Homework I

M.Sc. Data Science and Engineering - AY 2022-2023

Network Dynamics and Learning

Professors: F. Fagnani, G. Como, L. Cianfanelli

Pietro Cagnasso
s300801
pietro.cagnasso@studenti.polito.it

Nicolò Vergaro[1]
s295633
nicolo.vergaro@studenti.polito.it

Exercise 1

The graph $\mathcal{G}_1$ on which the first exercise focuses is a directed graph with five nodes and six links, it contains a source node called o and a sink node called d. The capacities assigned to each link are $c_1 = c_3 = c_5 = 2,\ c_2 = c_4 = c_6 = 1$. The graph is shown in Fig. 1.

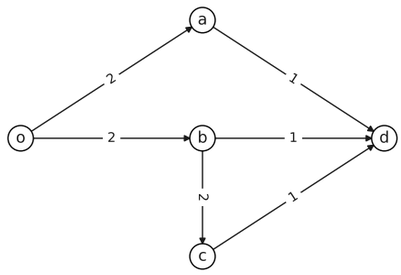

Figure 1: Graph $\mathcal{G}_1$ of the first exercise with capacities associated to the corresponding links.

Theoretical basis

Definition 1 (Distance) *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ we call dist(i, j) the length of the shortest path from i to j in $\mathcal{G}$, with the convention that dist$(i, j) = +\infty$ if no such path exists.*

Definition 2 (Exogenous net flow) *Given a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ we thus call a vector $\nu \in \mathbb{R}^{|\mathcal{V}|}$ satisfying the zero-sum constraint:*

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

This vector represents the quantities that come in and out the graph in every node.

Definition 3 (Flow) *Given a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the associated exogenous net flow vector ν we thus call a non-negative vector $f \in \mathbb{R}_+^{|\mathcal{E}|}$ whose entries satisfy the balance constraint:*

$$\nu_i + \sum_{e \in \mathcal{E}: k(e) = i} f_e = \sum_{e \in \mathcal{E}: \theta(e) = i} f_e, \forall i \in \mathcal{V}$$

using the node-link incidence matrix it can be wrote as:

$$Bf = \nu.$$

This vector represents the absolute quantities that move on each edge. The way B is constructed adds information on the direction of the flow on the edge.

Definition 4 (Throughput) *We thus call the total flow passing through the graph denoting it by τ, specifically a flow from a node o to a node d is denoted as τ_{od}.*

[1] With whom I collaborated on the entire homework.

Definition 5 (o-d Cut) *Given a capacitated multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c)$ we call thus way any partition of $\mathcal{V}$ such that $o \in \mathcal{U}$ and $d \in \mathcal{V} \setminus \mathcal{U}$. The capacity associated to a cut is computed as:*

$$c_{\mathcal{U}} = \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{V} \setminus \mathcal{U}} c_{ij}.$$

Theorem 1 (Max Flow Min Cut) *Given a capacitated multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c)$ and two distinct nodes o and d: we have that the optimal throughput τ^*_{od}, solution of the maximization of the throughput from o to d, is such that:*

$$\tau^*_{od} = c^*_{od} \triangleq \min_{\substack{\mathcal{U} \subseteq \mathcal{V} \\ o \in \mathcal{U},\ d \in \mathcal{V} \setminus \mathcal{U}}} c_{\mathcal{U}}.$$

This theorem allows us to compute in an easier way the maximum flow between nodes.

1.a

To find the minimum aggregate capacity that should be removed from the graph to have no feasible flow from o to d we have to look at the bottleneck of the system. As a direct consequence of Theorem 1 we have that the bottleneck of the system is the minimum cut, so the minimum aggregate capacity we need to remove is the capacity of the minimum cut. Using the method `networkx.minimum_cut` we can retrieve its capacity and the partitions in which this cut splits the graph (if there is more than one cut at minimum capacity it returns information on just one possible partition).

Results

As a result, we have that the minimum cut of this graph has a capacity 3 and the partitions in which it splits the graph are: {o, a, b, c} and {d}. Fig. 2 highlights the edges composing the minimum cut.

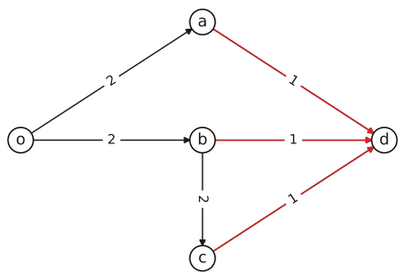

Figure 2: $\mathcal{G}_1$ with the edges that should be removed to stop the flow highlighted in red.

1.b

To find the maximum capacity that can be removed from the graph without affecting the maximum flow on it we can exploit once again Theorem 1. Since minimum cuts are already the bottleneck of the system, we can avoid considering them in the set of candidate edges of which we would like to reduce the capacity.

Since, as said above, the built-in method returns only information about one cut even if the graph has more than one minimum cut we need to define a function `find_minimum_cuts` that returns all the possible minimum cuts. To do so we produce all the possible combinations of internal nodes ($\mathcal{E} \setminus \{o, d\}$), each time add to the set the node o and save this cut (this represents a cut since a partition of nodes containing o uniquely defines a cut) associated with its capacity in a list. From this list return just the ones having minimum capacity.

To define the set of *candidate edges* we take all the edges that do not compose any minimum cut. To do so we make the union of all the edges composing a minimum cut returned by the function `find_minimum_cuts`: $\mathcal{A} = \bigcup_{e \in min\_cuts} e$. The candidate edges are defined as $\mathcal{C} = \mathcal{E} \setminus \mathcal{A}$ this set is returned by our function `candidate_edges`.

We define a secondary graph $\mathcal{G}'_1$ having the capacity on all the candidate edges, we just found, set to 1. On it we run a recursive algorithm `find_minimal_cap_rec` that does the following:

- Select a candidate edge e
- Increase e's capacity by 1
- Proceed in the recursion

- If the result of the recursion is locally optimum save it as optimum
- Reduce e's capacity by 1
- Return the locally optimum result.

Termination criteria for this recursive algorithm are: if the graph at this step of the recursion has the target flow (flow of $\mathcal{G}_1$) or if the amount of capacity added during the recursion is the same amount we remove from $\mathcal{G}_1$ producing $\mathcal{G}_1'$.

Results

Running this procedure above on $\mathcal{G}_1$ we found out that the maximum capacity we can remove from the graph without affecting the maximum flow is 2. In detail, we can reduce to 1 the capacity on the edges (o, a) and (b, c) as shown in Fig. 3.

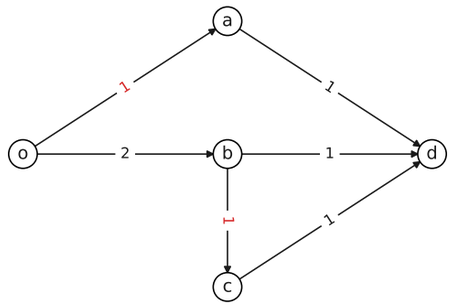

Figure 3: $\mathcal{G}_1$ with the minimum needed capacities to have the same flow of the original graph. The new capacities are highlighted in red.

1.c

Once more we exploit the fact that the minimum cuts are the bottleneck of the graph, a result coming from Theorem 1. We can use this result because if we want to increase the maximum flow on the graph we need to act exactly on the bottleneck.

We define an iterative method as follows:

- Find all minimal cuts with `find_minimum_cuts`
- For each edge in at least one minimum cut compute its absolute frequency of appearance counting the edges in which it appears
- Select from this set of edges the ones that have the highest appearance frequency
- From this selection of edges select the one with the lowest capacity and add 1 to its capacity.

Results

After the first four iterations in which the algorithm increases the capacity of (a, d), (b, d), (o, b), and (c, d) it stabilizes and creates a pattern: first (o, b), then (b, d). This produces the pattern shown in detail in Fig. 4-b. This kind of behavior increase-stop is also justified by the distance o-d in the graph which is, in fact, 2.

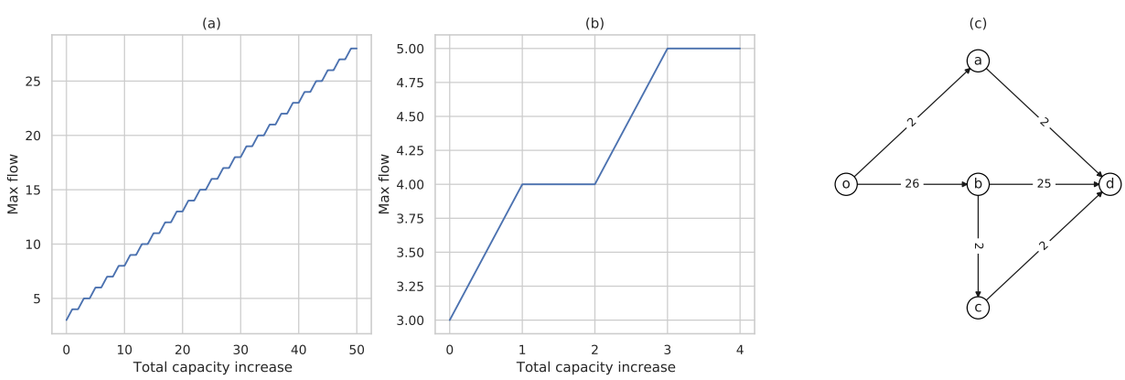

Figure 4: a: The trend of the flow adding up to 50 units of capacity. b: Detail of the pattern. c: Capacity distribution on $\mathcal{G}_1$ after the addition of 50 units of capacity.

Exercise 2

The graph $\mathcal{G}_2$ on which the second exercise focuses is a directed graph with 8 nodes and 9 links (it would be bipartite if the links were not directed). Each node p_i represents a person, while each node b_i represents a book. An edge `pi->bj` represents the fact that the person i is interested in the book j. The graph is shown in Fig. 5.

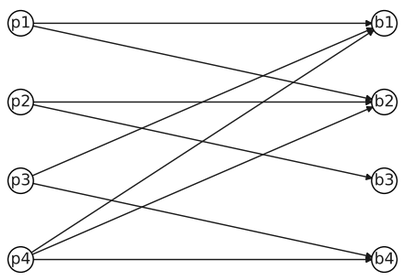

Figure 5: Graph $\mathcal{G}_2$ of the second exercise.

Theoretical basis

Definition 6 (Matching) *Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ without self-loops, we call thus way a subset of links $\mathcal{M} \subseteq \mathcal{E}$ that have no node in common. More precisely, we require that:*

$$(i,j) \in \mathcal{M} \iff (j,i) \in \mathcal{M} \text{ and } \{(i,j),(h,k)\} \in \mathcal{M} \implies \{i,j\} = \{j,k\} \text{ or } \{i,j\}\{h,k\} = \varnothing.$$

Under a matching $\mathcal{M}$ we call matched a node i if $(i,j) \in \mathcal{M}$ for some node j.

Definition 7 (Perfect matching) *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a matching $\mathcal{M}$ we call thus way a matching in which all nodes are matched.*

Theorem 2 (Hall's theorem) *For a simple bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ there exists a $\mathcal{V}_0$-perfect matching in the graph if ad only if:*

$$|\mathcal{U}| \leq |\mathcal{N}_{\mathcal{U}}|, \ \forall \mathcal{U} \subseteq \mathcal{V}_0$$

where $\mathcal{N}_{\mathcal{U}} = \bigcup_{i \in \mathcal{U}} \mathcal{N}_i$ is the neighborhood of $\mathcal{U}$ in $\mathcal{G}$.

2.a

Given the graph's similarity to a bipartite graph, to exploit Theorem 1 we can draw inspiration from the sufficiency proof of Theorem 2 by modifying the graph as follows:

- Add a node o with links of capacity 1 connecting it to nodes p_i
- Set the link capacity between nodes p_i and b_j to infinity (since in `networkx` links without capacity are considered with infinite capacity this is already the case)
- Add a node d with links of capacity 1 coming from nodes b_i.

The resulting graph is shown in Fig. 6-a.

In the same way, it is used in the proof of Hall's theorem the Max Flow Min Cut theorem gives us the maximum cardinality of a subset of $\mathcal{V}_0$ that can be matched, since we have four people and four books we need the maximum flow on this graph to be 4 to have a perfect matching. Using the method networkx.maximum_flow we can retrieve both the value of the maximum flow and its description.

Results

As a result, we have that the maximum flow is 4, so a perfect matching exists. A possible perfect matching is (p1, b2), (p2, b3), (p3, b1) and (p4, b4) as shown in Fig. 6-b.

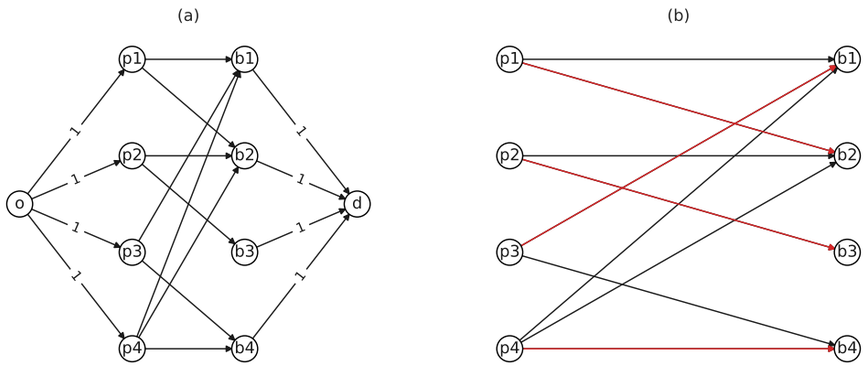

Figure 6: a: $\mathcal{G}_2$ modified drawing inspiration from Hall's theorem sufficiency proof. b: A perfect matching on $\mathcal{G}_2$.

2.b

Taking inspiration from the solution of the previous step, let us modify the graph again:

- To make sure that we have no prior bottlenecks we set the link capacities from node o to nodes p_i to infinity
- So that each person can take at most one copy of each book we set the link capacity between nodes p_i and b_j to 1
- To control the maximum number of copies of each allocated book we set the link capacity from nodes b_i to d equal to the number of available copies of the given book.

The resulting graph is shown in Fig. 7-a.

This kind of construction of capacities allows us to model at the same time the fact that each person can take an arbitrary number of *different* books and that the library has a fixed number of copies of each book. As in the previous step, the maximum flow on this graph represents the maximum amount of books we can assign with this kind of availability. Exploiting again the `networkx.maximum_flow` we get the maximum flow and its description.

Results

The resulting maximum flow on this graph is 8, a possible corresponding assignment is (p1, {b2}), (p2, {b2, b3}), (p3, {b1, b4}), (p4, {b1, b2, b4}) as shown in Fig. 7-b.

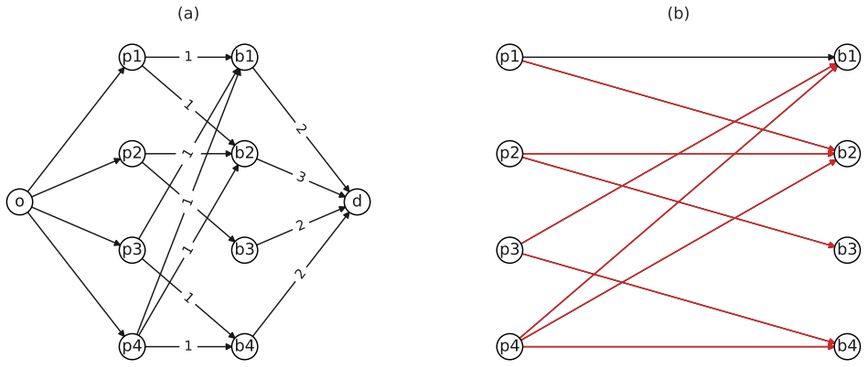

Figure 7: a: $\mathcal{G}_2$ modified to address the request b. b: A maximum assignment of books on $\mathcal{G}_2$ with the given constraints on copies.

2.c

Starting from the graph described in Fig. 7-a to maximize the number of assignable books we have to modify the capacities of links from b_i to d. In this specific case, we just need to find a book to sell and a book to buy.

To find this optimal books' availability we can set up an iterative method as follows:

- Select an edge e among the edges from b_i to d
- Decrease e's capacity by 1

- Cycle over all the other edges from b_i to d and try to add capacity 1 to each of them separately
- Save possible local optima (evaluated in terms of maximum flow as done in the previous request of the problem)
- Increase e's capacity by 1.

This algorithm can be easily extended to find how to sell and buy n books with a recursive algorithm, of which our implementation is a special case.

Another interesting solution involves the usage of in-degrees: since we set all the p_i to b_j edges' capacity to 1 we can reduce the capacity of each b_j to d edge exceeding the b_j's in-degree and adding up to the same quantity we removed to each b_j to d edge with a capacity lower than the in-degree.

Results

Running our algorithm on $\mathcal{G}_2$ we found out that there is an optimal choice of book for which we can assign all the books we have satisfying all customers' wishes. If we sell a copy of b3 and buy a copy of b1 we can assign 9. The modified graph with the new capacities is shown in Fig. 8.

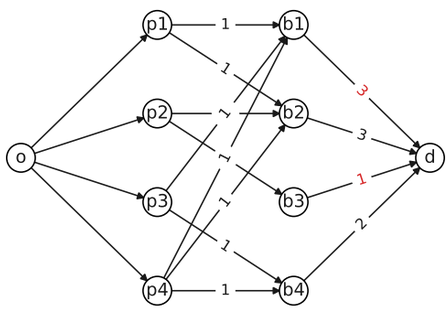

Figure 8: $\mathcal{G}_2$ with modified capacities maximizing the amount of assigned books highlighted in red.

Exercise 3

The graph $\mathcal{G}_3$ on which the third exercise focuses is a directed graph with 17 nodes and 28 links. It represents the highway network in Los Angeles, having node 0 Santa Monica and node 16 Santa Ana.

To construct the graph we read the node-link incidence matrix B provided. Each of its columns represents a link with a 1 on the node that is its tail and a -1 on the node that is its head. Reading column by column this matrix we extracted all the edges shown in Fig. 9.

N.B. For flow values associated with requests d-f please refer to the Jupyter notebook.

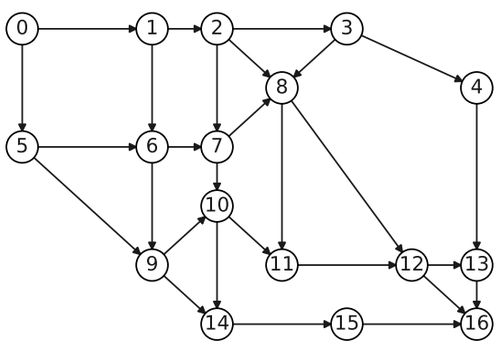

Figure 9: Graph $\mathcal{G}_3$ of the third exercise.

Theoretical basis

Definition 8 (Shortest Path Problem) *Given multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, c)$ let $o \neq d$ be two nodes such that d is reachable from o and let $\Gamma_{o,d}$ be the set of o-d paths. Let each edge $e \in \mathcal{E}$ be assigned a positive weight l_e representing its physical length and consider a linear cost*

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

This linear function does not take into account congestion effects having constant cost per unit of flow.

The flow f that minimizes the total cost function passes on the shortest path.

Definition 9 (Inflow) *Given multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and it exogenous net flow ν we define for each node i the inflow $\nu_i^+ = \max \nu_i, 0$ and so the total external inflow is defined as:*

$$TOT_{inflow} = \sum_{i \in \mathcal{V}} \nu_i^+ \ .$$

Definition 10 (System Optimum Traffic Assignment Problem) *In modeling congested transportation networks, links e in $\mathcal{E}$ represent roads and nodes represent junctions, while the link cost functions ψ_e are defined in terms of non-decreasing delay function τ_e, continuously differentiable in $[0, c_e)$ where $c_e = sup\{x \geq 0 : \tau_e(x) < +\infty\}$, representing the delay encountered by the users traversing link e when the flow on that link is f_e.*

The SO-TAP is the network flow optimization problem

$$\inf_{0 \leq f \leq c,\ Bf = \nu} \sum_{e \in \mathcal{E}} f_e \cdot \tau_e(f_e)$$

Definition 11 (Wardrop equilibrium) *We call thus way a network flow $f^{(0)} = A^{(o,d)} z$ where A is the link-path incidence matrix. z is such that $z \geq 0,\ \mathbb{1}^T z = \nu$ and for every $\gamma \in \Gamma_{o,d}$*

$$z_\gamma > 0 \implies T_\gamma(z) < T_{\tilde{\gamma}}(z)\ \forall \tilde{\gamma} \in \Gamma_{o,d}$$

where $T_\gamma(z)$ is the total travel time on a path γ.

This flow $f^{(0)}$ can be proven to be the solution to the user optimum traffic assignment problem, solution of

$$\inf_{0 \leq f \leq c,\ Bf = \nu} \sum_{e \in \mathcal{E}} \int_0^{f_e} \tau_e(s) ds \ .$$

Definition 12 (Price of Anarchy) *Given the solution of the SO-TAP f^* and the solution of the Wardrop equilibrium $f^{(\omega)}$ we call thus way:*

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

By construction this quantity is always a value greater or equal to 1.

Definition 13 (Tolls) *A way used by designers to force people to follow flow closer to social optimum is to add tolls ω. Tolls are designed as marginal cost tolls in a way that the solution of the Wardrop equilibrium and the solution of the SO-TAP are equivalent, this brings $PoA(\omega) = 1$.*

3.a

This exercise is a plain shortest path problem (Def. 8), in which the length l_e of each link is the time required to travel that stretch of highway at a fixed speed of 60mph. As a shortest path problem its cost function is

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

we set the exogenous net flow as $\nu = \delta^{(0)} - \delta^{(16)}$ to find the shortest of the paths from 0 to 16. So, the problem we deal with is:

$$f_{sp} = \underset{\substack{f \geq 0 \\ Bf = \nu}}{\operatorname{argmin}} \sum_{e \in \mathcal{E}} l_e f_e.$$

Using `cvxpy` library we can optimize the given function subject to those constraints. To extract the path followed by the flow we can use the method `numpy.isclose` to identify the edges with a flow close to 1.

Results

The result of this optimization step is that the shortest path is: $0 \to 1 \to 2 \to 8 \to 12 \to 16$ as shown in Fig. 10, with a total traveltime of 0.533.

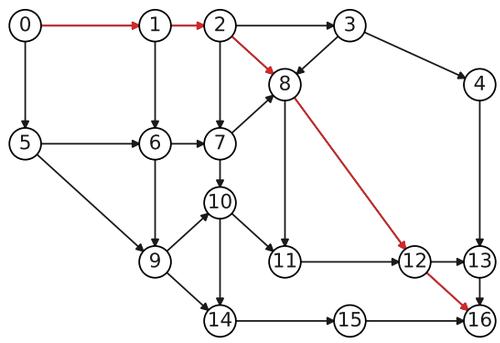

Figure 10: The shortest path on $\mathcal{G}_3$ for 0 to 16 is highlighted in red.

3.b

Exploiting Theorem 1 we can find the maximum flow form 0 to 16 using the method `networkx.maximum_flow`.

Results

The maximum flow identified on this graph is 22448.

3.c

To compute the exogenous net flow we can exploit the definition of feasible flow given the node-link incidence matrix (Def. 3). Having ν's value at our disposal, the computation of the total exogenous inflow (Def. 9) is direct:

$$\mathrm{TOT}_{inflow} = \sum_{i \in \mathcal{V}} \nu_i^+.$$

Results

Using the node-link incidence matrix and flow given we get:

$$\begin{aligned}\nu &= Bf \\ &= (16806, 8570, 19448, 4957, -746, 4768, 413, -2, \\ &\qquad -5671, 1169, -5, -7131, -380, -7412, -7810, -3430, -23544)^T\end{aligned}$$

and so the total exogenous inflow is 56131.

3.d

From now on the exogenous net flow we will consider is $\nu = 16806(\delta^{(0)} - \delta^{(16)})$.

Considering as delay function the well-known delay

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

and as cost function for the social optimum (Def. 10) $\psi_e(f_e) = \tau_e(f_e) f_e$, we define the problem to be solved as:

$$f^* = \underset{\substack{f \geq 0 \\ Bf = \nu}}{\operatorname{argmin}} \sum_{e \in \mathcal{E}} \left(\frac{l_e c_e}{1 - f_e/c_e} - l_e c_e \right).$$

Results

The optimal cost for this optimization problem resulted to be: 25943.62.

3.e

3.e.1 Wardrop equilibrium

Moving to the computation of the Wardrop equilibrium (Def. 11) we have now a different cost function

$$\begin{aligned}\psi_e(f_e) &= \int_0^{f_e} \tau_e(s)\ ds \\ &= \int_0^{f_e} \frac{l_e}{1 - s/c_e}\ ds \\ &= -l_e c_e \Big[\log(c_e - s) \Big]_0^{f_e} \\ &= -l_e c_e \log \frac{c_e - f_e}{c_e}.\end{aligned}$$

So, we the problem we have to deal with is:

$$f^{(0)} = \underset{\substack{f \geq 0 \\ Bf = \nu}}{\operatorname{argmin}} \sum_{e \in \mathcal{E}} -l_e c_e \log \frac{c_e - f_e}{c_e}.$$

Results

The optimal cost for this optimization problem resulted to be 15729.61; we've also computed the price of anarchy associated to this solution that was $PoA(0) = 1.013$.

3.e.2 Introduction of tolls

The introduction of tolls is used to restore the price of anarchy to 1. To do so we compute the marginal cost tolls as:

$$\omega^* : \psi'_{SO} = \psi'_{Wardrop} + \omega^*.$$

In this case the tolls were defined as $\omega_e^* = f_e^* \tau_e'(f_e^*)$, having as f^* the solution of the system optimum problem.

So, we can modify the cost function to include the tolls as

$$\begin{aligned}\psi_e(f_e, \omega_e^*) &= \int_0^{f_e} \tau_e(s) + \omega_e^*\ ds \\ &= -l_e c_e \log \frac{c_e - f_e}{c_e} + f_e f_e^* \frac{c_e l_e}{(c_e - f_e^*)^2},\end{aligned}$$

and define the new optimization problem as:

$$f^{(\omega^*)} = \underset{\substack{f \geq 0 \\ Bf = \nu}}{\operatorname{argmin}} \sum_{e \in \mathcal{E}} \left(-l_e c_e \log \frac{c_e - f_e}{c_e} + f_e f_e^* \frac{c_e l_e}{(c_e - f_e^*)^2} \right).$$

Results

The optimal cost for this optimization problem resulted to be 61885.97. To check the fact that the introduction of tolls achieved its goal we computed the price of anarchy that was, as expected, $PoA(\omega^*) = 0.9999$.

3.f

If we change the cost function used to compute the social optimum in the total additional delay compared to the total delay in free flow, namely

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

we can now compute a new system optimum flow f^* as the solution to this optimization problem:

$$f^* = \underset{\substack{f \geq 0 \\ Bf = \nu}}{\operatorname{argmin}} \sum_{e \in \mathcal{E}} \left(\frac{l_e c_e}{1 - f_e/c_e} - l_e c_e - l_e f_e \right).$$

Keeping the definition of the Wardrop equilibrium given in Def. 11, we can now construct the adequate tolls that can make the solution of the Wardrop equilibrium coincide with the new system optimum

$$\psi'_{SO} = \tau(f) + f\tau'(f) - l \qquad\qquad \psi'_{Wardrop} = \tau(f)$$

$$\omega^* : \ \psi'_{SO} = \psi'_{Wardrop} + \omega^* \implies \omega^*_e = f^*_e \tau'_e(f^*_e) - l_e.$$

Adding these tolls to the Wardrop equilibrium we get as cost function

$$\begin{aligned}\psi_e(f_e, \omega^*_e) &= \int_0^{f_e} \tau_e(s) + \omega^*_e \ ds \\ &= -l_e c_e \log \frac{c_e - f_e}{c_e} + f_e f^*_e \frac{c_e l_e}{(c_e - f_e)^2} - f_e l_e,\end{aligned}$$

and so the optimization problem becomes:

$$f^{(\omega^*)} = \underset{\substack{f \geq 0 \\ Bf = \nu}}{\operatorname{argmin}} \sum_{e \ in \mathcal{E}} -l_e c_e \log \frac{c_e - f_e}{c_e} + f_e f^*_e \frac{c_e l_e}{(c_e - f_e)^2} - f_e l_e.$$

Results

Optimal costs for social optimum and Wardrop with tolls were in this case 15095.51 and 50795.81, respectively. To check the fact that the introduction of tolls achieved its goal we computed the price of anarchy that was, as expected, $PoA(\omega^*) = 0.9999$.