Homework 1
Network Dynamics and Learning

1st Anna Roma
Politecnico di Torino
Matricola: 345819
anna.roma@studenti.polito.it

2nd Stefano Giulianelli
Politecnico di Torino
Matricola: 347911
stefano.giulianellli@studenti.polito.it

***Abstract*—This report investigates three core problems in network dynamics. First, a 5-node flow network is analyzed to determine its minimum cut and the optimal strategy for throughput maximization via capacity augmentation. Second, Katz and PageRank centrality algorithms are implemented and compared on a 15-node graph, identifying structural hubs and evaluating parameter sensitivity. Finally, the Los Angeles highway network is modeled to compute the shortest path , maximum flow , social optimum , and Wardrop equilibrium. This analysis demonstrates how marginal-cost tolls can align the user equilibrium with the social optimum, mitigating the inefficiency of selfish routing.**

I. Exercise I

We consider a directed network with 5 nodes and link capacities, shown in Figure 1, with capacity vector

$$c = [c_1, c_2, c_3, c_4, c_5, c_6] = [3, 3, 3, 3, 2, 1]. \tag{1}$$

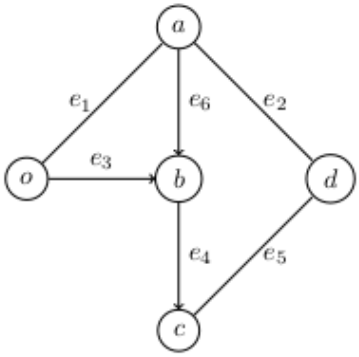


Fig. 1: Six-node Graph *G* from Exercise 1.

A. Find the minimum capacity

We start by computing the capacity of all s–t cuts separating the source node o from the destination node d, and determining the minimum total capacity whose removal prevents the existence of any feasible flow from o to d. To do so, we enumerate every admissible partition (U, V) with $o \in U$ and $d \in V$, compute for each cut the sum of the capacities of all directed edges from U to V and report the resulting values in a concise table (Table I-C).

To address this problem, we implemented a *MATLAB* script `es1.m`, where the directed graph (1) and the link capacities (1) are first defined. We also employed two auxiliary functions:

- **`compute_cuts.m`** constructs all admissible cuts. This is achieved by enumerating all possible combinations of intermediate nodes, that is, all subsets of nodes excluding the source and destination. Each subset defines a set U that always contains the source node, while its complement defines the set V, which necessarily contains the destination node. For each pair (U, V), the function checks for the presence of directed edges from U to V and sums their capacities. These values are stored in a vector collecting the capacity of each cut. The function then outputs a table listing, for every cut, the corresponding sets U and V, along with the computed total capacity.
- **`create_cut_r.m`** is used to generate and display all non-empty combinations of the intermediate nodes. This allows us to verify that the cut enumeration algorithm correctly accounts for all possible partitions of the node set.

Upon execution, the script returns the complete set of cuts in the network together with their associated capacities. The smallest among these values represents the minimum cut capacity, which corresponds to the minimal amount of capacity that must be removed in order to eliminate all feasible flows from o to d.

```
Combined results table:
      U                V          C
  ____________    _____________    _

  {[       1]}    {[5 2 3 4]}     6
  {[     1 2]}    {[  5 3 4]}     7
  {[     1 3]}    {[  5 2 4]}     6
  {[     1 4]}    {[  5 2 3]}     8
  {[  1 2 3]}     {[     5 4]}     6
  {[  1 2 4]}     {[     5 3]}     9
  {[  1 3 4]}     {[     5 2]}     5
  {[1 2 3 4]}     {[       5]}     5

  Minimum cut value: 5
```

Finally, from the computed results, the smallest cut value is identified as 5. This value represents the minimum amount of capacity that must be removed in order to disconnect the source o from the destination d, thus preventing any feasible o–d flow. In other words, the minimum cut capacity quantifies

the bottleneck of the network: it is the smallest total capacity of edges whose removal eliminates all possible paths from o to d. It is also possible to observe that two distinct cuts attain this minimum value. These cuts are reported below, together with the edges contributing to the total capacity of 5:

- **Cut 1:** $U = \{1, 3, 4\},\ \ V = \{5, 2\}$ The edges crossing the cut are:
  - $o \rightarrow a$ (capacity 3)
  - $c \rightarrow d$ (capacity 2)
- **Cut 2:** $U = \{1, 2, 3, 4\},\ \ V = \{5\}$ The edges crossing the cut are:
  - $a \rightarrow d$ (capacity 3)
  - $c \rightarrow d$ (capacity 2)

B. Capacity augmentation and throughput as a function of x

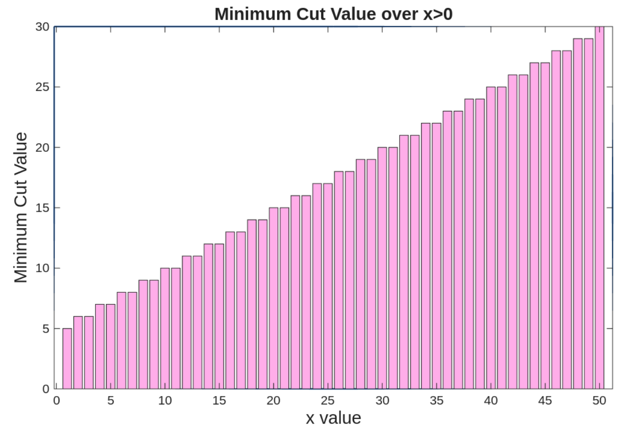
We now study how to allocate $x > 0$ additional integer units of capacity among the links of the network in Fig. 1, with the goal of maximizing the achievable throughput from the source o to the sink d. To do so, we implemented a new *MATLAB function* called `extra_units_capacity.m` that takes as input the directed graph G, node vector `nodes` and parameter `maxIter`. This function outputs the vector `min_cut` containing the evolution of the minimum cut value over time. At each iteration we compute the minimum cuts using the updated capacity vector. For each minimum cut, we collect the edges crossing it (via `findedge` on all pairs in $U \times V$) and decide to which edge we should give the extra capacity. To make this decision, at first, we increment the weight of the first such edge by 1. If no edge is common to all min-cuts, we choose the edge with the highest support (appearing most frequently across the min-cuts) and increment it.
The procedure stops either when `maxIter` is reached. For logging, we print the minimum-cut value at each iteration.

The plots in Figure 2 show that the minimum-cut value increases as x grows. The growth is not linear: when $x = 50$, the maximum value reached is approximately 30, whereas for $x = 100$ the minimum-cut value reaches about 55. This suggests diminishing returns, with early increments producing larger improvements than later ones: as x increases, the minimum cut value continues to grow, but the slope becomes smaller. This shows that the curve grows more slowly for larger x, confirming the presence of diminishing returns.

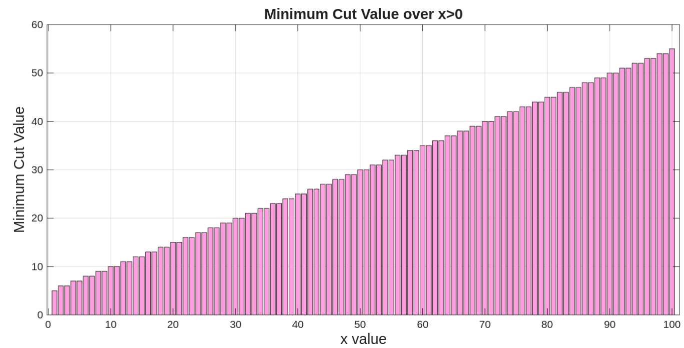
C. Adding a new link e_8

We are now allowed to add to the network a directed link e_8 with initial capacity $c_8 = 1$, together with $x > 0$ extra units of capacity ($x \in \mathbb{Z}$). This means that we may arbitrarily choose where to connect this new unit-capacity edge. We perform several tests and compare the resulting behaviours.

As a first attempt, we connect node b to node d, assigning the new edge an initial capacity of 1. We then compute all cuts using the function `compute_cuts`, which returns a minimum-cut value increased to 6.



(a) Minimum cut value as x increases up to 50.



(b) Minimum cut value as x increases up to 100.

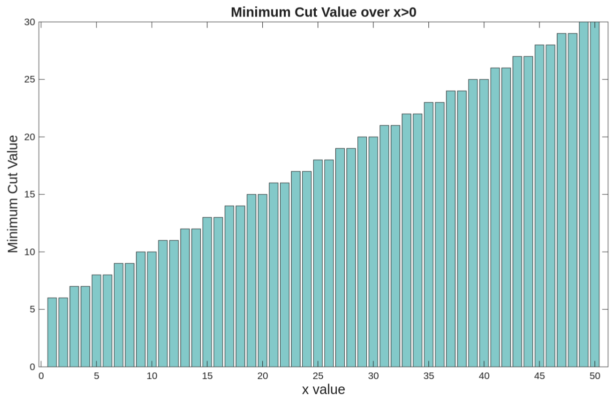
Fig. 2: Evolution of the minimum-cut value for different capacity increments.

```
   Combined results table:
        U                   V            C
   _____________       _____________     _

   {[        1]}       {[5  2  3  4]}    6
   {[     1  2]}       {[   5  3  4]}    7
   {[     1  3]}       {[   5  2  4]}    7
   {[     1  4]}       {[   5  2  3]}    8
   {[  1  2  3]}       {[      5  4]}    7
   {[  1  2  4]}       {[      5  3]}    9
   {[  1  3  4]}       {[      5  2]}    6
   {[1  2  3  4]}      {[         5]}    6

Minimum cut value: 6
```

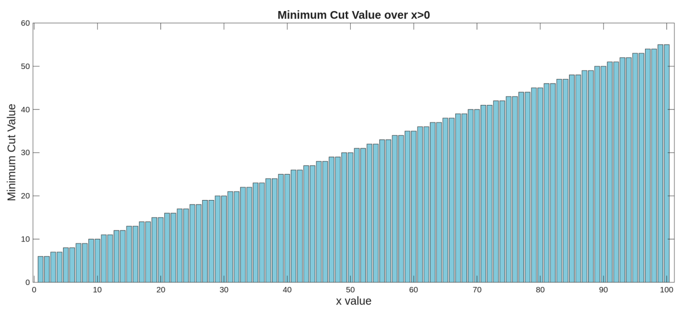
Next, we allow $x > 0$ additional units of capacity to be allocated, and as in the previous section we call the function `extra_units_capacity`. We plot the resulting minimum-cut values for a maximum of $x = 50$ and $x = 100$ and we compare the behaviour of the network with the case in which the new edge e_8 was not present.

From Fig. 4 we observe that, compared with the results in Fig. 2, there is little improvement. Therefore, adding an edge in this position does not significantly enhance the behaviour of the network.

We then try a different configuration and connect node



(a) Minimum cut value as x increases up to 50.



(b) Minimum cut value as x increases up to 100.

Fig. 3: Evolution of the minimum-cut value for different capacity increments.

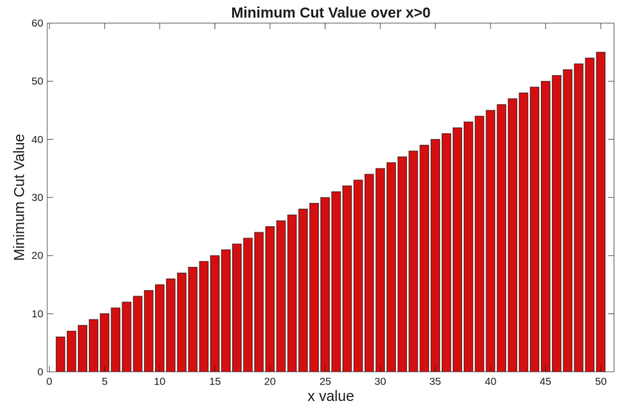
o directly to node d. We run again all functions in order to recompute and compare the results. The output of `compute_cuts.m` is the following:

```
    Combined results table:
           U                 V          C
    ____________      ____________      _

    {[        1]}     {[5  2  3  4]}     7
    {[     1  2]}     {[   5  3  4]}     8
    {[     1  3]}     {[   5  2  4]}     7
    {[     1  4]}     {[   5  2  3]}     9
    {[  1  2  3]}     {[      5  4]}     7
    {[  1  2  4]}     {[      5  3]}    10
    {[  1  3  4]}     {[      5  2]}     6
    {[1  2  3  4]}    {[         5]}     6

Minimum cut value: 6
```

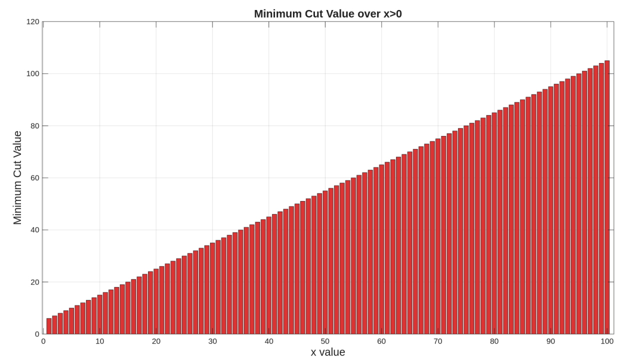
We notice that the capacities of the cuts increase slightly overall, but the relevant behaviour emerges when we add the extra integer units of capacity.

Here the situation changes considerably. When adding $x = 50$, the minimum-cut value reaches 55, which is the same value obtained in the first scenario (Fig. 2) only when $x = 100$. When we now add $x = 100$, we obtain a capacity of 105, a value never reached in the previous configuration.

Although the growth rate still slows down once $x > 50$, as seen before, the overall capacities are significantly higher. This



(a) Minimum cut value as x increases up to 50.



(b) Minimum cut value as x increases up to 100.

Fig. 4: Evolution of the minimum-cut value for different capacity increments.

happens because connecting o directly to d reinforces the most critical bottleneck of the network: the new edge immediately increases the number of independent paths from source to destination. As a result, each additional capacity unit is used more efficiently, especially for small and medium values of x.

Among the tested configurations, placing the new edge directly between o and d produces the most substantial improvement. It strengthens the main flow bottleneck and allows the additional capacity units to translate into a much larger increase in the minimum cut, particularly for moderate values of x.

II. Exercise II

We consider a new 15-nodes Graph $G = (V, \mathcal{E})$, shown in Figure 5.To perform the exercise, a *MATLAB* script, `es2.m`, was created, which first generates the graph and allows its analysis. By inspecting the generated graph G, we observe that it is divided into two densely connected sub-graphs and that two nodes in particular act as structural hubs, namely nodes 6 and 9, which have the highest number of incident edges. We will therefore analyze the behaviour of these two nodes by examining their centrality values.

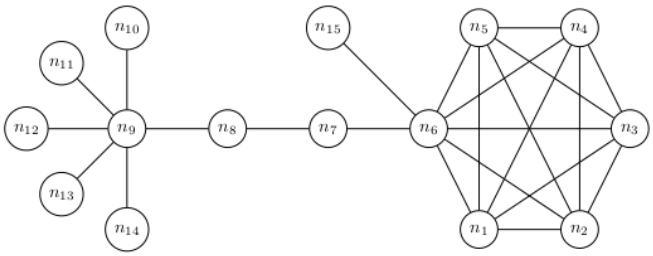


Fig. 5: Fifteen-node Graph G from Exercise 2.

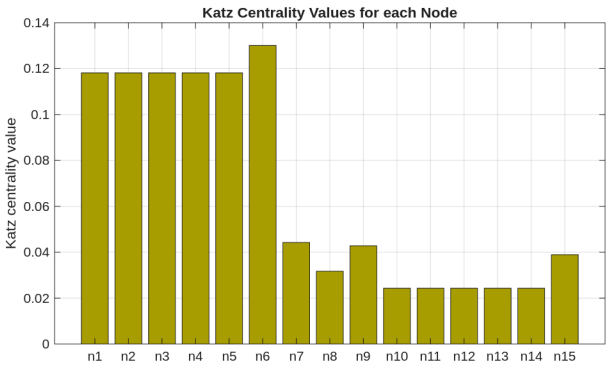


Fig. 6: Katz centrality results

A. Katz Centrality Computation

We compute the Katz centrality with $\beta = 0.15$ and uniform intrinsic centrality μ. Katz centrality measures the importance of a node in a network, taking into account not only its direct connections but also the importance of its neighbours. In the algorithm, each node receives a score proportional to its direct neighbours and to neighbours of neighbours, with decreasing weights as the distance from the node increases. The centrality vector z for all nodes is computed iteratively as:

$$\begin{cases} z(t+1) = \frac{1-\beta}{\lambda_W} W' z(t) + \beta\mu, \\ z(0) = z_0, \end{cases} \tag{2}$$

where $\frac{1-\beta}{\lambda_W}$ is a normalization factor that ensures convergence and W is the adjacency matrix of the graph G(Figure5). We also note $\mu = \frac{ones(1,15)}{15}$ is the uniform intrinsic centrality, a preferences vector that contains the intrinsic weight for each node. Importantly, each node does not rely solely on its local neighbours: the factor λ_W incorporates global information about the network structure. λ_W is obtained as the spectral radius of the adjacency matrix W.

The exercise was carried out by implementing a *MATLAB function*, `Katz_centrality.m`, which returns the vector **z** (III-G) containing the centrality values computed for each node, according to the Katz centrality algorithm.

```
Katz centrality computed:
    Node 1: 0.118140
    Node 2: 0.118140
    Node 3: 0.118140
    Node 4: 0.118140
    Node 5: 0.118140
    Node 6: 0.130080
    Node 7: 0.044247
    Node 8: 0.031719
    Node 9: 0.042811
    Node 10: 0.024303
    Node 11: 0.024303
    Node 12: 0.024303
    Node 13: 0.024303
    Node 14: 0.024303
    Node 15: 0.038930
```

Those results are also shown in the bar plot from Figure 6.

B. Page-Rank centrality

Using the same parameters as before, we now develop a distributed algorithm for the computation of Page-Rank centrality. This algorithm is defined as *distributed* because each node updates its centrality using only information from its neighbours, without relying on any global parameter such as λ_W.

We introduce the transition matrix P, a *stochastic matrix* (each row or column sums to 1), obtained as a normalized version of the adjacency matrix W. The matrix P represents the probability of moving from one node to another. By definition, a stochastic matrix has rank 1 and satisfies $\max(\rho(P)) = 1$, where $\rho(P)$ denotes the spectral radius. Therefore, we can set $\alpha = (1 - \beta)$ and rewrite the algorithm that computes the centralities as:

$$\begin{cases} z(t+1) = \alpha P z(t) + \beta\mu, \\ z(0) = z_0 \end{cases} \tag{3}$$

As in the previous case, we implemented the *MATLAB function* `pageRank_centrality.m`, which computes the algorithm and returns the following vector of centrality values:

```
PageRank centrality computed:
    Node 1: 0.074680
    Node 2: 0.074680
    Node 3: 0.074680
    Node 4: 0.074680
    Node 5: 0.074680
    Node 6: 0.114450
    Node 7: 0.048611
    Node 8: 0.058149
    Node 9: 0.194044
    Node 10: 0.037490
    Node 11: 0.037490
    Node 12: 0.037490
    Node 13: 0.037490
    Node 14: 0.037490
    Node 15: 0.023898
```

The obtained results are also shown in the bar plot from Figure 7.

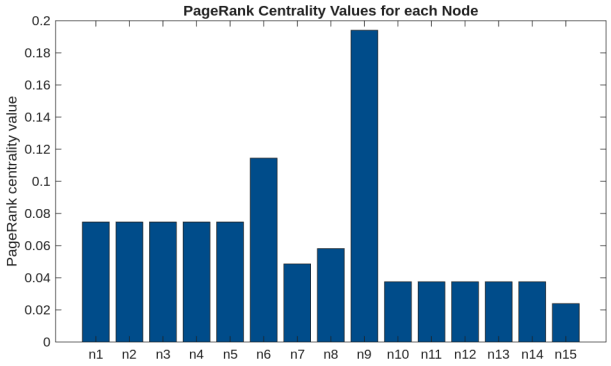


Fig. 7: Page-Rank centrality results

C. Compare the results

As anticipated above, the bar plots shown in Figures 6 and 7 indicate that nodes n_6 and n_9 attain the highest centrality values. This occurs because these nodes are structurally central within the graph and are connected to a larger number of edges, which increases both their direct influence and their ability to propagate information through the network. As a result, they play a key role in the overall connectivity and flow dynamics of the system.

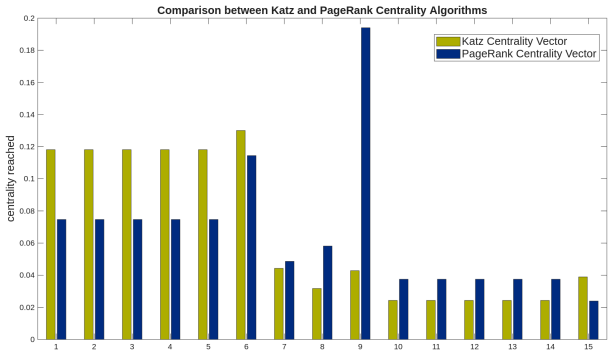


Fig. 8: Katz VS Page-Rank centrality Algorithms

From Figure 8, we observe a clear correspondence between network structure and centrality values. Nodes that are highly connected, by being incident to multiple edges, exhibit significantly higher centrality scores, while nodes connected to only one neighbour show much lower values. This is evident when comparing nodes n_{10}–n_{14}, all linked exclusively to node n_9, with nodes n_1–n_5, which are instead connected to n_6 and to one another.

Some differences between the two algorithms also emerge. On average, Katz centrality tends to reward nodes that are close to highly connected regions of the graph, even if they have low degree themselves. A notable example is node n_{15}: although it is connected to only one node (n_6), it receives a relatively high Katz centrality score because n_6 is itself connected to several highly central nodes. In contrast, PageRank does not amplify this effect to the same extent, as it redistributes importance proportionally through the transition probabilities; consequently, n_{15} attains one of the lowest PageRank values.

Overall, Katz centrality better captures the hierarchical influence created by multi-step connections, whereas PageRank primarily reflects the stationary distribution of a random walk on the graph. This explains the differences seen in peripheral nodes and the consistent prominence of the two hub nodes, n_6 and n_9, in both measures.

D. Page-Rank centrality with different values of β

In Figure 9, we report the results of the Page-Rank centrality computation for different values of the parameter β:

$$\beta \in \{0,\ \frac{1}{4},\ \frac{1}{2},\ \frac{3}{4},\ 1\} \tag{4}$$

These results can be interpreted directly from the update rule defined in Equation 3.

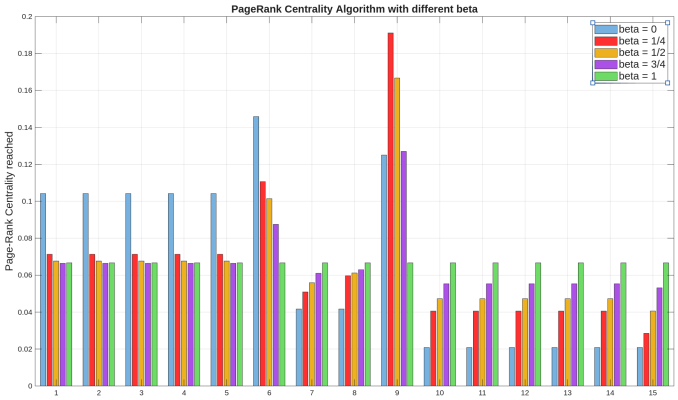


Fig. 9: Page-Rank Centrality with different β

When $\beta = 0$, the update reduces to $z(t+1) = Pz(t)$, meaning that the intrinsic centrality vector μ has no influence on the dynamics. In this case, the final centrality values depend solely on the graph topology: nodes with a higher number of connections, such as n_6 and n_9, obtain significantly larger centrality values, while peripheral nodes receive much smaller scores.

On the other hand, when $\beta = 1$, the update becomes $z(t+1) = \mu$. Since in our analysis μ is uniform, all nodes converge to the same centrality value. Here the structure of the graph plays no role, as the centrality depends entirely on the intrinsic weights.

For intermediate values of β, the influence of the two terms is balanced. Increasing β reduces the contrast between highly connected and weakly connected nodes, driving the centrality vector toward the uniform distribution induced by μ. As a result, nodes with initially high centrality (e.g., n_6 and n_9) show decreasing values as β increases, while less connected nodes experience a rise in centrality. This behaviour reflects the gradual shift from a purely topology-driven regime ($\beta \approx 0$) to one dominated by intrinsic factors ($\beta \approx 1$).

III. EXERCISE III

We now consider the highway network in Los Angeles, represented as a directed graph in Figure 10. In the *MATLAB* script `es3.m`, we initially analyze the available data, which are necessary to represent and study the graph G:

- **B = `traffic.mat`:** is the incidence matrix $B \in \mathbb{R}^{n \times m}$, where the n rows correspond to the nodes of the network and the m columns correspond to the edges of the network. The i-th column of B:
  - contains 1 in the row corresponding to the starting node of the edge e_i,
  - contains -1 in the row corresponding to the ending node of the edge e_i.

  Each node represents an intersection between highways (and part of the surrounding area).
- **C = `capacities.mat`:** is a vector of 28 elements containing the capacity of each edge in the graph.
- **l = `traveltime.mat`:** is a vector that represents the free-flow travel time for each edge, meaning the time a driver takes to traverse the edge when there is no traffic. This will be used as the weight vector w when constructing the graph *G* in the *MATLAB* script.

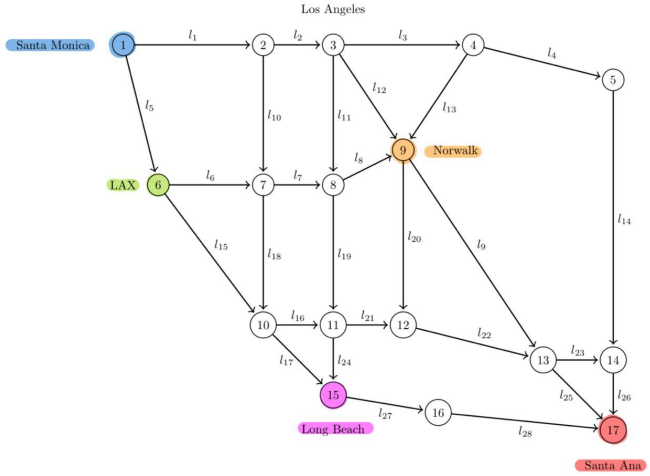


Fig. 10: Seventeen-node Graph *G* from Exercise 3.

A. Compute Shortest Path

The first task is to find the fastest path from node 1 (Santa Monica, blue node) to node 17 (Santa Ana, red node) in an empty network. This corresponds to the shortest path using the minimum travel times l_e as edge weights. As implemented in `es3.m`, we construct a digraph object G with l as the weights. We then use the *MATLAB function* `shortestpath` to find the shortest path from node n_1 to node n_17:

```
Shortest path from n1 to n17:
    {'n1' }
    {'n2' }
    {'n3' }
    {'n9' }
    {'n13'}
    {'n17'}

Path length: 5.598330e-01
```

The shortest-path obtained shows that the driver needs to pass through only four highway intersections to reach Santa Ana, which is advantageous since intersections are typically congestion points. This indicates that the computed route is not only time-optimal but also likely to be more reliable in terms of traffic conditions.

B. Maximum Flow

To find the maximum flow between node n_1 and node n_17, we must use the link capacities c_e. We modify the graph G from `es3.m` to use the capacity vector C as its edge weights. Using the *MATLAB function* `maxflow`:

```
mf = maxflow(G, 1, 17);
Maximum flow from n1 to n17: 2.87e-01
```

This resulting maximum flow value quantifies the overall capacity of the network to transfer traffic from node n_1 to node n_17 and reveals that only a moderate flow can be sustained before reaching saturation. This highlights the presence of structural bottlenecks that significantly limit the network's performance.

C. Compute Flow Vector ν

We are now given a flow vector f (from `flow.mat`) and compute the vector ν satisfying the equation

$$\nu = Bf \ , \tag{5}$$

where $B \in \mathbb{R}^{n \times m}$ is the incidence matrix and ν is the exogenous flow vector defined on the nodes, it specifies how much flow enters (or leaves) each node:

- $\nu_i > 0$ if node i injects flow into the network (*source* node).
- $\nu_i < 0$ if node i absorbs flow (*sink* node).
- $\nu_i = 0$ if node i is a transit node and inflow is equal to outflow.

To calculate ν it is sufficient a direct matrix-vector multiplication. For the following parts, we set all entries of ν to zero except for ν_1 (*source*) and ν_{17} (*sink*), such that $\nu_{17} = -\nu_1$, preserving the total flow computed from the file.

```
Vector nu satisfying Bf = nu:
       16806
        8570
       19448
        4957
        -746
        4768
         413
          -2
       -5671
        1169
          -5
       -7131
        -380
       -7412
       -7810
       -3430
      -23544
```

From the resulting ν, we understood that positive values of ν show where flow is being generated, while the negative values show where it is being absorbed. The fact that they do not cancel out to zero at most nodes might mean that the given flow f is not a circulation, but a flow with multiple sources and sinks spread across the network. The result tells us how unbalanced the flow is at each node.

D. Compute Social Optimum f^*

We find the social optimum f^* by minimizing the total delay $\sum_e f_e \tau_e(f_e)$. This is a convex optimization problem and we use the *MATLAB function* `fmincon` to solve it.

$$\begin{cases} \min_f & \sum_e \frac{f_e\, l_e}{1-f_e/c_e} \\ \text{Constraints:} & Bf = \nu, \\ & 0 \le f_e < c_e \quad \forall e. \end{cases} \tag{6}$$

The computed social optimum f^* redistributes traffic across the network in a way that minimizes the total travel time, with most edges carrying flows in the range of roughly $6,000$ to $12,000$ units and a few links exceeding $10,000$, indicating the presence of key corridors that bear a disproportionate share of the load. The resulting total cost of 1.6504105 represents the system-wide delay under this centrally optimized assignment, highlighting which parts of the network are most critical for maintaining efficient circulation and where capacity improvements or demand-management strategies would yield the greatest benefits.

```
Social optimum f*:
   1.0e+04 *

    0.7364
    0.7456
    1.1377
    0.9345
    0.9442
    0.6227
    0.6326
    0.5559
    0.5697
    0.8478
    0.7632
    0.7895
    0.6989
    0.8599
    0.7983
    0.6320
    1.1624
    0.8793
    0.8397
    0.9075
    0.7760
    0.9704
    0.5879
    0.6952
    0.9142
    0.7066
    1.0766
    0.7336

Total cost (total travel time):
   1.6504e+05
```

E. Wardrop Equilibrium $f^{(0)}$

To find the Wardrop Equilibrium $f^{(0)}$, we solve a similar optimization problem but use the Potential Function as the objective function, defined as:

$$\sum_e \int_0^{f_e} \tau_e(s) ds \ . \tag{7}$$

We also define the delay function as

$$\tau_e(s) = \frac{l_e}{1 - s/c_e}, \tag{8}$$

and its integral as

$$\int_0^{f_e} \tau_e(s)\, ds = -\, l_e c_e \ln\!\left(1 - \frac{f_e}{c_e}\right) \ , \tag{9}$$

in order to compute the following convex optimisation problem:

$$\begin{cases} \min_f & \sum_e -l_e c_e \ln(1 - f_e/c_e) \\ \text{Constraints:} & Bf = \nu, \\ & 0 \le f_e < c_e \quad \forall e. \end{cases} \tag{10}$$

```
Wardrop equilibrium f^(0):
   1.0e+04 *

    0.7149
    0.7455
    1.1078
    0.9218
    0.9657
    0.6328
    0.6255
    0.5581
    0.5741
    0.8264
    0.7686
    0.8139
    0.6818
    0.8472
    0.8097
    0.6165
    1.1852
    0.8750
    0.8357
    0.9125
    0.7767
    0.9761
    0.5837
    0.6750
    0.9285
    0.6897
    1.0792
    0.7362

Value of Wardrop objective function:
   5.1400e+04

Total travel time for Wardrop
    equilibrium flow:
   1.6708e+05
```

The Wardrop equilibrium flow $f^{(0)}$ yields a total travel time of 1.6708105, which is higher than the socially optimal cost, confirming the expected inefficiency of selfish routing. The difference between the two solutions highlights how individual, uncoordinated decisions lead to suboptimal network performance, especially on the most congested links.

F. Wardrop Equilibrium with Tools (1)

We first compute the tolls

$$\omega_e = f_e^* \tau_e'(f_e^*) \ , \tag{11}$$

where f_e^* is the social optimum from 6 and the derivative $\tau_e'(f_e)$ is defined as

$$\frac{l_e/c_e}{(1 - f_e/c_e)^2} \ . \tag{12}$$

We then find the new Wardrop Equilibrium $f^{(\omega)}$ with the new link costs $\tau_e(f_e) + \omega_e$ by minimizing a new Potential Function:

$$\begin{cases} \min_f & \sum_e \left(\int_0^{f_e} \tau_e(s) ds + f_e \omega_e \right) \\ \text{Constraints:} & Bf = \nu, \\ & 0 \le f_e < c_e \quad \forall e. \end{cases} \tag{13}$$

Upon computation, we observe that the new equilibrium flow $f^{(\omega)}$ is very similar to the social optimum f^* from 6, as predicted by theory.

```
Wardrop equilibrium with marginal-
    cost tolls:
   1.0e+04 *

    0.7364
    0.7456
    1.1377
    0.9345
    0.9442
    0.6227
    0.6326
    0.5559
    0.5697
    0.8478
    0.7632
    0.7895
    0.6989
    0.8599
    0.7983
    0.6320
    1.1624
    0.8792
    0.8397
    0.9075
    0.7760
    0.9704
    0.5879
    0.6952
    0.9142
    0.7066
    1.0766
    0.7336

Total travel time for Wardrop with
    tolls: 165039.29198386
Total travel time for social optimum
    (previously computed):
    165039.29187702
```

G. Wardrop Equilibrium with Tools (2)

Finally, we consider a new social cost: the total additional travel time, defined as

$$\psi_e(f_e) = f_e(\tau_e(f_e) - l_e) \ . \tag{14}$$

We first compute the new social optimum f^* by minimizing $\sum_e \psi_e(f_e)$ subject to the flow constraints.
Then, we construct a new toll vector

$$\omega_e^* = \psi_e'(f_e^*) \tag{15}$$

where ψ_e' is the derivative of the new cost function. We compute the Wardrop Equilibrium $f^{(\omega^*)}$ using the link costs $\tau_e(f_e) + \omega_e^*$.
As in 13, we verify that the resulting equilibrium $f^{(\omega^*)}$ successfully coincides with the new social optimum f^*.

```
Wardrop equilibrium under constructed tolls:
   1.0e+04 *

    0.7367
    0.7456
    1.1385
    0.9346
    0.9439
    0.6226
    0.6328
    0.5557
    0.5697
    0.8481
    0.7630
    0.7889
    0.6996
    0.8600
    0.7981
    0.6324
    1.1618
    0.8793
    0.8398
    0.9075
    0.7760
    0.9704
    0.5881
    0.6957
    0.9140
    0.7069
    1.0766
    0.7336

System cost (additional travel time) at Wardrop with tolls:
    142267.18546540
System cost at system optimum:
    142267.18544205
```

IV. Conclusion

This project successfully analysed network flow, node centrality, and traffic equilibrium.

In **Exercise I** [I], the minimum cut of a 5-node network was identified as 5. While capacity augmentation showed diminishing returns, a new link directly connecting the source and destination proved to be the most effective strategy for throughput improvement.

In **Exercise II** [II], Katz and Page-Rank algorithms identified nodes 6 and 9 as the primary structural hubs in a 15-node graph. The analysis also confirmed the role of the Page-Rank parameter β, demonstrating how it interpolates between a purely topology-driven centrality ($\beta = 0$) and a uniform distribution ($\beta = 1$).

In **Exercise III** [III], the Los Angeles traffic network was modelled to compute the social optimum f^* and Wardrop Equilibrium $f^{(0)}$. The analysis confirmed the expected inefficiency of selfish routing, with the Wardrop equilibrium yielding a higher total travel time ($1.6708e + 05$) than the social optimum ($1.6504e+05$). Crucially, it was demonstrated that applying marginal-cost tolls successfully aligns the user equilibrium flow with the social optimum, restoring system efficiency.

Acknowledgments: A public repository is available online at *this* link and includes the folder *Homework1* with the Matlab scripts useful to complete the exercises.