IE 456 PROJECT

Random Graph Generation With Prescribed Degrees

Hasancan Cebeci-20149402036
Burak Caymaz-2019402003

Introduction

Generating random graphs is an important asset for studying real-word networks and their properties. It has numerous applications in network modeling, hypothesis testing, benchmarking, null models and sampling (Newman,2010). Random graphs help researchers understand the behavior and characteristics of complex networks and technological networks such as World Wide Web, social networks, peer-to-peer networks and biological networks by providing controllable settings and inputs for analysis and comparison.

There are several algorithms for random graphs with given degree distributions. Most of these algorithms get stuck or produce multiple edges and loops while trying to come up with a random connected graph with the prescribed degree sequence. Those algorithms try to tackle those situations by restarting the whole process and trying again. As the degree of parameters grow, the probability that a restart is needed approaches to 1. This results in a necessity of a large number of trials to come up with a simple graph. The paper "A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees" by Joseph Blitzstein and Persi Diaconis introduces a new algorithm for generating random graphs with a given degree sequence, which does not get stuck like the former algorithms thanks to the Erdös-Gallai characterization and a carefully chosen order of edge selection, addressing an important issue in the literature. This report gives an understanding of different algorithms that generate random graphs, provides an overview of the Sequential Importance Sampling Algorithm and comparison of it to other methods in the literature with the help of implementation on Python, and observation of results.

Algorithms to Generate Random Graphs

Generating random graphs is a process that evolved. Many algorithms were introduced and as time passed, algorithms became more sophisticated. Some algorithms that generate random graphs include:

1- Erdos-Renyi Model: One of the simplest approaches to generate a random graph was found by Erdos and Renyi. Erdos-Renyi (in short, ER) graphs are a model for generating random graphs, introduced by Paul Erdos and Alfred Renyi. In the ER model, a graph G(n, p) is created by taking n nodes and connecting each pair of nodes with an edge independently with a probability p. These graphs are characterized by having a binomial degree distribution, and they have been extensively used as a basis for analyzing various properties of random graphs (Erdos & Rényi, 1959). However, since all vertices have the same expected degree, a different approach was preferable for handling dependent edges and non-binomial degree distributions. For instance, numerous researchers have found that the Web and a number of other massive networks follow a power-law degree distribution (often with an exponential cutoff or truncation at some point), where the likelihood of a vertex having degree k is proportional to k(-a) for some positive constant a. These graphs with power-law degree distributions are called scale free graphs. This type of graph is commonly found in real-world

networks, including the internet, social networks, and biological systems (Barabási & Albert, 1999). The Barabási-Albert (BA) model is one method for generating scale-free networks, employing a mechanism called preferential attachment. In this model, nodes are incrementally added, with each new node connecting to existing nodes based on a probability relative to their degree. This procedure results in a network exhibiting a power-law degree distribution (Barabási & Albert, 1999). In their paper, Blitzstein and Diaconis also state that the sequential algorithm could be utilized to generate graphs with a power-law degree distribution.

2- The Pairing Model: The Pairing Model fixes n as the number of vertices and d as the degree such that nd is even. Each vertex becomes a cell with d points inside of it. Then, the model looks for a perfect matching of those nd points uniformly. The matching induces a d-regular multigraph. The algorithm generates multigraphs in this manner until a simple graph is obtained. The downside of The Pairing Model is that for large number of d, expected trials to obtain a simple graph goes to infinity.

3- Havel-Hakimi and its variants: In Havel-Hakimi Algorithm, first of all, we choose a vertex according to a rule (highest degree first/smallest degree first/random). Then, we distribute this vertex's degree among other vertices again according to a rule by creating edges among vertices. For each vertex created, we subtract 1 from the degree of vertices and we continue this process until we reach the zero vector, i.e, all the degrees in the sequence becomes zero.

4- Markov Chain Monte Carlo (MCMC) Algorithms: MCMC methods enable the creation of random graphs by sampling from a vast range of potential graphs. The process begins with an initial graph and introduces minor modifications, such as deleting or adding edges, resulting in a proposed graph. The algorithm then decides to accept or reject this proposal based on the comparative probabilities of the current and proposed graphs. This decision-making involves random elements. With repeated iterations, the resulting graphs tend to mirror those found in the intended distribution. Despite this, there is a necessary "burn-in" phase and the method does not ensure a uniform distribution across all conceivable graphs.

5- The Sequential Algorithm: The Sequential Algorithm sees degrees as "residual degree", which is number of edges needed to saturate that vertex. It is a similar algorithm to Havel-Hakimi with some slight but important changes. The algorithm chooses the smallest degree vertex and picks another vertex with probability proportional to its degree and puts an edge between them, subtracting 1 from each vertices' degrees. The algorithm continues until the zero vector is reached.

Implementation of Havel-Hakimi Algorithms and The Sequential Algorithm

In order to conduct a benchmark and analyze the advantages of Sequential Importance Sampling Algorithm over the similar studies in the literature, we implemented five different algorithms, four Havel-Hakimi variants and The Sequential Algorithm. All of these algorithms take inputs of graphical random degree sequence of length n and try to give a connected graph as an output. The following algorithms are implemented using Python:

1- Havel-Hakimi (Random Vertex Selection, distributed to the Highest Degree):

Here is the algorithm;

1- First, we give a graphical degree sequence $d=\{d_1,d_2,\ldots,d_n\}$ as the input.
2- Then, we choose a random vertex i, with degree d_i and connect it with the d_i vertices in a non-increasing order according to the degree sequence.
3- Next, we subtract 1 from the degrees of the chosen vertices and naturally, remaining degree of vertex i becomes zero. We reorder other degrees in the sequence in a non-increasing order.
4- If $d_i = 0$ for $i = 1,2,3,\ldots,n$, we stop. Else, go to step 2.
5- We check connectivity with Depth First Search (DFS).
6- If the graph is connected, give output G, a connected graph of a degree sequence d. Else, do pairwise interchanges until the graph is connected. Repeat step 6.
7- Stop the timing and report it.

3 more variants of Havel-Hakimi were implemented. They only differ in vertex selections. For example, In High-to-high Havel-Hakimi, in step 2 we choose the vertex with highest degree and distribute it among the vertices with degree sequences in a non-increasing order. All other steps are the same as described. Those three additional Havel-Hakimi variants are:

1- Havel-Hakimi (Highest Degree Vertex First, distributed to the Highest Degree):

2- Havel-Hakimi (Smallest Degree Vertex First, distributed to the Highest Degree):

3- Havel-Hakimi (Smallest Degree Vertex First, distributed to the Smallest Degree):

2- The Sequential Algorithm: The Sequential Algorithm differs from Havel-Hakimi algorithm in the way it does the distribution. In Havel-Hakimi, the distribution is done in either non-increasing order or non-decreasing order. On the other hand, in sequential algorithm, the vertices are chosen with probability proportional to their degrees while distributing. This is called importance sampling approach.

Here is the algorithm;

1- First, we give a graphical degree sequence $d=\{d_1,d_2,\ldots,d_n\}$ as the input.
2- Then, we choose the vertex i with the smallest degree di.
3- Connect vertex i with another vertex j in d with probability proportional to its degree d_j. Repeat this until d_i edges are connected to vertex i.
4- Next, we subtract 1 from the degrees of the chosen vertices, naturally making $d_i = 0$, and update d.

5- If $d_i = 0$ for i = 1,2,3,…,n , we stop. Else, repeat steps 3 and 4.
6- We check connectivity with Depth First Search (DFS).
7- If the graph is connected, give output G, a connected graph of a degree sequence d. Else, do pairwise interchanges until the graph is connected. Repeat step 6.
8- Stop the timing and report it.

The sequential algorithm presented by Blitzstein and Diaconis offers a significant advantage over other algorithms by efficiently generating random graphs with a prescribed degree sequence using an importance sampling approach, that is choosing a vertex with probability proportional to its degree. In their paper, Blitzstein and Diaconis showed an interesting comparison. They observed that when estimating the number of graphs, the degree based selection resulted in a much less percent error than the selection with uniform candidates. The reason behind that is thee importance sampling algorithm can decrease the estimator variance by emphasizing the "more important" values by sampling more frequently. The essential advantage of The Sequential Algorithm is that it never gets stuck thanks to the importance sampling algorithm and Erdos-Gallai characterization.

Checking Connectivity

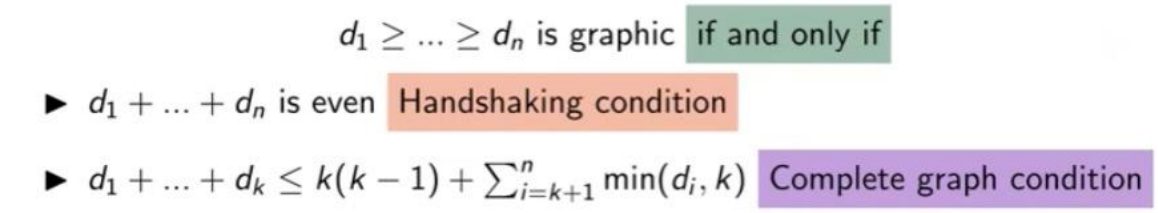
Connectivity of the graphs checked by the Depth First Search algorithm. If a graph is connected that means starting from any node, someone can reach all the nodes without lose of connectivity. In DFS, algorithms selects a random vertex and checks if its adjacent nodes are visited. When it finds a node which is not visited, it jumps to that node and again checks if there is any adjacent node that is not visited. It does it until it cannot find any node to visit. When it is stuck, it jumps back to the previous node and checks for the other adjacent nodes which is not visited. It repeats this process until algorithm visits all the nodes that can be reached from starting node. At the end, if the algorithm can reach all the nodes from starting point, the graph is said to be connected.

Is The Given Degree Sequence Graphical or Not?

In all the algorithms we implemented, it starts with the step “Give a graphical degree sequence as an input.”. The obvious question that comes to mind is: “How do we know if a given degree sequence is graphical or not?”.

The Erdos-Gallai graphicality criterion is a necessary and sufficient condition to determine if a given degree sequence is graphical (i.e., corresponds to a simple, undirected graph). Choudum (1986) provides a simple proof of this criterion using induction. The proof starts by showing that the criterion holds for the base case of a sequence with two elements: Choudum shows that, for a two-element degree sequence, the criterion is satisfied if and only if the sum of the degrees is even, which implies that there exists a simple graph corresponding to the sequence. Then, it shows that if the criterion holds for sequences of length k, it also holds for sequences of length k+1: Choudum demonstrates this by considering a degree sequence of length k+1 and removing the node with the highest degree. The result is a new degree

sequence of length k. The induction hypothesis is then applied to this reduced sequence, and it is shown that the Erdos-Gallai criterion holds for the original k+1 length sequence if and only if it holds for the reduced sequence. Choudum (1986) proves that the Erdos-Gallai criterion is valid for all degree sequences by concluding the induction.



Erdös-Gallai Characterization

The Erdös-Gallai Characterization has two conditions for a degree sequence to be graphical:

1- Handshaking condition: The sum of all degrees must be even. This is because an edge contributes 2 to the total degree number, therefore a graph with m edges have a total degree of 2m. Regardless of m, this number is even.

2- Complete graph condition: The sum of all degrees have an upper bound k*(k-1), which is the total degree number of a k-regular graph, plus an error term.

How to Make a Graph Connected?

This is another important concept for these algorithms. Although, a sequence that can form a connected graph is given to the algorithms, they cannot form a connected graph. In such situations, an algorithm is needed to transform these not-connected graphs into connected graphs. Pairwise interchange method is used to solve this problem. To perform pairwise change, first the subgraphs of the given graphs are found with the help of the DFS algorithm. Since the given graph is not connected, starting the search from a node will not end up in visiting the whole graph but a subgraph of the given graph. By repeating this process, all the subgraphs of the given graph can be determined. After determining the subgraph, the crucial part is to find the edges whose removal from this subgraph will not change the nodes in the subgraph. Let this edge be (x,y), connecting vertex x and y. In order to connect this subgraph to another subgraph, a random edge is selected on the other subgraph. Let this edge be (a,b), connecting vertex a and b. Then connecting vertex x to vertex a and vertex y to vertex b will give connect this two subgraphs. Then, it is easy to conclude that if the number of edges whose removal will not change the nodes in the selected subgraph is bigger or equal to the total number of subgraphs in given graph, a transformation to a connected graph can be performed.

Results

Small-to-Small algorithms turned out to be an ineffective algorithm. When the algorithm first connects small vertices to other small vertices, big vertices cannot find enough vertices to connect. Therefore, after some iterations remaining sequence ends up in being non-graphical sequence and algorithm cannot create a graph.

N	Distribution	Algorithm	Time
40	Power	Sequential	0.010055383046
40	Power	HTH	0.001989841461
40	Power	RTH	0.00335407257080
40	Power	STH	0.00099706649780
100	Power	Sequential	0.09971308708190
100	Power	HTH	0.015825033187
100	Power	RTH	0.022868712743
100	Power	STH	0.003988345464
250	Power	Sequential	1,3488252162
250	Power	HTH	0.129705349604
250	Power	RTH	0.184001763661
250	Power	STH	0.027160962422
40	Binomial 0.7	Sequential	0.1960050264994
40	Binomial 0.7	HTH	0.0030037562052
40	Binomial 0.7	RTH	0.0023277600606
40	Binomial 0.7	STH	0.0023266474405
100	Binomial 0.7	Sequential	4,85319558779
100	Binomial 0.7	HTH	0.01832675933837
100	Binomial 0.7	RTH	0.014484564463297
100	Binomial 0.7	STH	0.0164642333984
250	Binomial 0.7	Sequential	165,67045044898
250	Binomial 0.7	HTH	0.168679237365
250	Binomial 0.7	RTH	0.1596525510152
250	Binomial 0.7	STH	0.16005794207255

Table 1: The required time for different number of vertices(N) and different distributions.

N	Distribution	Algorithm	Connectivity	Pairwise Interchanges
40	power	Sequential	0	10,333333333
40	power	HTH	0	16,666666667
40	power	RTH	0	11,333333333
40	power	STH	1	0,000000000
100	power	Sequential	0	21,666666667
100	power	HTH	0	35,666666667
100	power	RTH	0	24,000000000

100	power	STH	1	0,000000000
250	power	Sequential	0	57,666666667
250	power	HTH	0	102,666666667
250	power	RTH	0	70,666666667
250	power	STH	1	0
40	Binomial 0.2	Sequential	1	0
40	Binomial 0.2	HTH	1	0
40	Binomial 0.2	RTH	1	0
40	Binomial 0.2	STH	1	0
100	Binomial 0.2	Sequential	1	0
100	Binomial 0.2	HTH	1	0
100	Binomial 0.2	RTH	1	0
100	Binomial 0.2	STH	1	0
250	Binomial 0.2	Sequential	1	0
250	Binomial 0.2	HTH	1	0
250	Binomial 0.2	RTH	1	0
250	Binomial 0.2	STH	1	0
40	Binomial 0.5	Sequential	1	0
40	Binomial 0.5	HTH	1	0
40	Binomial 0.5	RTH	1	0
40	Binomial 0.5	STH	1	0
100	Binomial 0.5	Sequential	1	0
100	Binomial 0.5	HTH	1	0
100	Binomial 0.5	RTH	1	0
100	Binomial 0.5	STH	1	0
250	Binomial 0.5	Sequential	1	0
250	Binomial 0.5	HTH	1	0
250	Binomial 0.5	RTH	1	0
250	Binomial 0.5	STH	1	0
40	Binomial 0.7	Sequential	1	0
40	Binomial 0.7	HTH	1	0
40	Binomial 0.7	RTH	1	0
40	Binomial 0.7	STH	1	0
100	Binomial 0.7	Sequential	1	0
100	Binomial 0.7	HTH	1	0
100	Binomial 0.7	RTH	1	0
100	Binomial 0.7	STH	1	0
250	Binomial 0.7	Sequential	1	0
250	Binomial 0.7	HTH	1	0
250	Binomial 0.7	RTH	1	0
250	Binomial 0.7	STH	1	0

Table 2: Connectivity (1 if connected, 0 otherwise) and number of pairwise interchanges needed.

As we can see from the results, Small-to-High(STH) algorithm is the best algorithm to create the connected graphs. Required number of pairwise interchanges for STH algorithm is minimum. Because all of the small degree vertices are connected to the high degree vertices.

Therefore, this algorithm inhibits small vertices to stay out and create small subsets. On the other hand, high-to-high (HTH) algorithm is the worst algorithm to create connected graphs. Unlike small-to-high algorithm, it connects high degree vertices to the other high degree vertices, which causes the limit of the higher value vertices to fill with the other high degree vertices and small degree vertices to stay outside. Sequential algorithm is better than Random-to-High and High-to-High in terms of required number of pairwise interchanges. That is because it again connects the small vertices to other possibly bigger vertices. Also it is significant that binomial sequences are better for algorithms to create connected graphs compared to power-series sequences. Power series includes high amounts of small degree vertices, such as 1,2,3,4 and few high degree vertices. Therefore, if the small degree vertices are not prioritized while connected, chances of ending up not connected graphs is high.

When we look at the average running times of the algorithms, sequential algorithm is the worst in almost every case. While creating the edges, sequential algorithm checks every time whether the remaining sequence is graphical or not. This control mechanism makes it fall behind other algorithms in terms of average time. Again, in binomial sequence, the high number of high order degree vertices makes it disadvantageous compared to power-law sequences. It increases the total number of edges. Therefore, sequential algorithm needs to check its selection of vertices for every edge. There are no significant differences for Havel-Hakimi algorithms in binomial sequences. Although, there is no big numerical value difference in power-law series, ratio of their time requirement of RTH and HTH to STH is significant.

In overall comparison, STH seems like the best alternative.

References

- Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. Science, 286(5439), 509-512.
- Blitzstein, J., & Diaconis, P. (2006). A sequential importance sampling algorithm for generating random graphs with prescribed degrees.
- Choudum, S. A. (1986). A simple proof of the Erdős-Gallai theorem on graph sequences. Bulletin of the Australian Mathematical Society, 33(1), 67-70.
- Erdos, P., & Rényi, A. (1959). On random graphs I. Publ. Math. Debrecen, 6, 290-297.
- M. Newman, Networks: An Introduction. New York, NY, USA: Oxford University Press, 2010.