SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF INFORMATION ENGINEERING, ELECTRONICS AND TELECOMMUNICATIONS (DIET)

Challenge #1 - Topology Design

NBD - NETWORKING FOR BIG DATA AND LABORATORY

Professor:
Andrea Baiocchi

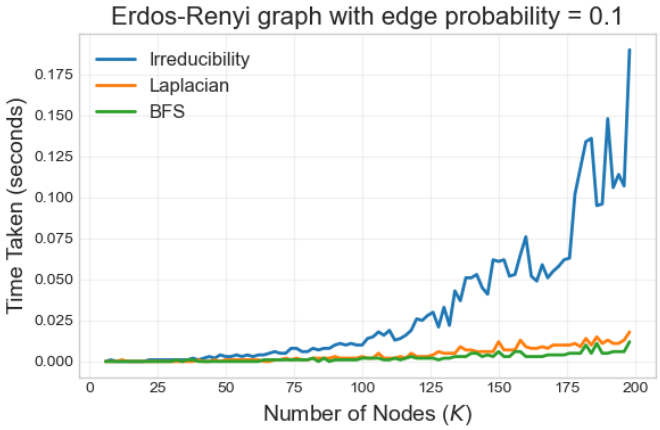
Group Viterbi:

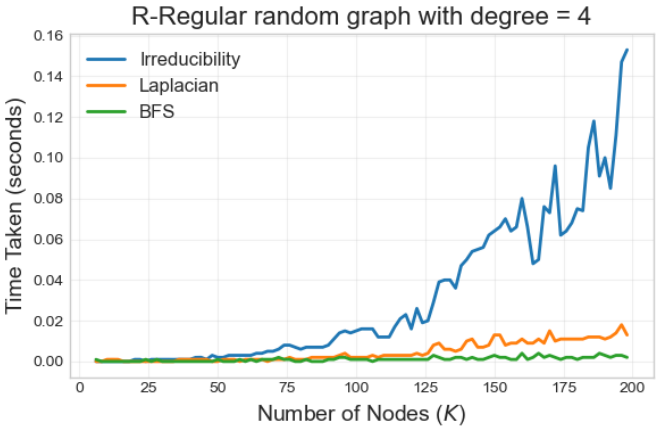
Giovanni Giunta	1177327
Mehrzad Jafari Ranjbar	1937944
Borbála Sára Tóth Gy	2066306

Academic Year 2022/2023

1 Assignment - Part 1

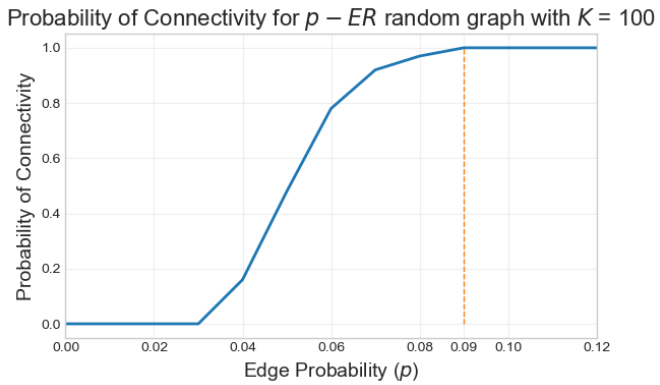
(i) We implemented three different Python functions to check the connectivity of Erdős–Rényi and r-regular random graphs, and then tested their complexity for different number of nodes K.
In the plots below, we can see that the Irreducibility and the Laplacian approach have the complexity of $O(K^3)$. However, the Irreducibility method is less efficient, as it requires performing several matrix multiplications, which is a computationally expensive task. The BFS algorithm is the most efficient among the three proposed methods, with a time complexity of $O(V + E)$. These results are consistent for both p-ER and r-regular random graphs.

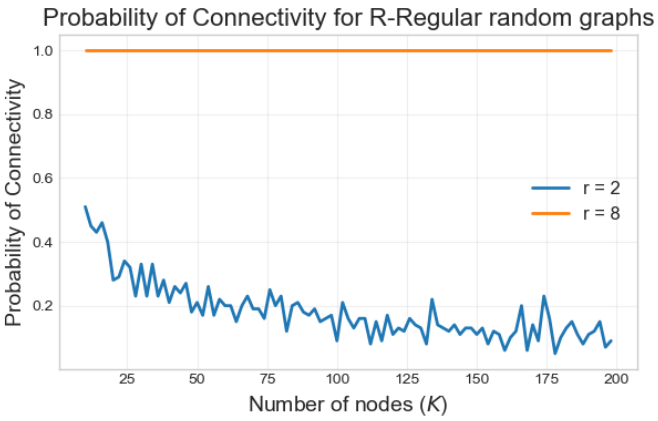




(ii) In the bottom left plot, we observe the behavior of a p-ER random graph with $K = 100$ nodes at different levels of edge probability p. It appears that Erdős–Rényi random graphs are almost certainly connected even with a relatively low edge probability of 0.09.

(iii) In the bottom right plot we observe the probability for a R-regular random graph to be connected, as we increase the number of nodes K, in the case of $r = 2$ and $r = 8$. When $r = 8$, the graph is always connected, as the probability of connectivity is 1 for all values of K. When $r = 2$, the probability of connectivity fluctuates and is lower than when $r = 8$. As we can see, a lower degree of regularity can lead to a lower probability of connectivity as we increase the value of K due to fewer edges connecting the vertices.





2 Assignment - Part 2

(i) To evaluate the **Mean Response Time**, we ran a loop ranging from 1 to $10,000$ servers. At each iteration N, we:

1. Create a dictionary in which the keys are the N closest servers to A, and the values are the number of hops from A to the server
2. Compute the average throughput, that is $TH_i = C\frac{1/T_i}{\sum_{j=1}^{N} 1/T_j}$ for each server i
3. Measure the time needed to send a fraction of data to each of the N servers: $\frac{(L_f + (L_f * f))/N}{TH_i}$
4. Run a simulation with 500 steps, where at each iteration we simulate:
   (a) The time needed by each server i to compute its share of processing, equal to $T_0 + E[X_i]$. In short, we sum the given constant T_0 to samples of size N drawn from a negative exponential distribution with mean equal to $E[X_i] = E[X]/N$. Each result is appended to the **job computations** list after each iteration
   (b) The fraction of data $L_{o,i}$ returned by each server i at the end of its share of job computation. We draw samples of size N from a uniform distribution defined in $[0, 2L_o/N]$. Each result is appended to the **outputs** list after each iteration
5. Compute the average of all the **job computations** samples (returns a single list)
6. Compute the average of all the **outputs** samples (returns a single list)
7. Add the overhead to each value of the **outputs** list, and measure the time needed to send each fraction of output back to server A: $\frac{(L_{o,i} + (L_{o,i} * f))/N}{TH_i}$
8. Sum the time needed to send the fraction of data from A to i (see 3.), the job computation time of i (see 5.), and the time needed to send the output from i back to A (see 7.). Now we have a single list with the average response times for each server i
9. Return the **Mean Response Time**, that is, the maximum value in the list with all the average response times (see 8.), since all the servers work in parallel

Selecting the N closest servers to A (and their distance from A in hops) is a trivial task, and can be easily computed. However, we proceeded nonetheless with generating a Fat-Tree and a Jellyfish network and performed a simulation starting from the actual graphs. In any case, both approaches lead to the same result. It is also worth mentioning that both the Fat-Tree and the Jellyfish networks have been given the same number of servers, so to compare them in a meaningful way. In fact, given a number of ports $n = 64$, the Jellyfish network was built with **servers** $= \frac{n^3}{4}$ and **switches** $= \frac{servers}{r}$, with $r = \frac{n}{2}$. This way, we end up having $65,536$ servers in both networks.

(ii) In the bottom left plot we have the results for the **Mean Response Time**. At first, it decreases dramatically as we distribute the job over the first 31 servers. Then it increases all of a sudden for $N = 32$, exactly when we start taking into account servers that are more distant from A. The effect is mitigated by adding more and more servers. For Fat-Tree we have an additional jump at $N = 1024$, when each additional server starts having $hops = 6$. For Jellyfish the jump is at $N = 1055$, when additional servers start having $hops = 4$. By adding more and more servers the mean response time decreases again, until it becomes almost flat towards $N = 10,000$: the job computation time becomes so small that T_0 becomes more relevant than $E[X_i]$.

(iii) In the bottom right plot we have the **Job Running Cost**. We can use it as a measure to explain the overall job computation cost of our networks, and is given by $S = E[R] + \xi E[\Theta]$.
In this case, $E[R]$ and $E[\Theta]$ are in contrast with each other. In fact, we can observe that after each jump we improve our performance, but as we add more servers, $E[\Theta]$ prevails and the function increases. This happens because $E[\Theta]$ is computed as the sum of the computation times of all the N servers.

(iv) It seems that the optimal number of servers is the same for both the Fat-Tree and the Jellyfish networks, and it is equal to 30. It corresponds to the point of minimum of the **Job Running Cost** function, and it has been highlighted on the chart. It is worth noting, however, that slightly different results can be achieved with a different number of Monte Carlo simulations.

(v) In general, we can say that the Jellyfish network topology performs better than the Fat-Tree, both in terms of **Mean Response Time** and **Job Running Cost**. In both scenarios, it is evident that adding more and more servers is not always convenient. Distributing the job over many servers can be an advantage only if the distance from A (and consequently the throughput), is not significant. Adding to our pool of servers even one single node that is too distant from A, can dramatically decrease the overall performance of our network. Once the threshold is passed, bringing back the performance to acceptable levels requires too many additional resources, which leads to a waste of computational power, as we can see from the **Job Running Cost** plot: we would perhaps get close to the same response time, but at a higher job running cost.