Traffic Engineering in Packet Switched Networks

Desempenho e Dimensionamento de Redes

Prof. Amaro de Sousa (asou@ua.pt)
DETI-UA, 2016/2017

Traffic engineering

- Consider a network composed by a set of point-to-point links and supporting a set of flows S, such that all flows have the same average packet size.
  - The network is modelled by a graph $G=(N,A)$ where N is the set of network nodes and A is the set of network links. The element $(i,j) \in A$ represents the link between nodes $i \in N$ and $j \in N$ from i to j whose capacity is c_{ij} (in packets/second).
  - Each flow $s \in S$ is defined by its origin node o_s, destination node d_s and average packet arrival rate λ_s.
  - For each flow $s \in S$, P_s is the set of the routing paths existent in graph G from its origin node o_s to its destination node d_s.

The traffic engineering task is the task of choosing for each flow $s \in S$ the percentage of its average arriving rate λ_s that must be routed through each of the routing paths of P_s.

Traffic engineering with single path routing

- In the single path routing, each flow must be routed through one single path (no flow bifurcation is allowed).
- Moreover, it is also usual to require symmetrical routing, *i.e.*, 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_{sp} associated to each flow $s \in S$ and each routing path $p \in P_s$ that, when is 1, indicates that flow s is routed through path p.

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

- For each flow $s \in S$, one of its associated variables x_{sp} must be 1 and all other associated variables must be 0.
- For each link $(i,j) \in A$, the sum of the average arriving rates λ_s of all flows routed through it cannot be higher that its capacity c_{ij}.

Traffic engineering objectives

The traffic engineering task aims to:

- optimize one (or more) parameter(s) related with either the network performance and/or the quality of service provided by the network;
- optionally, guarantee (maximum or minimum) bounds for other parameters related with either the network performance and/or the quality of service provided by the network.

Examples of optimization parameters:

- global average packet delay (using, for example, the Kleinrock approximation)
- worst case average packet delay (to maximize the fairness among all flows)
- worst link load (to maximize the robustness of the network to unpredictable traffic growth)

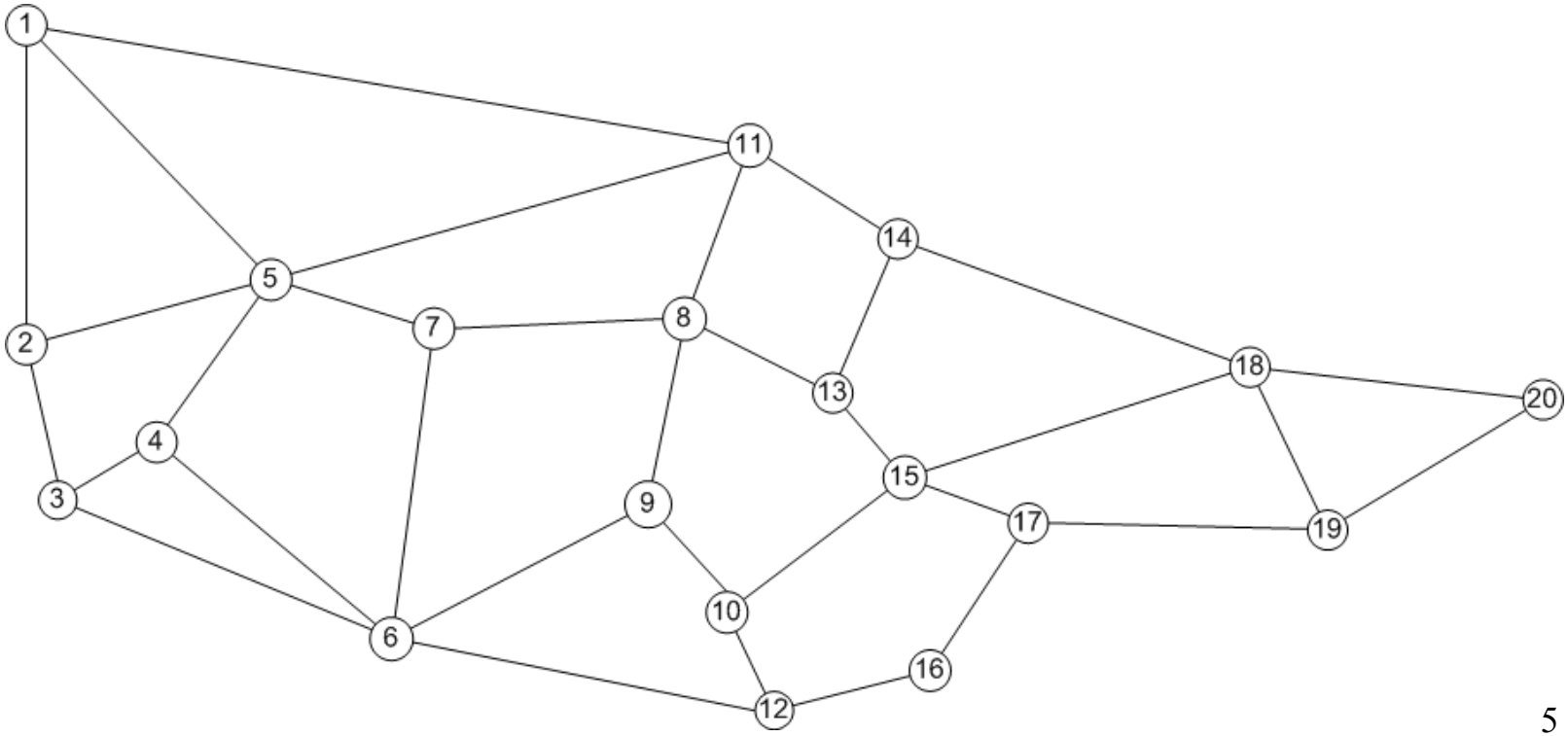
The best traffic engineering solution depends on the objective function defined by the network operator. Different objective functions might be contradictory

- for example, the solution that minimizes the global average delay can have a bad worst link load and vice-versa.

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 determined by the speed of light which is 3 x 10^8 meters/s.



Example – flow matrix

Consider the following flow matrix (values in Mbps) where all flows have an average packet size of 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 flow $s \in S$ 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:

Global average packet delay = 2.45 ms

Worst case average packet delay = 6.06 ms

Worst link load = 99.3%

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

Example – optimal solutions

Minimization of the global average delay:

Global average delay = 2.34 ms
Worst case average delay = 5.23 ms
Worst link load = 89.0%

Minimization of the worst case average delay:

Global average delay = 2.34 ms
Worst case average delay = 5.21 ms
Worst link load = 93.6%

Minimization of the worst link load:

Global average delay = 2.75 ms
Worst case average delay = 8.63 ms
Worst link load = 69.9%

Results analysis:

- The 1st and 2nd solutions are similar and exhibit an high worst link load.
- The 3rd solution is much more robust to unpredictable traffic growth but it exhibits more delay unfairness between flows.

Example – minimization of the energy consumption

Recently, energy aware routing has become a concern.

The technologies are evolving such that network links (and nodes) turn into a sleeping mode (with a very low energy consumption) if they have no traffic to route.

In the previous example, we consider a link energy consumption proportional to the link capacity. The optimal solution is:

Global average delay = 3.12 ms

Worst case average delay = 10.54 ms

Worst link load = 82.4%

Number of active links = 26 out of 33

Result analysis:

- We obtain an energy consumption reduction of 21.2% (=(33–26)/33) at the cost of worst values on delay performance parameters.
- Worst link load is not a problem since the sleeping links can turn into active if traffic grows.

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 from the scratch.
   - Examples: *random*, *greedy, greedy randomized, etc…* .
2. To get a better solution from a known solution.
   - Examples: *local search, tabu search, simulated annealing, etc... (*we will address only the local search strategy).

Building a solution from the scratch (I)

Random strategy:

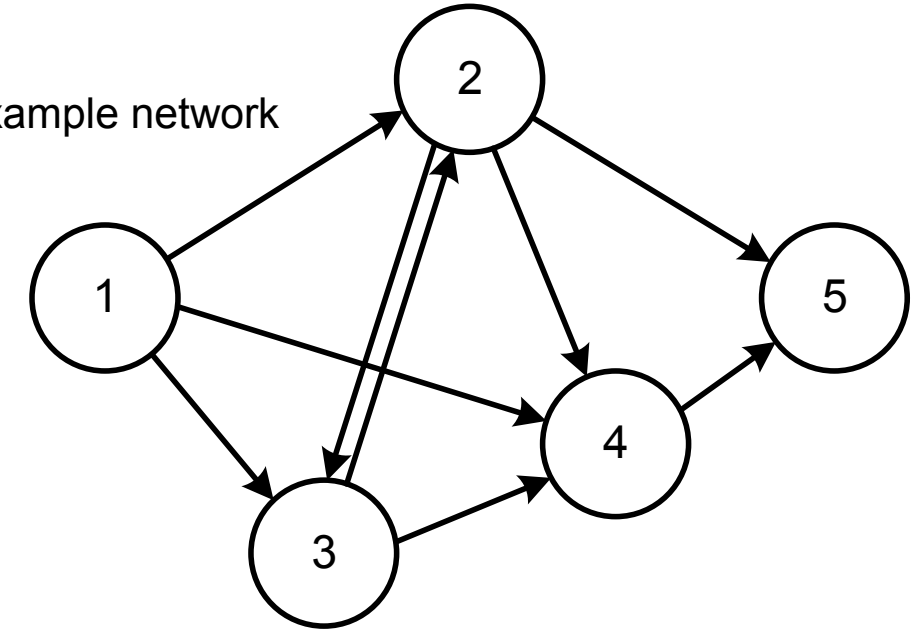
- We assign a random routing path $p \in P_s$ to each flow $s \in S$
- We obtain a slightly better performance if we consider higher probabilities to routing paths $p \in P_s$ with "better characteristics"
  - For example, paths with a smaller number of links or paths containing links of larger capacity,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

Greedy strategy:

- We start by considering the network without any routing path
- For each flow $s \in S$:
  - We assign the routing path $p \in P_s$ that, together with the previous assigned 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

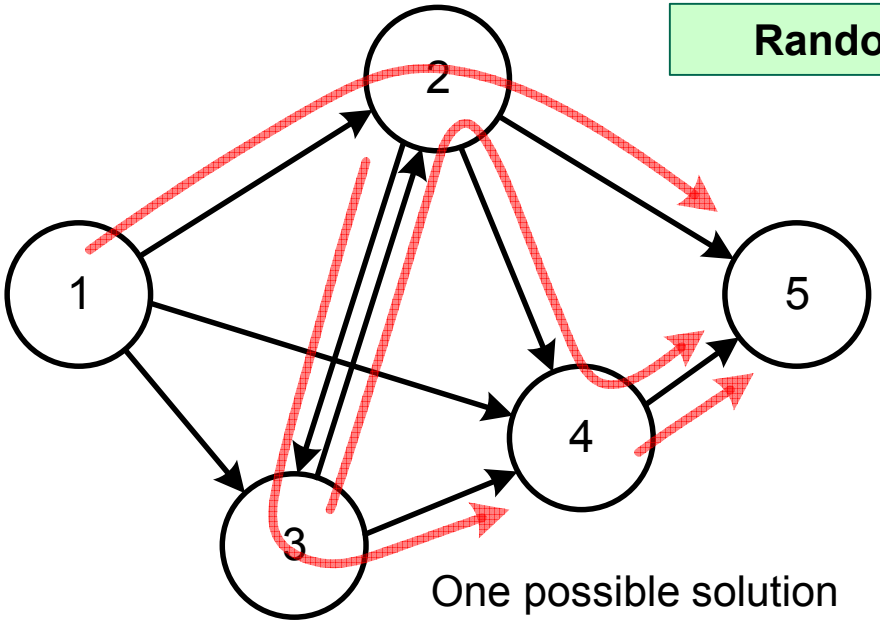
From 3 to 5:
- 3-4-5
- 3-2-5
- 3-2-4-5

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

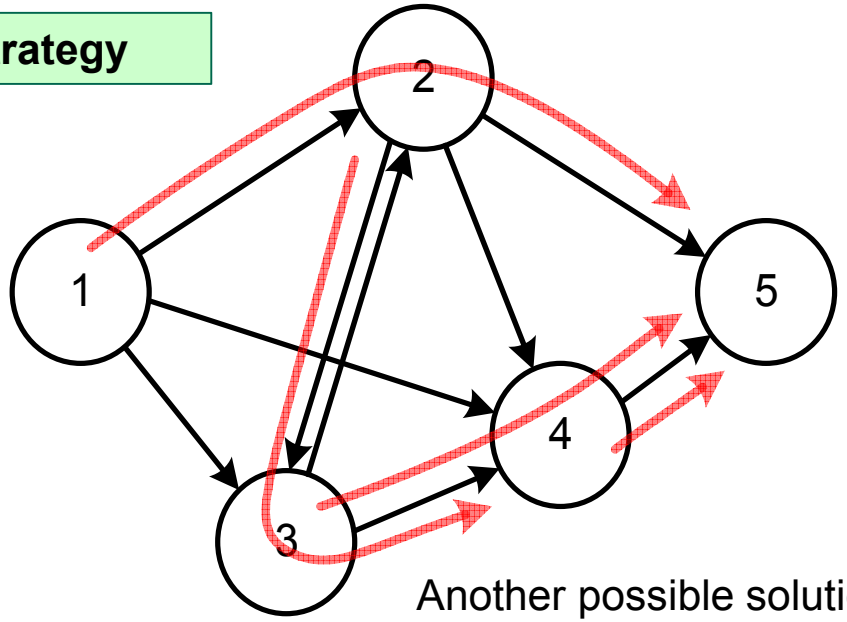
From 4 to 5:
- 4-5

Candidate paths for each flow

Random strategy

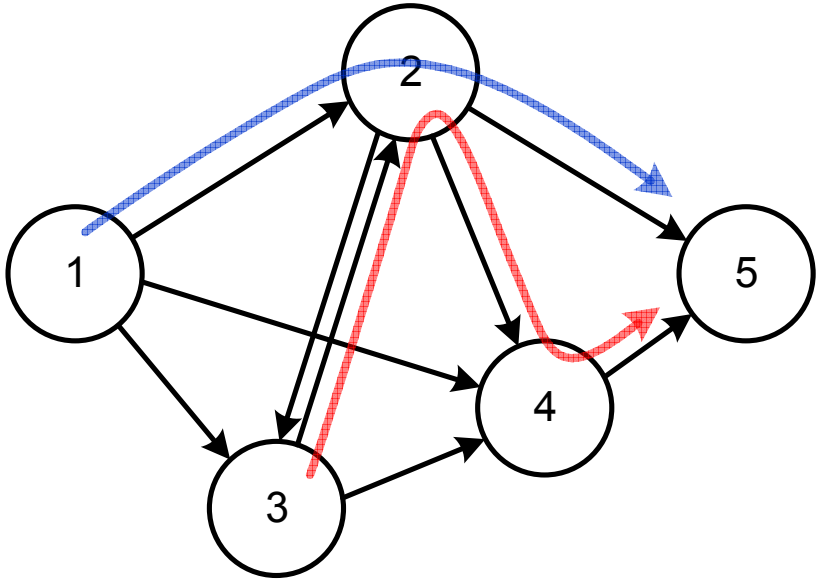


One possible solution

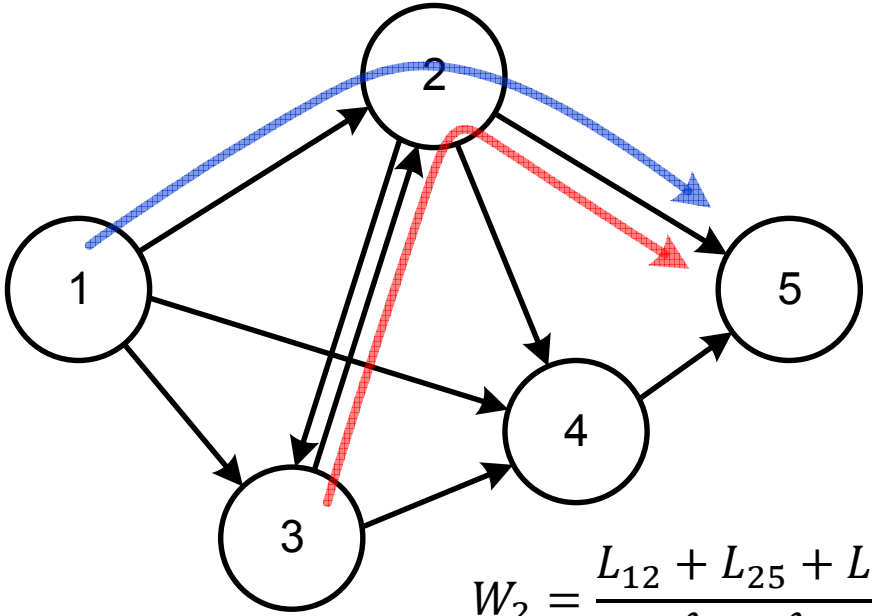


Another possible solution

Building a solution from the scratch (III)



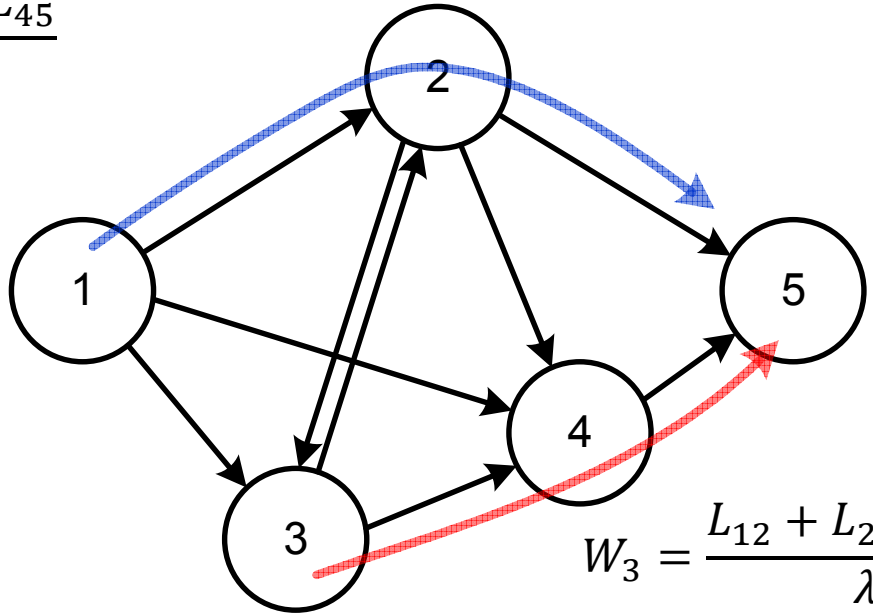
$$W_1 = \frac{L_{12} + L_{25} + L_{32} + L_{24} + L_{45}}{\lambda_1 + \lambda_2}$$



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

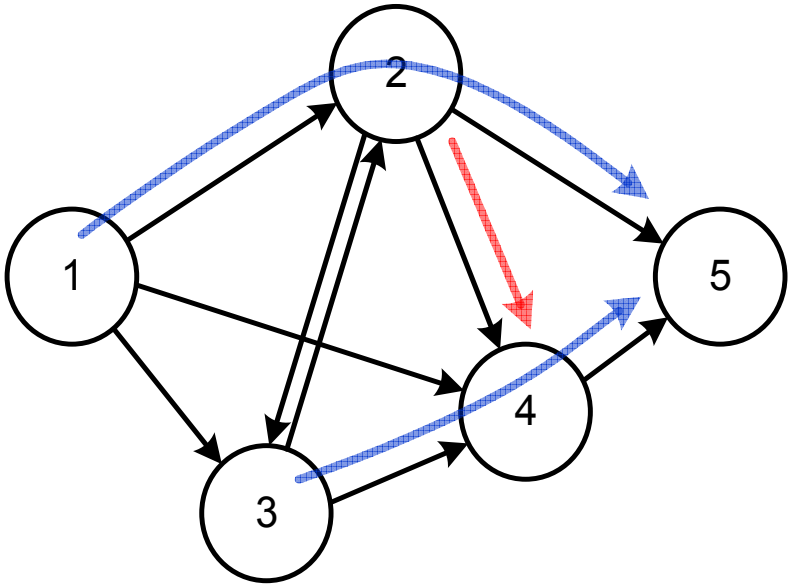
Greedy strategy:

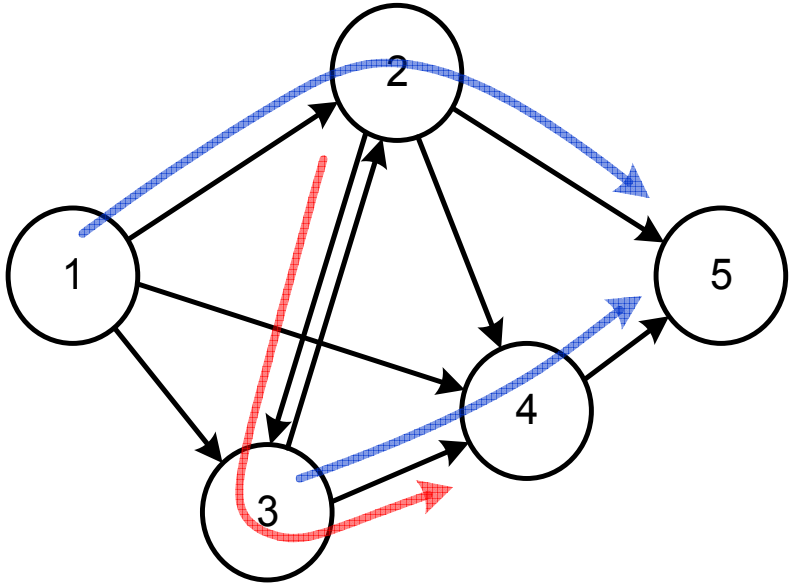
- 1st flow (blue) already installed
- Try all candidate paths for 2nd flow (red) and select the solution with lowest global average delay



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

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 installed
- Try all candidate paths for 3rd flow (red) and select the solution with lowest global average 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 compute the routing paths of the flows $s \in S$

Second alternative:

- For each flow $s \in S$, to 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 have a combination of the 2 previous alternatives

Getting a better solution from a known solution (I)

Local search strategy – best neighbour move variant

In this strategy, we start by an initial solution and try to move to a better solution by making local changes. 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 the best neighbour solution
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)

Note that in step 1., all neighbour solutions are computed (this is the best neighbour move variant).

Getting a better solution from a known solution (II)

Local search strategy – first neighbour move variant

If the evaluation of all neighbours of a current solution is too heady (either because a solution is hard to compute or because the neighbour set is very large), the previous variant might not be 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 the neighbour solutions until we find a solution better than the current one
2. If such a neighbour solution exists, we set the current solution with the computed 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 (III)

Local search strategy – defining the set of neighbour solutions

The set of neighbour solutions (of a given solution) is problem dependent.

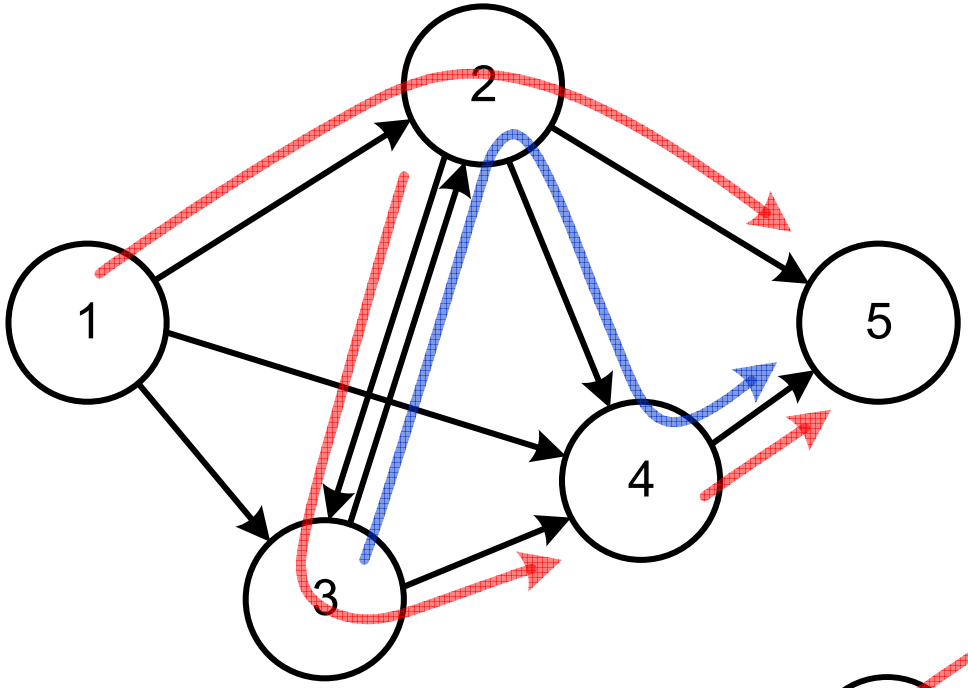
The neighbour set must be carefully defined in order to allow the algorithm to compute all neighbour solutions in reasonable runtime.

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

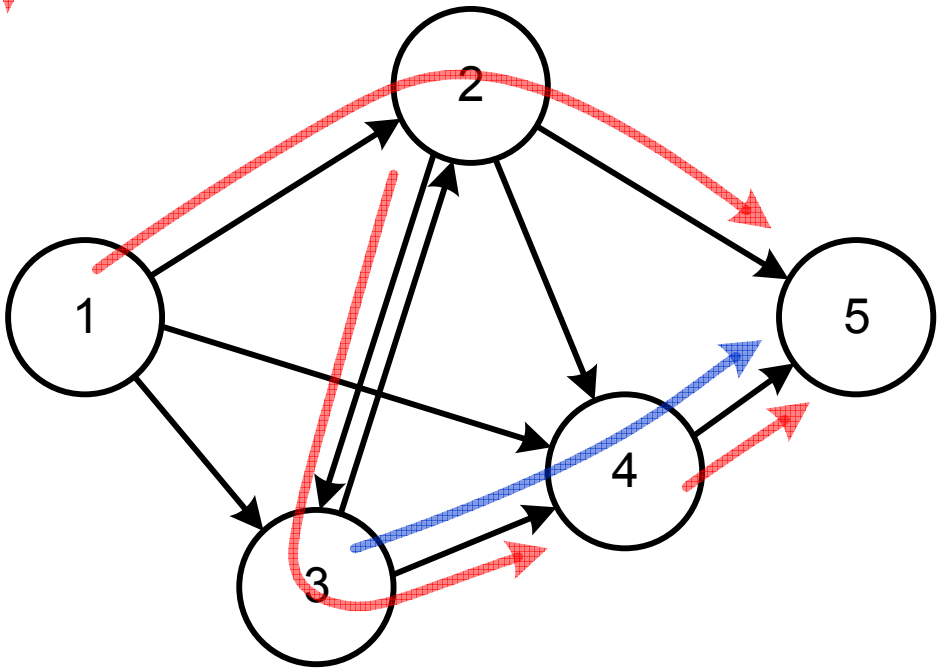
- For a current solution (that defines a routing path $p \in P_s$ for each flow $s \in S$), a neighbour solution is a solution that differs from the current one in the routing path of a single flow.
- When the set of routing paths P_s is defined, the number of neighbour solutions is:

$$\sum_{s \in S} (|P_s| - 1)$$

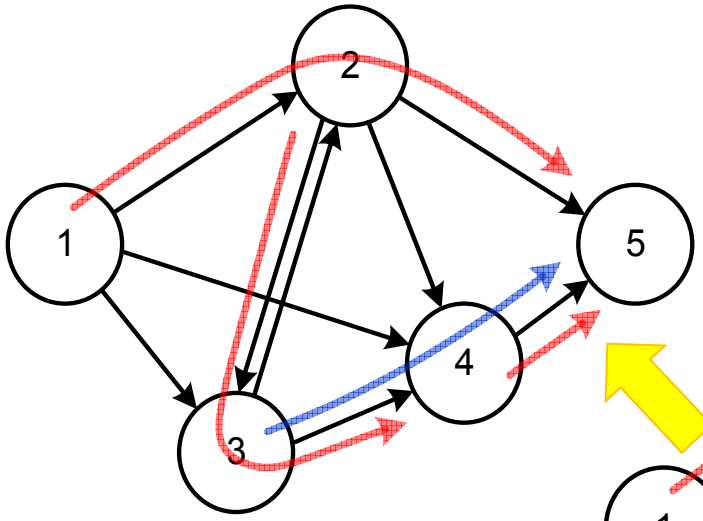
Getting a better solution from a known solution (IV)



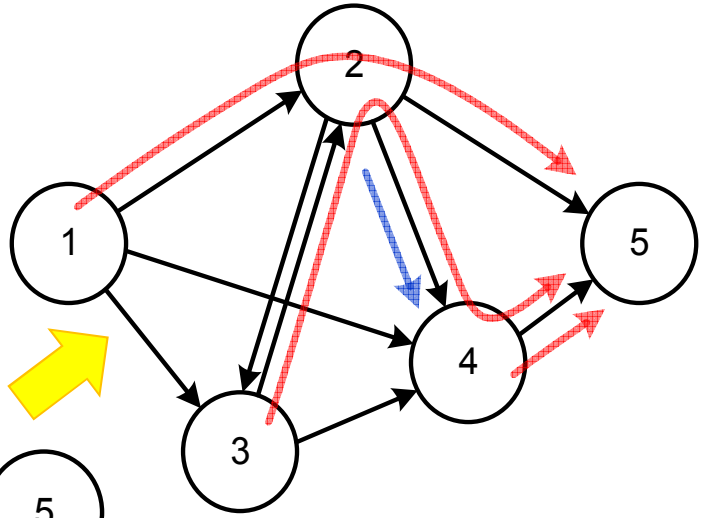
Neighbour solutions



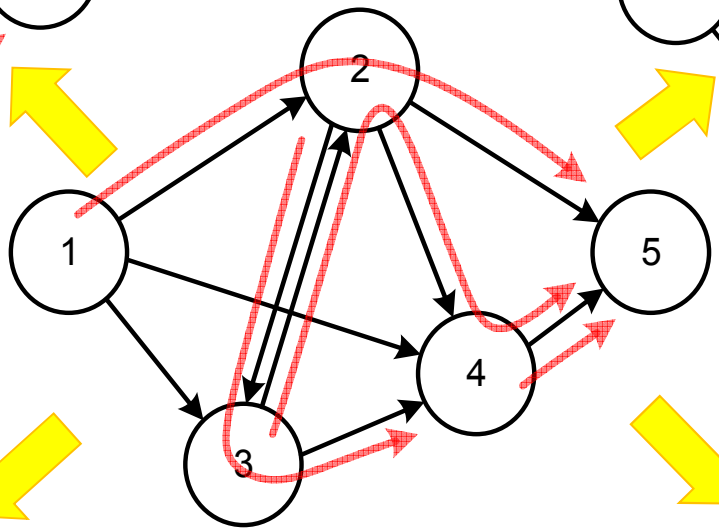
Getting a better solution from a known solution (V)



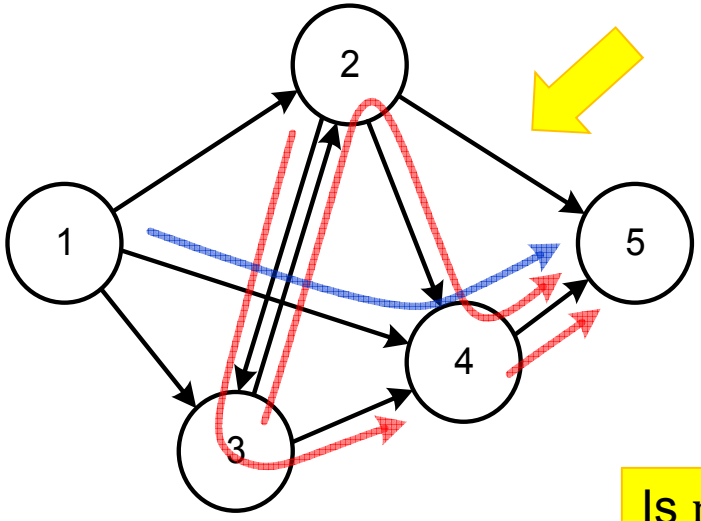
neighbour solution, W_1



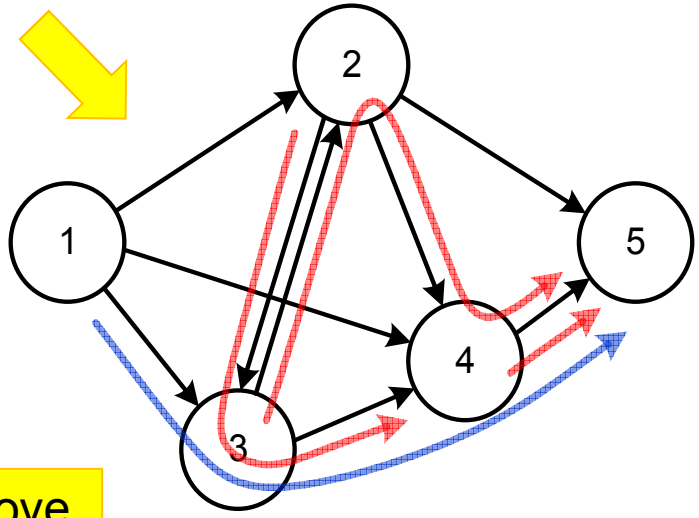
neighbour solution, W_2



current solution, W



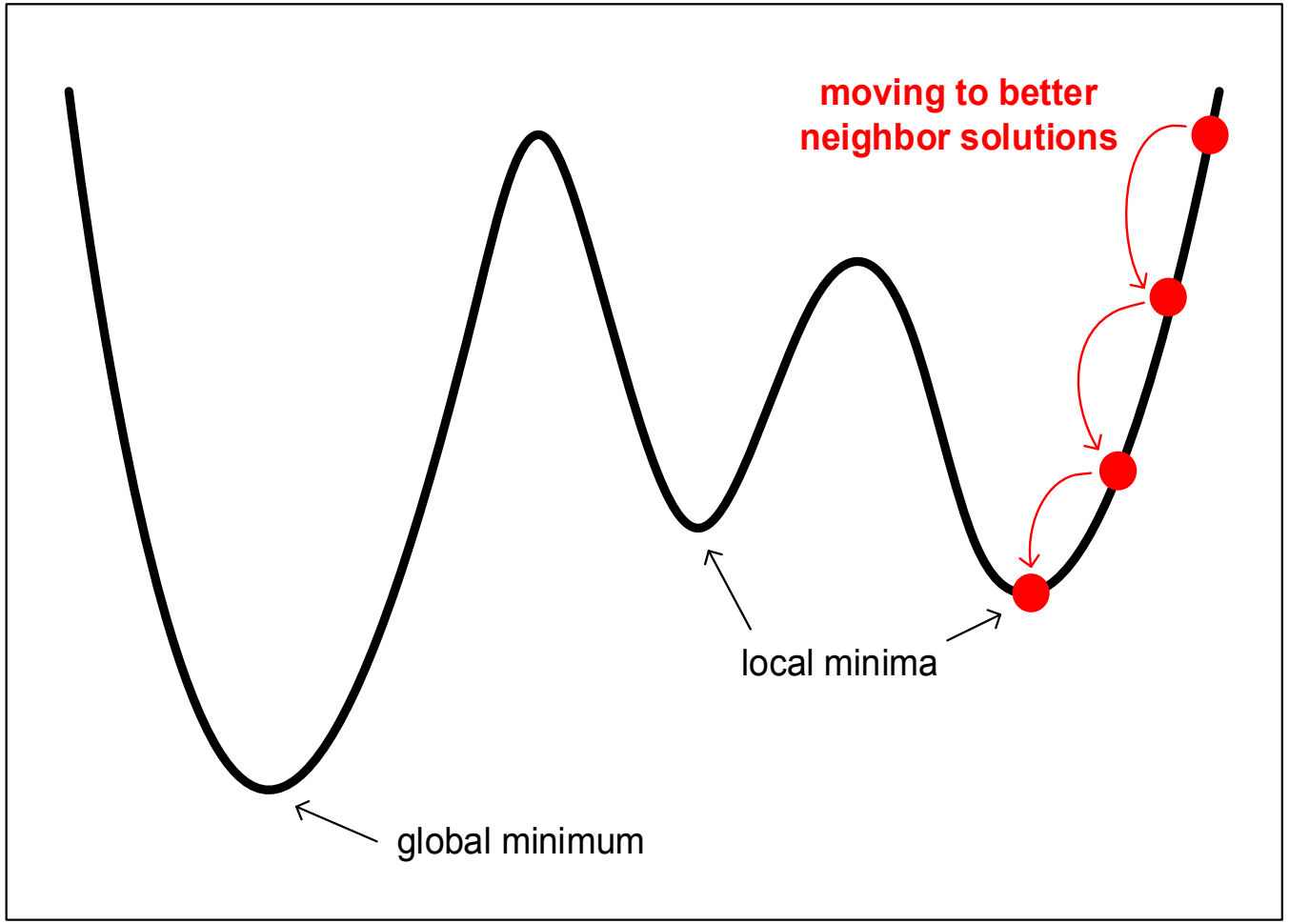
neighbour solution, W_3



neighbour solution, W_4

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

Getting a better solution from a known solution (VI)



Multi Start Local Search Heuristic (I)

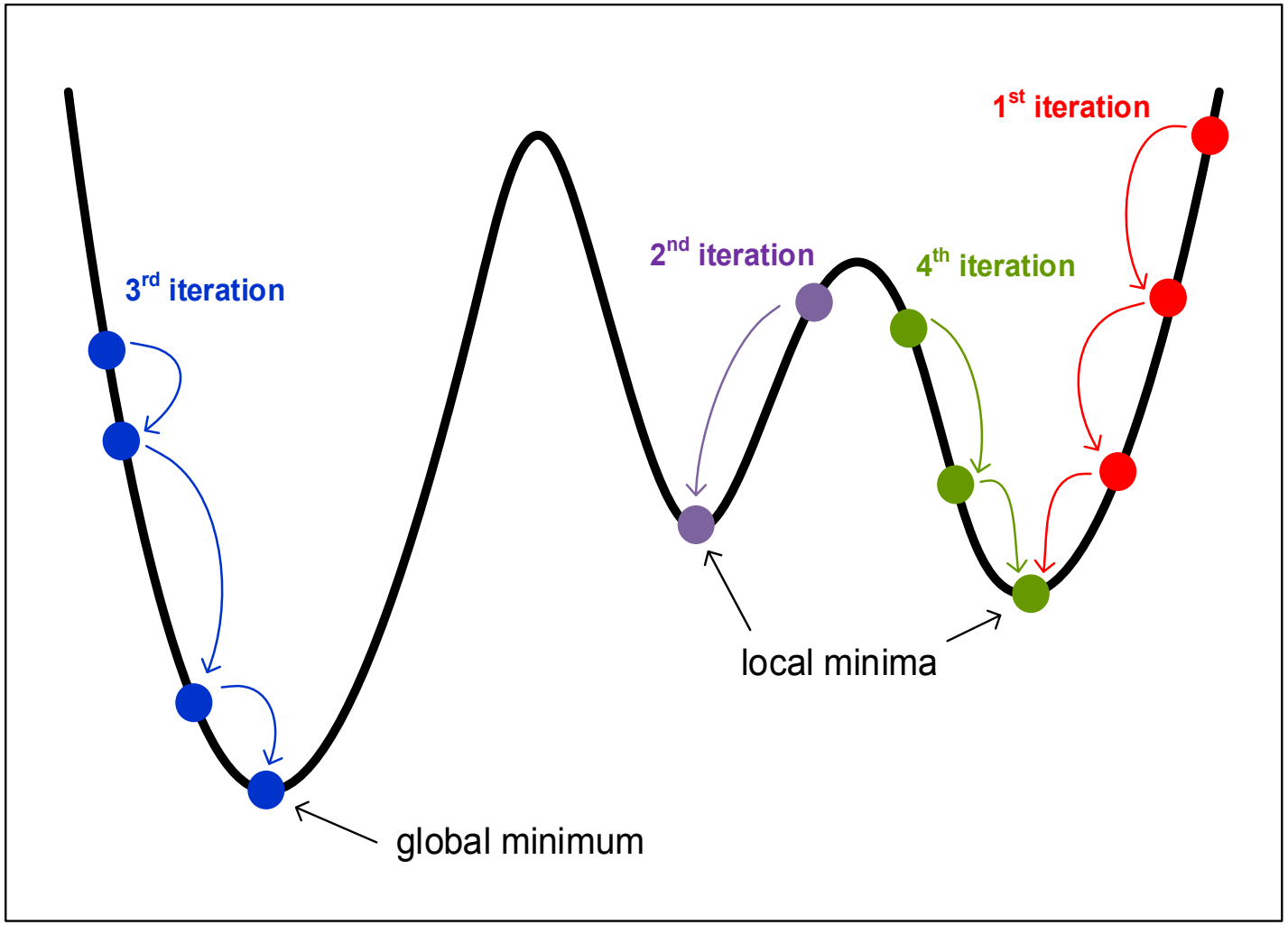
- This heuristic combines the two algorithmic strategies:
  1. to build a solution from the scratch
  2. to get a better solution from a known solution

- In a problem aiming to minimize function $F(z)$, it works as follows:

$F_{best} = +\infty$
repeat
$x = BuildSolution\ ()$
$z = LocalSearch\ (x)$
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 number of iterations
  - Run until F_{best} not improving a predefined number of iterations

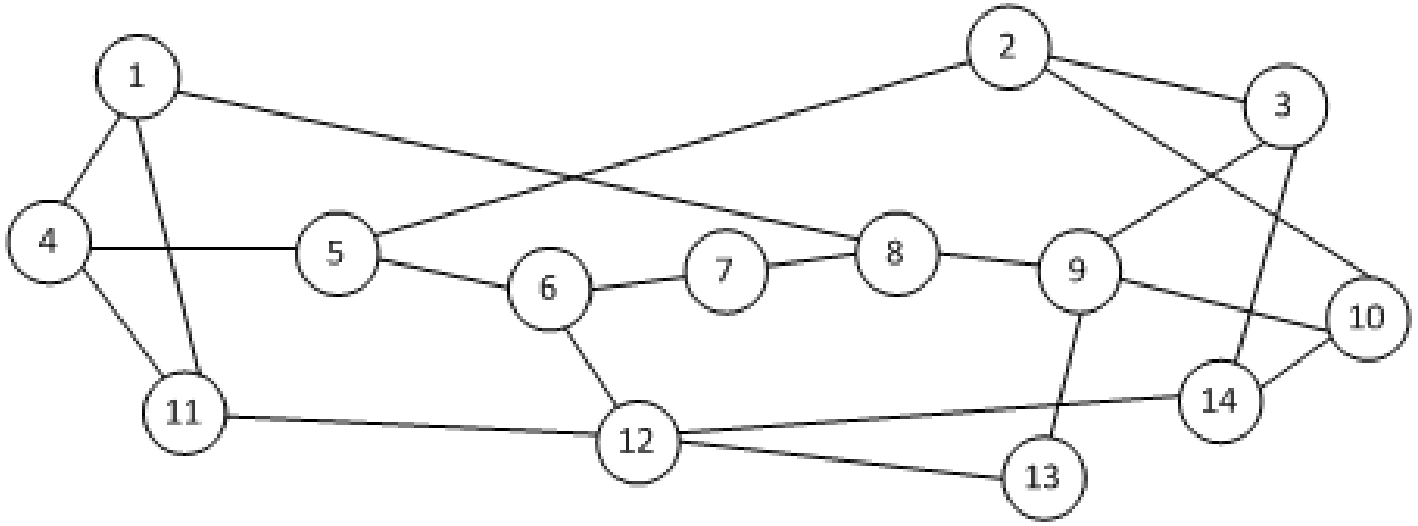
Multi Start Local Search Heuristic (II)



Non defined P_s sets (I)

The sets P_s (with the available routing paths for each flow $s \in S$) might be too large.

For example, in this network, there are more than 200 possible routing paths for any pair of network nodes.



If $|P_s|$ are large, the neighbour set of a local search algorithm becomes too large and the performance of the algorithm is penalized.

Non defined P_s sets (II)

In these cases, an alternative approach is to use a minimum cost path algorithm (like the Dijkstra's algorithm).

When a routing path is to be assigned to a flow:

1. We first assign an "appropriate cost value" to each link.
2. We run Dijkstra's algorithm to determine the minimum cost path.

The "appropriate cost values" must be carefully selected and they should depend on the objective to be optimized.

Examples:

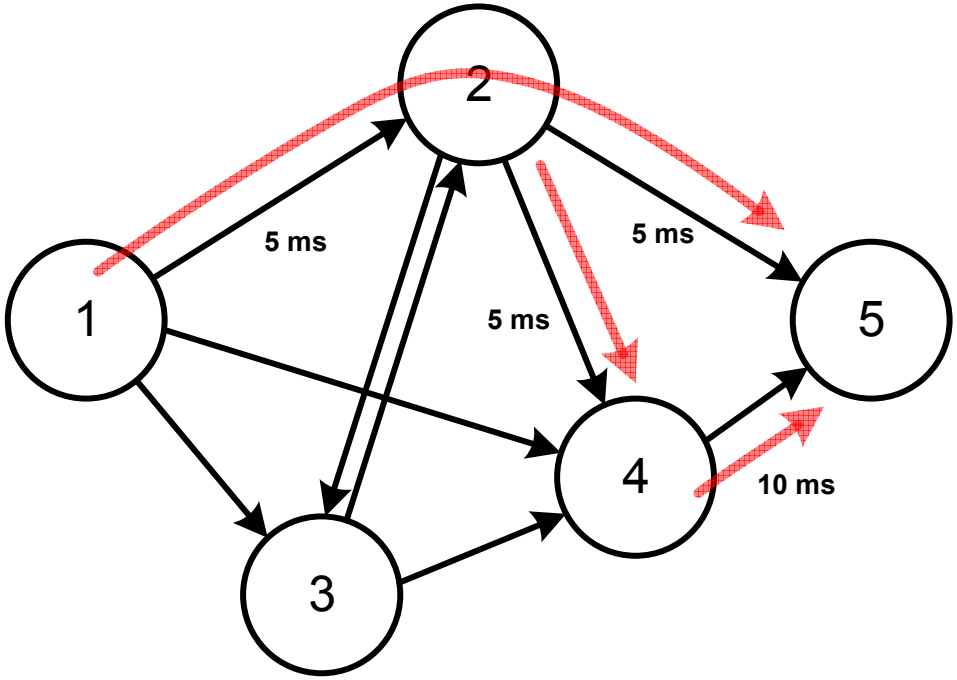
- If the objective function is related with flow packet delays, the cost values must be based on the link delays.
- If the objective function is related with link loads, the cost values must be based on the link loads.

Non defined P_s sets (III)

Greedy randomized strategy:

- We start by considering the network without any routing path
- We determine a random order to compute the routing paths of the flows $s \in S$
- For each flow $s \in S$, and by the previous determined order:
  - we assign a cost to each link based on the current routed flows
  - we execute Dijkstra's algorithm to compute the routing path p of flow s
  - we update the network by routing the average packet rate λ_s of flow s through its assigned path p

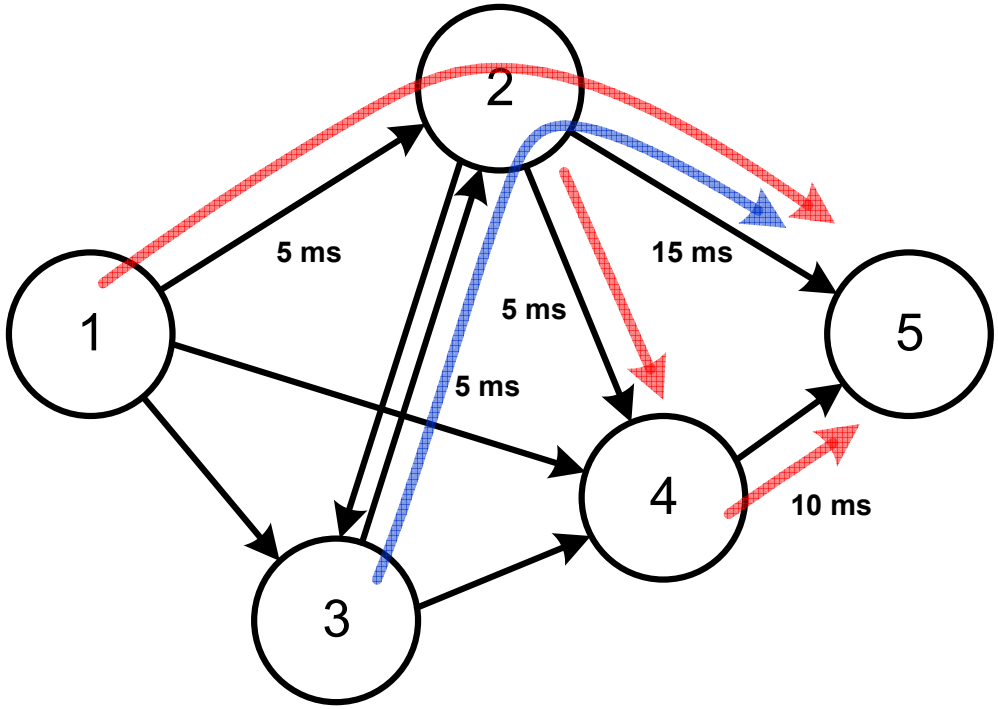
Non defined P_s sets (IV)



- Previous solution, link delays calculated based on installed flows

$$W_{ij} = \frac{1}{\mu_{ij} - \lambda_{ij}} + d_{ij}$$

Next solution, shortest path based on link delays of previous solution



Non defined P_s sets (V)

Local search strategy:

(Remember that) the neighbour set is defined as:

- For a current solution (that defines a routing path *p* for each flow $s \in S$), a neighbour solution is a solution that differs from the current one in the routing path of a single flow.

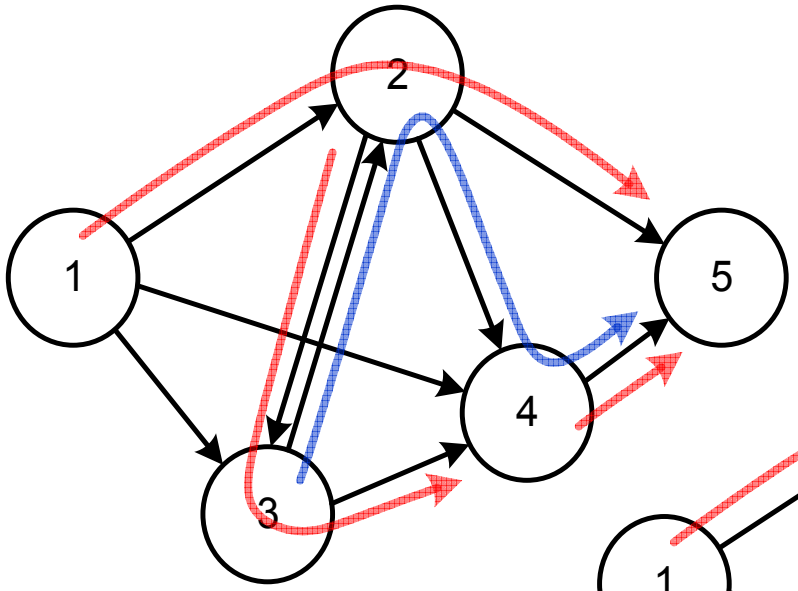
The neighbours of a current solution are computed as follows:

➢For each flow $s \in S$:

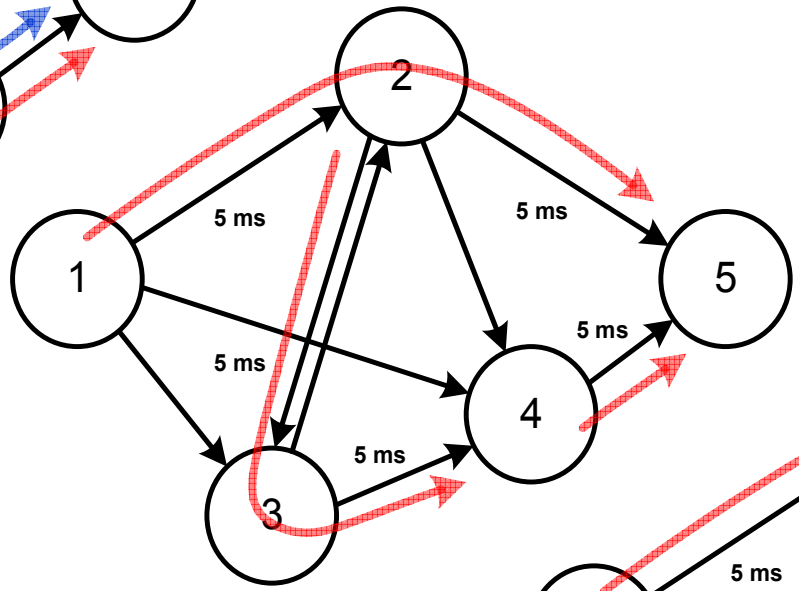
- we determine the network solution with all flows except *s*
- we assign a cost to each link based on the routed flows
- we execute Dijkstra's algorithm to compute the routing path *p* of flow *s*

Note that, in this case, a neighbour solution might be equal to the current solution.

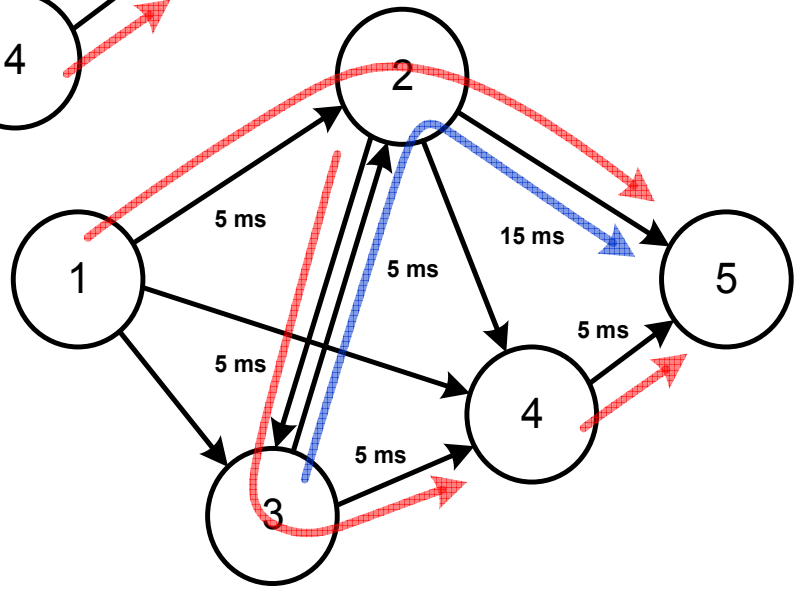
Non defined P_s sets (VI)



1. Remove path



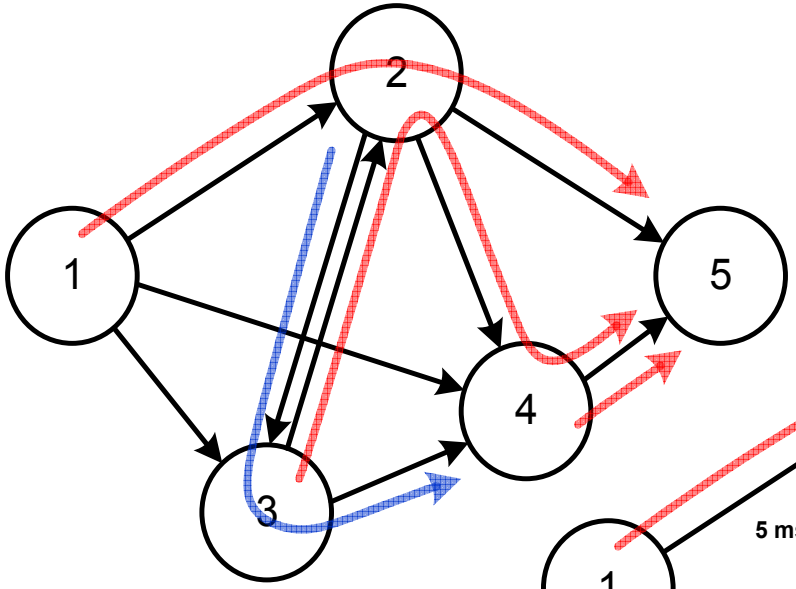
2. Assign costs to links



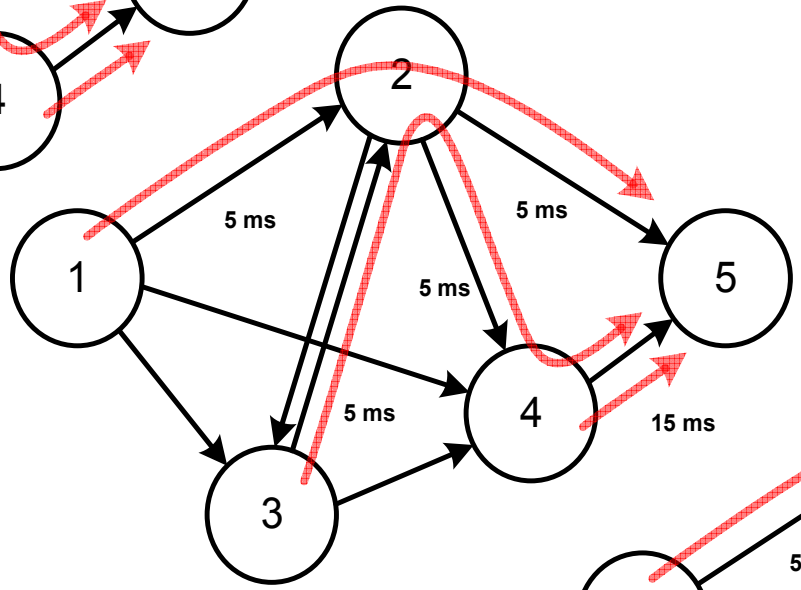
3. Determine shortest path of removed flow (using Dijkstra)

Steps in building neighbor solution

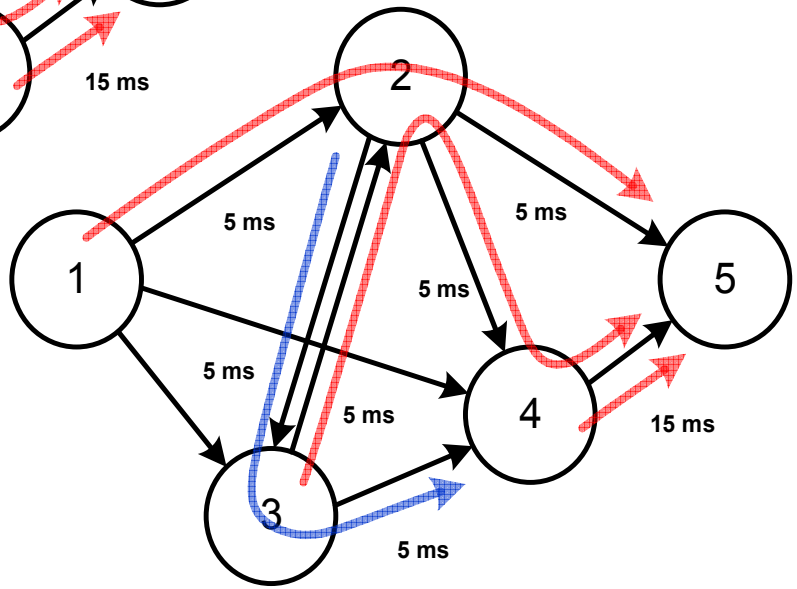
Non defined P_s sets (VII)



1. Remove path



2. Assign costs to links



3. Determine shortest path of removed flow (using Dijkstra)

Neighbor solution may equal current solution

Multi Start Local Search Heuristic with non defined P_s

Example of a MATLAB code:

```
GlobalBest= Inf;
for iter=1:Iterations
    CurrentSolution= GreedyRandomized();
    CurrentObjective= Evaluate(CurrentSolution);
    repeat= true;
    while repeat
        NeighbourBest= Inf;
        for i=1:size(CurrentSolution,1)
            NeighbourSolution= BuildNeighbour(CurrentSolution,i);
            NeighbourObjective= Evaluate(NeighbourSolution);
            if NeighbourObjective < NeighbourBest
                NeighbourBest= NeighbourObjective;
                NeighbourBestSolution= NeighbourSolution;
            end
        end
        if NeighbourBest < CurrentObjective
            CurrentObjective= NeighbourBest;
            CurrentSolution= NeighbourBestSolution;
        else
            repeat= false;
        end
    end
    if CurrentObjective < GlobalBest
        GlobalBestSolution= CurrentSolution;
        GlobalBest= CurrentObjective;
    end
end
```

Computes a solution with a Greedy Randomized algorithm

Computes a neighbour solution

Computes the objective function value of a solution

Multi Start Local Search Heuristic with non defined P_s

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

Example of a MATLAB code:

```
GlobalBest= Inf;
for iter=1:Iterations
    CurrentSolution= GreedyRandomized();
    CurrentObjective= Evaluate(CurrentSolution);
    repeat= true;
    while repeat
        NeighbourBest= Inf;
        for i=1:size(CurrentSolution,1)
            NeighbourSolution= BuildNeighbour(CurrentSolution,i);
            NeighbourObjective= Evaluate(NeighbourSolution);
            if NeighbourObjective < NeighbourBest
                NeighbourBest= NeighbourObjective;
                NeighbourBestSolution= NeighbourSolution;
            end
        end
        if NeighbourBest < CurrentObjective
            CurrentObjective= NeighbourBest;
            CurrentSolution= NeighbourBestSolution;
        else
            repeat= false;
        end
    end
    if CurrentObjective < GlobalBest
        GlobalBestSolution= CurrentSolution;
        GlobalBest= CurrentObjective;
    end
end
```

Main Cycle

$x = BuildSolution()$

$z = LocalSearch(x)$

Calculation of Best Neighbour Solution

Move to Neighbour Solution

Exit Local Search

Update Global Best Solution