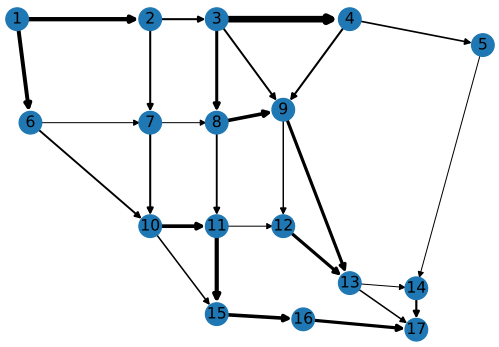

Figure 14: Flow in User Optimum (Wardrop equilibrium) with tolls constructed

This time however PoA in theory should become 1, because each driver pays back his 'selfishness' with little amount depending on what road he/she chooses (ω_e). In fact as is calculated,

$$PoA = 1.$$

(f) Instead of the total delay, let the cost be the total additional delay compared to the total delay in free flow be given by

$$c_e(f_e) = f_e(d_e(f_e) - l_e)$$

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

First of all, cost function becomes:

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

Then ω_e is found as

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

Then in order to find new Wardrop equilibrium new integral funcion is needed to be generated:

$$\sum_{e\in\mathcal{E}} \int_0^{f_e} (d_e(s) - l_e)ds + f_e\omega_e^* = \sum_{e\in\mathcal{E}} -l_e C_e \left(ln(C_e - s)\right) - l_e s \Big|_0^f + f_e\omega_e^* =$$

$$= \sum_{e\in\mathcal{E}} \left[l_e C_e \left(ln(C_e) - ln(C_e - f_e)\right) - l_e f_e + f_e\omega_e^*\right]$$

New System Optimum with the new equation is found by applying the same functions in library used in previous exercises. Here cost is found a bit less than previous Social Optimum, which is understandable because now cost function became total additional delay in the free flow. Results in social optimum are: Cost: 15095.513524607866 Flow: [6653.297, 5774.662, 3419.717, 3419.711, 10152.703, 4642.78, 3105.84, 2662.185, 3009.079, 878.634, 0.007, 2354.938, 0.006, 3419.711, 5509.923, 3043.693, 4881.805, 2415.575, 443.663, 2008.05, 3487.353, 5495.403, 2203.778, 0.002, 6300.704, 5623.489, 4881.807, 4881.807].

Results in new Wardrop equilibrium with constructed tolls are: Total delay: 15095.513256019116 Flow: [6653.13186254, 5775.41861223, 3419.48015483, 3419.47940279, 10152.86809007, 4642.42326506, 3105.49323142, 2661.73318847, 3009.16879616, 877.71325031, 0.00094138, 2355.93751602, 0.00075204, 3419.47940279, 5510.44482501, 3043.37621851, 4881.71189044, 2414.64328395, 443.76098434, 2008.50266036, 3487.13690893, 5495.6395693, 2204.0696741, 0.00029391, 6300.73869136, 5623.54907689, 4881.71218436, 4881.71218436].

Now PoA can be calculated based on the results above:

$$PoA(\omega^*) = \frac{\sum_{e\in\mathcal{E}} f_e^{(\omega^*)} d_e(f_e^{(\omega^*)})}{min \sum_{e\in\mathcal{E}} f_e^* d_e(f_e^*)} = 1$$

It proves that with the tolls constructed one can approach flow in the social optimum just like previous toll case

References

[1] G. Como and F. Fagnani, *Lecture notes on Network Dynamics*, 2021.

[2] Networkx library of python. [Online]. Available: https://networkx.org/documentation/stable/reference

[3] Cvxpy library of python. [Online]. Available: https://www.cvxpy.org/api_reference/cvxpy.problems.html