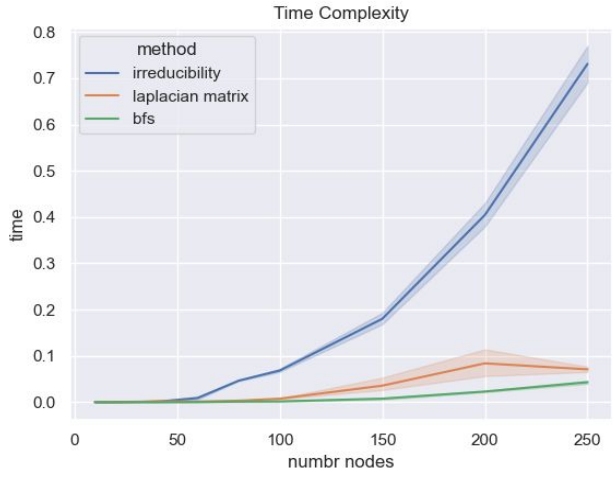
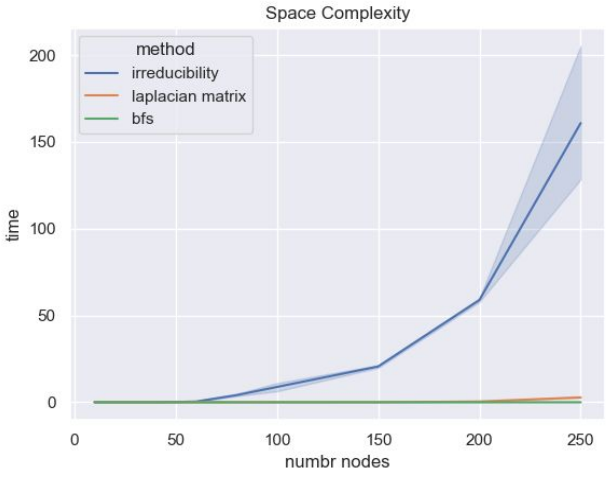
DATA CENTER CHALLENGE 1

Group Name: Iverson

Group members	Matricula
Grieco Nicola	2081607
Grimaldi Enrico	1884443
Vigneri Davide	2058036

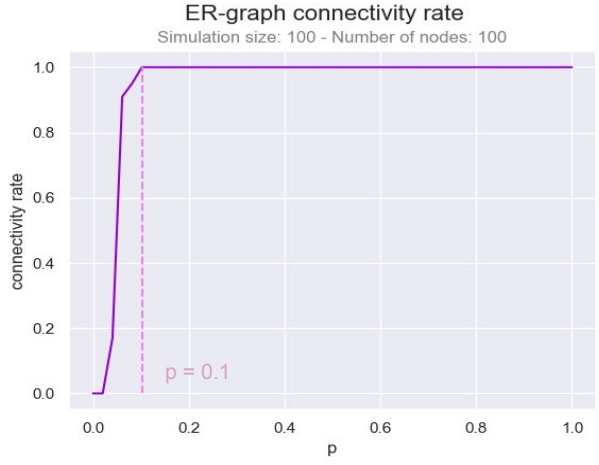
COMPLEXITY vs NUMBER OF NODES

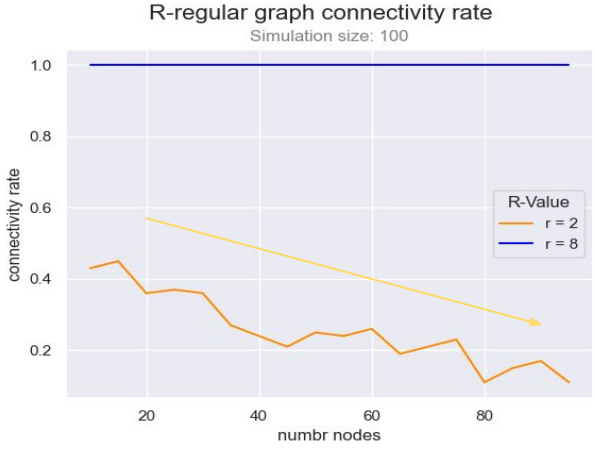




The two figures describe respectively time and space complexity of three different algorithms: *irreducibility* of the associated adjacency matrix, study of the *laplacian matrix*, and *bfs*.
We see that in terms of both time and space complexity, the *irreducibility* algorithm is the clear loser among the three methods. In fact, the *irreducibility* method is efficient only for small graphs (nodes < 50) and can be computationally expensive for larger graphs (since it needs to work with a large adjacency matrix).
Although the *laplacian matrix* and *bfs* algorithms scale in a comparable manner until 100 nodes, the *bfs* is consistently more efficient (with classical queue implementation) than its competitor for larger graphs. In the the *laplacian matrix*, we need to verify the positive definiteness of a square matrix, a job that becomes more and more computationally involved with an increasing number of nodes, since for each node we have to control over a matrix with both one additional row and column. In this example the *bfs* becomes more efficient since, although the number of nodes (V) grows, we keep nearly constant the number of edges (E) and this search algorithm has complexity O(|V| |E|).

CONNECTIVITY PROBABILITY DISTRIBUTION FUNCTION





We can notice that the connectivity rate gets pretty quickly to 1 (when p = 0.1). This rapid increase in connectivity rate is due to the fact that as more edges are more probable to be added to the graph, the number of connected components decreases, hence we have a higher probability to have a connected graph.

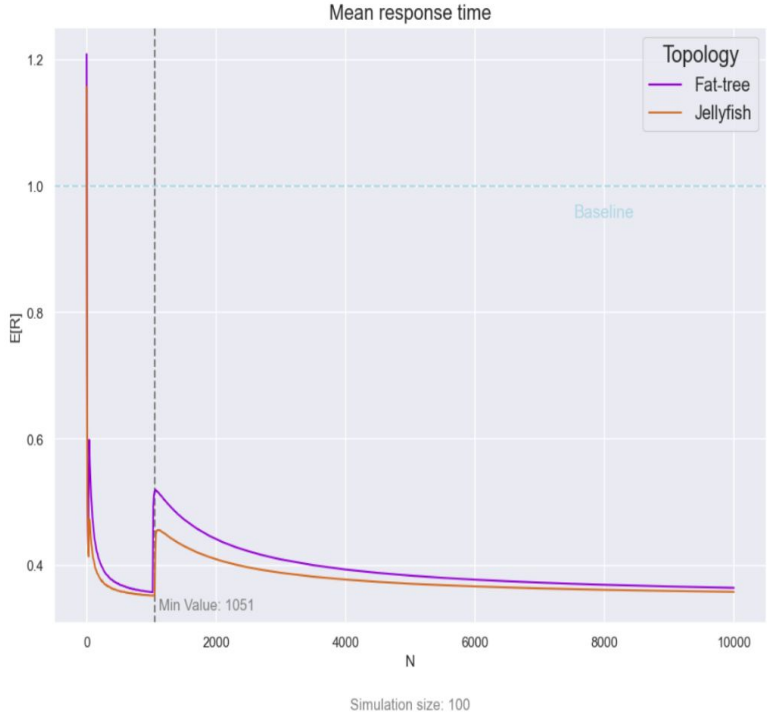
The blue line (r=8) is flat and uninformative, staying at the max value of 1. The orange curve (r=2) shows that the connectivity rate decreases with the number of nodes since each node has only two neighbors and there is a higher probability of having multiple connected components as the number of nodes increases. In contrast, with degree 8 the graph is highly connected, so a decrease in the connectivity rate won't happen even with a very huge number of nodes. This results are well justified by the so called *six-grade of separability theory*.

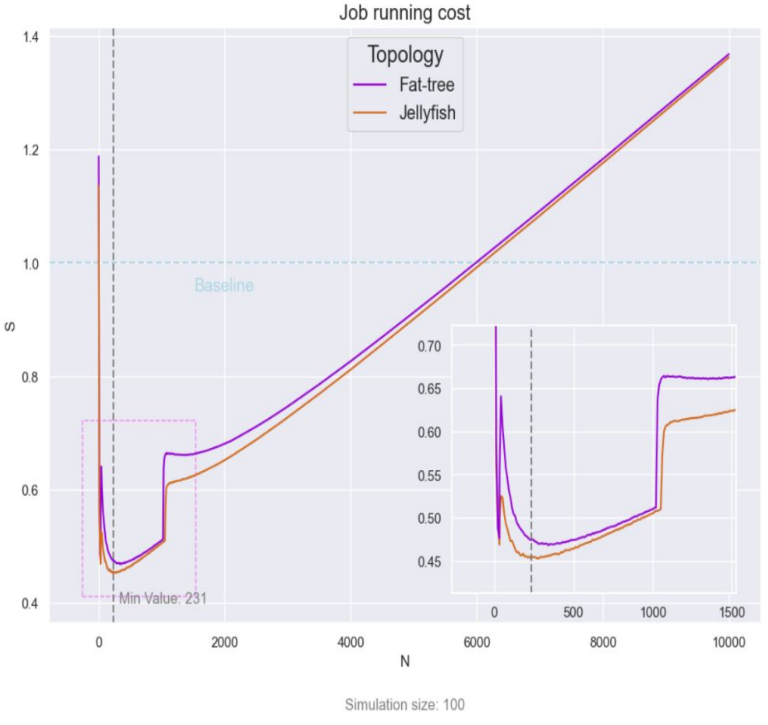
ALGORITHM FOR THE PERFORMANCE EVALUATION OF THE TOPOLOGIES

- If the topology type is a:
  - Fat-tree : build the topology outside the for loop because all is deterministic.
  - Jellyfish: build it many times to capture the randomness of its nature;
- Build a list with different number (n) of used servers in the range (1:10000)
- For each simulation iteration (i):
  - select a random source server from the topologies;
  - compute all the shortest paths from source to all the other servers
  - For each n:
    - select the n servers that are nearest to the source;
    - calculate the round trip time (RTT_i) for each neighbour server;
    - calculate the average throughput (phi_i) for each neighbour server;
    - calculate the forth transmission time (forth_i) for each neighbour
    - calculate the back transmission time (back_i) for each neighbour
    - calculate the computing time on nodes (X_i) for each neighbour
    - compute the server usage time (Θ) as the sum of setup time and computing time on nodes for all neighbours and store it
    - compute the response time (R) as the maximum (to take into account the Straggler problem) of the forth transmission time, back transmission time, setup time, and computing time on nodes for all neighbours and store it

- Compute mean response time (E[R]) as the average of all response times (R)
- Compute mean server usage time (E[Θ]) as the average of all server usage times (Θ)

PLOTS OF OUR RESULTS FROM SIMULATED TOPOLOGIES





ANALYSIS OF RESULTS

First, we explicate that in the plots we report the performance in terms of **average response time** and **job running cost** of the specified topology. Both evaluation metrics are normalized against *baseline* performance (topology consisting of only one node).
Main **macro-features**:

1. Parallelization of the job on several nodes allows better levels of efficiency given by higher speed of response, i.e., performance grows very fast as the number of nodes increases until it reaches a minimum threshold around 1000 servers in use. This value is variable as the input hyperparameters such as *link capacity* (C), *heaviness of the file* (L_f) to be processed, and *complexity of the overall job* (X) to be done.
2. Beyond a certain number of servers, parallelization in subtasks loses efficiency. Probably the trade-off dynamic between *network (communication) complexity* and the level of actual *resource utilization* determines a real drawback to scalability.
3. Finally, for larger and larger values of N, the topology will tend asymptotically to the minimum value of response time. It is therefore important to emphasize that a larger network and a larger number of available servers is not always optimal but the correct choice depends on the complexity of the problem under consideration. The minimum value cannot be improved because communication costs cannot be eliminated (and therefore are not negligible).
4. Regarding jub running cost, this metric incorporates the concept of efficient resource utilization by also considering $E[\Theta]$. The minimum achievable level in this case is global and equal to **231**: as the number of nodes increases (with the same complexity of the problem), the average level of utilization decays linearly. Thus, considering also the mean response time, the optimal choice of N lies in the interval between the minimum points of the two curves ($231 \leq N^* \leq 1050$)
5. It is this second metric that makes it possible to identify a topology size for which it becomes really inconvenient to parallelize tasks (from 6000 servers onward it is more efficient to run the job locally)

In addition we have some differences between the two topologies: in general the jellyfish seems to get better results especially at inflection points.
The Fat-tree has two peak points for the sudden increase in the number of hops when using servers not directly tied to the same edge switch but in the same pod ($32<N<1024$) or not into the same pod ($N>1024$).
Therefore, since the number of hops passes from 2 to 4 when N =33 and from 4 to 6 when N = 1025, the response time will consequently get affected by the presence of the last added node (which is the farthest node from the source) because of the Straggler Problem.
Jellyfish also has such peaks but they are somewhat mitigated by the lack of pod structure, granting a leaner architecture (fewer involved switches and edges in the link) and with less pronounced hop jumps between non-adjacent nodes, thereby mitigating the Straggler problem causing bottleneck into the topology.

CONCLUSIONS

In conclusion, we can say that the Jellyfish is a topology that grants a shorter response time and at the same time allows a better exploitation of the available resources. In our case, for fixed levels of computational complexity, the gap in efficiency with respect to Fat-tree is not so large, but for particularly elaborate tasks it could be far more decisive.
Probably, however, the real advantage lies in the agility of this architectural choice: the costs of extending Jellyfish for higher computational capacity requirements are far lower than those required by other topologies given the "lean" aspect of the network itself (fewer switches and edges)