Homework 1

Network Dynamics and Learning

Marco Tasca
Politecnico di Torino
Student id: s285174
s285174@studenti.polito.it

***Abstract*—The aim of this report is to present the solution to Homework 1. We used python to numerically solve the problems. For the sake of full reproducibility, the code is available at github.com/MarcTasca/network_dynamics.**

I. Exercise 1

Consider the network in fig. 1 with link capacities:

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

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) You are given $x > 0$ extra units of capacity $x \in Z$. 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 $x \geq 0$.

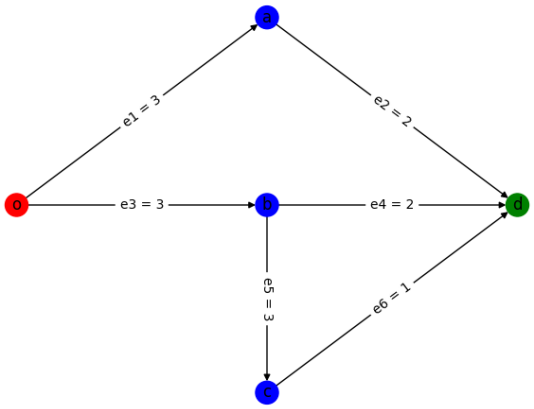

Fig. 1. Graph with capacities

A. Minimum removable capacity

The maximum flow is corresponding to the minimum aggregate capacity we need to subtract to isolate o and d.

The **Max-Flow Min-Cut Theorem** is a fundamental concept in graph theory that provides a powerful tool for solving network flow problems. One important application of this theorem is in finding the minimum capacity that needs to be removed from a graph to disconnect the origin (o) and destination (d) nodes.

The Max-Flow Min-Cut Theorem states that in any flow network (a directed graph with capacities assigned to edges), the maximum flow from the source to the sink is equal to the minimum cut capacity of the network.

There are different algorithms and theorem to compute it. In our case, we exploit the power of python library **networkx [1]**, which provides a function to directly calculate the minimum cut: **minimum_cut_value(G, weight = 'capacity')**.
In our case:

$$f_{max} = 5$$

B. Maximum removable capacity

To find the maximum aggregate capacity that can be removed without affecting the throughput, we can calculate the flow across the network, for each edge $e \in E$, from source (o) to sink (d). Then we can subtract the flow from each capacity, in order to find the residual unused capacity r_e:

$$r_e = c_e - f_e \text{ , for } e \in E$$

Finally, summing up the residual capacities, we find the maximum removable capacity r_{max}:

$$r_{max} = \sum_{e \in E} (c_e - f_e) = 3$$

To calculate the flow in each edge we use again **networkx [1]** library, this time with the function **maximum_flow(G, source, dest, capacity)**.

This function, uses the **Edmond-Karp algorithm**, an implementation of the **Ford-Fulkerson method** [2], a method for finding the maximum flow in a network. The basic idea is to iteratively find augmenting paths from the source to the sink and update the flow along these paths until no more augmenting paths exist.

Algorithm 1 Edmonds-Karp Algorithm

Input: Graph G, source s, sink t
Create residual graph G_f
Initialize $c_f(u,v) = c(u,v)$ for all $(u,v) \in G$
Initialize $f(u,v) = 0$ for all (u,v) in G
Initialize $max\_flow = 0$
while $\exists P$ (augmenting path) in the residual G_f **do**
    Find the minimum residual capacity along P : $c_f(P) = \min\{c_f(u,v) : (u,v) \in P\}$
    for each edge (u,v) in P **do**
        $f(u,v) \leftarrow f(u,v) + c_f(P)$
        $c_f(u,v) \leftarrow c_f(u,v) - c_f(P)$
        $c_f(v,u) \leftarrow c_f(v,u) + c_f(P)$
    end for
    $max\_flow \leftarrow max\_flow + c_f(P)$
end while
return max_flow

C. Maximize the throughput

In order to maximize the throughput of the network, allocating additional capacity, we need to carefully select the edges. We decided to follow a **greedy iterative procedure**, adding one unit of capacity each time:

1) Find the minimum cut partition (with the method described above).
2) Add one unit of capacity of one of the edges in the cut, chosen at random.
3) Repeat until all the capacity has been added.

We chose to add 10 units of capacity, in fig. 2 we can see the resulting plot, and in fig. 3 the final state of the graph.

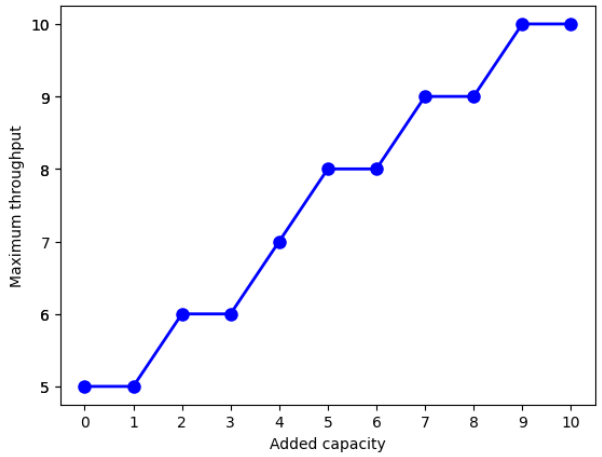

Fig. 2. maximum throughput

II. 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

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

Resulting in the bipartite graph modeling in fig. 4.

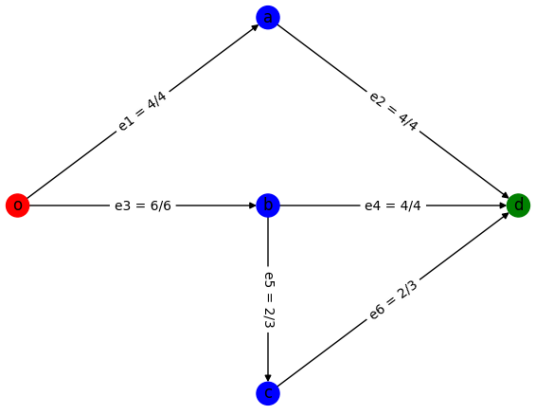

Fig. 3. maximum throughput

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

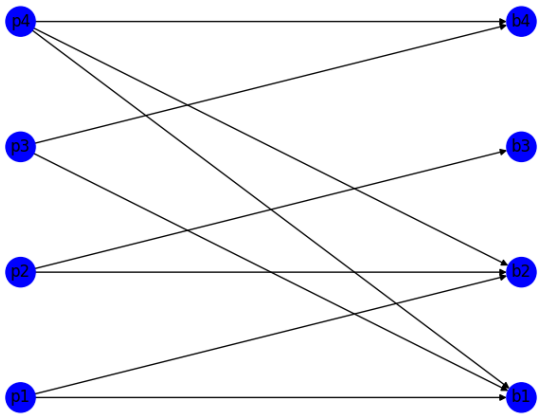

Fig. 4. Relationship between people and books

A. Find a perfect matching

In the context of graph theory, a perfect matching ensures that all nodes are paired, and it serves as a minimum-size link cover. The existence of such a matching can be proven by **Hall's theorem** [3], which states that for a simple bipartite graph $G = (V, E)$ with a subset $V_0 \subset V$:

$$V_0\text{-perfect matching} \iff \forall U \mid U \subseteq V_0 \to |U| \leq |N_U|$$

Where N_U is the set of the neighborhoods of U.

Drawing an analogy between perfect matching and maximal flow in the auxiliary network G, we find that a V_0-perfect matching exists if and only if there is a flow with a throughput equal to the cardinality of V_0 in the network G. Leveraging the Ford-Fulkerson algorithm [2], we can determine the maximum flow that can be sent through the network.

To identify the matching, we can expand the network into an auxiliary graph, denoted as G', and transform the problem into one of **maximum flow**.

The initial step in constructing G' involves introducing a source and a sink node to the original graph. Additionally, we add an edge from the source to each node in V_0 and from each node in V_1 to the sink. All newly added links are assigned the same capacity, set equal to 1. Due to the graph's topology, each node participating in any flow vector from the source to the sink has both incoming and outgoing flow equal to 1. Consequently, given that there are 4 nodes in V_0, the maximum flow is, at most, 4, as we can see in fig. 5.

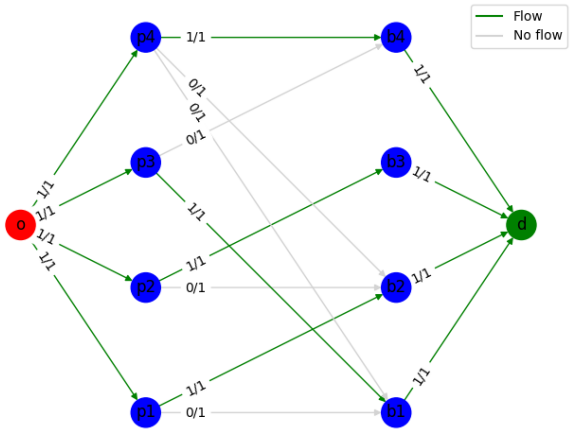

Fig. 5. The maximum flow in G' is equal to 4

Furthermore, we can show the resulting perfect matching, in fig. 6.

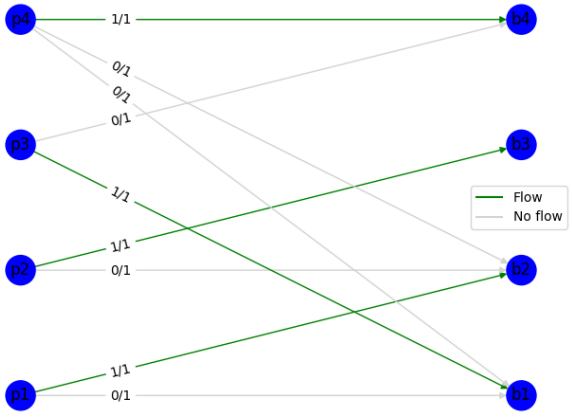

Fig. 6. TThe Resulting perfect matching

B. Find the maximum number of assigned books

The problem at hand can be effectively reformulated as a Max-Flow problem. This reduction is contingent upon comprehending how variations in book availabilities impact the capacity of G'. Specifically, the information regarding the number of copies available prompts an adjustment in the capacity of edges between nodes in V_1 and d, now specified as $(2, 3, 2, 2)$.

Given that each individual has the liberty to acquire an arbitrary number of distinct books, the capacities of edges between V_0 and V_1 remain set at 1 (reflecting the limitation to one copy per distinct book per person). However, it is imperative that each individual has the potential to acquire all four different books, should they choose to do so. Consequently, the weights of the links between o and the nodes in V_0 are assigned a value of plus infinity.

The solution to the Max-Flow problem can be see in fig. 7 resulting in a throughput of 8. This means that **the maximum number of books is 8**.

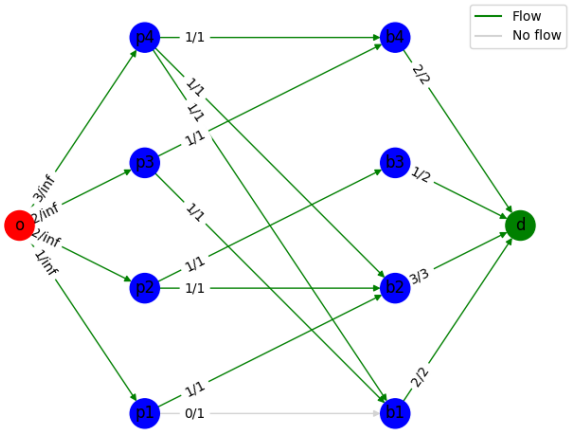

Fig. 7. The resulting maximum flow, after adapting the capacities to the new problem

C. Find the right books to sell and buy

The problem can be approached in at least two distinct manners. One approach involves rephrasing it as a **maximization problem**, where the goal is to optimize the throughput of the network without increasing the total capacity.

Alternatively, we adopted an explicit approach, undertaking a detailed examination of the actual graph configuration to optimize it. Our observation revealed that unsaturated edges between books and the sink signify that the corresponding book cannot be purchased, allowing for a reduction in their capacity until saturation. Simultaneously, if an edge between individuals and books remains unsaturated, it indicates that the corresponding book could be purchased. In response, we increase the capacity of the edge between that book and the

sink, thereby adding a copy for potential acquisition. after modifying capacities, we calculate again the maximum flow. **The maximum number of books is now 9**, and the resulting graph is in fig. 8.

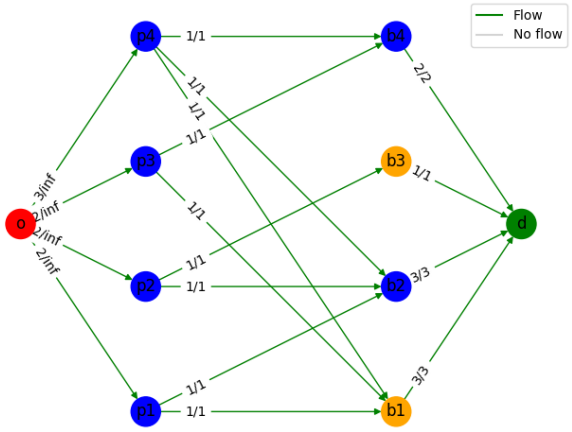

Fig. 8. After selling b3 and buying b1, the resulting throughput is 9

III. Exercise 3

We are given the highway network in Los Angeles. To simplify the problem, an approximate highway map is given in fig. 9, covering part of the real highway network.

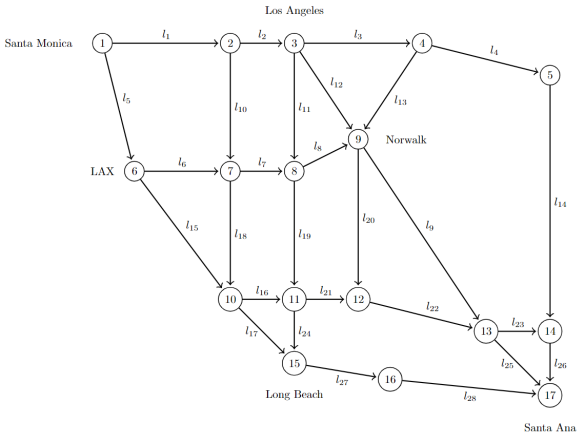

Fig. 9. Some possible paths from Santa Monica (node 1) to Santa Ana (node 17)

The node-link incidence matrix B, for this traffic network is given in the file traffic.mat, Each link $e_i \in \{e_1, ..., e_28\}$, 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.

First of all, we add edges and nodes starting from the B matrix, then we complete the graph with edge's attributes: **travelling times** and **capacities**. In fig. 10 the resulting graph.

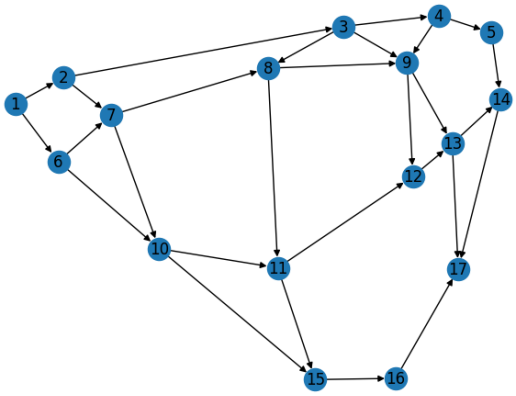

Fig. 10. The graph computed starting from the B matrix

A. Shortest Path Calculation

There are two ways to find the shortest weighted path from o (source) to d (destination):

1) Consider it as a flow optimization problem.
2) Use a greedy algorithm.

In our case, we adopted the second approach; we used a greedy algorithm developed by Dijkstra [4]. The shortest path is $[1, 2, 3, 9, 13, 17]$.

Algorithm 2 Dijkstra's Algorithm for Shortest Path

Require: Graph $G = (V, E)$ with vertices V and edges E, source vertex s
Ensure: Shortest paths and distances from s to all vertices
Initialize distance array $dist[]$ to ∞ for all vertices, and $dist[s]$ to 0
Create a priority queue Q and insert all vertices with their corresponding distances
while Q is not empty **do**
    Extract the vertex u with the minimum distance from Q
    for all neighbors v of u **do**
        $alt \leftarrow dist[u] + \text{weight}(u, v)$
        if $alt < dist[v]$ **then**
            $dist[v] \leftarrow alt$
            Update priority of v in Q to alt
        end if
    end for
end while

B. Find the maximum flow between 1 and 17

Solving the problem is possible through established algorithms like **Ford-Fulkerson** [2]. However, how previuosly described, **networkx [1]** presents an alternative off-the-shelf method with its **maximum_flow** function. Specifically, **the maximum flow** between the two nodes is determined to be **22448**.

C. Compute the external inflow ν

We can use the following equation:

$$Bf = \nu$$

The resulting vector is:

$$\nu = (16282, 9094, 19448, 4957, -746, 4768, 413, -2, -5671, 1169, -5, -7131, -380, -7412, -7810, -3430, -23544)$$

After, we compute the **exogenous inflow** useful for the next point (d).

$$f_{ex} = (\nu_1, 0, ..., 0, -\nu_1), \text{ with } f_{ex} \in \mathbb{R}^{|V|}$$

D. Find the social optimum f^*

In graph theory, the social optimum refers to a configuration or state in which the **overall welfare** or benefit of the entire network is **maximized**. In our scenario, the network represents roads, each with its **travel time**. The speed of each road is influenced by congestion, which is described by a **delay function**. The social optimum involves minimizing a **cost function** derived from this delay function. Essentially, it entails finding the optimal configuration of road usage to collectively **minimize the total travel time**. Consider the following delay function, for each link e:

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

and this cost function ψ_{so} to be minimized, in order to find f^*:

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

The complete math modeling:

$$\begin{aligned} \min_{\substack{Bf = \nu \\ f \in \mathbb{R}^{|E|} \\ f \geqslant 0 \\ f \leqslant c}} \sum_{e \in E} \left(\frac{l_e c_e}{1 - \frac{f_e}{c_e}} - l_e c_e \right) \end{aligned}$$

In order to solve this minimization problem, we'll exploit the power of the Python library **cvxpy**, able to solve **DCP** problems. The final optimized cost is:

$$\psi_{so}(f^*) = 23835.487$$

E. Find the Wardrop equilibrium $f^{(0)}$

In the same context, the **Wardrop equilibrium** refers to a state where travelers **selfishly select routes** to minimize their **individual** travel time. At equilibrium, no traveler can unilaterally decrease their travel time by switching to an alternative route. This condition is often described as a situation where the travel times on all used routes are equal, and any unused route has a longer travel time. The Wardrop equilibrium captures the idea that, over time, individuals naturally adjust their route choices to achieve a **balanced and stable distribution** of traffic across the network. In this way, the resulting $f^{(0)}$ will always be **less optimized** than f^*. This can be proven by computing the **Price of Anarchy**:

$$PoA = \frac{\psi_{so}(f^{(0)})}{\psi_{so}(f^*)} = \frac{\sum_{e \in E} f^{(0)} \tau_e(f^{(0)})}{\sum_{e \in E} f^* \tau_e(f^*)}$$

In fact, it happens to always be:

$$PoA \geqslant 1$$

To calculate the Wardrop equilibrium, our objective is to minimize the following ψ_{we} cost function:

$$\psi_{we}(f) = \sum_{e \in E} \int_0^{f^e} \tau_e(s) ds$$

Here is the complete modeling:

$$\begin{aligned} \min_{\substack{Bf = \nu \\ f \in \mathbb{R}^{|E|} \\ f \geqslant 0 \\ f \leqslant c}} \sum_{e \in E} \int_0^{f^e} \tau_e(s) ds \end{aligned}$$

We attempted to calculate the integral using **cvxpy**, by leveraging the fundamental theorem of integration, which involves breaking down the integral of a continuous function into a sum of rectangles. However, since we were unable to make it **Disciplined Convex Programming** (DCP), we opted to manually compute the primitive function first. Subsequently, we used **cvxpy** to address the minimization problem. The final cost function, after computing the primitive:

$$\psi_{so}(f) = \sum_{e \in E} c_e l_e \log(1 - f_e / c_e)$$

The final cost optimized using $f^{(0)}$ is:

$$\psi_{so}(f^{(0)}) = 24162.201$$

Meanwhile the Price of Anarchy is:

$$PoA = \frac{\psi_{so}(f^{(0)})}{\psi_{so}(f^*)} = 1.014$$

The final maximum flow can be observed in fig. 12.

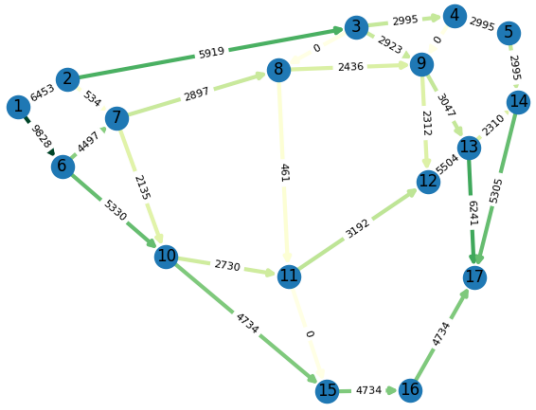

Fig. 11. Maximum flow of social optimum in point (d).

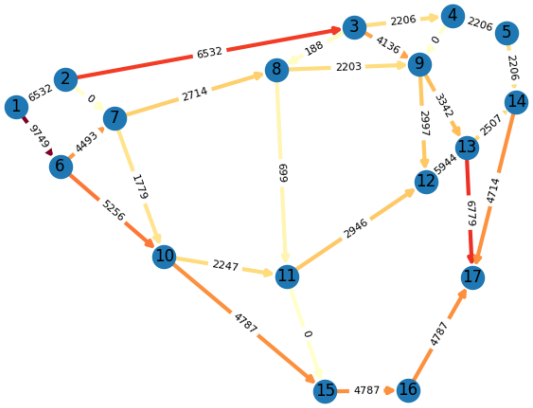

Fig. 12. Maximum flow of Wardrop equilibrium in point (e).

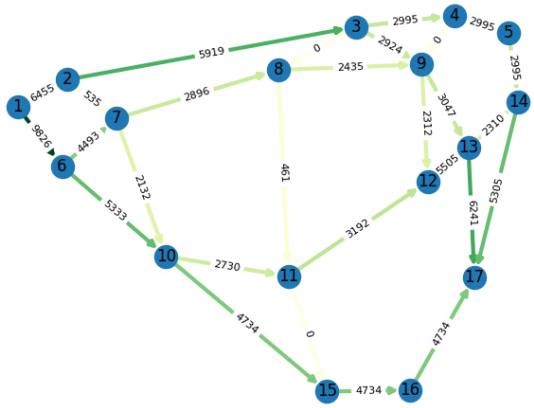

Fig. 13. Maximum flow of Wardrop equilibrium with tolls in point (f).

F. Wardrop equilibrium with tolls

Tolls in the **Wardrop equilibrium alter route choices** by introducing a **cost factor**.The use of tolls in the context of **Wardrop equilibrium** can lead to the **social optimum** by influencing travelers to choose routes that collectively minimize overall congestion and travel time. Tolls act as incentives, guiding individuals to make choices that align with the broader goal of optimizing traffic flow and reducing congestion across the entire road network.It's really important to choose a vector of tolls ω^* which allows the new **Wardop equilibrium** $f^{(\omega^*)}$ to coincide with the **social optimum** f^*. This is true when the **Price of Anarchy** tends to 1:

$$PoA \cong 1$$

Here is the general function describing the toll for each edge e:

$$\omega_e = \psi'_e(f^*_e) - \tau_e(f^*_e) \text{ , where } \begin{cases} \psi_e(f_e) \text{ is the cost function} \\ f^*_e \text{ is the optimal flow} \end{cases}$$

In our case:

$$\omega_e = f^*_e \tau'_e(f^*_e)$$

Consider the new delay function τ^1_e for edge e:

$$\tau^1_e(f_e) = \tau_e(f_e) + \omega_e$$

and so the new cost function:

$$\psi_{we}(\vec{f}) = \sum_{e \in E} \int_0^{f_e} \tau_e(s) + \omega_e ds$$

which, after computing the primitive, becomes:

$$\psi_{we}(\vec{f}) = \sum_{e \in E} \{\omega_e f_e - c_e l_e \log(1 - f_e/c_e)\}$$

The final cost optimized using $f^{(0)}$ is:

$$\psi_{so}(f^{(0)}) = 23835.490$$

Meanwhile the Price of Anarchy is:

$$PoA = \frac{\psi_{so}(f^{(0)})}{\psi_{so}(f^*)} = 1.000$$

The final maximum flow can be observed in fig. 13.

G. Recalculate, with a different cost function

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:

$$\psi_{so}(f) = \sum_{e \in E} f_e(\tau_e(f_e) - l_e)$$

1) Compute the new f^*: The new complete math modeling:

$$\begin{array}{l} \min \\ Bf = \nu \\ f \in \mathbb{R}^{|E|} \\ f \geqslant 0 \\ f \leqslant c \end{array} \sum_{e \in E} \left(\frac{l_e c_e}{1 - f_e/c_e} - l_e c_e - f_e l_e\right)$$

As always, solved using **cvxpy**.

$$\psi_{so}(f^*) = 13334.305$$

The final maximum flow can be observed in fig. 14.

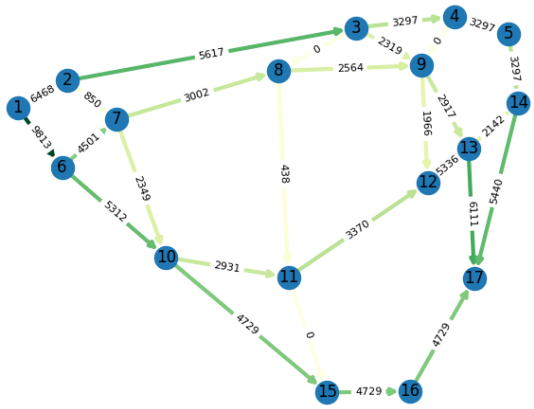

Fig. 14. Maximum flow of social equilibrium with the new cost function in point (g.1).

2) Compute the new ω^*: From the general function to compute ω described before, we can derive our:

$$\omega_e^* = \psi_e'(f_e^*) - \tau_e(f_e^*) = \frac{d}{df_e^*} f_e^*(\tau_e(f_e^*) - l_e) - \tau_e(f_e^*) =$$
$$= \frac{d}{df_e^*} f_e^*(\frac{l_e}{1-f_e^*/c_e} - l_e) - \frac{l_e}{1-f_e^*/c_e} = \frac{l_e(c_e^2 - 3c_e f_e^* + f_e^{*,2})}{(c_e - f_e^*)^2}$$

In the end we can say:

$$\omega_e^* = \frac{l_e(c_e^2 - 3c_e f_e^* + f_e^{*,2})}{(c_e - f_e^*)^2}$$

3) Compute the new $f^{(0)}$: In order to compute the new Wadrop equilibrium we have to define the new cost function:

$$\psi_{we,e}(f_e) = \int_0^{f_e} (\tau_e(s) - l_e + \omega_e) ds$$

After computing the primitive:

$$\psi_{we,e}(f_e) = \omega_e f_e - l_e f_e - c_e l_e \log(1 - f_e/c_e)$$

Final results:

$$\psi_{so}(f^{(0)}) = 13334.315$$

$$PoA = 1.000$$

Final flow vivible in fig. 15.

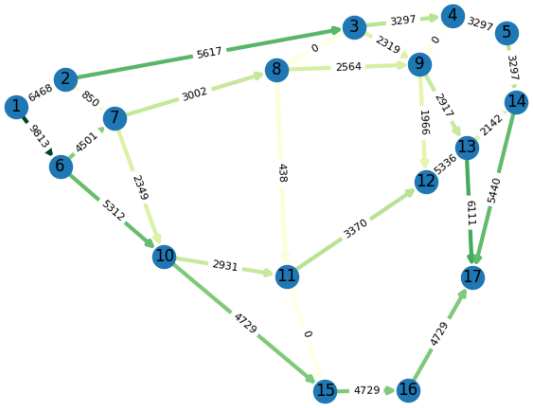

Fig. 15. Maximum flow of Wardrop equilibrium with tolls and the new cost function in point (g.3).

REFERENCES

[1] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15, 2008.

[2] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, p. 399–404, 1956.

[3] P. Hall, "On representatives of subsets," *Journal of the London Mathematical Society*, vol. 10, no. 1, pp. 26–30, 1935.

[4] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.