IE 456 PROJECT REPORT

"A SEQUENTIAL IMPORTANCE SAMPLING ALGORITHM FOR GENERATING RANDOM GRAPHS WITH PRESCRIBED DEGREES "

By Joseph Blitzstein and Persi Diaconis

Group 9

Alican Yılmaz 2016402093

Gökay Besler 2016402108

Instructor: Tınaz Ekim Aşıcı

BOGAZICI UNIVERSITY

December 2021

Table of Contents

1-Introduction

Advances in technology led to a paradigm shift in network theory, some questions asked for small graphs have turned out to be redundant, and new questions emerged such as “What are the large-scale statistical properties of graphs?” Creating model of network via random graph generation is among the crucial steps in order to find answers to such questions [1]. Furthermore, generating random graphs with a given vertex degree sequence has applications in various areas ranging from social networks and biological networks to matrix theory and string theory. With the goal of better understanding or predicting the behavior of those systems, researchers have benefited from generating random graphs with a given degree sequence. Different techniques and algorithms have been developed and used to generate random graphs. This paper provides a novel approach to solving the random graph generation problem, namely 'Sequential Importance Sampling Algorithm'.

In this project, we further analyzed the proposed algorithm and compared it with different algorithms in terms of running time. In Section 2, the main contribution of the proposed algorithm is explained briefly. In Sections 3 and 4, Erdos-Renyi graphs and scale-free graphs are explained, respectively. In Section 5, the sufficient condition of the Erdos-Gallai graphicality criterion is shown. (The necessary condition is trivial and is already shown in the paper.) In Sections 6,7 and 8, the Havel-Hakimi Algorithm, the Configuration (Pairing) Model and the Sequential Algorithm are explained together with their implementation details, respectively. Finally, the experimental results are discussed in Section 9 and final remarks are made in Section 10.

2-Contribution of the Paper

The proposed algorithm handles some of the critical issues resulted naturally from the previous algorithms, such as the high probability of a restart needed as degree parameters grow and the getting stuck of the algorithm at some point. The algorithm proposed can generate a random graph with given degree sequence as well as given degree distribution. Furthermore, every realization of (d_1, d_2,…,d_n) has positive probability in the output of the algorithm. Although the output distribution is nonuniform, importance-sampling techniques can be used to simulate a general distribution with this approach [2].

3-Erdos-Renyi Graphs

The classical and simple random graph generation model due to Erdos-Renyi refers to the process of connecting each edge (i,j) with probability p, independently. Therefore, the degree of each vertex follows a Binomial(n-1,p) distribution. All vertices have same expected degree in this model. Erdos- Renyi calls the model defined by this process as G(n,p). In this notation, a graph of m edges has probability $p^m(1-p)^{M-m}$, where $M = \frac{1}{2}n(n-1)$ is the maximum possible number of edges [1]. Due to the independent property, the limiting case of n resulted in so-called “Poisson random graphs” and their structure depends on the value of p. For further information,

we refer to the [1]. Clearly, a different model is needed to account for dependent edges and different degree distributions that are different from binomial in the random graph generation process.

4-Scale-Free Graphs

Scale-free graphs refer to graphs (or networks) containing some hubs and central vertices that have a seemingly unlimited number of links and many vertices that have a very low degree [3]. Contrary to the Erdos-Renyi random graph model, researchers have observed that 'scale-free' graphs are not uncommon in the real world. Those kinds of graphs are heterogenous and unequal in terms of how influential the different nodes in the network are and it has been shown that the degree of connectivity and the frequency of the node follows “power-law” property. In other words, the number of nodes having degree k is proportional to $P(k) = k^{-\gamma}$ for large k. In [3], Barabasi and Eric Bonabeau gives various examples from different domains where scale-free networks are observed. For instance, the internet network, where nodes refer to routes and links to optical and other physical connections, shows properties of scale-free networks. In biology, network of cellular metabolism or protein regulatory networks are some of the examples of scale-free networks. For further examples, we encourage the reader to look for [3]. It has also been remarked that in most scale-free networks the value of γ in $k^{-\gamma}$ tends to fall between 2 and 3. By their very nature, scale free networks are also robust against accidental failures but more prone to coordinated attacks [3].

In [3], authors dive into the work of Erdos and Renyi model and reveal two main reasons to explain the scale-free networks and existence of hubs: “growth” and “preferential attachment”. On the contrary to Erdos-Renyi model, growth theory is based upon the idea that the number of nodes is not constant beforehand; rather, the network constantly expands. Also, during that growing process, new nodes tend to connect to the more connected nodes. This process can be thought of as 'rich gets richer'.

As a result, the authors claim that those aspects should be considered in the graph-generating process. Instead of uniformly randomly linking new nodes to others, the new nodes should be connected such that the new nodes are more likely to be connected to a node of higher degree. This process would lead to more realistic networks than random graph models. However, one needs to realize that this method also has some details that need to be addressed, such as how the process should be handled in the starting condition where all degrees are initially zero.

5-Erdos-Gallai Graphicality Criterion

$$Theorem(Erdos-Gallai): Let\ d_1 \geq d_2 \geq \ldots\ldots \geq d_n\ be\ nonnegative\ integers\ with \sum_{i=1}^{n} d_i$$

$$even. Then\ d = (d_1, d_2, \ldots d_n)\ is\ graphical\ if\ and\ only\ if$$

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} min(k, d_i)\ for\ each\ k \in \{1,2,\ldots n\}.$$

Degree sequence $(d_1, d_2, \dots, d_n)$ is called graphical sequence if there is a labeled simple graph with vertex set $\{1, \dots, n\}$ in which vertex i has degree d_i [2]. The necessity condition is trivial and was explained in [2]. Thus, it is omitted here and only the sufficient condition due to Choudum [4] will be summarized.

The proof by Choudum is done with the induction on s, where s equals sum of the degrees in the sequence. It is true that theorem holds for $s = 0$ or $s = 2$. Let us assume that the theorem is true for sequences whose sum is $s - 2$. And let π: $d_1 \geq d_2 \geq \dots\dots \geq dp$ be a sequence whose sum s is even and satisfies (EG). If π is proven to be graphical, then the proof is done by induction. First, a new sequence $\pi*$ is defined as follows:

$$\pi*: d_1 \geq \dots \geq d_{t-1} \geq d_t - 1 \geq d_{t+1} \geq \dots \geq d_{p-1} \geq d_p - 1$$

First, the author proves that the degree sequence $\pi*$ satisfies (EG) by dividing the proof into five cases. These steps are omitted here and can be found in [4]. By induction hypothesis, it is graphical. (See that $S(\pi*) = s - 2$). Let G be a realization of $\pi*$ on the vertices $v_1, v_2, \dots, v_p$. If $(v_t, v_p) \notin E(G)$, then $G \cup (v_t, v_p)$is a realization of π. Let $(v_t, v_p) \in E(G)$. $deg_G(v_t) = d_t - 1 \leq p - 2 \Rightarrow$ there is a v_m such that $(v_m, v_t) \notin E(G). deg_G(v_m) \geq deg_G(v_p) =>$ there is a v_n such that $(v_m, v_n) \in E(G)\ and\ (v_n, v_p) \notin E(G)$. By deleting edges $(v_t, v_p), (v_m, v_n)$ and adding the edges $(v_t, v_m), (v_n, v_p)$ new graph is created. Then by adding (v_t, v_p) to the graph the realization of π is created. The proof is done! [4].

6-Havel-Hakimi Algorithm

Havel-Hakimi Algorithm is used to solve whether a simple graph for the given degree sequence exists or not and it constructs a deterministic solution if such a graph exists. Although the solution is deterministic, this simple graph can be used as a first step for other random graph generation algorithms. Other variants of the Havel-Hakimi algorithm have also been proposed to add randomness to this process [2]. Due to its recursive nature, the algorithm ends at most n iterations, each consisting of setting a component i to 0 and subtracting 1 from the d_i components [2]. In our implementation, the degree sequence is sorted after each iteration, and degree subtraction is done to the next d_i highest degree vertices other than i itself. The details of the implementation are explained in the following section.

6.1- Implementation Details

Input

Line 1: *degree sequence*

Output

If graphical: *list of edges*
If not graphical: *False*

The function takes a degree sequence as input. First, it checks whether sum of the degrees is even, if not, returns *False* (Handshaking Lemma). Then, in an infinite while loop, at each step, the

degree sequence is sorted in non-increasing order. If all the values in the degree sequence equal 0, it breaks the while loop and returns the edge list as a dictionary. Otherwise, it first checks whether the degree d_i of the highest degree vertex is greater than the number of the remaining highest d_i vertices. If this is the case, it returns *False*. Next, the degree of the next d_i highest degree vertices are subtracted by 1. If there is any vertex whose degree is smaller than 0, it returns *False*. At the end of the while loop, it updates the edge list and the degree of the highest degree vertex updated from d_i to 0. This while loop runs until all the degrees become zero or the degree sequence is found to be non-graphical (i.e., the algorithm returns *False*). Thus, this method can be used for graphicality test or generating simple graph with given degree sequence. This implementation is based on the Theorem 2 explained in [2].

7-The Configuration (Pairing) Model

In Pairing Model, each vertex is called as disjoint *cells*, each of which consists of d_i points. Then, the goal is to choose a perfect matching of these $\sum(d_i)$ points uniformly random; that is, at each step random two unmatched points are selected and matched. It can be observed that the matchings might lead to loops or multigraphs. In that case, the algorithm is re-run again until a simple graph is obtained. The algorithms works better when the degree of the vertices is relatively small, conversely, as degrees get higher the probability of having loops or multiple graphs approaches 1 very rapidly [2].

7.1- Implementation Details

Input

Line 1: *degree sequence*

Output

If graphical: *list of edges*
If not graphical: *False*

The function uses three helper functions, namely *random_select()*, *find_the_vertex()* and *Havel_Hakimi()*. The first function, random select, takes a list consisting of all possible points as input, and randomly choose two elements from the list and returns their values. For that, *sample* function is imported from *random* library in Python. This process refers to selecting two random points in the pairing model, as the name implies. The second function, find the vertex, takes a point and a degree sequence as input and returns its corresponding vertex as a result. Vertex information of a point is needed when deciding whether a loop or multigraph is formed in the matching process. The third function is basically used to decide whether the given degree sequence is graphical.

The *pairingModel()* function, takes the degree sequence as input. First, it checks whether the sum of the degrees is even. Then the Havel-Hakimi algorithm is called to check whether the given degree sequence is graphical, if not, it returns *False*. Before while loop, initialization step takes place. Unmatched points, set of edges, and adjacency matrix are initialized. Then the matching

process starts in an infinite while loop. First, two random points are selected from the unmatched points list and their corresponding vertices are labeled. Their adjacency matrix is also updated. Then, in an if statement, it is checked whether a multigraph occurred due to new matching. If it is the case, the process restarts, that is, all of the lists are initialized again. Else, the new edge is added to the set of edges list. In the second if statement, it is checked whether a loop is occurred after the matching. If it is the case, the process restarts similarly. The algorithm ends when the unmatched points list becomes an empty list, that is, a perfect matching is constructed.

8-The Sequential Algorithm

The sequential algorithm was coined by Joseph Blitzstein and Persi Diaconis. The algorithm always terminates in a realization and every realization has positive probability, although not uniformly distributed. Importance sampling techniques and some procedures can be used to speed up the implementation if desired. However, we implemented the basic sequential algorithm for random graph generation. Its pseudocode is given as follows by the authors [2]:

PSEUDOCODE OF THE SEQUENTIAL ALGORITHM

Input: a graphical degree sequence $(d_1, d_2, \dots d_n)$.
1. Let E be an empty list of edges.
2. If d = 0, terminate with output E.
3. Choose the least i with d_i a minimal positive entry.
4. Compute candidate list $J = \{j \neq i : \{i,j\} \notin E \text{ and } \Phi_{i,j} d \text{ graphical}\}$
5. Pick $j \in J$ with probability proportional to its degree d.
6. Add the edge $\{i,j\}$ to E and update d to $\Phi_{i,j} d$.
7. Repeat steps 4-6 until the degree of i is 0.
8. Return to step 2.

Output: E.

The proposed algorithm eliminates the possibility of loops and multiple graphs and checks graphicality at each step. Thus, it constructs a simple graph with given degree sequence in the end. The edges are connected with probability proportional to their degrees at each step. The details of our implementation are explained in the next section.

8.1- Implementation Details

Input

Line 1: *degree sequence*

Output

If graphical: *list of edges*
If not graphical: *False*

The function uses four helper functions, namely *Erdos_Gallai(), negative_update(), compute_candidate_list()* and *pick_j().* The first function, Erdos_Gallai, is used for graphicality

check as recommended by authors. The function is based on the Theorem in Section 5, and it takes degree sequence as input and returns *True* if the degree sequence is graphical, or *False* if it is not graphical. Negative update function takes degree sequences and two indices as input and subtracts 1 from each of the corresponding degrees. It returns new degree sequence. Compute_candidate_list function takes index i, edge list E, and degree sequence as input and returns candidate list of possible j values as output. The candidate list is determined identically to the candidate list set J at step 4 in the pseudocode. Pick_j function corresponds to the Step 5 in the pseudocode. It takes the candidate list j and the degree sequence as input and selects a j with probability proportional to its degree in d_j. It returns selected index as output.

In the sequential algorithm, first, it is checked whether the degree sum is even and calls Erdos_Gallai function to check if it is graphical. If not *False*,then, it initializes the adjacency matrix and edge set E. Then, in an infinite while loop, at each step, it chooses the least i with d_i a minimal positive entry. (Step 2 in pseudocode). In a second while loop the following steps are followed until the degree of i equals 0:

First, candidate list is created with the help of the compute_candidate_list function. Then, a vertex is selected with weighted probability of their degree values. The degree sequence is updated with the negative_update function. Finally, the edge list E, and adjacency matrix is updated. The while loop terminates if all the degree sequences equal 0. The function returns adjacency matrix as output.

9-Computational Experiments

Total of 85 test instances are generated to test the performance of the three graph generation algorithms. Instances 1-15 are generated such that the resulting degree sequence corresponds to 'scale-free graph' networks. 1-5, 6-10, and 11-15 each refer to the instances of 20, 50, and 100 vertex size, respectively. In the generation of scale-free graphs the 'preferential attachment' idea is implemented to ensure that some central / hub nodes (i.e., nodes having high vertex degree) are observed. Instances 16-45 refer to the random (Erdos-Renyi) graphs. Graphs are generated such that instances 16-30 have vertex degree mean 6, and instances 31-45 have vertex degree mean 4. K-regular graphs are generated with instances 46-60 having k=3 and 61-75 having k=6. Finally, non-graphical graphs are generated. Instances 76-82 refer to the graphs that satisfy Handshaking Lemma, but still non-graphical, whereas instances 83-85 refer to the graphs which do not satisfy Handshaking Lemma, that is, graphs whose degree sum is odd. Computational experiments were carried out using Python-Jupyter Notebook on a 8-GB RAM and M1-Processor. For implementation details, we refer to the Appendix section in the code file.

Each type of graphs (scale-free, random graphs with degree mean 6, random graphs with degree mean 4, 3-regular graphs, 6- regular graphs) having sizes of 20,50 and 100 are randomly generated five times and average run times of each algorithm are summarized in the Table-1 below. Detailed table can be found in the Appendix.

Table-1

Average Run-Time Results

Input Number	Graph Size	Graph Type	Average Run Times(s)		
			Havel-Hakimi	Pairing Model	Sequential Algorithm
1-5	20	Scale-free	0,000386286	>300	0,041732788
6-10	50	Scale-free	0,000688314	>300	1,504075432
11-15	100	Scale-free	0,00210309	>300	24,02740502
16-20	20	Random (mean 6)	0,000292969	27,3136651	24,02740502
21-25	50	Random (mean 6)	0,000613451	23,95663099	1,634712839
26-30	100	Random (mean 6)	0,001228857	60,43206036	24,26723218
31-35	20	Random (mean 4)	0,000209951	0,008150196	0,032924175
36-40	50	Random (mean 4)	0,000499201	0,103322029	1,094398594
41-45	100	Random (mean 4)	0,000850773	0,241214561	16,33385916
46-50	20	3-regular	0,000182772	0,000846672	0,029189348
51-55	50	3-regular	0,000505638	0,001438522	0,881642628
56-60	100	3-regular	0,000817585	0,015219307	12,71307478
61-65	20	6-regular	0,000366354	1,365354156	0,052604151
66-70	50	6-regular	0,000685644	4,258741045	1,690732431
71-75	100	6-regular	0,001142168	6,478450918	25,15077991

In all the instances, average run time is the smallest in Havel-Hakimi Algorithm. This is not surprising, since Pairing Model restarts whenever a matching results in loops and multi-edges. Similarly, Sequential Algorithm is more complex than Havel-Hakimi and it checks graphicality at each step when creating candidate list. However, one must realize that the resulting graph is deterministic in Havel-Hakimi Algorithm. Pairing model fails to generate a random graph in reasonable time for scale-free graphs in all three graph sizes. This might be because of hubs/central nodes, which increase the possibility of loops and multi-edge generation, thus restarting the algorithm for long times. Pairing Model performs better than Sequential Algorithm when the average degree is low, and the distribution is binomial/uniform. This can be seen from Table-1 above, in 3-regular and random (degree mean 4) graphs. In general, it can be concluded that the run times and variance of the pairing model increase significantly when the average vertex degree is high or the degree distribution follows the 'power law' property, rather than uniform.

10- Conclusion

Generating random graphs with a given degree sequence, or degree distribution, has been an important step in understanding today's complex real-world networks. Researchers from different domains have been studying complex networks and analyzing their properties for years. The random graphs generated with given degree sequence might be illuminating to understand the large-scale properties of those networks. Thus, researchers have developed some procedures and algorithms for that purpose.

In this project, three algorithms, namely Havel-Hakimi Algorithm, Pairing Model and Sequential Algorithm are implemented based on the paper “A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees” by Blitzstein and Diaconis. Experiment results showed that the Havel-Hakimi model runs relatively faster in all instances, unsurprisingly. Pairing model runs relatively faster with small sized graphs but might take more than 150 seconds to generate a random graph for some of the instances. Moreover, the running time of the Pairing Model exceeds 300 seconds in the generation of scale-free graphs, even the graph size is as small as 20. In that sense, Pairing Model is not as robust as the other two in terms of running time. Sequential Algorithm performs better than pairing model in most of the instances, but worse than Havel-Hakimi algorithm. This might be due to the graphicality checks at each step. It is important to mention that Sequential Algorithm consistently generates random graphs in acceptable times, whereas the run times in pairing model might get higher due to loops/multi-edges in some of the instances. For instance, in the instance-28, pairing model fails to find a graphical solution in 300 seconds whereas, sequential algorithm returns a random graph in 27 seconds. Although Havel-Hakimi performs the best, the resulting graph is deterministic, which might limit its area of usage. Besides having successful run-times on average, Sequential Algorithm has structural benefits, e.g., every realization has positive probability.

11- Appendix

Input Number	Graph Size	Graph Type	Run Times		
			Havel-Hakimi	Pairing Model	Sequential Algorithm
1	20	Scale-free	0,0002770	52,1914868	0,044700146
2	20	Scale-free	0,0002241	>300	0,030438900
3	20	Scale-free	0,0006142	>300	0,043054819
4	20	Scale-free	0,0006080	>300	0,049542189
5	20	Scale-free	0,0002081	>300	0,040927887
6	50	Scale-free	0,0009179	>300	1,584666729
7	50	Scale-free	0,0006161	>300	1,586340904
8	50	Scale-free	0,0005591	>300	1,417215824
9	50	Scale-free	0,0006912	>300	1,266504765
10	50	Scale-free	0,0006573	>300	1,665648937
11	100	Scale-free	0,0011272	>300	23,198739052
12	100	Scale-free	0,0048220	>300	25,107738018
13	100	Scale-free	0,0016918	>300	21,942027092
14	100	Scale-free	0,0016692	>300	22,826641083
15	100	Scale-free	0,0012052	>300	27,061879873
16	20	Random(mean 6)	0,0003152	1,1402512	0,046943903
17	20	Random(mean 6)	0,0002549	125,7955987	0,053472996
18	20	Random(mean 6)	0,0002859	0,8620470	0,044651031
19	20	Random(mean 6)	0,0002198	1,0724468	0,049616098
20	20	Random(mean 6)	0,0003891	7,6979818	0,051419020
21	50	Random(mean 6)	0,0006421	35,8147368	1,589771748
22	50	Random(mean 6)	0,0006399	64,2135839	1,610183001
23	50	Random(mean 6)	0,0006473	6,8425391	1,787675858
24	50	Random(mean 6)	0,0005600	11,3727789	1,726948738
25	50	Random(mean 6)	0,0005779	1,5395162	1,458984852
26	100	Random(mean 6)	0,0012851	165,6305890	25,017300129
27	100	Random(mean 6)	0,0013063	18,5110898	23,046193838
28	100	Random(mean 6)	0,0011120	>300	27,656367064
29	100	Random(mean 6)	0,0011311	19,2093599	22,047215939
30	100	Random(mean 6)	0,0013099	38,3772027	23,569083929

31	20	Random(mean 4)	0,0002038	0,0034781	0,032226801
32	20	Random(mean 4)	0,0001669	0,0093801	0,032667875
33	20	Random(mean 4)	0,0001709	0,0015779	0,032706261
34	20	Random(mean 4)	0,0003211	0,0024760	0,029303074
35	20	Random(mean 4)	0,0001869	0,0238390	0,037716866
36	50	Random(mean 4)	0,0004401	0,3172963	1,119650126
37	50	Random(mean 4)	0,0006108	0,0319901	1,137907028
38	50	Random(mean 4)	0,0004361	0,0489810	1,031039000
39	50	Random(mean 4)	0,0005422	0,1000278	1,086941957
40	50	Random(mean 4)	0,0004668	0,0183151	1,096454859
41	100	Random(mean 4)	0,0008380	0,3918991	17,334735155
42	100	Random(mean 4)	0,0008829	0,3593018	16,948484898
43	100	Random(mean 4)	0,0008690	0,0264180	14,849025965
44	100	Random(mean 4)	0,0009022	0,1591902	17,131420851
45	100	Random(mean 4)	0,0007617	0,2692637	15,405628920
46	20	3-regular	0,0001719	0,0006638	0,029135942
47	20	3-regular	0,0001531	0,0011272	0,028676987
48	20	3-regular	0,0001891	0,0005093	0,028622150
49	20	3-regular	0,0002120	0,0002990	0,029769421
50	20	3-regular	0,0001879	0,0016341	0,029742241
51	50	3-regular	0,0004599	0,0015907	0,886073112
52	50	3-regular	0,0004129	0,0015960	0,890457153
53	50	3-regular	0,0003881	0,0014789	0,890887022
54	50	3-regular	0,0008862	0,0015059	0,868184090
55	50	3-regular	0,0003810	0,0010211	0,872611761
56	100	3-regular	0,0008769	0,0064180	12,740639925
57	100	3-regular	0,0007267	0,0175700	12,712875843
58	100	3-regular	0,0008812	0,0272141	12,846214056
59	100	3-regular	0,0007281	0,0083327	12,642174959
60	100	3-regular	0,0008750	0,0165617	12,623469114
61	20	6-regular	0,0004098	1,4312460	0,052423000
62	20	6-regular	0,0003419	0,0857759	0,052829981
63	20	6-regular	0,0003941	0,5480380	0,051977873
64	20	6-regular	0,0003500	0,2804379	0,053101778
65	20	6-regular	0,0003359	4,4812729	0,052688122
66	50	6-regular	0,0006411	1,4587271	1,682643175
67	50	6-regular	0,0005221	7,8895550	1,688972950

68	50	6-regular	0,0006819	3,3870091	1,696132898
69	50	6-regular	0,0006940	2,3275781	1,691071033
70	50	6-regular	0,0008891	6,2308359	1,694842100
71	100	6-regular	0,0012050	3,4142902	25,259857178
72	100	6-regular	0,0010083	1,8068490	25,114775896
73	100	6-regular	0,0011709	1,7225971	25,204868793
74	100	6-regular	0,0011327	22,9498391	25,207875729
75	100	6-regular	0,0011940	2,4986792	24,966521978
76	20	Non-graphical(even)	0,0000629	0,0000350	0,000032902
77	20	Non-graphical(even)	0,0003500	0,0000830	0,000169992
78	20	Non-graphical(even)	0,0000312	0,0000303	0,000029802
79	20	Non-graphical(even)	0,0000420	0,0000379	0,000034809
80	20	Non-graphical(even)	0,0001800	0,0000379	0,000057936
81	20	Non-graphical(even)	0,0001569	0,0000329	0,000053883
82	20	Non-graphical(even)	0,0003219	0,0000482	0,000136852
83	20	Non-graphical(odd)	0,0000269	0,0000081	0,000009298
84	20	Non-graphical(odd)	0,0000589	0,0000131	0,000014067
85	20	Non-graphical(odd)	0,0001290	0,00001001	0,00001693

References

[1] Newman, M. E. J. (2003). The Structure and Function of Complex Networks. *SIAM Review*, *45*(2), 167–256.

[2] Blitzstein, J., & Diaconis, P. (2011). A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees. *Internet Mathematics*, *6*(4), 489–522.

[3] R., C., & Barabasi, A. L. (2008). Scale-free networks. *Scholarpedia*, *3*(1), 1716.

[4] Choudum, S. (1986). A simple proof of the Erdos-Gallai theorem on graph sequences. *Bulletin of the Australian Mathematical Society*, *33*(1), 67–70.