Traffic Engineering of Unicast Services

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2021/2022

Traffic engineering of unicast services

A unicast service is defined by a set of point-to-point traffic flows on a given telecommunication network.

- Consider a network composed by a set of point-to-point links and supporting one unicast service defined by a set of traffic flows T, such that all flows have the same average packet size B.
  - The network is modelled by a graph $G=(N,A)$. N is the set of network nodes. A represents the set of network links: the arc $(i,j) \in A$ represents the link between nodes $i \in N$ and $j \in N$ from i to j whose capacity is given by c_{ij} in bps (usually $c_{ij} = c_{ji}$).
  - Each traffic flow $t \in T$ is defined by its origin node o_t, destination node d_t, average throughput from origin to destination b_t (in bps) and average throughput from destination to origin $\underline{b_t}$ (in bps).
  - For each flow $t \in T$, P_t is a set of the candidate routing paths in graph G from its origin node o_t to its destination node d_t.

The traffic engineering task is the task of choosing for each flow $t \in T$ the percentage of its average throughput that must be routed through each of its candidate routing paths of P_t in each direction.

Traffic engineering with single path routing

- In the single path routing, each traffic flow must be routed through one single path (no flow bifurcation is allowed).
- Symmetrical routing might be required or not; when required, the routing path from a node $j \in N$ to a node $i \in N$ must use the same links as the routing path from node $i \in N$ to node $j \in N$.

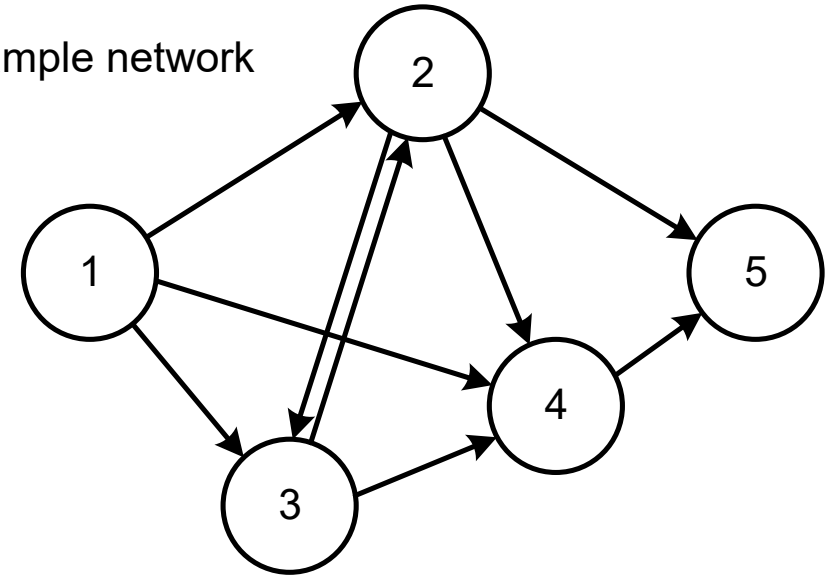
Consider a binary variable x_{tp} associated to each traffic flow $t \in T$ and each routing path $p \in P_t$ that, when is 1, indicates that traffic flow t is routed through path p.

Any traffic engineering solution with single path routing must be compliant with the following constraints:

- For each flow $t \in T$, one of its associated variables x_{tp} must be 1 and all other associated variables must be 0.
- At each arc $(i,j) \in A$, the sum of the throughput values (either b_t or $\underline{b}_t$) of all flows routed through it cannot be higher that its capacity c_{ij}.

Example of a graph and sets of candidates paths

Example network



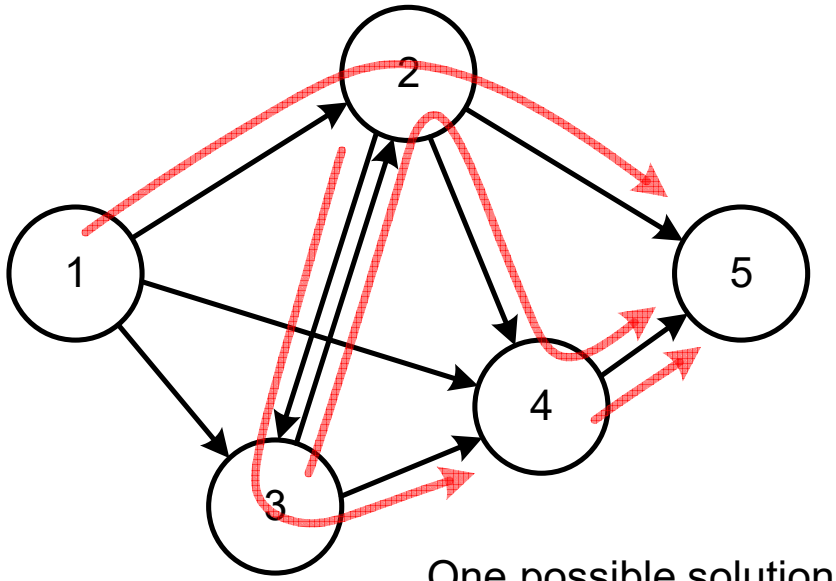
From 1 to 5:
- 1-2-5
- 1-4-5
- 1-2-4-5
- 1-3-4-5
- 1-3-2-5
- 1-3-2-4-5
- 1-2-3-4-5

From 3 to 5:
- 3-4-5
- 3-2-5
- 3-2-4-5

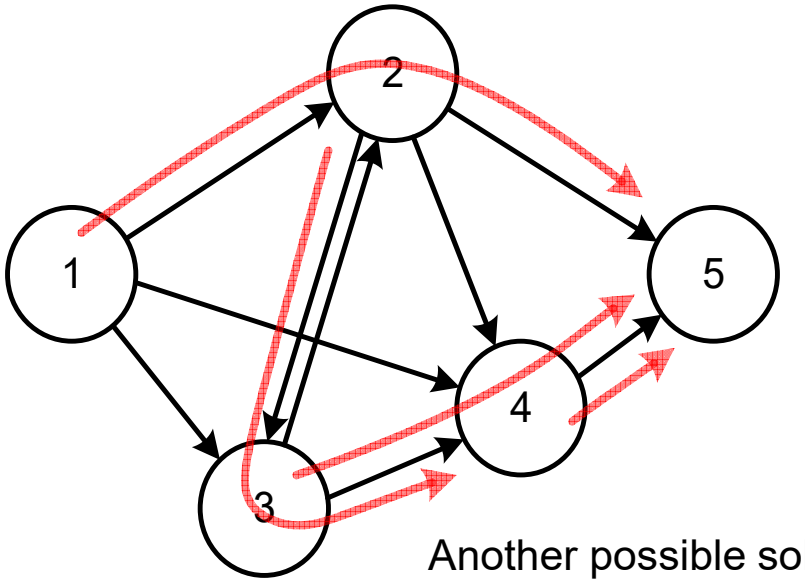
From 2 to 4:
- 2-4
- 2-3-4

From 4 to 5:
- 4-5

- 4 unidirectional flows
- All candidate paths for each flow



One possible solution



Another possible solution

Traffic engineering objectives

The traffic engineering task aims to:

- optimize at least one parameter related with either the performance or the operational cost of the network;
- optionally, guarantee (maximum or minimum) values for other parameters.

Examples of optimization parameters:

- the average service packet delay (to minimize the delay performance of the service);
- the worst average packet delay among all traffic flows (to minimize the delay performance fairness among all traffic flows);
- the worst link load (to maximize the robustness of the network to unpredictable traffic growth);
- the energy consumption of the network (to minimize operational costs).

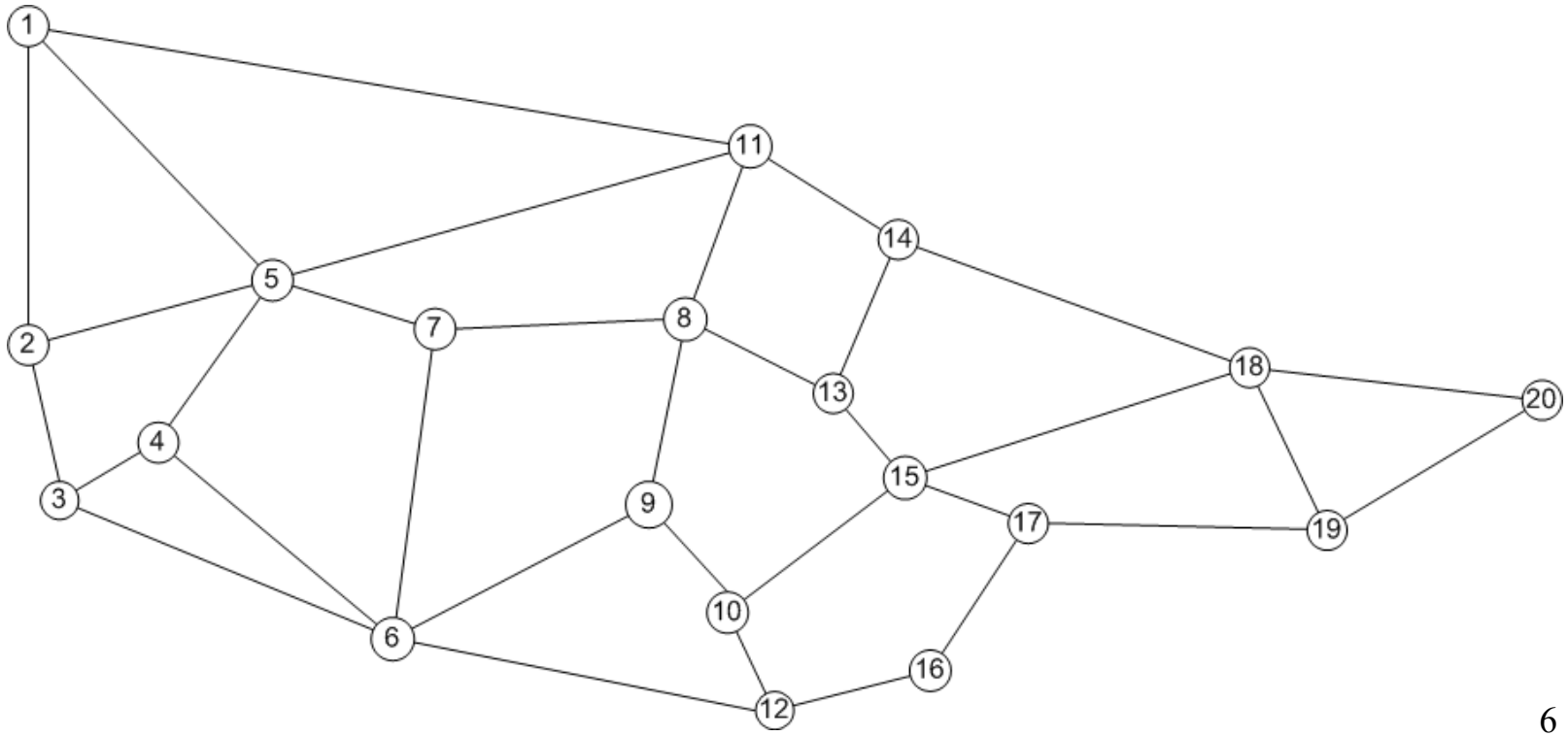
The best traffic engineering solution depends on the optimization objective of interest. Different optimization objectives might be conflicting:

- for example, to reduce the energy consumption, more links must be put in sleeping mode; consequently, the same traffic is routed through less links and the worst link load increases.

Example - network

Consider the following network with 20 routers and 33 links where all links have a capacity of 1 Gbps.

The length of the links varies between 88 km (between nodes 10 and 12) and 759 km (between nodes 1 and 11) and the link propagation delay is the light speed over fiber (2 x 10^8 meters/s).



Example – traffic flow matrix

Consider the following flow matrix (values in Mbps) where all flows have an average packet size B = 1000 Bytes:

0.0	47.7	64.4	13.6	10.6	45.5	10.6	12.9	9.8	9.8	13.6	11.4	11.4	41.7	12.9	10.6	9.1	12.9	11.4	22.0
47.0	0.0	68.2	32.6	33.3	189.4	59.8	47.0	49.2	38.6	59.1	53.0	38.6	212.1	31.8	48.5	40.9	43.9	54.5	74.2
65.9	69.7	0.0	8.3	13.6	53.0	13.6	14.4	18.9	14.4	11.4	36.4	12.9	63.6	13.6	14.4	11.4	13.6	15.9	22.7
13.6	46.2	13.6	0.0	12.9	31.1	14.4	12.1	11.4	12.9	31.1	12.1	9.1	31.8	11.4	10.6	14.4	12.9	17.4	24.2
7.6	31.8	10.6	14.4	0.0	55.3	11.4	9.1	9.8	12.9	36.4	11.4	12.9	46.2	12.9	7.6	12.1	15.9	16.7	28.0
52.3	174.2	53.8	55.3	38.6	0.0	39.4	47.0	41.7	40.9	53.8	44.7	42.4	212.1	40.9	71.2	62.9	46.2	56.8	72.0
13.6	32.6	11.4	11.4	12.9	54.5	0.0	7.6	12.1	12.9	12.1	14.4	56.8	55.3	9.8	9.1	13.6	15.9	9.8	21.2
13.6	47.0	14.4	11.4	9.8	37.9	10.6	0.0	12.1	9.1	11.4	13.6	12.1	57.6	9.8	10.6	9.8	17.4	12.9	34.1
9.8	33.3	16.7	12.1	14.4	38.6	12.9	9.8	0.0	11.4	13.6	9.1	13.6	56.1	11.4	13.6	12.1	15.2	18.9	18.9
12.9	53.0	10.6	9.8	11.4	46.2	13.6	10.6	10.6	0.0	9.1	9.8	11.4	42.4	10.6	11.4	10.6	15.9	10.6	25.8
9.1	36.4	9.8	31.1	33.3	40.9	9.8	10.6	12.1	14.4	0.0	9.8	10.6	47.7	9.1	11.4	8.3	9.8	12.9	32.6
10.6	61.4	35.6	9.8	9.8	59.8	14.4	8.3	8.3	10.6	12.1	0.0	9.8	33.3	9.8	28.0	9.8	9.1	14.4	25.0
10.6	32.6	9.1	12.1	9.1	35.6	59.8	10.6	9.8	14.4	9.8	13.6	0.0	41.7	11.4	12.9	13.6	15.9	16.7	40.9
40.9	181.8	49.2	56.1	42.4	189.4	55.3	64.4	57.6	31.8	31.8	33.3	46.2	0.0	40.9	57.6	40.2	48.5	51.5	69.7
12.1	37.1	10.6	9.8	10.6	37.1	8.3	14.4	8.3	10.6	9.8	12.1	11.4	47.7	0.0	11.4	10.6	10.6	9.1	28.8
10.6	47.7	9.8	11.4	11.4	44.7	9.8	11.4	10.6	9.1	9.1	12.9	9.1	56.8	14.4	0.0	11.4	9.1	12.9	30.3
13.6	34.1	10.6	10.6	13.6	55.3	12.1	12.9	9.8	11.4	10.6	12.9	9.1	44.7	10.6	9.1	0.0	10.6	9.8	20.5
13.6	40.9	11.4	9.8	18.9	40.9	11.4	18.2	13.6	18.9	12.9	10.6	17.4	40.9	11.4	12.9	9.8	0.0	34.1	24.2
7.6	49.2	18.9	15.9	12.9	53.0	12.1	9.8	15.9	13.6	15.2	10.6	18.9	47.0	12.9	9.8	9.8	30.3	0.0	18.9
23.5	68.2	26.5	28.8	34.1	65.9	25.8	30.3	15.9	22.7	30.3	28.0	34.1	65.2	34.1	25.8	24.2	21.2	15.9	0.0

Example – one possible solution

One possible solution is to route each traffic flow $t \in T$ by the routing path with the shortest length (minimizing, in this way, the propagation delay of each flow).

Using the Kleinrock approximation, we obtain the following performance parameters:

Worst average packet delay = 6.06 ms

Worst link load = 99.3%

Number of active links = 33 out of 33

However, it is possible to obtain better traffic engineering solutions through appropriate optimization algorithms.

Example – optimal solutions

Minimization of the worst average packet delay:

Worst average packet delay = 5.21 ms
Worst link load = 93.6%
Number of active links = 33 out of 33

Minimization of the worst link load:

Worst average packet delay = 8.63 ms
Worst link load = 69.9%
Number of active links = 33 out of 33

Minimization of the number of active links:

Worst average packet delay = 10.54 ms
Worst link load = 82.4%
Number of active links = 26 out of 33

Conclusion:

- Each traffic engineering solution is a different trade-off between the 3 optimization objectives.
- It is up to the operator to select the best routing solution.

Optimization algorithms

Exact algorithms

- Based on mathematical models (for example, Integer Linear Programming)
- In the general case, computationally hard
- Theoretically, they are able to compute the optimal solutions
- Inefficient for large problem instances (they either take too long to even compute feasible solutions or finish due to out-of-memory)

Heuristic algorithms

- Based on simple programming algorithms
- Easy to implement and quick to find solutions
- Do not guarantee optimality
- For larger runtimes, they find better solutions
- Efficient for large problem instances

Heuristic method versus heuristic algorithm

Heuristic method: a generic approach to search for good solutions that can be applied to any optimization problem.

Heuristic algorithm: an optimization algorithm that has resulted from applying an heuristic method to a particular optimization problem.

Many heuristic methods (usually, also the simplest ones) are based on two algorithmic strategies:

1. To build a solution starting from the scratch.
   - Examples: *random*, *greedy, greedy randomized, etc…*

2. To get a better solution from a known solution.
   - Examples: *hill climbing, tabu search, simulated annealing, etc...* (we will address only the hill climbing strategy).

Building a solution from the scratch

Building a solution from the scratch (I)

Random strategy:

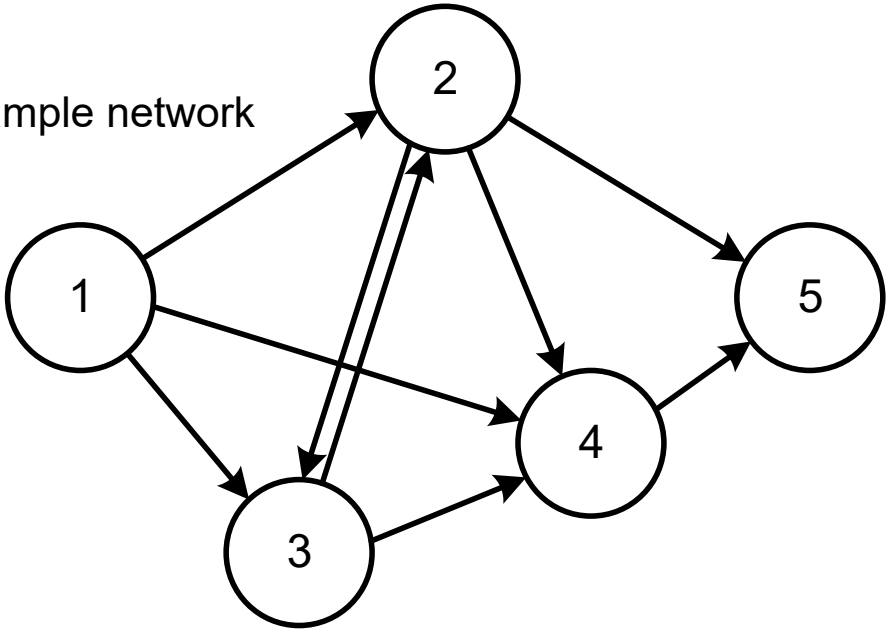
- We select a random routing path $p \in P_t$ to each flow $t \in T$
- We obtain a slightly better performance if we consider higher probabilities to routing paths $p \in P_t$ with "better characteristics"
  - For example, paths with a smaller number of links, paths containing links of larger capacity, etc…

Greedy strategy:

- We start by considering the network without any routing path
- For each flow $t \in T$:
  - We select the routing path $t \in P_t$ that, together with the previous selected routing paths, gives the best objective function value

Building a solution from the scratch (II)

Example network



From 1 to 5:
- 1-2-5
- 1-4-5
- 1-2-4-5
- 1-3-4-5
- 1-3-2-5
- 1-3-2-4-5
- 1-2-3-4-5

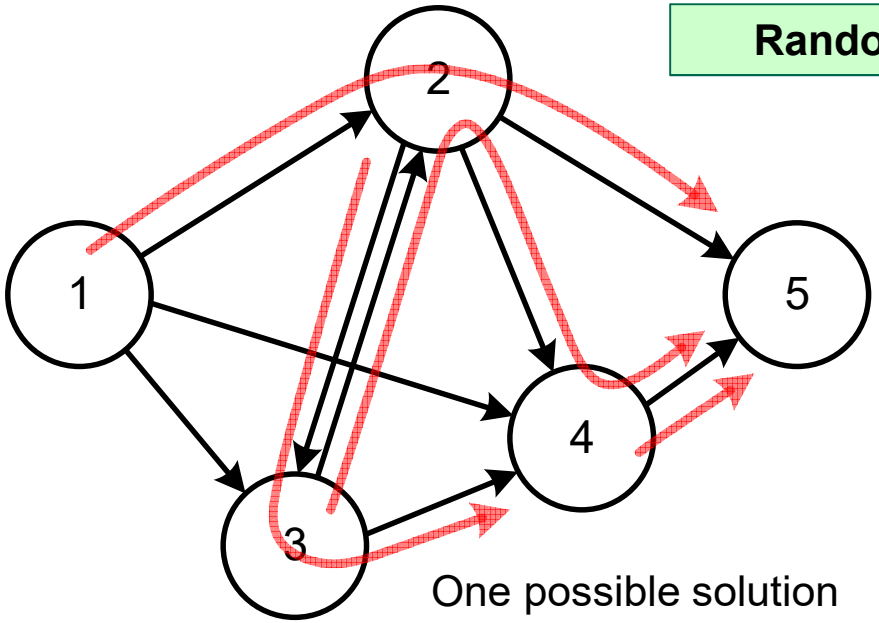
From 3 to 5:
- 3-4-5
- 3-2-5
- 3-2-4-5

From 2 to 4:
- 2-4
- 2-3-4

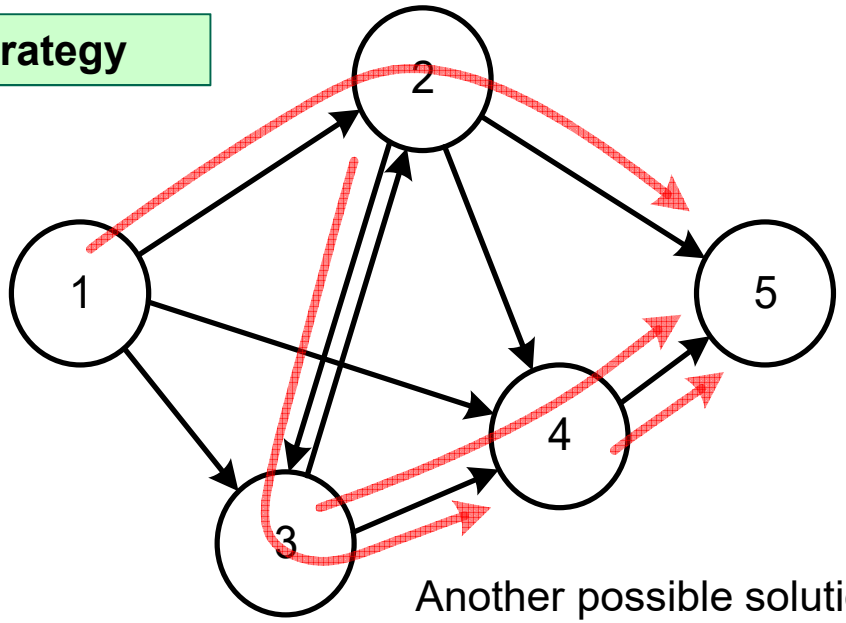
From 4 to 5:
- 4-5

Candidate paths for each flow

Random strategy

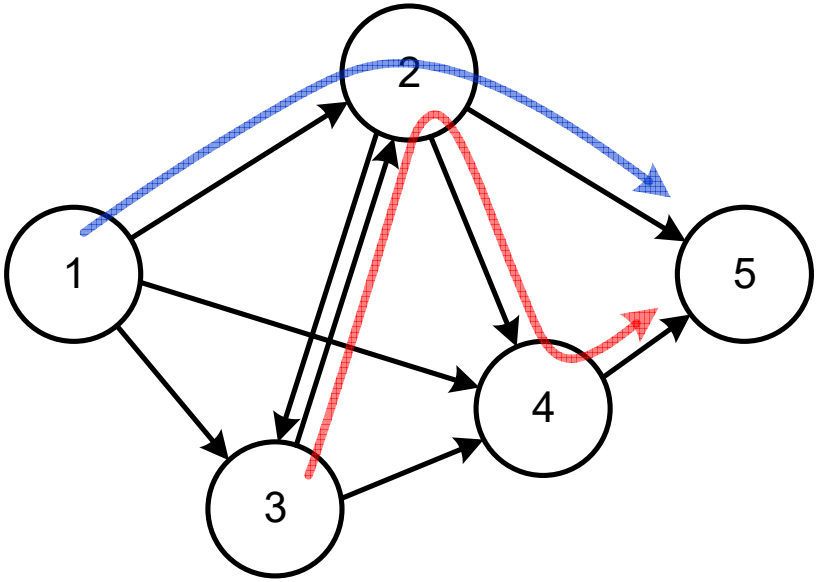


One possible solution

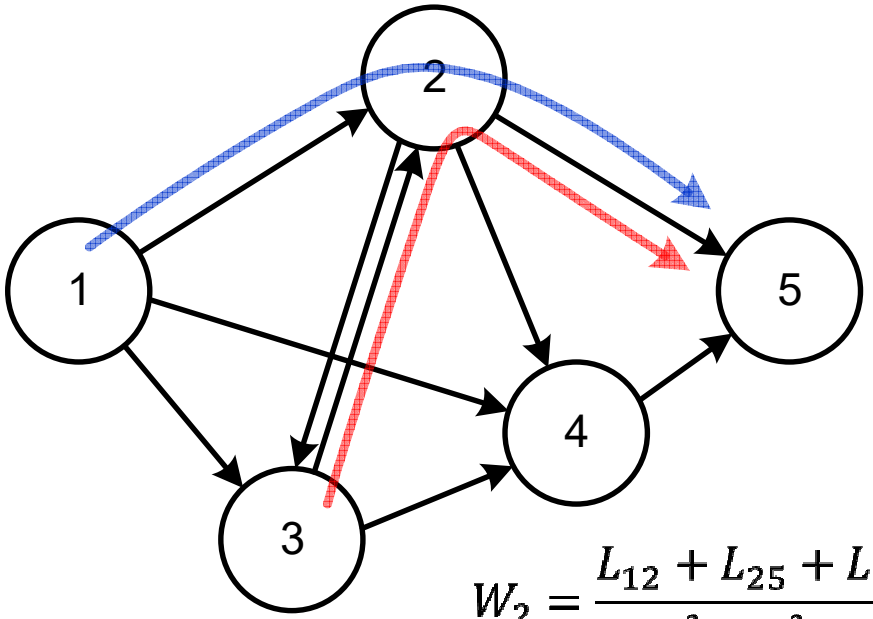


Another possible solution

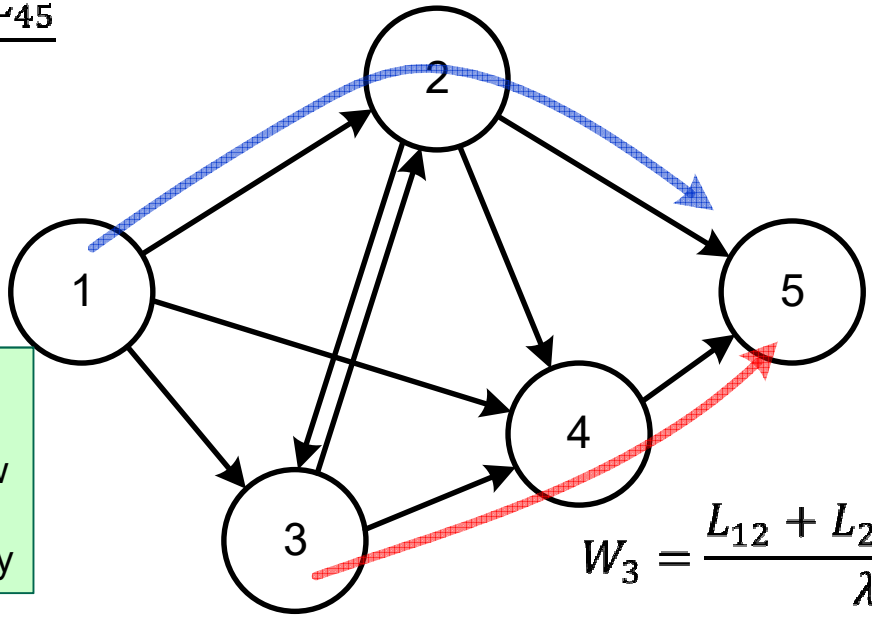
Building a solution from the scratch (III)



$$W_1 = \frac{L_{12} + L_{25} + L_{32} + L_{24} + L_{45}}{\lambda_1 + \lambda_2}$$



$$W_2 = \frac{L_{12} + L_{25} + L_{32}}{\lambda_1 + \lambda_2}$$

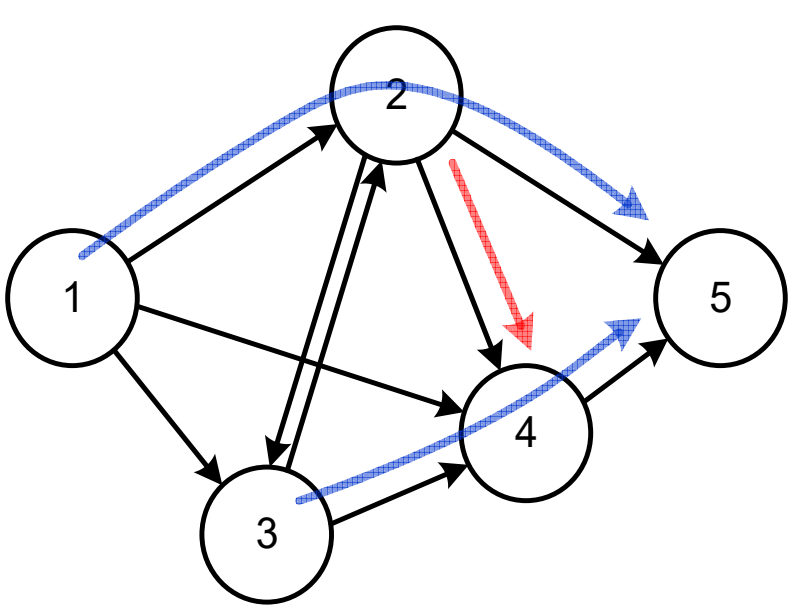


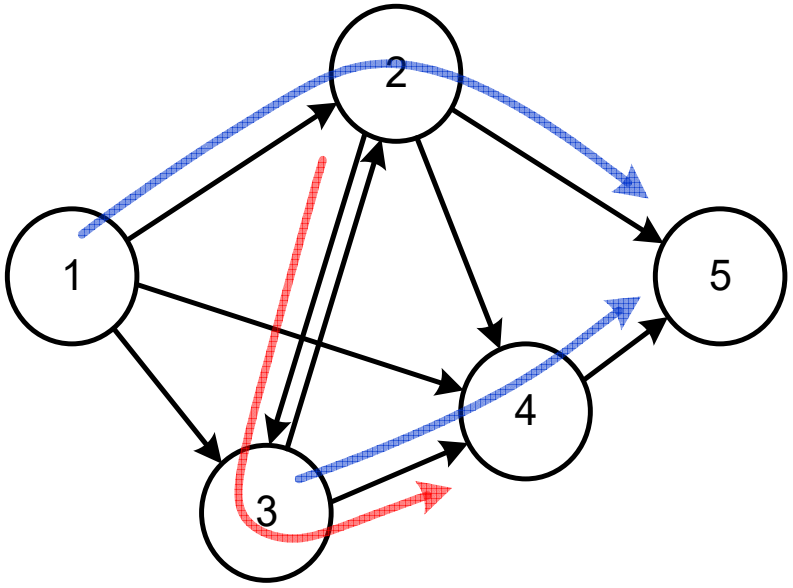
$$W_3 = \frac{L_{12} + L_{25} + L_{34} + L_{45}}{\lambda_1 + \lambda_2}$$

Greedy strategy:

- 1st flow (blue) already selected
- Try all 3 candidate paths for 2nd flow (red) and select the solution with lowest service average packet delay

Building a solution from the scratch (IV)





$$W_1 = \frac{L_{12} + L_{25} + L_{24} + L_{34} + L_{45}}{\lambda_1 + \lambda_2 + \lambda_3}$$

$$W_2 = \frac{L_{12} + L_{25} + L_{23} + L_{34} + L_{45}}{\lambda_1 + \lambda_2 + \lambda_3}$$

Greedy strategy:

- 1st and 2nd flow (blue) already selected
- Try all 2 candidate paths for 3rd flow (red) and select the solution with lowest service average packet delay

Building a solution from the scratch (V)

Greedy randomized strategy:

The aim is to obtain a different solution on different runs.

First alternative:

- Choose a random order to the flows $t \in T$ to select their routing paths.
- Apply the greedy strategy by the previous order.

Second alternative:

- For each flow $t \in T$, select randomly one routing path among the best α routing paths (*i.e.*, the α paths that, together with the previous assigned routing paths, give the best objective function values)
  - α is a parameter of the algorithm

Third alternative:

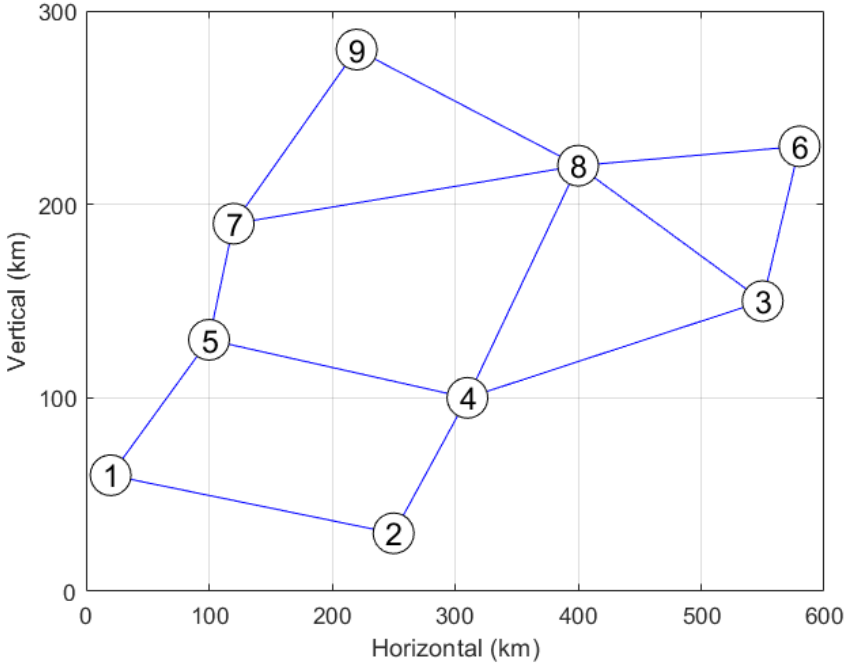
- To combine the 2 previous alternatives

Practical Task

Consider the network with all links with a capacity of 10 Gbps.

Consider a unicast service with the following flows (throughput values b_t and $\underline{b}_t$ in Gbps):

t	o_t	d_t	b_t	$\underline{b}_t$
1	1	3	1.0	1.0
2	1	4	0.7	0.5
3	2	7	3.4	2.5
4	3	4	2.4	2.1
5	4	9	2.0	1.4
6	5	6	1.2	1.5
7	5	8	2.1	2.7
8	5	9	2.6	1.9



A. Based on the provided MATLAB code, compute a symmetrical single routing path solution with minimum worst link load using all possible routing paths and resorting to the first alternative of the greedy randomized strategy.

B. Compare the efficiency of this algorithm with the one resorting to the random strategy.

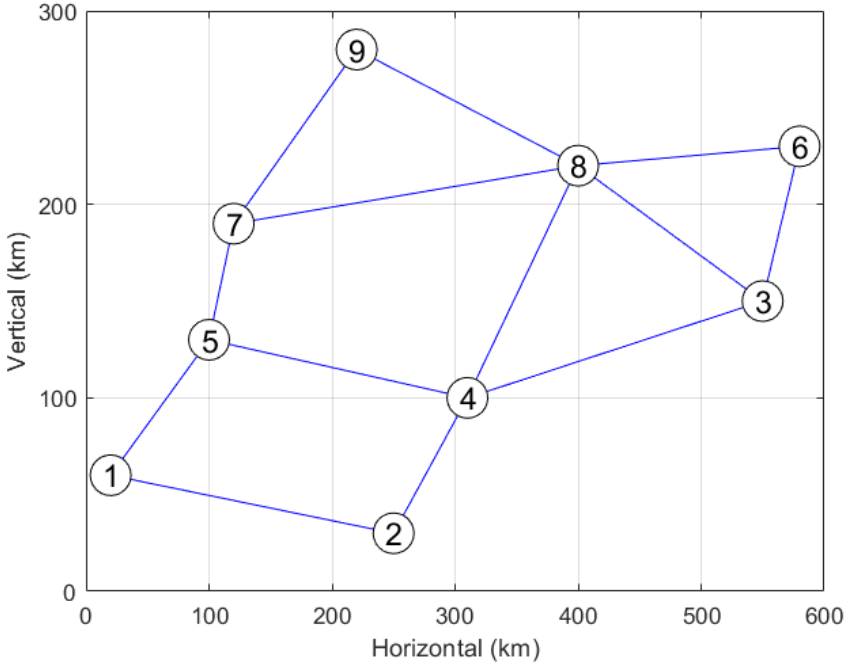
C. Run this algorithm using only the 6 shortest paths of each flow and compare the obtained solutions with the previous ones.

Practical Task

Consider the network with all links with a capacity of 10 Gbps.

Consider a unicast service with the following flows (throughput values b_t and $\underline{b_t}$ in Gbps):

t	o_t	d_t	b_t	$\underline{b_t}$
1	1	3	1.0	1.0
2	1	4	0.7	0.5
3	2	7	3.4	2.5
4	3	4	2.4	2.1
5	4	9	2.0	1.4
6	5	6	1.2	1.5
7	5	8	2.1	2.7
8	5	9	2.6	1.9



D. Consider that the energy consumption of each link is proportional to its length. Adapt both algorithms (based on the random strategy and on the greedy randomized strategy) to compute a symmetrical single routing path solution minimizing the energy consumption of the network.

E. Compare the efficiency of both algorithms.

F. Run both algorithms using only the 6 shortest paths of each flow and compare the obtained solutions with the previous ones.

Getting a better solution from a known solution

Getting a better solution from a known solution (I)

Hill climbing strategy – best neighbour move variant

In this strategy, we start by an initial solution and try to move to a better solution by making one local change. The moves are repeated until no possible local change produces any better solution.

This strategy works with the following steps:

1. For a given current solution (in the first iteration, the current solution is the initial known solution), we compute all neighbour solutions and select the best one.
2. If the best neighbour solution is better than the current solution, we move to this solution (*i.e.*, we set the current solution with the best neighbour solution) and go to step 1.
3. If not, we stop and the current solution is the final result (we say it is a local optimum solution).

Getting a better solution from a known solution (II)

Hill climbing strategy – best neighbour move variant:

Consider a minimization problem with a feasible solution set S, a minimization function $f(s)$ and a set $V(s)$ of neighbours of each $s \in S$

$s' \leftarrow$ Initial($s \in S$) ← *s'* is set with a initial solution

improved ← TRUE

While *improved* **do** ← the algorithm stops when no improvement can be obtained by a neighbour solution

  $s \leftarrow$ Best($s \in V(s')$) ← the best neighbour *s* of *s'* is selected

  If $f(s) < f(s')$ **do**

    $s' \leftarrow s$ ← if $f(s)$ is better than $f(s')$, *s* becomes the current solution *s'*

  Else do

    improved ← FALSE ← otherwise, no improvement can be obtained by a neighbor solution

  EndIf

EndWhile

The final result of the algorithm is solution *s'* whose function value is $f(s')$

Getting a better solution from a known solution (III)

Hill climbing strategy – first neighbour move variant

If the evaluation of all neighbour solutions is computationally heavy (either because each neighbour solution is hard to compute or because the neighbour set has too many solutions), the previous variant might be non efficient.

This variant works with the following steps:

1. For a given current solution (in the first iteration, the current solution is the initial solution), we compute neighbour solutions until we find a solution better than the current one or until all neighbour solutions are computed.
2. If a neighbour solution better than the current one is found, we set the current solution with the neighbour solution and go to step 1.
3. If not, we stop and the current solution is the final result.

Usually, it is more efficient to use the best neighbour move variant, although in some problems it requires a careful definition of the neighbour set.

Getting a better solution from a known solution (IV)

Hill climbing strategy – defining the set of neighbour solutions

- The set of neighbour solutions (of a given solution) is problem dependent.
- The neighbour set must be carefully defined in order to allow the algorithm to compute all neighbour solutions in reasonable runtime.

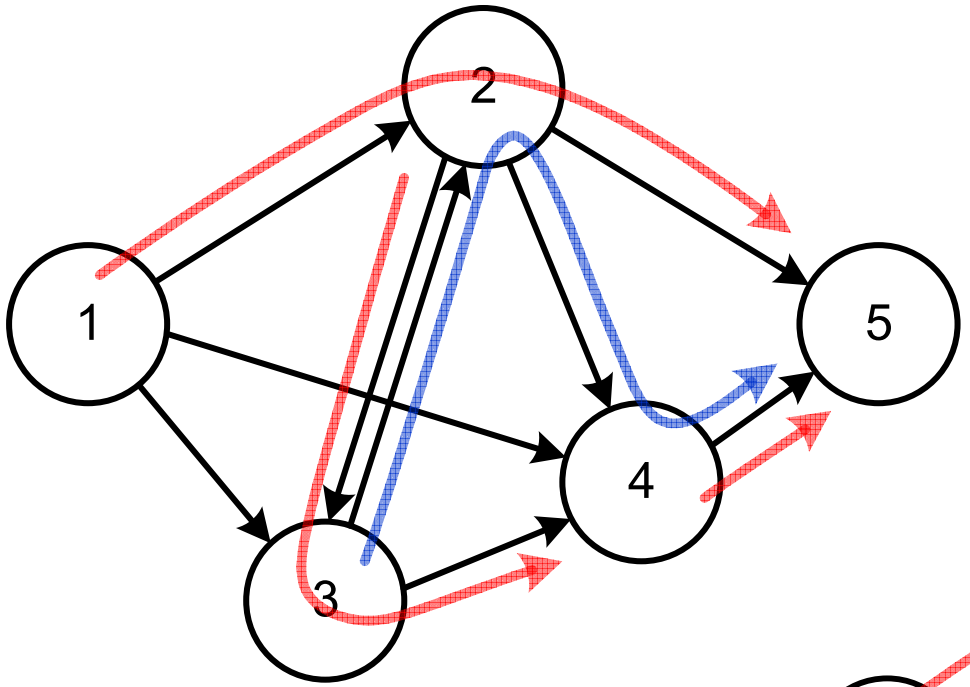
In traffic engineering of telecommunication networks, the neighbour set is usually defined as follows:

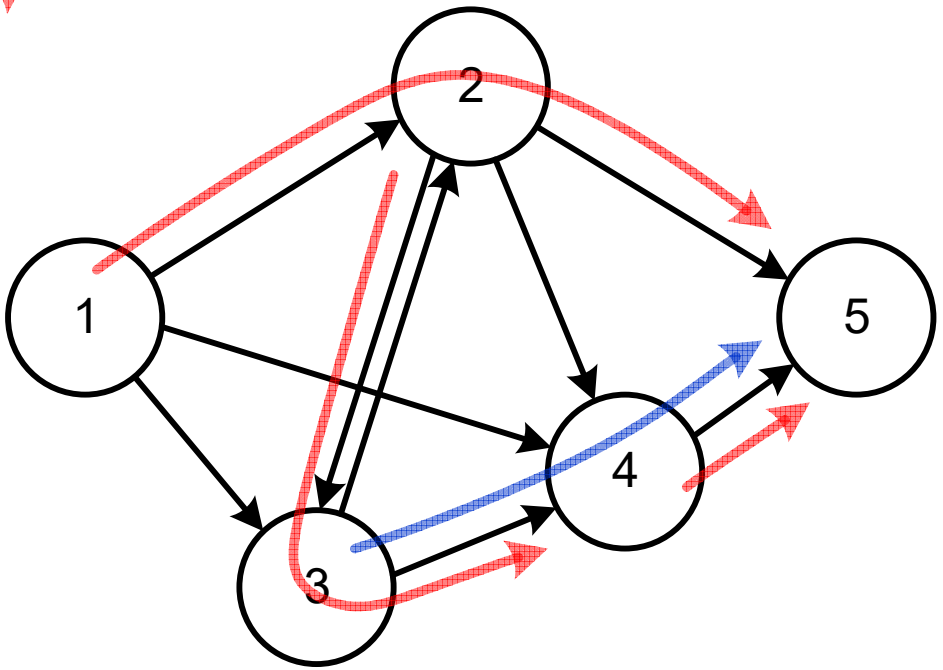
- For each flow $t \in T$, set P_t is computed by a k-shortest path algorithm.
- For a current solution (that defines a routing path $p \in P_t$ for each flow $t \in T$), a neighbour solution is a solution that differs from the current one in the routing path of a single flow.
- So, the number of neighbour solutions is:

$$\sum_{t \in T} (|P_t| - 1)$$

where $|P_t|$ is the number of paths of set P_t.

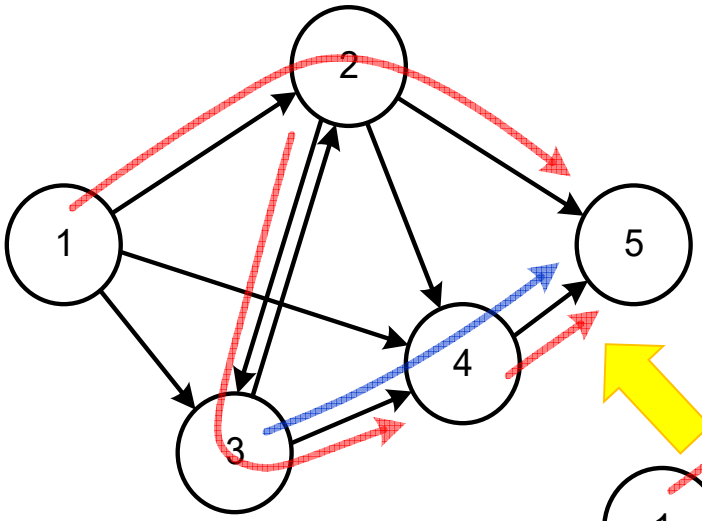
Getting a better solution from a known solution (V)



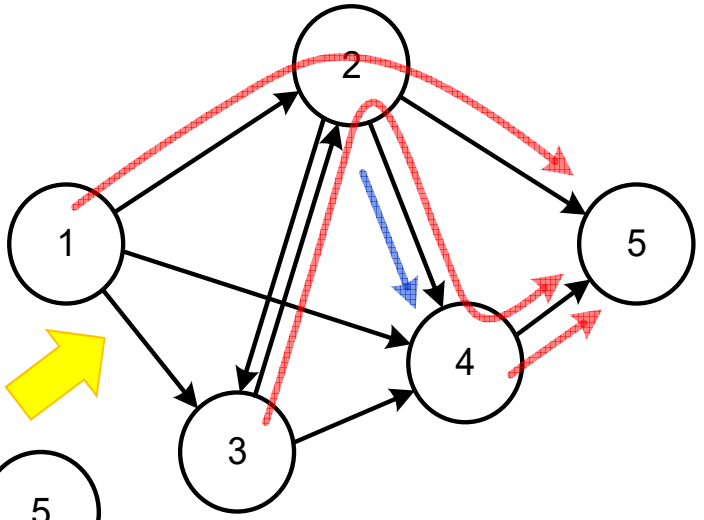


Neighbour solutions

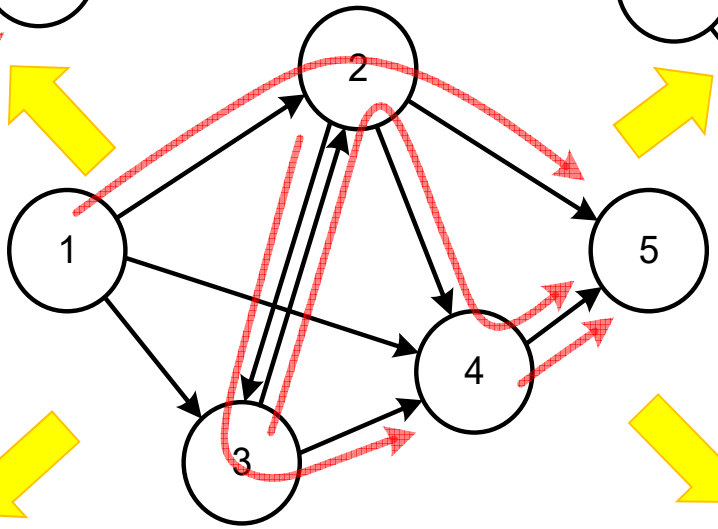
Getting a better solution from a known solution (VI)



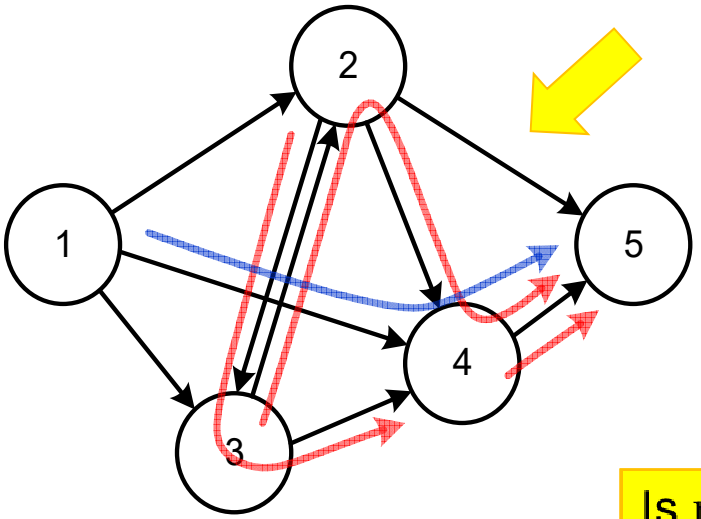
neighbour solution, W_1



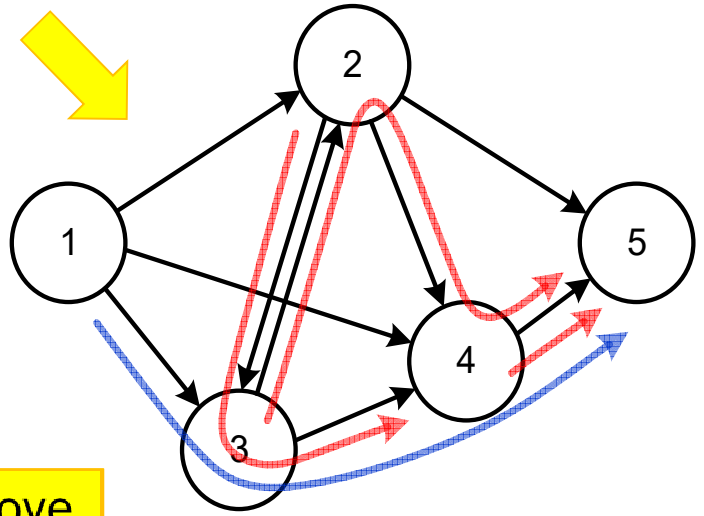
neighbour solution, W_2



current solution, W



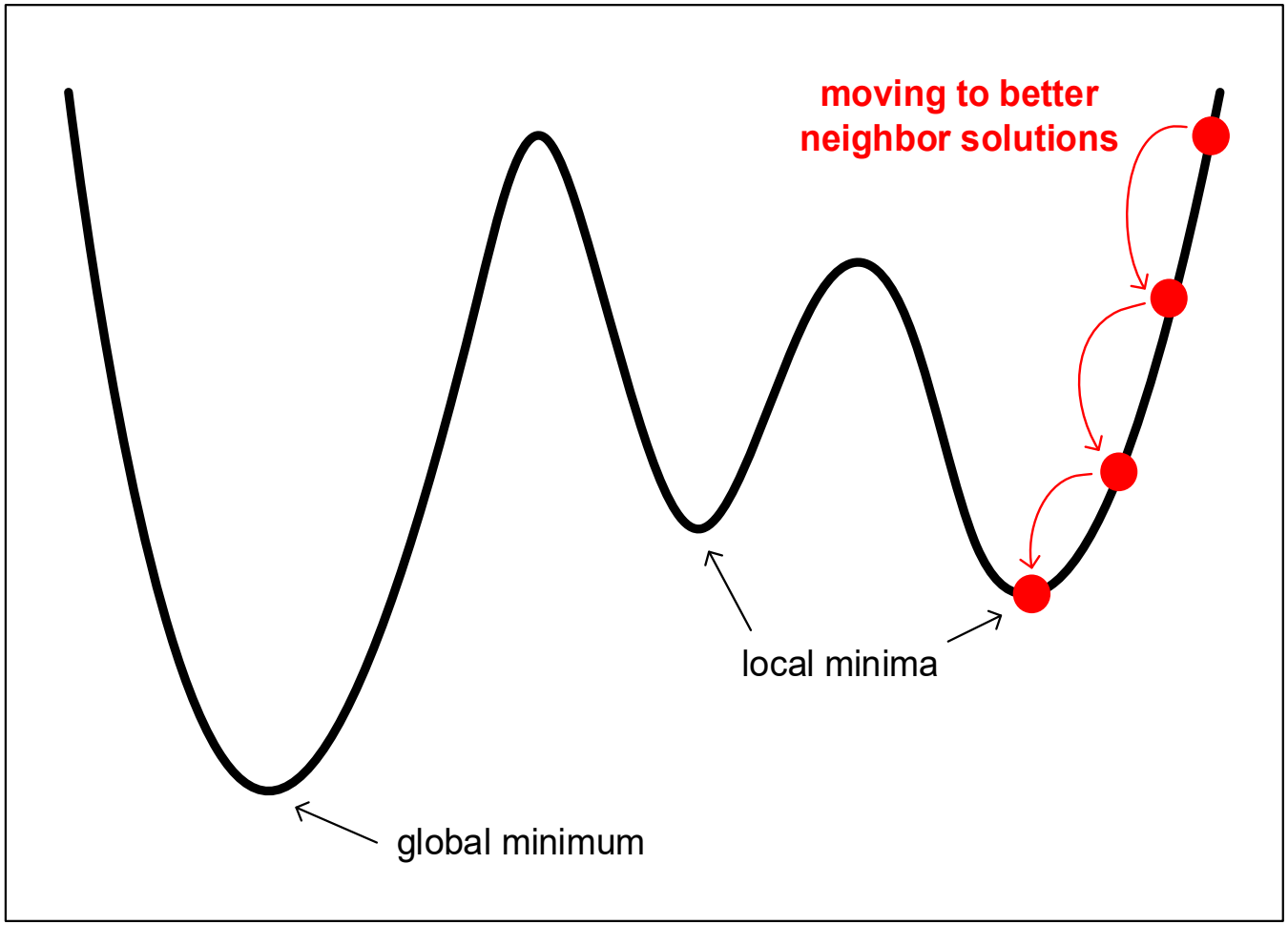
neighbour solution, W_3



neighbour solution, W_4

Is $\min W_i < W$? If yes, move to best neighbour solution.

Getting a better solution from a known solution (VII)



Multi Start Hill Climbing Heuristic

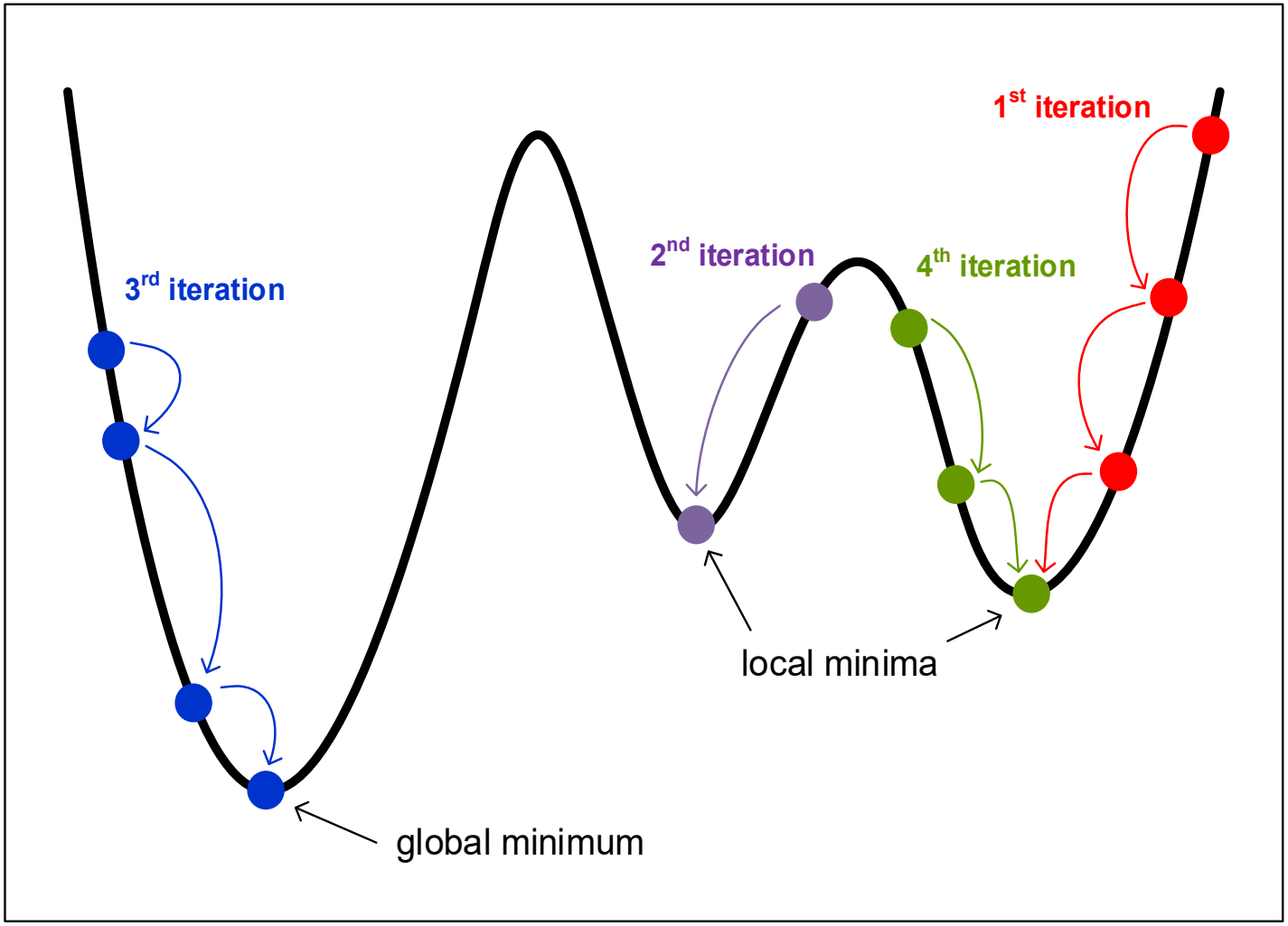
Multi Start Hill Climbing Heuristic (I)

- This heuristic combines the two algorithmic strategies:
  1. to build a solution from the scratch
  2. to get a better solution from a known solution
- In a problem aiming to minimize function $F(z)$, it works as follows:

$F_{best} = +\infty$
repeat
$x = BuildSolution\ ()$
$z = HillClimbing\ (x)$
if $F(z) < F_{best}$ **then** $x_{best} = z$ **e** $F_{best} = F(z)$
until Stopping Criteria is met

- Examples of Stopping Criteria:
  - Run a predefined time duration
  - Run a predefined number of iterations
  - Run until F_{best} not improving a predefined number of iterations

Multi Start Local Search Heuristic (II)

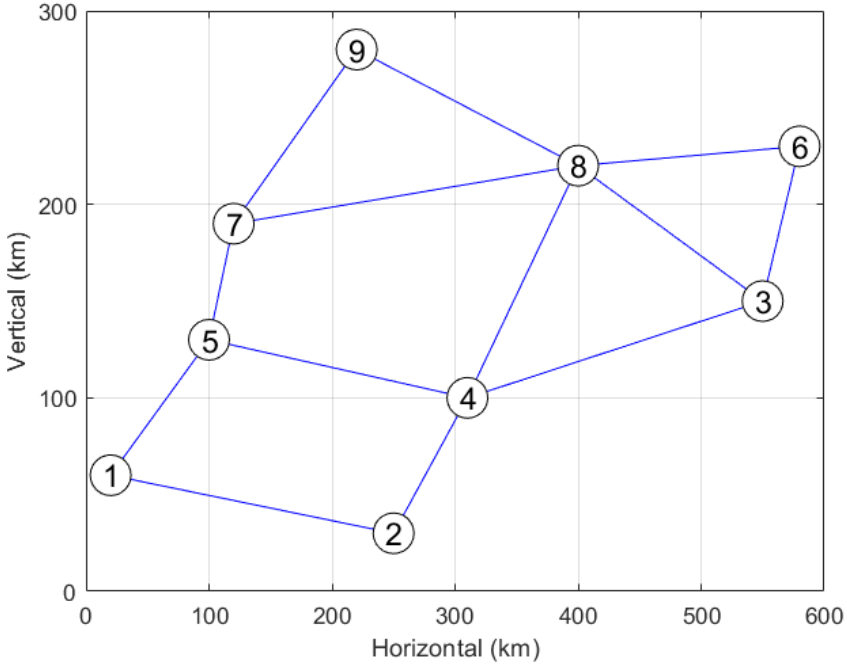


Practical Task 2

Consider the network with all links with a capacity of 10 Gbps.

Consider a unicast service with the following flows (throughput values b_t and $\underline{b}_t$ in Gbps):

t	o_t	d_t	b_t	$\underline{b}_t$
1	1	3	1.0	1.0
2	1	4	0.7	0.5
3	2	7	3.4	2.5
4	3	4	2.4	2.1
5	4	9	2.0	1.4
6	5	6	1.2	1.5
7	5	8	2.1	2.7
8	5	9	2.6	1.9



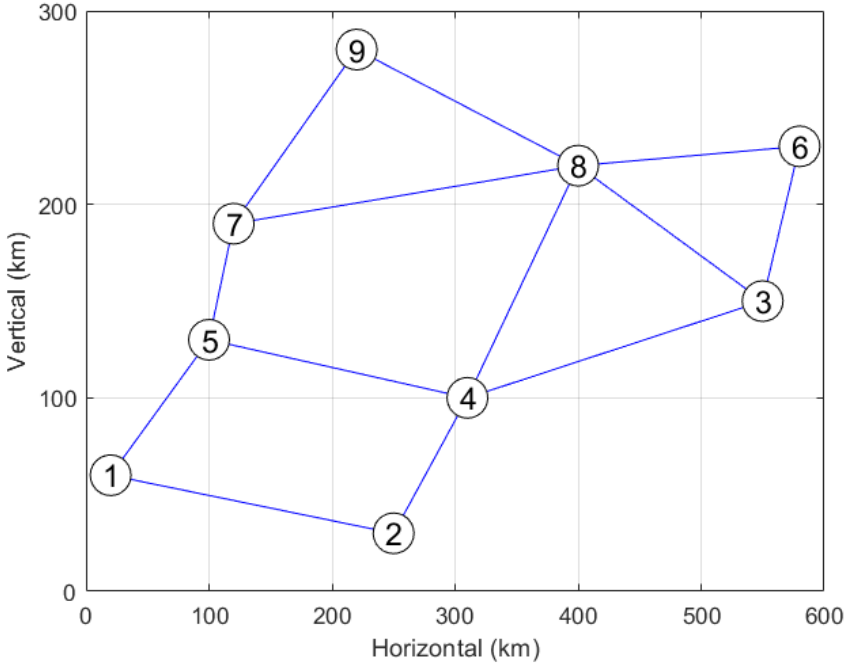
A. Based on the provided MATLAB code, compute a symmetrical single routing path solution with minimum worst link load using all possible routing paths and resorting to a multi start hill climbing heuristic.

B. Compare the efficiency of this algorithm with the algorithms tested in the Practical Task of the last class.

C. Run this algorithm using only the 6 shortest paths of each flow and compare the obtained solutions with the previous ones.

Practical Task 2

Consider the network with all links with a capacity of 10 Gbps.



Consider a unicast service with the following flows (throughput values b_t and $\underline{b}_t$ in Gbps):

t	o_t	d_t	b_t	$\underline{b}_t$
1	1	3	1.0	1.0
2	1	4	0.7	0.5
3	2	7	3.4	2.5
4	3	4	2.4	2.1
5	4	9	2.0	1.4
6	5	6	1.2	1.5
7	5	8	2.1	2.7
8	5	9	2.6	1.9

D. Consider that the link energy consumption is proportional to its length. Adapt the multi start hill climbing heuristic to compute a symmetrical single routing path solution minimizing the energy consumption of the network.

E. Compare the efficiency of this algorithm with the algorithms tested in the Practical Task of the last class.

F. Run this algorithm using only the 6 shortest paths of each flow and compare the obtained solutions with the previous ones.