# POLITECNICO DI TORINO

## Operational Research

### Laboratory 02 – Greedy Heuristics


## – Components of the team –

"We declare that what is written in this relation is the result of our work"

**Angius Marco** **S240515**

**Avalle Giorgio** **S241834**

## Introduction

The goal of this laboratory is to solve the LTD problem implementing, on our own, an heuristic algorithm: results has to be compared with the ones that a random approach would lead to.

Our choice is to work with Python language: in particular, we will use a third party library called "**NetworkX**[1]", which provides graph data structure support and several implementation of the classical algorithms (for example, the one used to find every simple path existing between a pair of nodes). This library provides also tools to visualize the resulting graph, using functions of another well-known third party library called "**Matplotlib**[2]".

Once we have read the documentation, we started developing the code: apart from the exercises implementations, the python files created are

- **LAB2_OpRes**: it contains the heuristic algorithms and, if executed, allows the user to specify the parameters of the LTD problem and to solve it
- **graph_topologies**: functions used to generate particular graph topologies
- **graph_traffic_matrix**: functions used to generate a traffic matrix, whose values are generated between the specified range(s)
- **flow_utilities**: functions used to retrieve, manipulate and display information about the traffic flow values of the graph's edges
- **ltd_utilities**: functions used by the main file, solving the LTD problem given
- **input_controls**: functions used to verify the correctness of an input parameter

These libraries are entirely realized by us, using functionalities of the ones declared above or the native ones. Please, note that the input controls are done exhaustively in the "*LAB2_OpRes.py*" functions: we are assuming that the other libraries will not be called directly by the user, so there is no need to verify again the input parameters. This assumption is done in order to avoid to recheck the same values multiple times, speeding up the computational time required to solve the LTD problem.

The code is fully commented: for this reason, we do not provide here the full documentation of every function we have written. Anyway the commented code will be provided, for your convenience, in double version: English and Italian language.

**WARNING**: if you do not have the libraries "*NetworkX*" and "*Matplotlib*" installed on your pc, you would not be able to run the programs we have wrote. In this case, please follow the setup instructions as declared in the official web pages we have reported in the footer.

## Forecasts

According to the theory, we generally expect not to reach optimum solutions: however, heuristics (if well implemented) are able to give us solutions which are close to the optimal one, saving computational time.

In particular, over the same traffic matrix, we expect to obtain topologies in which the maximum flow value is usually slightly lower (better) when the heuristic approach is adopted, with respect to the random approach. In fact, although a random algorithm could potentially lead us to the optimal solution, a heuristic one can take advantage of the knowledge of the problem we are going to face.

[1] NetworkX reference page: https://networkx.readthedocs.io/en/stable/
[2] Matplotlib reference page: https://matplotlib.org/

## The water filling technique

This is the routing strategy we adopted: the technique is used to evaluate the traffic flow values of the graph's edges, according to the values declared into the associated traffic matrix.

### How it works

Given a graph topology, we consider every possible pair of nodes (s, d) and we evaluate existing paths between them: the final goal is to distribute the traffic "*f*" exchanged by "*s*" and "*d*" (which can be seen from the traffic matrix), incrementing every edge's flow value of these paths.

Each of these path has a maximum value "***f_max***", that is the maximum flow value between the edges of that path; "water fill" means that we first increment the flows of the paths characterized by a lower value of "*f_max*", until it is equal to the one of the next path to fill. Once every path has the same "*f_max*" associated value, the remaining part of "*f*" (if exists) is distributed equally.

### Implementation details

Actually, this is valid only for pairs (s, d) which are **not directly connected**: if the arc "s-d" exists in the topology, the correspondent traffic matrix value is entirely associated to it. Also, the **paths search depth is limited** in order to preserve computational resources (otherwise, the required time would be enormous in topologies with lot of edges): the starting depth value is passed as parameter (its default value is six) and it is incremented only if needed (no path found).

The functions implementing this technique are "*complete_water_fill*" (over a topology) and "*water_fill*" (over a set of paths): they can both be found in the file called "*flow_utilities.py*" (usually imported under the name "*flows*".

### A posteriori considerations

We have launched a very large number of simulations, in order to test algorithms behavior in several network conditions: four simulation per each of the four adopted approaches, for each pair of N/Δ. This operation has been extremely costly and, unfortunately, we were not able to provide results for values:

- N = 30, Δ = 25
- N = 40, Δ = 25, 29, 35

which were not directly requested by the requirements of this laboratory, but something of our interest.

What we have noticed is that the traffic routing (according to the water filling principle) is **much more heavier**, computationally speaking, than the network topology definition. In fact, the topology creation is something which often requires less than a second: but the routing part could require hours, for a single simulation.

This was the reason we couldn't achieve the result we were looking for: our computational capacity was not sufficient and the problem became infeasible. Although we have asked the permission to launch the script on the university's Big Data Cluster, we found that this operation was not possible: the cluster accepts only jobs (especially Java ones) which can be parallelized and run on several machines.

In this report, you can find computations results in an aggregate form: max flow values and their comparisons, with some consideration about the adopted approaches (algorithms). If you want to see the complete output information, they can be found in the "***LOG.zip***" archive attached to this paper.

## The LTD algorithms

We have realized (and compared each other) four types of possible approaches to solve the LTD problem: two greedy heuristic algorithms (mesh and ring based), a casual one (random topology) and the Manhattan topology algorithm. Their implementation code can be found into the file called "*LAB2_OpRes.py*".

Once the final topology is well-defined, flow values are assigned to the edges according to the "water filling" principle and the resulting information (graph, log files and statistics) are returned to the user.

### Greedy heuristic mesh approach

This heuristic is implemented by the function "***greedy_LTD_mesh***": the idea is to start from the full mesh topology, removing edges from it until the requirements on the maximum number of receivers/transmitters is satisfied ("*delta*" parameters). The algorithm is a variant of the D-MLTDA (Decreasing Multi-hop Logical Topology Design Algorithm), in which the routing is executed only at the end (because of its computational complexity) and no random edges are generated after the removal (because we want to have a greedy deterministic approach to the LTD problem).

Edges we try to remove first are the ones between pairs of nodes (s, d) characterized by the **least** traffic exchanged values (the value reported into the traffic matrix): clearly, an edge is removed only if its removal will not disconnect the graph and if it is necessary (because of the "*delta*" requirements). Edges between nodes that both respect the Delta constraints are never removed.

### Greedy heuristic ring approach

This heuristic is implemented by the function "***greedy_LTD_ring***": the idea is to start from the ring topology, adding edges from it until the requirements on the maximum number of receivers/transmitters can be satisfied ("*delta*" parameters). The algorithm is a variant of the I-MLTDA (Increasing Multi-hop Logical Topology Design Algorithm), in which the routing is executed only at the end (because of its computational complexity) and the starting point is a ring topology (in order to guarantee the graph connectivity).

Edges we try to insert first are the ones between pairs of nodes (s, d) characterized by the **highest** traffic values exchanged (the value reported into the traffic matrix).

### Random approach

Implemented by the function "***LTD_random***": starting from a ring (in order to guarantee the connectivity between graph's nodes), random edges are continuously added until the target number of edges the graph must have is reached and the "*delta*" constraints are satisfied.

Final computations (such as edge swap/creation) are done in order to **adapt** the topology, if infeasible. It can happen, in fact, that at a certain point the algorithm needs to add edges but the topology created so far does not allow this operation. As you can see in the code, the problem is solved swapping an arc before adding the new one: several possible cases are considered, in order to find always an improvement and reach the final desired result, without enter in an infinite loop.

### Manhattan topology problem

Implemented by the function "***LTD_manhattan***" and "***LTD_manhattan_smart***": the LTD problem is solved taking into account that the topology to generate has to be a Manhattan one (eventually, disposing nodes in a smarter way). Here, nodes are disposed as a rectangle Nr · Nc, where Nr is the number of nodes per row and Nc is the number of nodes per column.

## Ex 01 & 02

The LTD problem is solved using the greedy algorithms described above (**mesh**, **ring**): results are then compared, also with **random** topologies characterized by the same number of nodes and edges.

### Tests explanation

For every LTD problem formulation, the traffic matrix values are generated as uniform variables distributed between **0.5** and **1.5**: the number of nodes "N" and the constraint on the maximum number of receivers/transmitters per node "Δ" vary.

Every possible N/Δ combination is tested **four times** for each possible approach: in this way, maximum flow and other quality parameters, such as computational required time, are estimated as an average between the four measured. Clearly, different algorithms will use the same traffic matrix, while different attempts of the same algorithm will use different ones.

### Possible values

N = [3, 4, 6, 8, 10, 16, 20, 30, 40]
Δ = [1, 2, 3, 5, 7, 9, 15, 19, 25, 29, 35, 39]

Note that, for each N value, only Δs lower than it are considered (in fact, greater ones would produce the same result: a full mesh topology).

### What we have experienced

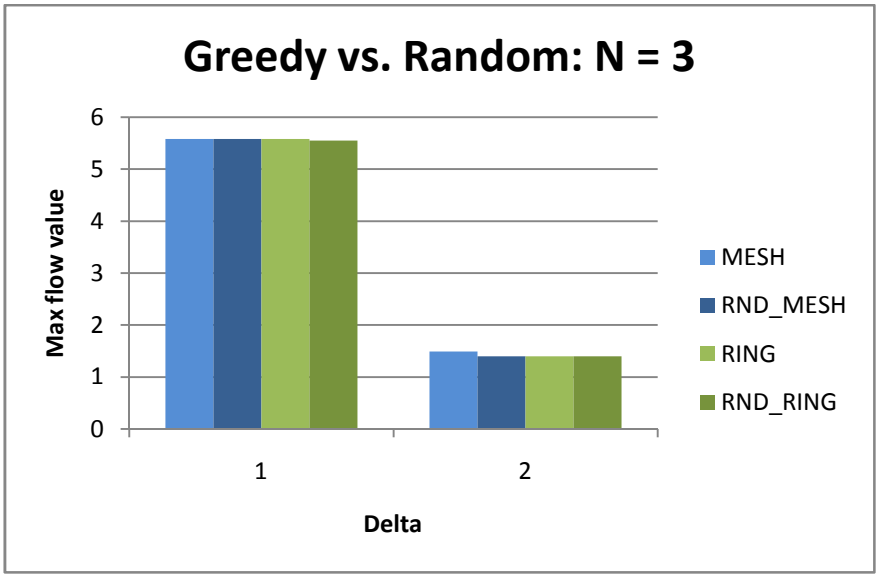
According to the results obtained (which are showed below), we notice that the **ring approach algorithm is usually preferable to the mesh one**. In fact, without consider the ring-case and the full-mesh one (Δ = 1 or N-1), although computational required time is higher the maximum flow value is lower. The reason is that the ring approach has higher probability to generate a topology with higher number of edges, often equal to the maximum number of edges a topology with N nodes can have (which is the product: N Δ).

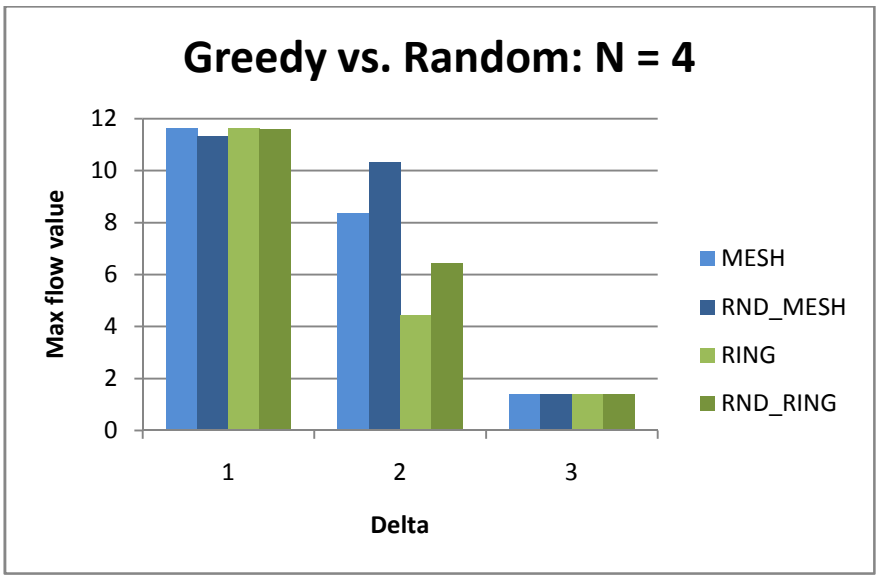
Another reason we can prefer the ring approach is that **the mesh algorithm can fail**: it hardly ever happens, but there are some traffic matrix for which the mesh greedy algorithm is not able to instantiate a feasible topology. In fact, at a certain point it happens that edges which should be removed are forced to remain in the graph, because their removal would disconnect it: the reason is, at that moment, all removable neighbor edges have been considered first (because of their lower associated flow value) and the algorithm has decided to remove them.

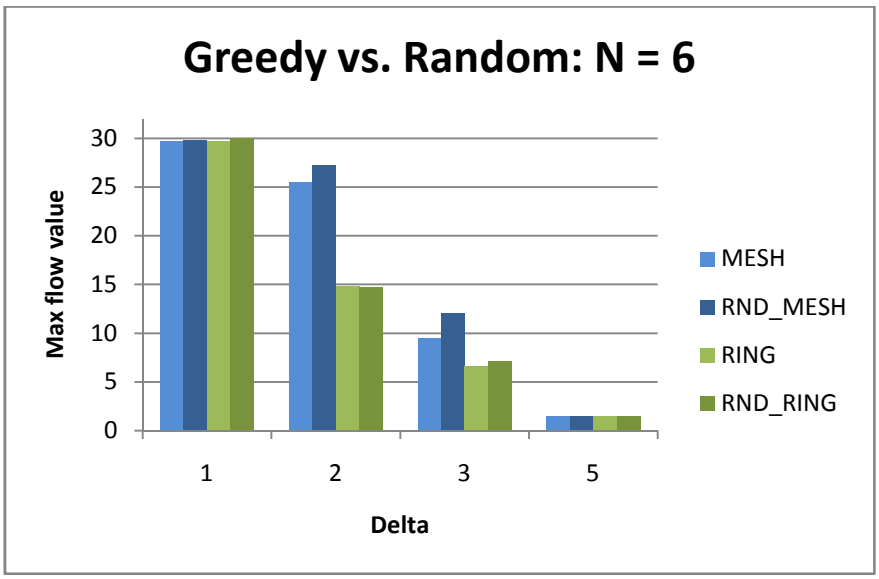
As we have said before, **the most complex task is not creating the topology, but routing the traffic** over it according to what the traffic matrix impose. In fact, the water filling function requires a lot of time in order to complete the computation: the reason is that we have to search, for each pair of node u-v of the graph not directly connected each other, all the simple paths between them.

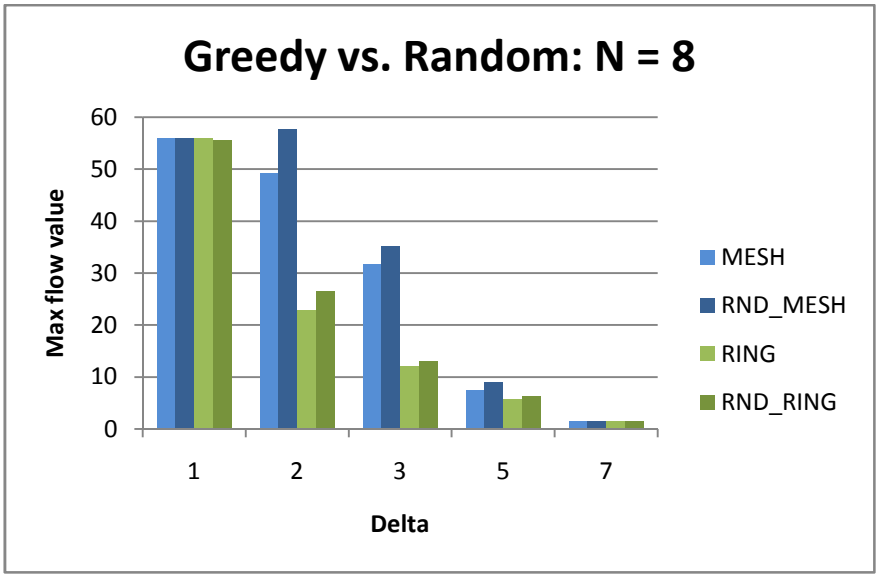
This is the reason why the search of possible paths is tried limiting the depth of the research to a value which is, by default, equal to six: otherwise, with higher values of N and Δ, the ***MemoryError*** **exception** is raised by the program.

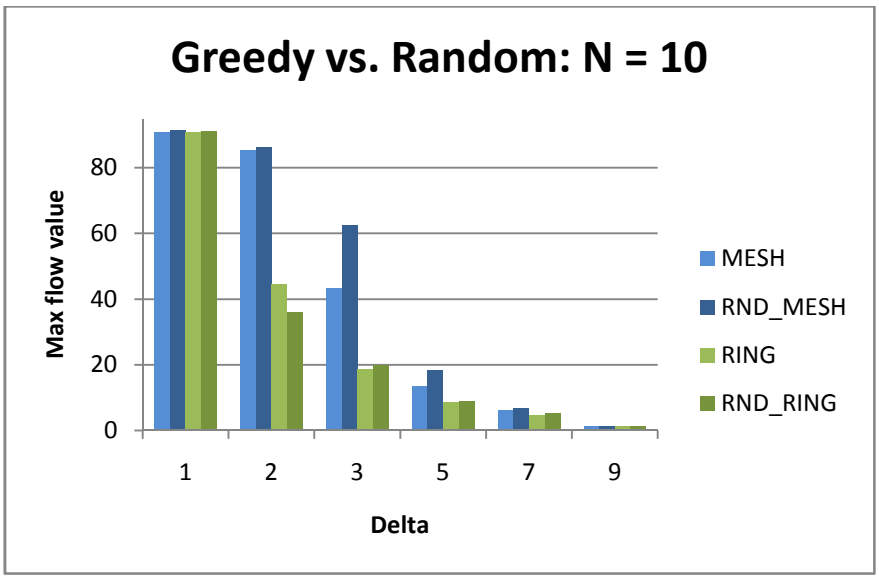
Only if no path is found, the depth value becomes greater and greater, until the maximum depth value N-1 is reached. In that case, if no path are found is because they do not exist. Once at least a path is found, the traffic is routed over them according to the water filling principle.

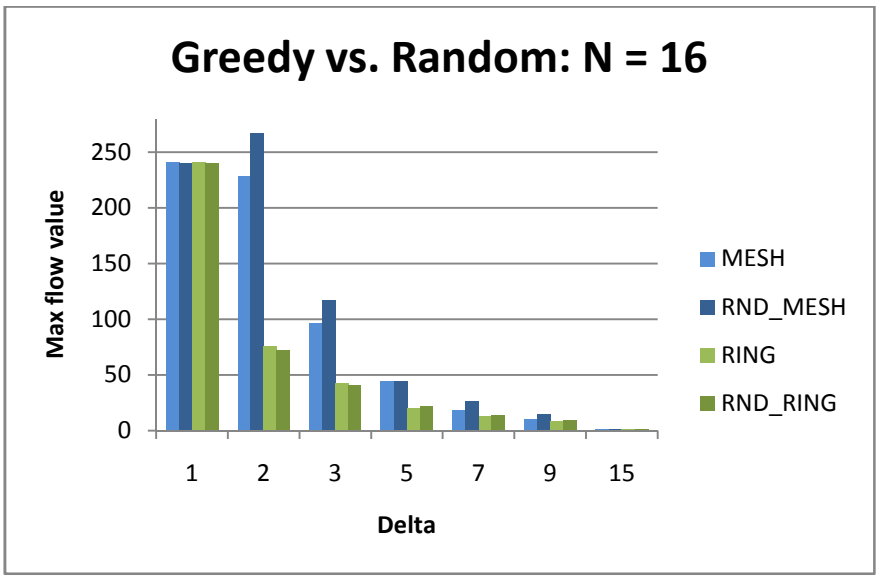

Greedy vs. Random: N = 3
Max flow value
0
1
2
3
4
5
6
1
2
Delta
MESH
RND_MESH
RING
RND_RING

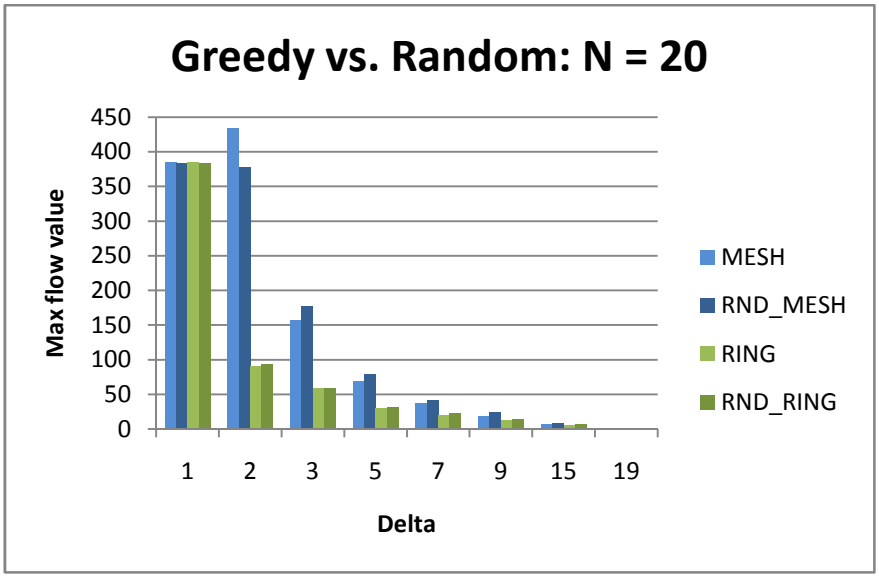

Greedy vs. Random: N = 4
Max flow value
0
2
4
6
8
10
12
1
2
3
Delta
MESH
RND_MESH
RING
RND_RING

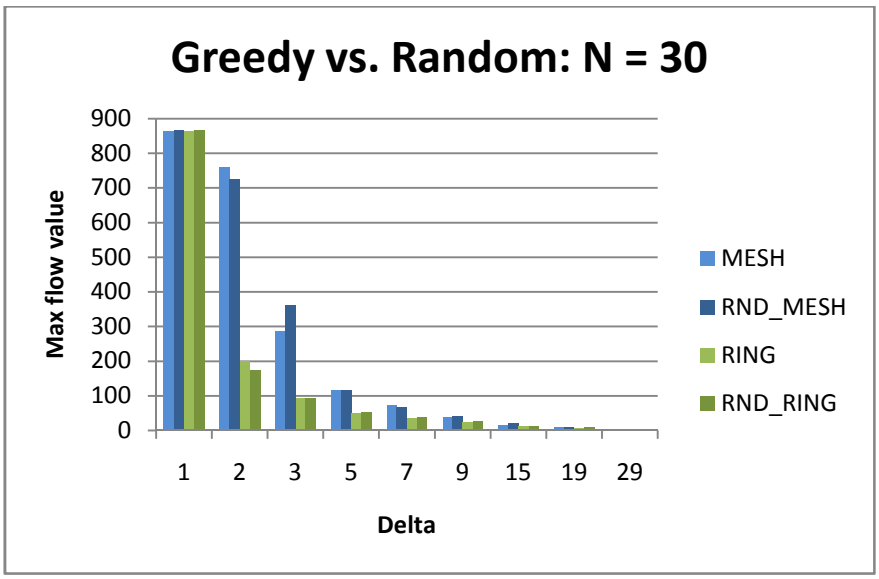

Greedy vs. Random: N = 6
Max flow value
0
5
10
15
20
25
30
1
2
3
5
Delta
MESH
RND_MESH
RING
RND_RING

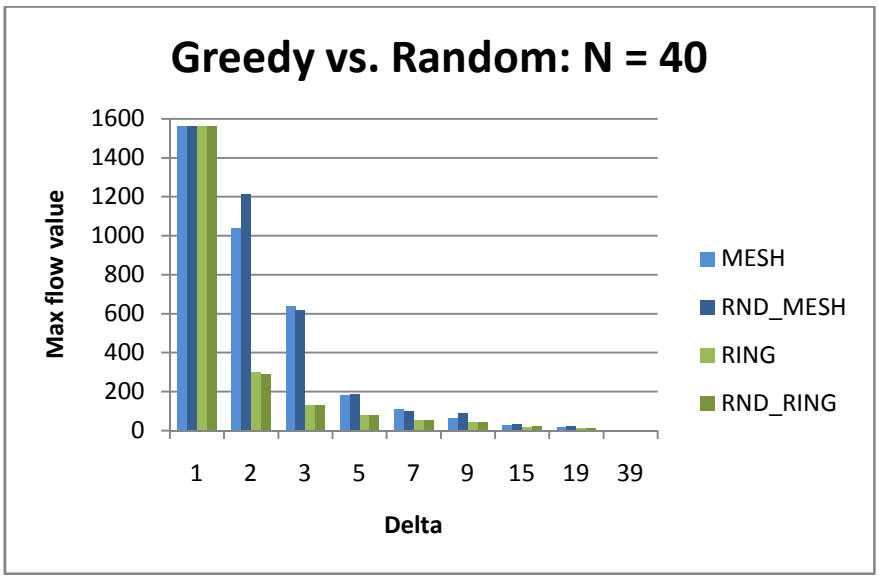

Greedy vs. Random: N = 8
Max flow value
60
50
40
30
20
10
0
1
2
3
5
7
Delta
MESH
RND_MESH
RING
RND_RING


Greedy vs. Random: N = 10
Max flow value
80
60
40
20
0
1
2
3
5
7
9
Delta
MESH
RND_MESH
RING
RND_RING


Greedy vs. Random: N = 16
Max flow value
250
200
150
100
50
0
1
2
3
5
7
9
15
Delta
MESH
RND_MESH
RING
RND_RING



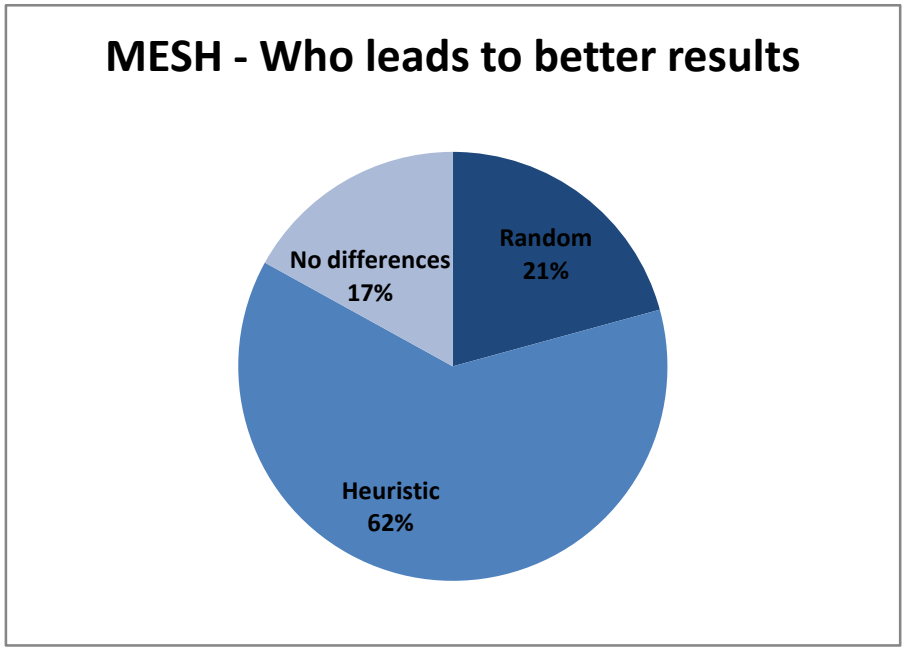

*Complete results can be found in Appendix A or in the attached files, into "LOG.zip" archive*

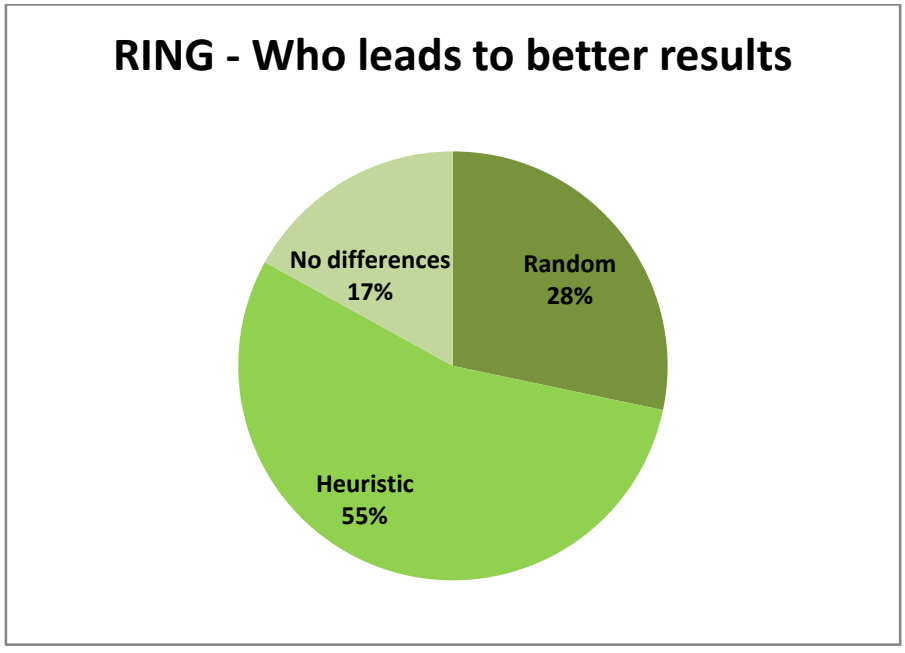
## Conclusions

As introduced before, we notice that the **ring approach** often gives us better results, in terms of max flow value in the topology. In the histograms, we can see how the traffic is better distributed through edges of the topology when Δ value increases.

Random approaches tend to lead us to better results when Δ is lower: the reason is that greedy algorithms have a fixed way to create the initial ring, ordered by name. Random algorithms can modify the initial disposal of the nodes, obtaining results which can be better or not (depending on the traffic matrix values).

In these graphs, we can see the comparisons between greedy and random approaches' results: clearly, we have no differences when the resulting topology is the full mesh.



## Ex 03

The exercise is the same as the previous one: but now, the traffic matrixes are generated uniformly using **two possible ranges**, decided according to a **probability** "p". In other words, some pairs of nodes require now to exchange traffic volumes which are ten times greater than the normal ones.

### Traffic classes

**High** traffic: U(5, 15)
**Low** traffic: U(0.5, 1.5)

### Tests explanation

In an LTD problem formulation, for every pair of nodes (the single traffic matrix value to consider) a **random number** between 0 and 1 is generated: if its value is lower than "p" = 0.1 (10% probability), this specific traffic matrix value is generated as uniform variable distributed in the high traffic class. Otherwise, the low class is chosen.

As in the previous exercise, every possible N/Δ combination is tested **four times** for each possible approach: in this way, maximum flow and other quality parameters, such as computational required time, are estimated as an average between the four measured. Clearly, different algorithms will use the same traffic matrix, while different attempts of the same algorithm will use different ones.

### Possible values

N = [3, 4, 6, 8, 10, 16, 20, 30, 40]
Δ = [1, 2, 3, 5, 7, 9, 15, 19, 25, 29, 35, 39]
p = 10 %

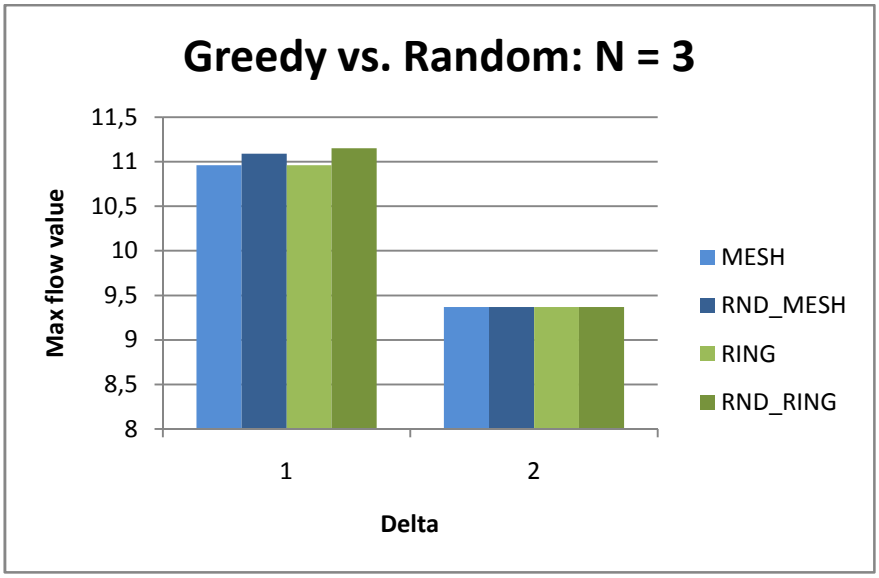
As before, we can notice that for each N value only Δs lower than it are considered. In fact, greater ones would produce the same result: a full mesh topology.

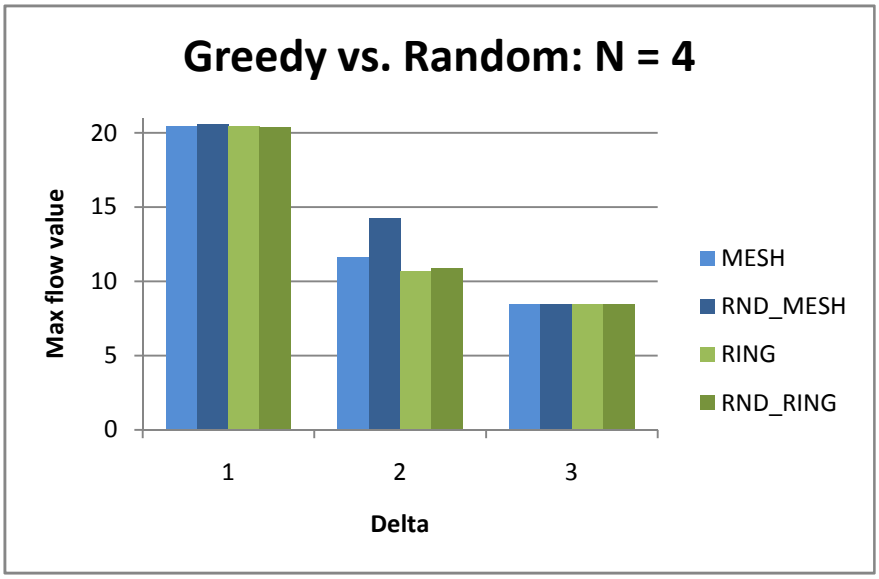
### What we have experienced

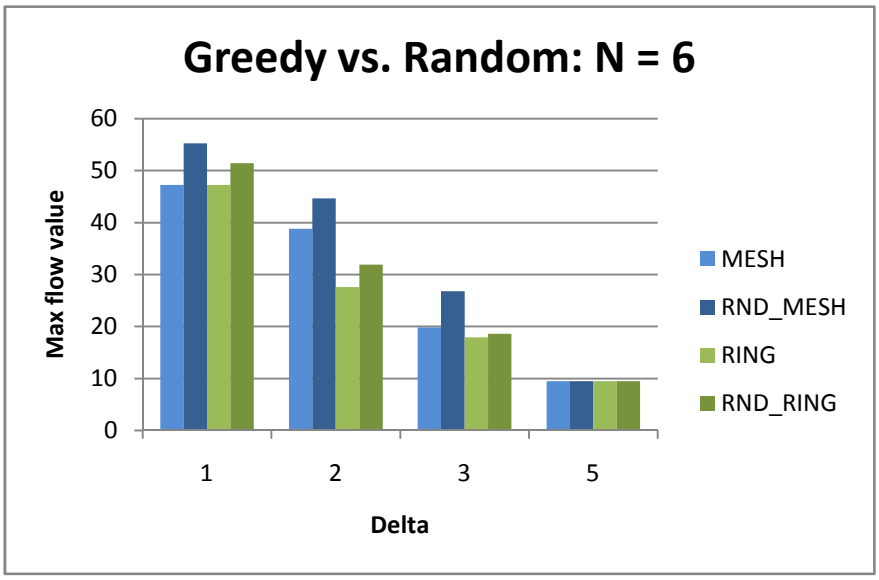
According to the results obtained (which are showed below), we notice that the ring approach algorithm is still preferable: anyway, the **effect of the initial disposal** of the nodes here is more evident.

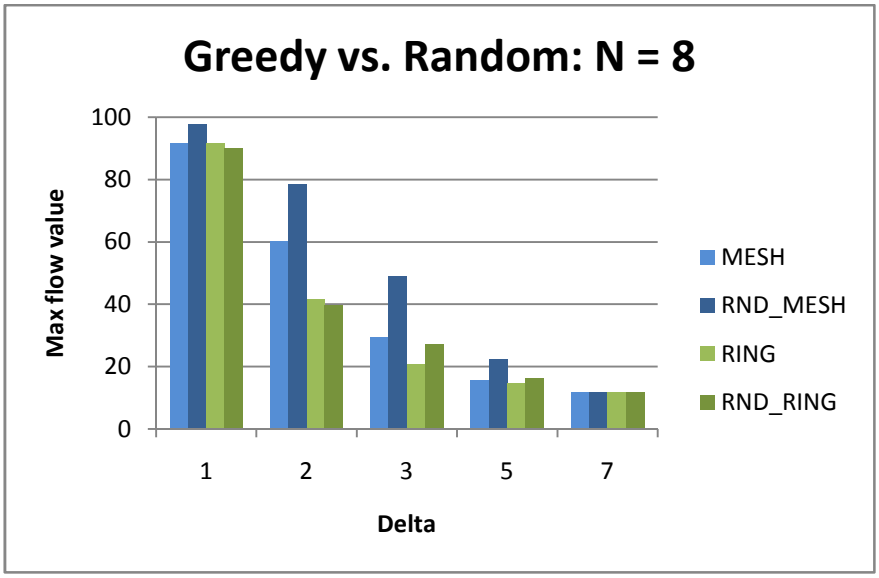
When Δ < 3 and N is high, the initial order of the nodes in the ring (the initial computed topology) usually determines better results in the random approach, where this order is casual. The fact that this effect is now more evident is a consequence of the presence of the pair of nodes who exchange greater amounts of traffic.

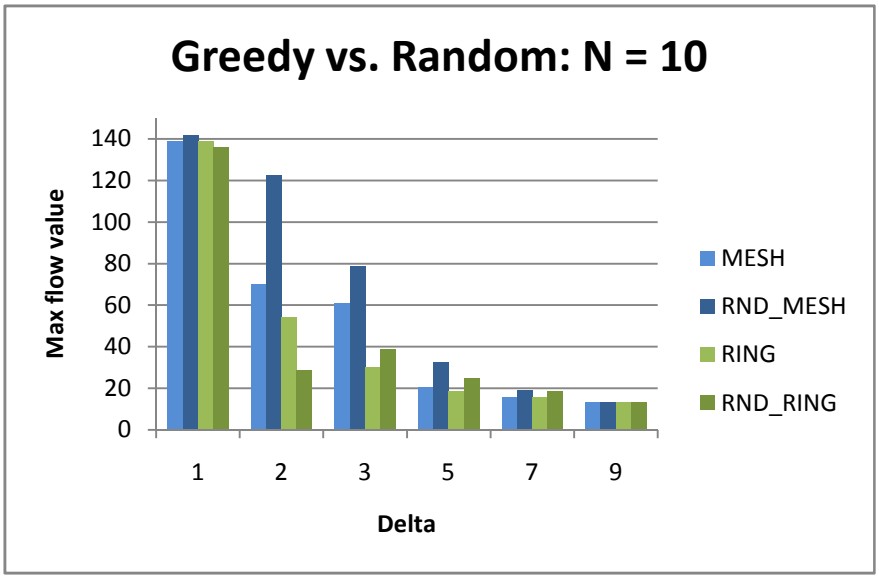
The fixed nodes sequence in the greedy ring algorithm (ordered by name) is clearly not always the best routing configuration: for that reason, in the average case a random one is able to guarantee lower max flow values.

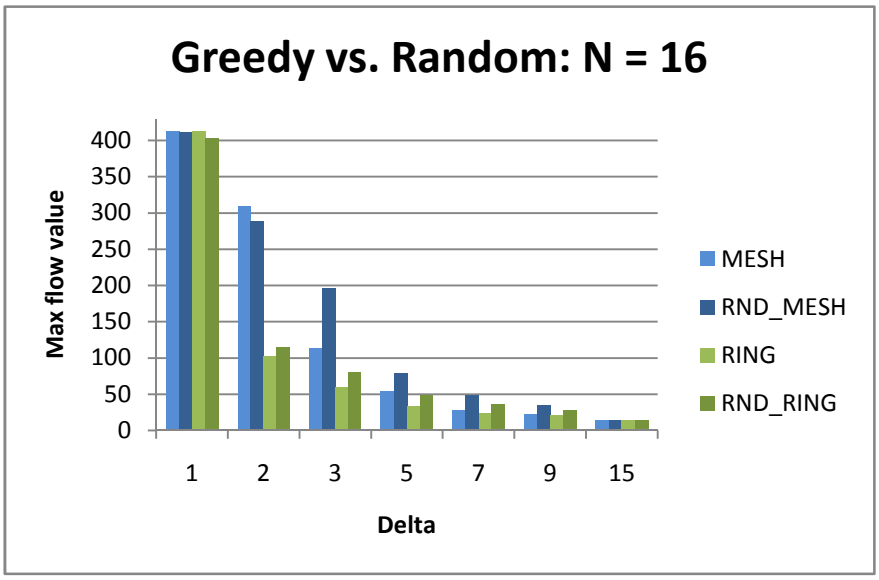

Greedy vs. Random: N = 3
Max flow value
11,5
11
10,5
10
9,5
9
8,5
8
1
2
Delta
MESH
RND_MESH
RING
RND_RING

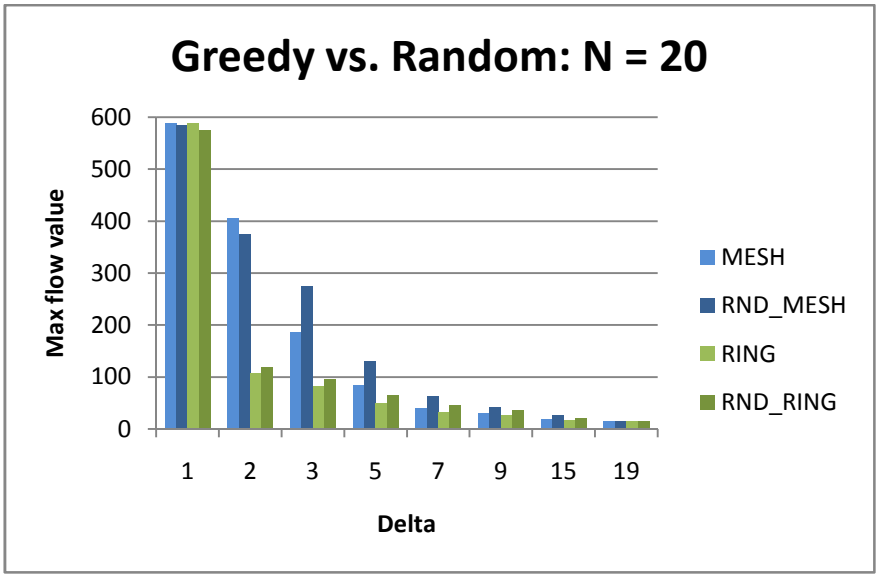

Greedy vs. Random: N = 4
Max flow value
20
15
10
5
0
1
2
3
Delta
MESH
RND_MESH
RING
RND_RING

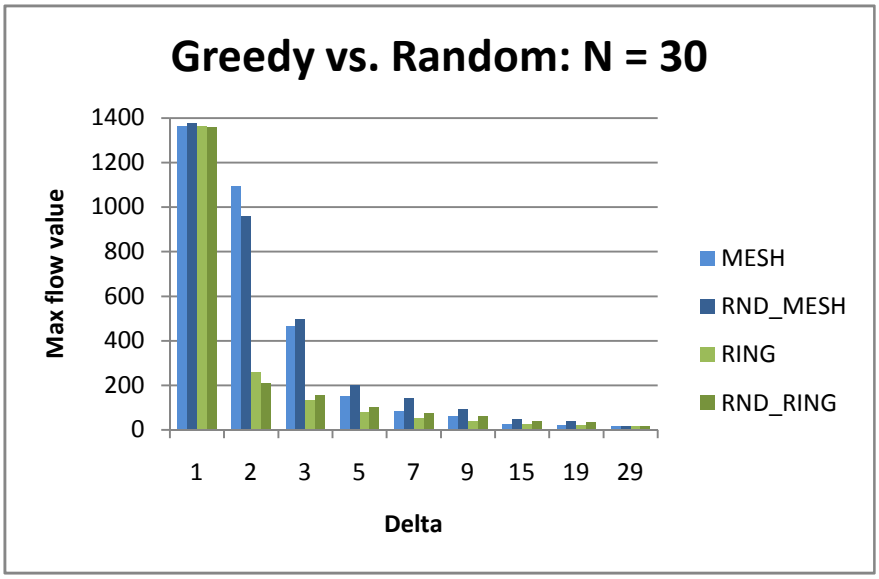

Greedy vs. Random: N = 6
Max flow value
60
50
40
30
20
10
0
1
2
3
5
Delta
MESH
RND_MESH
RING
RND_RING

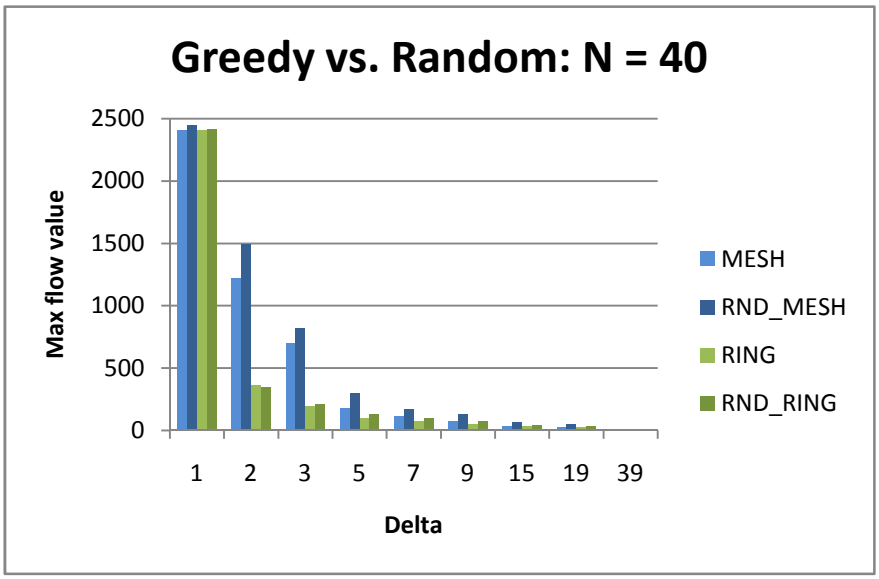

Greedy vs. Random: N = 8
Max flow value
100
80
60
40
20
0
1
2
3
5
7
Delta
MESH
RND_MESH
RING
RND_RING


Greedy vs. Random: N = 10
Max flow value
140
120
100
80
60
40
20
0
1
2
3
5
7
9
Delta
MESH
RND_MESH
RING
RND_RING


Greedy vs. Random: N = 16
Max flow value
400
350
300
250
200
150
100
50
0
1
2
3
5
7
9
15
Delta
MESH
RND_MESH
RING
RND_RING



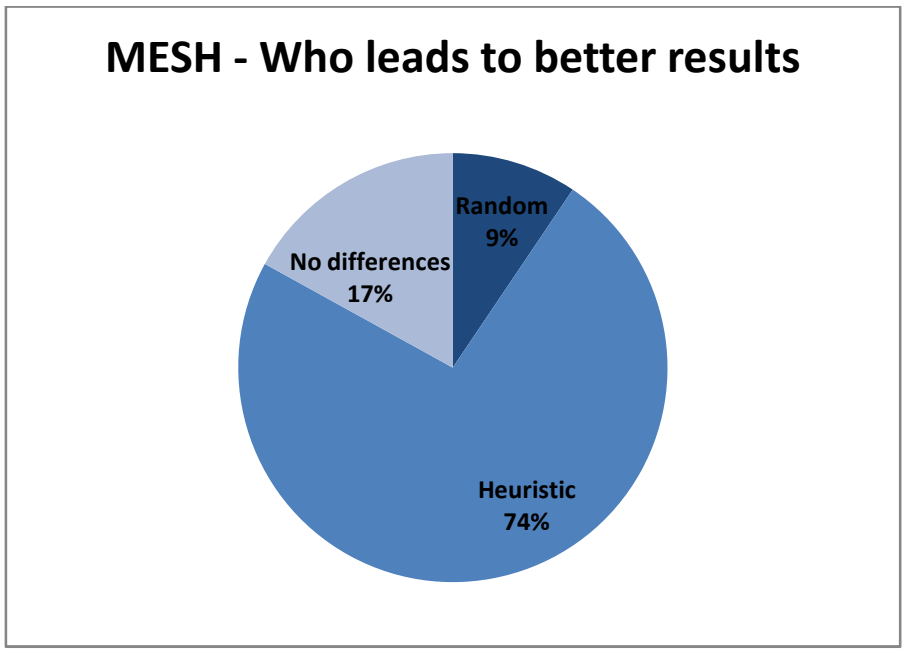

*Complete results can be found in Appendix B or in the attached files, into "LOG.zip" archive*

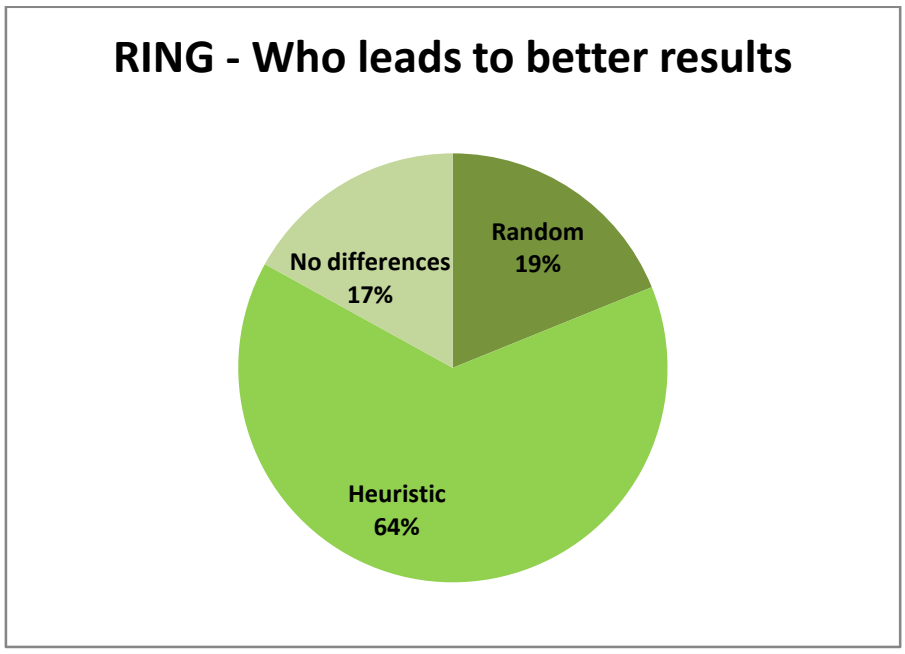
## Conclusions

As introduced above, we notice that the **ring approach** still often gives us better results, in terms of max flow value in the topology. In the histograms, we can see how the traffic is better distributed through edges of the topology when Δ value increases.

The presence of the high traffic class has highlighted that the initial node disposal in the ring greedy algorithm is not the optimal one: some pair of nodes are now exchanging much greater amount of traffic and they should be placed as near as possible. Instead, the algorithm sort and dispose them by name: which is still a good approach when we can add several edges into the topology (Δ> 3).

In these graphs, we can see the comparisons between greedy and random approaches' results: clearly, we have no differences when the resulting topology is the full mesh.



## Ex 04

Another possible approach to relax an LTD problem is to adopt, a priori, a well-defined regular topology on which route the traffic (according to the chosen routing strategy). Because of the topology's regularity, obtained result can be far from the optimal one if the traffic is not balanced (uniform).

In this exercise, we will solve the LTD problem on a **Manhattan topology**, in which nodes are placed according to its definition: every node has in/out degrees ("delta" constraint) equal to four. It is well-known that having a Manhattan topology leads to several benefits, in terms of shortest paths' length between pair of nodes and flow distribution over edges.

In particular, starting from the top-left corner, nodes will be disposed by ascending order of their names, as a rectangle: the variables "Nr" and "Nc" represents the number of nodes per row and per column. Clearly, we have that: $Nr \cdot Nc = N$

### Tests explanation

The number of nodes "N" change for every LTD problem instance: but now, because of the Manhattan topology structure, the constraint on the maximum number of receivers/transmitters per node **"Δ" is fixed** to four (and it clearly coincide with the actual in/out degree of the nodes).

For every N-nodes topology, **four different traffic matrixes** are generated in order to test the routing results: in this way, maximum flow and other quality parameters, such as computational required time, are estimated as an average between the four measured. Traffic matrixes values are uniformly distributed in the range [**0.5**, **1.5**], in order to have balanced traffic between nodes.

Finally, the maximum depth specified is the one that, also in the worst case, will produce at least a path for each pair of nodes: it is calculated as:

- depth = Nr – 1, if Nr = Nc
- depth = Nr/2 + Nc/2, otherwise

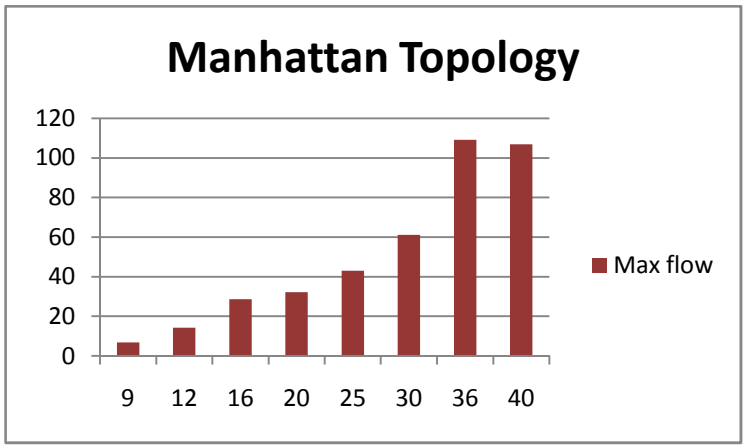
### Possible values

N = [9, 12, 16, 20, 25, 30, 36, 40]
Nr = [3, 4, 4, 5, 5, 6, 6, 8]
Nc = [3, 3, 4, 4, 5, 5, 6, 5]

### Results

Here, are reported the obtained results: as you can see, both max flow values and computational required time are lower than in the previous exercises. This is the consequence of the Manhattan topology implementation.

| N | Nr | Nc | Max flow | Time |
|---|---|---|---|---|
| **9** | 3 | 3 | 6,85 | 0,01 |
| **12** | 4 | 3 | 14,16 | 0,02 |
| **16** | 4 | 4 | 28,61 | 0,17 |
| **20** | 5 | 4 | 32,12 | 0,06 |
| **25** | 5 | 5 | 42,98 | 0,07 |
| **30** | 6 | 5 | 61,17 | 0,27 |
| **36** | 6 | 6 | 109,11 | 7,26 |
| **40** | 8 | 5 | 106,86 | 1,22 |



## Ex 05

We now repeat the previous exercise, trying to optimize the obtained results **swapping nodes' positions**. The idea is to place in adjacent positions of the topology nodes who need to exchange largest amount of traffic, in order to have a direct communication without involving (and saturating) other channels.

### Tests explanation

Experiments are the same as before, done following same criterions but in two ways: the first approach ("A") is identical to the one adopted in the previous exercise, while the second one ("B") is the attempt to optimize final results. What we do in the "B" phase, calling the function "***LTD_manhattan_smart***", is to swap nodes' position of the "A" Manhattan topology.

Speaking about the Python code implementation, the optimized solution of "B" approach is retrieved in three phases. In the first one, a copy of the "A" Manhattan topology is generated and nodes are labeled as "No name": in this way, placing a node in the resulting topology means to label a "no name" node. The second step is nodes placement (renaming): considering first pairs who exchange greater amount of traffic, nodes are progressively placed into the topology. When all the nodes are marked as placed, the phase three consists in the computation of the final Manhattan topology, where nodes are disposed according to the names of the ones attached into the "A" topology.

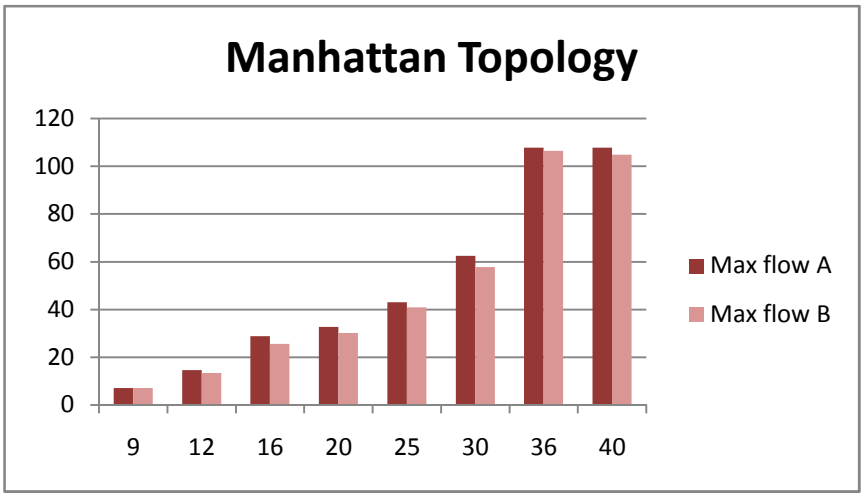
### Possible values

N = [9, 12, 16, 20, 25, 30, 36, 40]
Nr = [3, 4, 4, 5, 5, 6, 6, 8]
Nc = [3, 3, 4, 4, 5, 5, 6, 5]

### Results

We report now the results we have obtained: as you can see, apart from the first case (in which nodes are fewer and closed each other), the "B" approach leads to better outcomes.

| N | Nr | Nc | Max flow A | Max flow B | Time A | Time B |
|---|---|---|---|---|---|---|
| **9** | 3 | 3 | 7,05 | 7,15 | 0,01 | < 0.01 |
| **12** | 4 | 3 | 14,68 | 13,42 | 0,01 | 0,01 |
| **16** | 4 | 4 | 28,89 | 25,66 | 0,14 | 0,14 |
| **20** | 5 | 4 | 32,75 | 30,13 | 0,06 | 0,05 |
| **25** | 5 | 5 | 43,04 | 40,84 | 0,07 | 0,08 |
| **30** | 6 | 5 | 62,53 | 57,77 | 0,23 | 0,25 |
| **36** | 6 | 6 | 107,8 | 106,46 | 6,45 | 6,52 |
| **40** | 8 | 5 | 107,83 | 104,91 | 1,08 | 1,17 |



## Appendix A

These are the complete results obtained in exercises 01 and 02: where the "time" attribute is zero, it means that the computational time required to solve the LTD problem instance was lower than 0.01 seconds.

| N | Delta | MESH | | RND_MESH | | RING | | RND_RING | | BEST SOL. | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Max flow | Time | Max flow | Time | Max flow | Time | Max flow | Time | Mesh | Ring |
| **3** | **1** | 5,58 | 0 | 5,58 | 0 | 5,58 | 0 | 5,55 | 0 | 0,5 | 0 |
| **3** | **2** | 1,49 | 0,01 | 1,4 | 0 | 1,4 | 0,02 | 1,4 | 0,01 | 0 | 0,5 |
| **4** | **1** | 11,63 | 0 | 11,33 | 0 | 11,63 | 0 | 11,6 | 0 | 0 | 0 |
| **4** | **2** | 8,38 | 0,01 | 10,34 | 0 | 4,46 | 0 | 6,44 | 0 | 1 | 1 |
| **4** | **3** | 1,41 | 0 | 1,41 | 0 | 1,41 | 0,01 | 1,41 | 0 | 0,5 | 0,5 |
| **6** | **1** | 29,67 | 0 | 29,79 | 0 | 29,67 | 0 | 30,04 | 0 | 1 | 1 |
| **6** | **2** | 25,49 | 0 | 27,18 | 0,01 | 14,77 | 0,01 | 14,73 | 0 | 1 | 0 |
| **6** | **3** | 9,53 | 0 | 12,06 | 0 | 6,6 | 0,01 | 7,16 | 0,01 | 1 | 1 |
| **6** | **5** | 1,48 | 0 | 1,48 | 0 | 1,48 | 0,01 | 1,48 | 0 | 0,5 | 0,5 |
| **8** | **1** | 55,92 | 0,01 | 55,77 | 0,01 | 55,92 | 0 | 55,52 | 0,01 | 0 | 0 |
| **8** | **2** | 49,05 | 0 | 57,63 | 0,01 | 22,73 | 0,01 | 26,41 | 0 | 1 | 1 |
| **8** | **3** | 31,73 | 0,01 | 35,18 | 0,01 | 12,08 | 0,01 | 13,04 | 0,01 | 1 | 1 |
| **8** | **5** | 7,29 | 0,02 | 8,95 | 0,03 | 5,74 | 0,03 | 6,21 | 0,03 | 1 | 1 |
| **8** | **7** | 1,48 | 0,01 | 1,48 | 0,01 | 1,48 | 0,01 | 1,48 | 0,01 | 0,5 | 0,5 |
| **10** | **1** | 90,8 | 0,01 | 91,57 | 0,04 | 90,8 | 0,06 | 91,05 | 0,03 | 1 | 1 |
| **10** | **2** | 85,39 | 0,02 | 86,47 | 0,02 | 44,74 | 0,05 | 36,14 | 0,03 | 1 | 0 |
| **10** | **3** | 43,26 | 0,01 | 62,54 | 0,03 | 18,81 | 0,06 | 19,85 | 0,03 | 1 | 1 |
| **10** | **5** | 13,47 | 0,06 | 18,27 | 0,08 | 8,72 | 0,24 | 8,94 | 0,19 | 1 | 1 |
| **10** | **7** | 6,1 | 0,27 | 6,91 | 0,32 | 4,72 | 0,38 | 5,4 | 0,41 | 1 | 1 |
| **10** | **9** | 1,49 | 0 | 1,49 | 0,02 | 1,49 | 0,02 | 1,49 | 0,06 | 0,5 | 0,5 |
| **16** | **1** | 241,33 | 0,02 | 240,06 | 0,02 | 241,33 | 0,04 | 240,06 | 0,03 | 0 | 0 |
| **16** | **2** | 228,55 | 0,03 | 266,93 | 0,02 | 76,19 | 0,08 | 71,77 | 0,01 | 1 | 0 |
| **16** | **3** | 96,61 | 0,04 | 116,94 | 0,04 | 42,26 | 0,11 | 40,74 | 0,08 | 1 | 0 |
| **16** | **5** | 44,42 | 0,29 | 44,62 | 0,27 | 20,5 | 0,68 | 22,42 | 0,68 | 1 | 1 |
| **16** | **7** | 18,4 | 1,43 | 26,82 | 1,39 | 12,5 | 2,96 | 13,58 | 2,98 | 1 | 1 |
| **16** | **9** | 10,57 | 5,45 | 14,39 | 5,41 | 8,19 | 8,45 | 9,49 | 8,59 | 1 | 1 |
| **16** | **15** | 1,49 | 0,01 | 1,49 | 0,01 | 1,49 | 0,02 | 1,49 | 0,01 | 0,5 | 0,5 |
| **20** | **1** | 384,49 | 0,02 | 383,53 | 0,01 | 384,49 | 0,02 | 382,73 | 0,01 | 0 | 0 |
| **20** | **2** | 433,19 | 0,04 | 377,9 | 0,03 | 90,07 | 0,05 | 93,84 | 0,04 | 0 | 1 |
| **20** | **3** | 156,54 | 0,12 | 177,23 | 0,09 | 58,67 | 0,16 | 58,06 | 0,15 | 1 | 0 |
| **20** | **5** | 68,09 | 0,35 | 77,89 | 0,3 | 29,82 | 1,29 | 31,49 | 1,32 | 1 | 1 |
| **20** | **7** | 36,59 | 2,69 | 41,31 | 2,68 | 18,89 | 6,65 | 21,48 | 6,47 | 1 | 1 |
| **20** | **9** | 18,6 | 10,7 | 23,25 | 10,97 | 12,98 | 20,52 | 14,03 | 20,91 | 1 | 1 |
| **20** | **15** | 5,87 | 128,19 | 7,28 | 125,99 | 5 | 149,69 | 5,96 | 148,73 | 1 | 1 |
| **20** | **19** | 1,5 | 0,01 | 1,5 | 0,02 | 1,5 | 0,03 | 1,5 | 0,02 | 0,5 | 0,5 |
| **30** | **1** | 864,73 | 0,05 | 866,49 | 0,05 | 864,73 | 0,05 | 866,48 | 0,04 | 1 | 1 |
| **30** | **2** | 759,79 | 0,18 | 725,33 | 0,19 | 199,97 | 0,35 | 174,54 | 0,26 | 0 | 0 |
| **30** | **3** | 287,92 | 0,76 | 362,17 | 1,12 | 94,26 | 0,42 | 94,17 | 0,39 | 1 | 0 |
| **30** | **5** | 117,09 | 1,2 | 117,13 | 1,14 | 51,21 | 3,81 | 52,68 | 3,89 | 1 | 1 |
| **30** | **7** | 74,42 | 6,86 | 66,75 | 7,02 | 35,74 | 20,2 | 37,35 | 20,22 | 0 | 1 |
| **30** | **9** | 37,56 | 31,22 | 42,3 | 30,44 | 25,01 | 68,7 | 27,73 | 69,46 | 1 | 1 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **30** | **15** | 14,57 | 504,06 | 21,45 | 488,4 | 11,52 | 820,42 | 13,16 | 819,5 | 1 | 1 |
| **30** | **19** | 8,78 | 1692 | 11,05 | 1737,8 | 7,37 | 2267,3 | 8,86 | 2259,6 | 1 | 1 |
| **30** | **29** | 1,5 | 0,03 | 1,5 | 0,06 | 1,5 | 0,06 | 1,5 | 0,07 | 0,5 | 0,5 |
| **40** | **1** | 1560,36 | 0,14 | 1559,36 | 0,1 | 1560,36 | 0,15 | 1562,71 | 0,1 | 0 | 1 |
| **40** | **2** | 1038,14 | 0,78 | 1212,54 | 0,94 | 301,03 | 0,95 | 286,56 | 0,94 | 1 | 0 |
| **40** | **3** | 638,07 | 1,53 | 617,98 | 3,86 | 133,54 | 0,89 | 133,53 | 0,87 | 0 | 0 |
| **40** | **5** | 179,81 | 3,23 | 187,33 | 2,64 | 77,3 | 8,3 | 78,79 | 8,31 | 1 | 1 |
| **40** | **7** | 108,16 | 16,59 | 101,88 | 15,67 | 52,18 | 43,49 | 53,61 | 43,17 | 0 | 1 |
| **40** | **9** | 64,14 | 71,77 | 90,26 | 67,89 | 40,99 | 151,98 | 40,45 | 151,43 | 1 | 0 |
| **40** | **15** | 28,32 | 1125,3 | 31,78 | 1112,2 | 18,44 | 2079,8 | 20,6 | 2077,2 | 1 | 1 |
| **40** | **19** | 16,45 | 4181,7 | 20,9 | 4094,3 | 13,06 | 6177,8 | 14,29 | 6026 | 1 | 1 |
| **40** | **39** | 1,5 | 0,01 | 1,5 | 0,2 | 1,5 | 0,08 | 1,5 | 0,21 | 0,5 | 0,5 |
| | | | | | | | | | wins | **33** | **29** |
| | | | | | | | | | draws | **9** | **9** |
| | | | | | | | | | fails | **11** | **15** |
| | | | | | | | | | tests | 53 | 53 |

## Appendix B

These are the complete results obtained in exercise 03: where the "time" attribute is zero, it means that the computational time required to solve the LTD problem instance was lower than 0.01 seconds.

| N | Delta | MESH | | RND_MESH | | RING | | RND_RING | | BEST SOL. | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Max flow | Time | Max flow | Time | Max flow | Time | Max flow | Time | Mesh | Ring |
| **3** | **1** | 10,96 | 0,01 | 11,09 | 0 | 10,96 | 0 | 11,15 | 0,01 | 1 | 1 |
| **3** | **2** | 9,37 | 0 | 9,37 | 0 | 9,37 | 0 | 9,37 | 0 | 0,5 | 0,5 |
| **4** | **1** | 20,42 | 0 | 20,55 | 0,01 | 20,42 | 0 | 20,39 | 0 | 1 | 0 |
| **4** | **2** | 11,59 | 0,01 | 14,24 | 0 | 10,69 | 0 | 10,91 | 0 | 1 | 1 |
| **4** | **3** | 8,47 | 0,01 | 8,47 | 0 | 8,47 | 0 | 8,47 | 0,01 | 0,5 | 0,5 |
| **6** | **1** | 47,24 | 0 | 55,23 | 0 | 47,24 | 0,01 | 51,43 | 0 | 1 | 1 |
| **6** | **2** | 38,82 | 0 | 44,65 | 1,95 | 27,59 | 0 | 31,93 | 0 | 1 | 1 |
| **6** | **3** | 19,84 | 0 | 26,8 | 0 | 17,9 | 0 | 18,61 | 0 | 1 | 1 |
| **6** | **5** | 9,51 | 0,01 | 9,51 | 0,01 | 9,51 | 0 | 9,51 | 0 | 0,5 | 0,5 |
| **8** | **1** | 91,49 | 0 | 97,46 | 0 | 91,49 | 0 | 89,88 | 0 | 1 | 0 |
| **8** | **2** | 60,12 | 0,01 | 78,32 | 0 | 41,68 | 0,01 | 39,47 | 0 | 1 | 0 |
| **8** | **3** | 29,44 | 0,01 | 48,91 | 0,01 | 20,86 | 0,01 | 27,08 | 0,01 | 1 | 1 |
| **8** | **5** | 15,43 | 0,03 | 22,17 | 0,02 | 14,49 | 0,03 | 16,07 | 0,03 | 1 | 1 |
| **8** | **7** | 11,58 | 0 | 11,58 | 0,01 | 11,58 | 0,01 | 11,58 | 0 | 0,5 | 0,5 |
| **10** | **1** | 138,81 | 0 | 141,5 | 0,01 | 138,81 | 0,01 | 135,94 | 0,01 | 1 | 0 |
| **10** | **2** | 70,14 | 0,01 | 122,57 | 0,01 | 54,34 | 0,01 | 28,84 | 0,02 | 1 | 0 |
| **10** | **3** | 60,77 | 0,02 | 78,57 | 0,01 | 30,15 | 0,02 | 38,61 | 0,01 | 1 | 1 |
| **10** | **5** | 20,27 | 0,05 | 32,43 | 0,05 | 18,56 | 0,13 | 24,54 | 0,12 | 1 | 1 |
| **10** | **7** | 15,69 | 0,22 | 18,7 | 0,24 | 15,5 | 0,29 | 18,46 | 0,27 | 1 | 1 |
| **10** | **9** | 13,35 | 0,01 | 13,35 | 0 | 13,35 | 0 | 13,35 | 0,01 | 0,5 | 0,5 |
| **16** | **1** | 413,1 | 0,01 | 411,61 | 0,01 | 413,1 | 0,03 | 403,49 | 0,01 | 0 | 0 |
| **16** | **2** | 309,28 | 0,05 | 288,83 | 0,01 | 103,03 | 0,03 | 114,94 | 0,03 | 0 | 1 |
| **16** | **3** | 113,64 | 0,05 | 196,62 | 0,05 | 60,5 | 0,1 | 80,55 | 0,08 | 1 | 1 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **16** | **5** | 54,44 | 0,29 | 79,39 | 0,29 | 33,7 | 0,69 | 49,58 | 0,66 | 1 | 1 |
| **16** | **7** | 28,49 | 1,68 | 48,82 | 1,61 | 23,98 | 3,3 | 37,14 | 3,12 | 1 | 1 |
| **16** | **9** | 22,12 | 5,57 | 35,51 | 5,42 | 21,32 | 8,48 | 28,37 | 8,65 | 1 | 1 |
| **16** | **15** | 14,49 | 0,02 | 14,49 | 0,01 | 14,49 | 0,03 | 14,49 | 0,01 | 0,5 | 0,5 |
| **20** | **1** | 586,78 | 0,02 | 583,4 | 0,02 | 586,78 | 0,03 | 574,4 | 0,02 | 0 | 0 |
| **20** | **2** | 404,94 | 0,08 | 374,93 | 0,03 | 107,17 | 0,06 | 117,36 | 0,04 | 0 | 1 |
| **20** | **3** | 186,31 | 0,15 | 273,7 | 0,11 | 82,21 | 0,17 | 95,55 | 0,16 | 1 | 1 |
| **20** | **5** | 83,15 | 0,45 | 130,46 | 0,42 | 48,43 | 1,31 | 64,56 | 1,39 | 1 | 1 |
| **20** | **7** | 39,35 | 3,17 | 63,5 | 2,98 | 32,37 | 6,76 | 45,53 | 6,65 | 1 | 1 |
| **20** | **9** | 30,37 | 10,09 | 41,79 | 10,09 | 25,88 | 20,4 | 35,95 | 20,89 | 1 | 1 |
| **20** | **15** | 18,07 | 131,29 | 25,81 | 129,7 | 17,26 | 153,53 | 19,6 | 152,29 | 1 | 1 |
| **20** | **19** | 14,85 | 0,03 | 14,85 | 0,02 | 14,85 | 0,03 | 14,85 | 0,02 | 0,5 | 0,5 |
| **30** | **1** | 1362,67 | 0,05 | 1375,22 | 0,04 | 1362,67 | 0,05 | 1358,13 | 0,05 | 1 | 0 |
| **30** | **2** | 1095,1 | 0,19 | 959,29 | 0,15 | 257,04 | 0,43 | 208,6 | 0,29 | 0 | 0 |
| **30** | **3** | 462,68 | 0,91 | 493,41 | 0,69 | 133,42 | 0,45 | 155,59 | 0,43 | 1 | 1 |
| **30** | **5** | 149,69 | 1,28 | 198,28 | 1,19 | 77,35 | 3,92 | 99,7 | 3,91 | 1 | 1 |
| **30** | **7** | 83,42 | 6,87 | 141,99 | 6,89 | 52,66 | 20 | 71,06 | 20,49 | 1 | 1 |
| **30** | **9** | 58,11 | 34,04 | 89,89 | 32,79 | 40,38 | 70,51 | 58,48 | 70,87 | 1 | 1 |
| **30** | **15** | 25,4 | 520,55 | 47,44 | 513,98 | 23,47 | 856,31 | 35,02 | 863,78 | 1 | 1 |
| **30** | **19** | 20,12 | 1685,7 | 39,12 | 1661,7 | 19,34 | 2241,5 | 31,25 | 2240,8 | 1 | 1 |
| **30** | **29** | 14,92 | 0,03 | 14,92 | 0,07 | 14,92 | 0,06 | 14,92 | 0,08 | 0,5 | 0,5 |
| **40** | **1** | 2406,25 | 0,11 | 2451,59 | 0,08 | 2406,25 | 0,11 | 2415,57 | 0,09 | 1 | 1 |
| **40** | **2** | 1222,44 | 0,55 | 1492,88 | 0,81 | 364,24 | 0,8 | 349,08 | 0,82 | 1 | 0 |
| **40** | **3** | 697,92 | 3,53 | 821,01 | 2,43 | 197,98 | 0,82 | 216,12 | 0,81 | 1 | 1 |
| **40** | **5** | 176,35 | 3,83 | 297,55 | 2,33 | 102,93 | 7,65 | 131,13 | 7,71 | 1 | 1 |
| **40** | **7** | 118,81 | 14,81 | 173,44 | 13,47 | 73,28 | 40,43 | 98,64 | 40,55 | 1 | 1 |
| **40** | **9** | 75,15 | 58,86 | 130,97 | 57,34 | 53,7 | 138,2 | 73,69 | 137,78 | 1 | 1 |
| **40** | **15** | 38,75 | 1072,7 | 69,78 | 1036,8 | 32,93 | 1913,7 | 46,74 | 1916,2 | 1 | 1 |
| **40** | **19** | 29,21 | 3876,9 | 52,51 | 3789,5 | 26 | 5889,4 | 38,34 | 5929,6 | 1 | 1 |
| **40** | **39** | 14,93 | 0,04 | 14,93 | 0,17 | 14,93 | 0,11 | 14,93 | 0,18 | 0,5 | 0,5 |
| | | | | | | | | | wins | **39** | **34** |
| | | | | | | | | | draws | **9** | **9** |
| | | | | | | | | | fails | **5** | **10** |
| | | | | | | | | | tests | 53 | 53 |