Reimplementation of the Research Paper "Preferential deletion in dynamic models of web-like networks"

Implemented by Nabila Shahnaz Khan

***Abstract*— In this era of Internet and World Wide Web, where everything is connected, study of large and random web-like network has become of great eminence. The authors Deo and Cami proposed a discrete-time dynamic random graph model where nodes are both added (along with edge) and deleted preferentially. They studied different properties such as growth in the number of nodes and edges with respect to steps, degree distribution of deleted nodes and overall degree distribution of the graph model and showed that this model follows the power-law degree distribution. Here, I have tried to reimplement their work by generating their proposed network model and then analyzed the relation of growth rate of nodes, edges, degree distribution of deleted nodes and degree distribution of the whole graph with respect to the number of iterations. The results were promising and they conceded with the results of the paper discussed here.**

***Index Terms*—Web-like network, Dynamic random graph model, discrete-time, Preferential deletion, Scale-free, Reimplementation**

I. Introduction

According to previous observational studies, Internet and Web-like networks are statistically similar to various large real-life networks such as transportation and communication, ecology, economic dependency, research connection, power distribution, brain cell signaling, food chain, etc. In fact any connected structure in real-life can be presented and analyzed as a web-like network. Understanding such huge and dynamic networks can be really complex. The increasing prominence and applications of such networks have engendered a research field dedicated to analyzing and understanding their properties along with the functionalities.

Web-like networks are every large, dynamic and they are considered to have no centralized control. They are seemed to follow scale-free power-law and have a greater degree of clustering [1]. A dynamic random graph model is a graph model which starts with a fixed number of nodes and at different points of discrete-times, nodes with edges are added and existing nodes are deleted randomly following some stochastic or preferential rules. Different research works [2]–[7] related to such graph models have already been done and still going on. Among them, Birth-only ones are the most common which use either preferential attachment [8] or copying [7]. In contrast, models including both birth and deletion have been studied much less. In 2000, Dorogovtsev and Mendes [5] proposed a model with an uniform deletion of edges while Chung et al. [9] and Cooper et al. [4] separately proposed a model that combines birth with uniform deletion of edges and nodes.

In this paper proposed by Deo and Cami [10], the authors proposed a dynamic random graph model which combines addition of nodes and edges with a preferential deletion of nodes where nodes having small-degree have higher probability of getting deleted. They analyzed properties of their proposed graph model and according to their yielded results, it was pretty evident that their proposed preferential deletion model follows the known asymptotically power-law degree distribution. Here, I have implemented their proposed model, analyzed the same properties that they had analyzed and found my results in lined with their research work. This not only proofs the accuracy and authenticity of their work, but also emphasizes on the simplicity of this model.

The rest of the paper is organized as follows: Section II further defines the dynamic random graph model proposed in [10]. Section III demonstrates the algorithm I have followed and analyzes it's runtime. In Section IV I have analyzed the properties of my implemented model and compared my results with their results. Finally, Section V concludes the paper.

dynamic growth

II. Random Graph Model with Preferential Deletion

In this model, initially the graph G contains one node and one-loop. At each time step t, either a new node is added along with edge (Birth Process) or a node is deleted (Death Process). Both of the processes are described below:

1) Birth Process:: If probability $\leq$ p (user-defined), a new node (u) is added with an incident edge. From the existing nodes, a node is selected which will be the other end-node (v) of that edge. This node v is selected based on Equation 1:

$$\frac{degree(EachNode)}{2 * TotalEdges} \tag{1}$$

Based on this formula, a weight is assigned to each existing node and then a node is selected randomly. As nodes with higher degree will be assigned higher weight, they are more likely to get selected.

2) Death Process:: When probability is q (q = 1 - p), a node u is chosen for deletion using the Equation 2 given below:

$$\frac{TotalNodes - degree((EachNode)}{TotalNodes - 2 * TotalEdges} \tag{2}$$

Here, the nodes with lower degree are more likely to get deleted as they are assigned lower weights. While deleting a node, all the edges incident to it are also deleted.

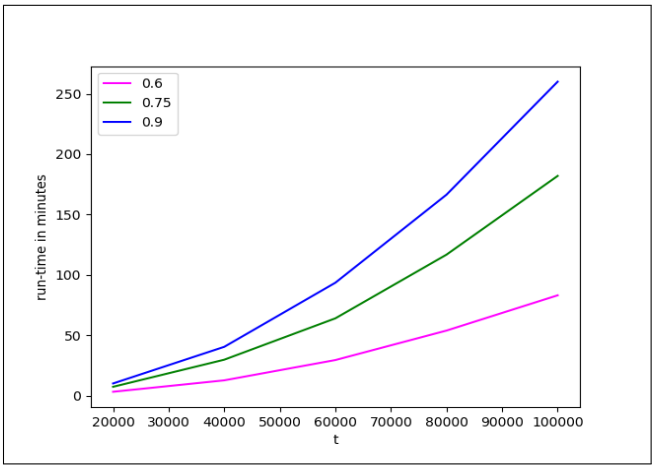

Fig. 1. Runtime comparison with respect to the number of iterations(20000, 40000, 60000, 80000, 100000) and probabilities (0.6, 0.75, 0.9)

III. Implementation of the Model

The algorithm has been implemented using the Networkx package in python (Code: **"Generate.Graph.py"**). Initially, instead of creating a loop-edge, I created the graph G with two nodes [0,1] and one edge (0,1) between them. In order to keep the runtime as small as possible, the code has been written efficiently, avoiding any extra calculations.

A. Algorithm

The basic algorithm [Algorithm 1] for generating the dynamic random graph model with preferential deletion shows the basic structure of the code.

B. Run-time Analysis

The runtime of this algorithm is exponential and it depends on the number of nodes. If number of nodes is N then runtime is O(e^N). With increasing number of nodes, the runtime can increase significantly. That's why while writing the code, special care was given to keep the run-time of the code as small as possible for a small subset of nodes. That way, even with large number of nodes, the program will finish running in finite time. For 1000 iterations, the runtimes (in seconds) of the implemented code for probabilities 0.6, 0.75, 0.9 are 1.484, 1.781, 2.234 respectively (approximate value). In case of 100,000 iterations, for the same probabilities the run times (in minutes) were 83.06, 181.98 and 260 approximately which is quite reasonable.

Figure 1 shows the growth of runtime with respect to iterations. Here, we can see that with increasing number of iterations, the runtime increases exponentially. Also, as the probability p increases, number of deaths decreases and number of nodes at any step increases. For example, number of nodes at iteration 40,000th will be more in case of probability 0.9 compared to probability 0.6. That's why with increase in probability, the runtime increases at any step t.

IV. Analyzing the Properties of the Graph

In this paper authors have analyzed how the numbers of nodes and edges change with respect to the number of

Algorithm 1 Generating Dynamic Random Graph with Preferential Deletion Model

```
N ← set_of_nodes
M ← set_of_edges
t ← 100001
p ← probability
node_to_be_added ← 1
// creating graph G with two nodes and one edge
G ← create_Graph()
G.add_node(0)
G.add_node(1)
G.add_edge(0, 1)

for iteration ← 1 to t do
    nt ← G.number_of_nodes()
    mt ← G.number_of_edges()
// Case Handling #1: When all nodes get deleted, again create the graph
    if nt = 0 then
        G.add_node(0)
        G.add_node(1)
        G.add_edge(0, 1)
        nt ← 2
        mt ← 1
// Case Handling 2: When all edges get deleted, create an edge between first two nodes
    else if mt = 0 then
        if nt = 1 then
            node_to_be_added ← node_to_be_added + 1
            G.add_node(node_to_be_added)
            nt ← 2
        G.add_edge(G.1st.node, G.2nd.node)
        mt ← 1
//Adding or Deleting node
    random ← any random value between 0 and 1

//Birth Process
    if random ≤ p then
        node_to_be_added ← node_to_be_added + 1
        for each node N_i in G do
            P[N_i] ← degree(N_i)/(2*mt)        // calculating probability distribution of all existing nodes "P[N]" based on their degree for adding edge
        node_to_add_edge ← Random(N, P[N])     // Given P[N], randomly select other end-node
        G.add_node(node_to_be_added)
        G.add_edge(node_to_be_added, node_to_add_edge)

//Death Process
    else if random > p then
        for each node N_i in G do
            P[N_i] ← (nt−degree(N_i))/(nt^2−2*mt)   // calculating probability distribution of all existing nodes "P[N]" based on their degree for deletion
        node_to_delete ← Random(N, P[N])       // Given P[N], randomly select a node to delete
        G.remove_node(node_to_delete)

return G
```

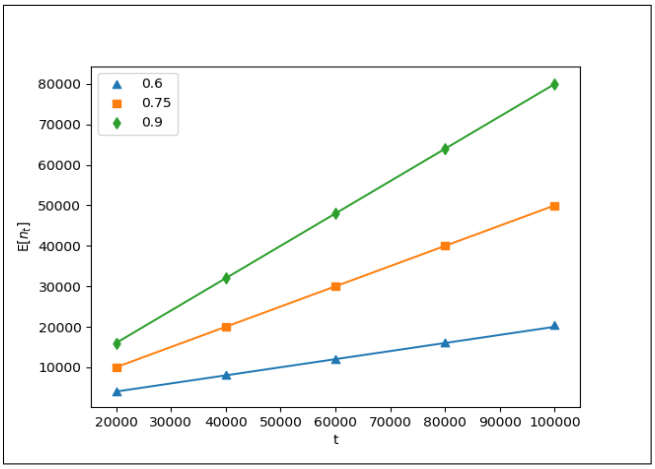

Fig. 2. Growth in the number of nodes with respect to steps (20000, 40000, 60000, 80000, 100000) for three different probabilities (0.6, 0.75, 0.9)

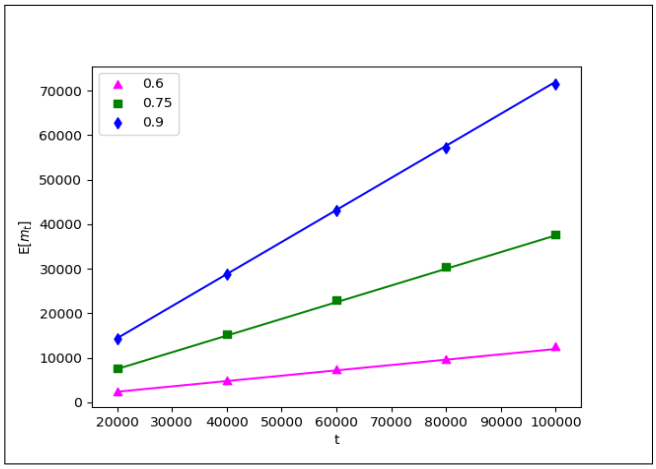

Fig. 3. Growth in the number of edges with respect to steps (20000, 40000, 60000, 80000, 100000) for three different probabilities (0.6, 0.75, 0.9)

iterations for their proposed model. They also analyzed the change in the degree distribution of a node to be deleted at a certain time (step) and also the overall degree distribution of the whole graph. They compared their simulated results with the theoretically predicted values and showed that the simulated results were in complete sync with the theoretical results. Thus, they proved that their proposed model is scale-free (follows power-law) like other web-like networks.

Here, after generating the model, the same features were tested for the simulated model (Graph G) and the results were compared with the theoretical results (just like the paper [10]). The comparisons are shown below along with the explanation:

A. Growth in the Number of Nodes and Edges:

The growth of the number of nodes and edges of the generated graph have been shown with respect to the number of iterations (up to 100,000) for different probabilities (0.6, 0.75, 0.9) and the results have been plotted using the code **"Plot1.and.Plot2.py"**.

1) Growth of Nodes:: The expected value of nodes at any time point (iteration) t can be represented using Equation 3:

$$E[n_t] = (p - q)t + 2q \tag{3}$$

In Figure 2, straight lines represent the expected value of nodes and the scatter plot represent the number of nodes found at different steps based on simulation. As we can see from the figure, the expected value and simulated value completely align with each other. This confirms the assumption of the authors.

2) Growth of Edges:: The expected value of edges at any time point (iteration) t can be represented using Equation 4:

$$E[m_t] = p(p - q)t \tag{4}$$

In Figure 3, straight lines represent the expected value of edges and the scatter plot represent the number of edges found at different steps based on simulation. In this figure also, they completely align, proving the analysis of the authors to be correct.

B. Degree Distribution in the First neighborhood of the Deleted Node

Here, the authors showed the degree distribution adjacent to the node chosen for deletion during a certain step t of a Graph G. In their simulation for p = 0.8 and step t = 40,000, they averaged 1000 realizations for the graph G. The expected value of edges at any time point (iteration) t can be represented using Equation 5:

$$E[N_{k,t}^{(1)}] = \frac{kN_{k,t}}{n_t - \overline{d_t}}[1 - \frac{2 * \overline{d_t}}{n_t}] \tag{5}$$

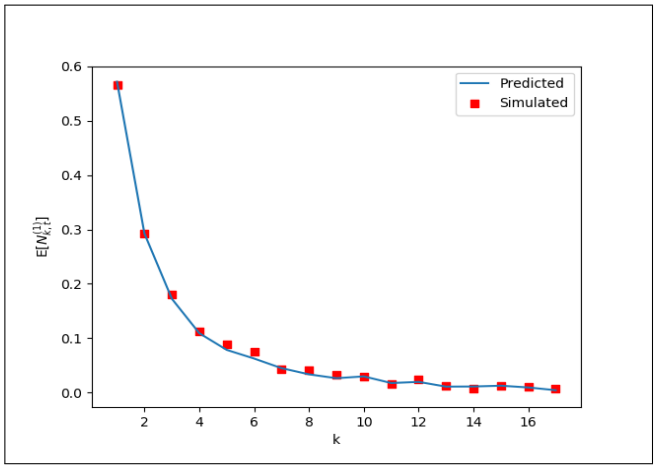

Fig. 4. The expected number of neighbors of degree k of a node chosen for deletion

In this paper, the simulation was also done for p = 0.8, t = 40,000 and for 1000 realizations and the comparison result has been shown in Figure 4. This figure has been plotted using the code **"Plot3.py"**. In this figure, straight lines represent the

Predicted values and the scatter plot represent the simulated values. As we can see, for degree less than 4, the simulated values are slightly less than the predicted values, but after that the expected values and simulated values seem to align. So, the assumption of the authors can be considered as true.

C. Degree Distribution

Next, they compared the degree distribution of the of the Graph G with the expected value using a log-log plot of the reverse cumulative degree distribution. The expected cumulative degree distribution of a graph can be represented using Equation 6:

$$a_k^{(1)} = \Theta[k^{-1-(2p/2p-1)}] \tag{6}$$

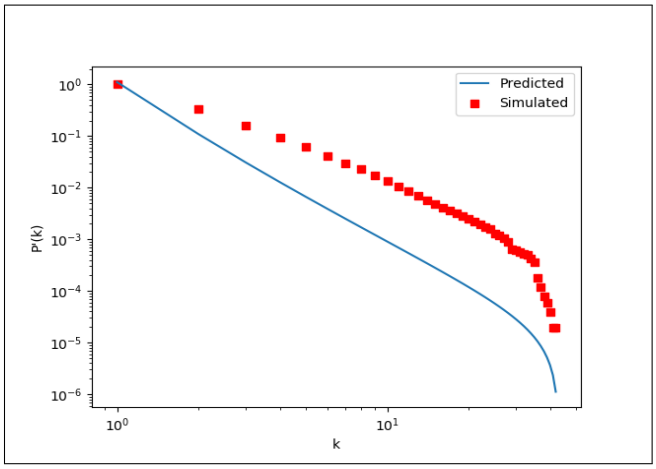

Fig. 5. Log-log plot of the cumulative degree distribution of the graph generated by the preferential model. Here, p = 0.8, t = 100,000 and prediction line followed Equation 6

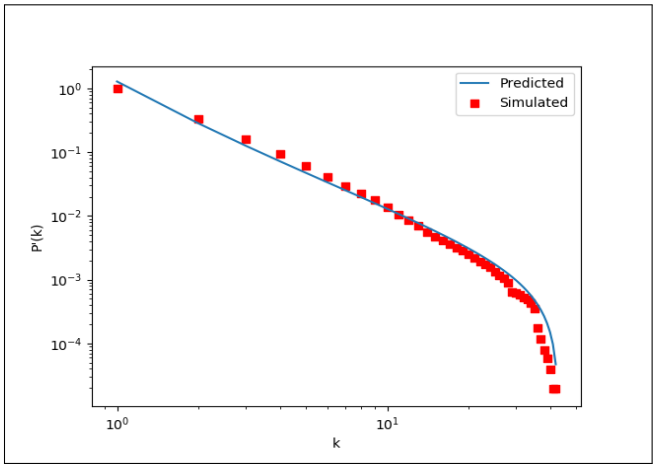

Fig. 6. Log-log plot of the cumulative degree distribution of the graph generated by the preferential model. Here, p = 0.8, t = 100,000 and prediction line followed Equation 7

Here, Figure 5 shows a comparison between the graph analytical prediction given by Equation 6 and the data obtained by the simulated model for p = 0.8 and 100,000 iterations. Here, $P'(k) = \sum_{i \geq k} P(i)$. As we can see, though the predicted and simulated values do not completely align, still the simulated values maintain a similar pattern while the $P'(k)$ decreases with the increasing value of k.

As we know for scale-free graph, $P(k) = k^{-\gamma}$ where γ can be any value between 0 to 2. So, changing the parameters of Equation 6 a bit (As shown in Equation 7), I got a better alignment which has been represented in Figure 6.

$$a_k^{(1)} = \Theta[k^{-1-(1.25*p/2p-1)}] \tag{7}$$

This indicates that the degree distribution of the predicted model maintains the power-law. Both Figure 5 and Figure 6 have been plotted using code **"Plot4.py"**.

V. Discussion and Conclusion

From this paper, it is clearly visible that for the reimplemented model, the simulated data closely aligned with the predicted data. It points out the fact that the the assumptions made in paper [10] are all justified. The dynamic random graph model presented in that paper does follow the power-law distribution and hence can be categorized as a scale-free graph.

References

[1] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, no. 2, p. 167–256, Jan 2003. [Online]. Available: http://dx.doi.org/10.1137/S003614450342480

[2] B. e. Bollobás, O. Riordan, J. Spencer, and G. Tusnády, "The degree sequence of a scale-free random graph process," *Random Structures & Algorithms*, vol. 18, no. 3, pp. 279–290, 2001.

[3] C. Cooper and A. Frieze, "A general model of web graphs," *Random Structures & Algorithms*, vol. 22, no. 3, pp. 311–335, 2003.

[4] C. Cooper, A. Frieze, and J. Vera, "Random deletion in a scale-free random graph process," *Internet Mathematics*, vol. 1, no. 4, pp. 463–483, 2004.

[5] S. N. Dorogovtsev and J. F. F. Mendes, "Scaling behaviour of developing and decaying networks," *EPL (Europhysics Letters)*, vol. 52, no. 1, p. 33, 2000.

[6] P. Krapivsky and S. Redner, "Rate equation approach for growing networks," in *Statistical mechanics of complex networks*. Springer, 2003, pp. 3–22.

[7] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, "Stochastic models for the web graph," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE, 2000, pp. 57–65.

[8] A.-L. Barabási, R. Albert, and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A: Statistical Mechanics and its Applications*, vol. 272, no. 1-2, pp. 173–187, 1999.

[9] F. Chung and L. Lu, "Coupling online and offline analyses for random power law graphs," *Internet Mathematics*, vol. 1, no. 4, pp. 409–461, 2004.

[10] N. Deo and A. Cami, "Preferential deletion in dynamic models of web-like networks," *Information processing letters*, vol. 102, no. 4, pp. 156–162, 2007.