Random Graph Generation Methods, Models and Their Applications

Group 9:Deniz Sürmeli

09 June 2022

1 Introduction

Generation of random graphs is crucial for many industries. Areas such as biology, computer science, operations research and many more require simulations and testing for various studies. Thus, the structures and methods are heavily studied in academia and there are many existent methods and models used for generating random graphs.However they all have drawbacks on their terms such as their topology and runtime tradeoffs.

The paper written by Blitzstein and Diaconis [2] focuses on a new approach for the generation of randomized structures. It uses a sequential approach that utilizes importance sampling bounded by better time complexities and the generated graphs are more feasible, by that we mean in terms of self loops and multiple edges, compared to the other methods used. We can see that sequential algorithm does not offer known characteristics by the previous algorithms. While many of the known algorithms for generating random graphs either get stuck as the input size gets bigger or produce multiple edges and self loops. The algorithm differs in these aspects. It never gets stuck and has a low probability to produce self loops and multiple edges.In this paper, we will discuss the comparisons of algorithms, the methods they use and their characteristics with their applications for various industries.

The study also gives real life examples for the application of random graphs and their usage for the problems, thus the paper constantly motivates the reader to learn more about the subject. It shows several methods for specific classes of problems with their analysis, while showing some erroneous approaches with the existing methods. One can say that the paper is fully equipped with both theory and applications.

2 Definitions and Preliminaries

Before the discussion, we should give some definitions about the structures we will study.

Definition: A **graph** is a structure that is represented by $G = (V, E)$ where V is the set of vertices and E is the set of edges.

Definition: A **degree sequence** $d = (d_1, d_2, ..., d_n)$ is the sequence of numbers that represent the degrees of vertices. It maps v_i to d_i, where v_i is a vertex in the graph and d_i is the related degree of the vertex.

Definition: A **simple graph** is a graph $G = (V, E)$ such that $\forall v_i \in V, \nexists e \in E, e = (v_i, v_i)$ and $\forall e_1 \in E, \nexists e_2 \in E, e_1 = e_2$. That is, a graph G does not contain any self-loops or multiple edges between two vertices.

Definition: A **realization** of a degree sequence d is if d is graphical, then there exists a equivalence class C of graphs such that $\forall g \in C$, they have the degree sequence d.

Theorem(Handshake Lemma): Let $G = (V, E)$ be a graph. Let $|V| = n$ and $deg(v)$ denotes the degree of a vertex. Then,

$$\sum_{i=1}^{n} deg(v) = 2|E|$$

3 Erdos-Renyi Graphs

Erdos-Renyi Models are famous structures that are used in the generation of the random graphs. We will discuss them by starting with their definitions.

Definition(Erdos-Renyi Model-I): In the $G(n, M)$ model, a graph is chosen uniformly at random from the collection of all graphs which have n nodes and M edges. The nodes are considered to be labeled, meaning that graphs obtained from each other by permuting the vertices are considered to be distinct.[6]

Definition(Erdos-Renyi Model-II): In the $G(n, p)$ model, a graph is constructed by connecting labeled nodes randomly. Each edge is included in the graph with probability p, independently from every other edge. Equivalently, the probability for generating each graph that has n nodes and M edges is $p^M(1-p)^{\binom{n}{2}-M}$. The parameter p in this model can be thought of as a weighting function; as p increases from 0 to 1, the model becomes more and more likely to include graphs with more edges and less and less likely to include graphs with fewer edges. In particular, the case $p = \frac{1}{2}$ corresponds to the case where all $2^{\binom{n}{2}}$ graphs on n vertices are chosen with equal probability.[6]

Even though the definitions look complex, at their core they are very simple. For the $G(n, p)$ model, imagine that you layout n labeled vertices such as $v_1, v_2, ...v_n$. There are exactly $\binom{n}{2}$ possible pairings between them. For every possible pairing,let (v_i, v_j) be one of them, you flip a coin such that probability of constructing an edge between v_i and v_j is p while not constructing is $1 - p$. After you complete this process for every pair, the graph is generated.

There are several properties for the $G(n, p)$ model which are described by Erdos and Renyi[5] in their paper for precise values of n and p.

- For $np < 1$, a graph in $G(n,p)$ will almost surely have no connected components of size larger than $O(log(n))$.
- For $np = 1$, a graph in $G(n,p)$ will almost surely have a largest component whose size is of order $n^{2/3}$.
- For $np \to c > 1$,where c is a constant ,a graph in $G(n,p)$ will almost surely have a unique giant component containing a positive fraction of the vertices. No other component will contain more than $O(log(n))$ vertices.
- For $p < \frac{(1-\epsilon)ln(n)}{n}$, a graph in $G(n,p)$ will almost surely contain isolated vertices, and thus be disconnected.
- For $p > \frac{(1+\epsilon)ln(n)}{n}$, a graph in $G(n,p)$ will almost surely be connected.

Following these properties, we can see that $\frac{ln(n)}{n}$ is a sharp threshold for connectedness of $G(n,p)$

As intuition suggests, same probability of selection for every vertex is not feasible for every case. In situations you want a graph with certain characteristics such as *scale-free* graphs, which we will discuss them in the next section, the method might not be sufficient to satisfy the conditions. Also, when you try to generate large sets of data, graphs start to become similar with each other quickly, due to the nature of uniform probability functions. This might not be the best case when you try to obtain sets with high variance such as modeling social networks with high level of clustering.

4 Scale-free Graphs

Scale-free graphs are heavily studied in the network science. Many of the real world structures like banking system, software dependency graphs, social networks and many more examples can be represented by these structures. Before we examine the applications of them, we layout some definitions.

Definition: A **Scale-free graph** is a graph whose degree distribution obeys a power law.

Definition: A **Power Law** is a functional relationship between two quantities, where a relative change in one results in a proportional change in the other, independent of the initial size: one varies as a power of other. [12]

In plain words, a scale-free graph simply states that in some graphs, most of the edges are generated by pairings of vertices selected from a small subset of vertices of the vertex set.

We have mentioned about the *Erdos-Renyi* graphs previously. Recall that in $G(n,p)$ model where two nodes are chosen uniformly at a random and an edge generated between them. However, by the definition of scale-free graphs, *Erdos-Renyi* graphs would simply not be able to generate scale-free graphs due to their dependence of vertex-independent probability function. It has the same selection probability for every vertex, and expected graphs generated by the

model have degree sequences that has low variance, which the scale free graphs has the contrary case with high variance degree sequences. For generating a scale-free graph, every vertex should have it's own probability of being selected, which implies that the vertices with higher degree in the degree sequence should have higher probability of being selected to be an adjacent vertex to the edge that will be added to graph and vice versa. It's also called the *rich get richer* generative model. This is the central thought of *Barabasi* and *Albert*[1] model. In this method, they use *preferential attachment* method, which simply can be stated as for one quantity, it attracts more value as proportional to it's degree value[13].

There is also the **copy** model by *Kumar et al.*[9], a vertex selected at random and added to the current with a copy of a fraction of the current edges of the paired vertex. This method practically resamples the network by it's previous state, this resampling scales the highly connected vertices to remain highly connected with respect to the rest of the network thus the generated structure obeys power laws.

There are many more models for scale-free graph generation such as *Two-Level Network Model[7], MDA Model[14], Fitness Model[3].*

These models are used for various research topics and examining interactions in real world. For some examples, we can give *global banking system*, where countries that have superior systems in banking generate most of the volume in the network. Or in social events, where some attendants are more popular than other individuals in the crowd, thus they interact with more people compared to others. *World Wide Web* and some *intranets* used by private companies for intercommunication and coordination do obey to some power law also. All of these examples could be represented as a scale-free graph.

5 Usage of Random Graphs

Many of the real world events are not deterministic. However, many of the systems we build should have some sort of deterministic results or at least an approximation for what class of outputs we will encounter for various inputs that is provided to them. The process of generating various inputs obviously utilizes different techniques but for the systems that expects graphs as inputs, random graphs are essential for generating the set of inputs that we will stress our systems against.

In social networks, one would want to train their *machine learning* model with realistic inputs. Even though most of the training would be subject to some user-collected data, there might be a case for a researcher to not being able to obtain a sufficient data set. In order to increase the model's precision rate, one could generate random graphs that are similar to collected data and train the model against them.

In software testing, system crucial software used by sectors such as health, finance and infrastructure must test their software extensively. *Fuzzing*[10] is an important technique in the field, where the inputs are randomly generated

and software tested against the randomly generated inputs. The tester tries to achieve highest possible coverage, which is the percentage of executed parts of the source code. Also, randomly generated inputs can detect some edge cases that have been missed before. Software that expects graphs as inputs heavily utilizes random graphs by incorporating fuzzing methods.

In finance, high-frequency trading algorithms must be benchmarked extensively. Since time is such a crucial aspect of the system, it requires highest level of optimizations in terms of running time. Simulating arbitrage opportunities between several commodities and exchanges are highly important for benchmarking and detect the parts that the algorithm can be optimized is extremely important for building profitable systems.

In epidemics, researches can simulate and try to predict the spread rate by generation of random graphs. If your generated models are accurate, early intervention of the individuals that spread the disease and accessing the contacted individuals help greatly to slow down the spread rate.

In computer networks, most of the systems desire higher uptimes, that is being accessible all of the time. If one can simulate the possible connections that can arise and build scale tiers for their systems ahead, the time for rescaling,adding more computing power to their services, their servers could be minimized and uptime of the network would be improved. Also reducing the cost of using unnecessary computing power help them to become more profitable.

Overall, some of the problems we work on involve graphs. The methods that we use to solve these problems need testing and simulation for better results, and generation of random graphs are crucial for determining the correctness of the solutions them and detecting optimizations for the methods.

6 Erdos-Gallai Graphicality Criterion and It's Proof

Erdos-Gallai Theorem is very important for generating random graphs that result in simple graphs. It's crucial to check some degree sequence is actually *graphical* before try to generate a graph according to it. Let's start by some definitions and theorems.

Theorem(Erdos-Gallai[11]): A sequence of non-negative integers $d_1 \geq d_2 \geq ... \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if $\sum_{i=1}^{n} d_i$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} min(d_i, k), \forall k, 1 \leq k \leq n$$

The proof given by Choudum[4] is a proof based on induction and it checks for five cases. One can think that given the previous sequence is graphical, adding an edge to some graph represented in the class of the sequence, proof shows that after increasing two vertices degree by one, following some constraints

which generate the cases, the updated sequence still complies with the inequality or not.

7 Implemented Algorithms and Experimental Results

We have implemented three algorithms for benchmarking. A pairing model based, the sequential algorithm described in the paper by Blitzstein and Diaconis[2] and a Havel-Hakimi variant. The implementations for these algorithms can be found in the bibliography[15].

7.1 Pairing Model

In this method of generation, implementation is very simple.

Input: A degree sequence $d = (d_1, d_2, d_3, ..., d_n)$ of size n.
1. Start by generating an auxiliary list s such that $\forall v_i \in d$, append exactly d_i of v_i to s.
2. Shuffle s.
3. Divide s into two sublists s_α and s_β.
4. Add an edge e to graph G such that $\forall i \in range(|s_\alpha|), e = (s_\alpha.at(i), s_\beta.at(i))$.

Output: A graph G with a degree sequence d.

In simple words, you put the label of a vertex to a bag k times where k is the degree of that vertex described in the degree sequence. You shake the bag and divide into exactly two bags. You start to pick one each of them, each round you add an edge to the vertices that have been selected. In the end, you perform a bipartite perfect matching between two bags.

7.2 Sequential Model

The exact definition of the algorithm can be found in the paper by Blitzstein and Diaconis[2]. At it's core, it utilizes *Importance Sampling* [8] technique, which is a variance reducing method for Monte-Carlo experiments. Here we will just give a verbal explanation. Given a degree sequence, assuming that it is sorted in descending order, you pick the highest index non-zero vertex.You match them with the vertices that have high probability of being selected, that is their degrees are higher compared to other vertices that are non-selected. Then you check that whether the candidates selected do not break the graphicality criterion of the degree sequence. If not, you repeat until you are left with zero-filled degree sequence. Else, you repick the candidates with their given probabilities.

7.3 Havel-Hakimi Model

The Havel-Hakimi Theorem[2] has a constructive definition. By using the definition, we can realize the sequence.

Input: A degree sequence $d = (d_1, d_2, ..., d_n)$. of size n.
1. Select highest index i with $d_i > 0$. Assume that there are d_i other vertices we can reduce their degrees by 1 since d is assumed to be graphical.
2. Reduce 1 starting from the lowest index j to k such that $k - j = i$.
3. If $d' = 0$, halt the algorithm.
4. Check the new sequence d' is graphical, if so go to *step 2.*
5. Else return an error.

Output: A graph G with degree sequence d.

7.4 Experimental Results

Implemented algorithms are benchmarked with randomly generated degree sequences with varying sizes such that maximum of length 100, which every size had 5 samples thus every result is the average of 5 runs. Results are represented in the table. There are several warnings the reader should be aware of. First of all, every algorithm checks whether the degree sequence is graphical or not using the Havel-Hakimi test in their first step. Thus, the non-graphical sequences are not represented in the table. Second, we utilize circuit breakers in Pairing Model, value -1 in the value implies that retries have not been successful and it has failed to generate a simple graph. Lastly, we don't check for self-cycles or multiple edges in Havel-Hakimi generator since it's a deterministic algorithm, if there is a rule-breaker for that sequence, it will be present no matter how many times we will try. The results are represented in milliseconds.

As reader can observe from the table, Havel-Hakimi algorithm is substantially faster than its competitors, but the reader should be aware of the fact that Havel-Hakimi implementation do not check for cycles or self-edges, thus it fails most of time even though it's fast. This is the reason why it's so fast compared to the other algorithms, it's deterministic nature executes once and concludes whether it will generate a simple graph or not. On the contrary side, one of the biggest reasons that the Pairing Model is such a slow algorithm compared to others is the fact that it recursively retries the generation process when it detects a cycle or a self-edge, but as the input size gets bigger, the probability of never producing a simple graph approximates to 1. In general, executing it on a regular PC, it starts to fail frequently in cases where n is greater than 10. Thus for larger values of n, we have implemented the algorithm using a circuit breaker with stack depth of 24. Higher stack depths again consumed too much computing power and memory, we have found that 24 is the most number of chances we can give to the pairing model.

Sequential Algorithm has properties such that while it is relatively performant in terms of time compared to Pairing Model, it generates graphs that are feasible most of the time. By feasible, we imply that the graphs that do not contain frequent multiple edges or self-edges. Scaling wise, Pairing Model fails due to it's recursive constraint of checking criterion for simple graphs and retrying, while Havel-Hakimi is the most scalable algorithm thanks to it's deterministic structure and Sequential Algorithm stays between them in terms of the comparison. Overall, Sequential Algorithm can be greatly utilized compared to the others since it can generate better structured graphs, by that we mean simple or quasi-simple, in acceptable times.

Input Size	Havel-Hakimi	Pairing Model	Sequential Algorithm
0	24	107.0	21.0
1	21	224.0	38.0
2	23	69.0	22.0
3	34	191.0	156.0
4	38	208.0	136.0

Input Size	Havel-Hakimi	Pairing Model	Sequential Algorithm
5	31	237.0	176.0
6	53	548.0	516.0
7	42	819.0	676.0
8	78	2177.0	1610.0
9	104	3310.0	2255.0
10	269	4460.0	2434.0
11	87	3111.0	3404.0
12	109	6002.0	4246.0
13	89	7705.0	4433.0
14	105	11064.0	7433.0
15	167	17191.0	10951.0
16	127	18860.0	11144.0
17	145	25727.0	18432.0
18	169	23909.0	15204.0
19	194	23133.0	21975.0
20	195	24395.0	19835.0
21	214	47740.0	36517.0
22	283	54799.0	45360.0
23	301	74023.0	58892.0
24	305	72281.0	61832.0
25	234	75021.0	54834.0
26	530	132721.0	97073.0
27	336	69285.0	80720.0
28	401	177097.0	150593.0
29	506	144668.0	133510.0
30	675	239406.0	194677.0
31	464	162590.0	145246.0
32	457	189550.0	170687.0
33	441	214842.0	229066.0
34	540	279373.0	288165.0
35	596	314207.0	357984.0
36	651	330670.0	371493.0
37	778	450754.0	525758.0
38	643	529885.0	600765.0
39	600	419209.0	424536.0
40	656	539253.0	572447.0
41	709	452258.0	483246.0
42	1121	801148.0	798440.0
43	1618	685274.0	936651.0
44	804	656405.0	734994.0
45	887	703300.0	999391.0
46	801	958081.0	1050550.0
47	793	1054700.0	1251430.0
48	1014	1130690.0	1359820.0
49	874	973021.0	1419540.0

Input Size	Havel-Hakimi	Pairing Model	Sequential Algorithm
50	849	1324700.0	1459370.0
51	907	1348850.0	1772860.0
52	1025	1041470.0	1350540.0
53	919	1581640.0	1975590.0
54	939	1443070.0	1961460.0
55	1476	2399130.0	3001070.0
56	972	1612280.0	2613250.0
57	1430	2026030.0	3202230.0
58	1176	2444010.0	3521060.0
59	1113	1748460.0	2899330.0
60	1272	2231940.0	3329820.0
61	1249	2203240.0	3024970.0
62	1367	2928120.0	3662760.0
63	1690	3037960.0	4503860.0
64	1271	3079010.0	6325450.0
65	2214	3878560.0	5285100.0
66	1621	3574840.0	4791080.0
67	1300	3269320.0	4483330.0
68	1490	4590920.0	6304970.0
69	1580	4529600.0	6676380.0
70	1644	4505630.0	7595370.0
71	1748	4791580.0	8299180.0
72	2026	5047230.0	8600180.0
73	1823	4989090.0	10693900.0
74	1917	4336050.0	8949820.0
75	1755	5085610.0	9042450.0
76	1934	6952500.0	11560100.0
77	1907	7914260.0	11316100.0
78	2172	6075700.0	12770800.0
79	2075	6608220.0	9872720.0
80	1853	7950230.0	12907600.0
81	2042	6815340.0	11569300.0
82	2438	7392990.0	14291400.0
83	2508	9998580.0	17582100.0
84	2372	11446400.0	19672000.0
85	2348	11554200.0	18113200.0
86	3240	11071000.0	16646600.0
87	3591	10556100.0	18661000.0
88	2704	10264100.0	19213600.0
89	2518	14560700.0	26156500.0
90	2592	10192400.0	23074500.0
91	2622	15731200.0	27480500.0
92	2482	13043000.0	22670300.0
93	2752	12003700.0	20878800.0
94	2893	16636100.0	31219400.0

Input Size	Havel-Hakimi	Pairing Model	Sequential Algorithm
95	3009	15439300.0	28281200.0
96	4511	15052000.0	30321500.0
97	2982	15780100.0	32829600.0
98	3313	18837700.0	32636600.0
99	3515	23666400.0	41346100.0

8 Discussion

We have seen that the new approach proposed by Blitzstein and Diaconis[2] is a relief for generating simple graphs compared to the existing methods. Since the generation of random algorithms are heavily needed on many fields, the Blitzstein and Diaconis's [2] work can be modified to produce some desired topologies for various fields. Thus, the work is extremely important to build better products and services while ensuring that they have desired characteristics by extensive testing. Also the authors effort to show the real-world applications by applying the algorithm and constructing models to them is very motivating for the reader, one can say that they wrote an usage manual for their algorithm.

Bibliography

[1] Reka Albert and Albert-Laszlo Barabasi. "Statistical mechanics of complex networks". In: *Reviews of Modern Physics* 74.1 (Jan. 30, 2002), pp. 47–97. ISSN: 0034-6861, 1539-0756. DOI: 10.1103/RevModPhys.74.47. arXiv: cond-mat/0106096. URL: http://arxiv.org/abs/cond-mat/0106096 (visited on 05/21/2022).

[2] Joseph Blitzstein and Persi Diaconis. "A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees". In: *Internet Mathematics* 6.4 (Mar. 9, 2011), pp. 489–522. ISSN: 1542-7951, 1944-9488. DOI: 10.1080/15427951.2010.557277. URL: http://www.internetmathematicsjournal.com/article/1494 (visited on 05/12/2022).

[3] G. Caldarelli et al. "Scale-Free Networks from Varying Vertex Intrinsic Fitness". In: *Physical Review Letters* 89.25 (Dec. 3, 2002), p. 258702. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.89.258702. URL: https://link.aps.org/doi/10.1103/PhysRevLett.89.258702 (visited on 06/04/2022).

[4] S.A. Choudum. "A simple proof of the Erdos-Gallai theorem on graph sequences". In: *Bulletin of the Australian Mathematical Society* 33.1 (Feb. 1986), pp. 67–70. ISSN: 0004-9727, 1755-1633. DOI: 10.1017/S0004972700002872. URL: https://www.cambridge.org/core/product/identifier/S0004972700002872/type/journal_article (visited on 06/06/2022).

[5] P Erdős and A Rényi. “ON THE EVOLUTION OF RANDOM GRAPHS by”. In: (), p. 45.

[6] *Erdős–Rényi model*. In: *Wikipedia*. Page Version ID: 1087229876. May 11, 2022. URL: https://en.wikipedia.org/w/index.php?title=Erd%C5%91s%E2%80%93R%C3%A9nyi_model&oldid=1087229876 (visited on 05/20/2022).

[7] Kamrul Hassan and Liana Islam. “Growing Scale-free Networks by a Mediation-Driven Attachment Rule”. In: *Physica A: Statistical Mechanics and its Applications* 469 (Mar. 2017), pp. 23–30. ISSN: 03784371. DOI: 10.1016/j.physa.2016.11.001. arXiv: 1411.3444[cond-mat,physics:physics]. URL: http://arxiv.org/abs/1411.3444 (visited on 06/04/2022).

[8] *Importance sampling*. In: *Wikipedia*. Page Version ID: 1091428930. June 4, 2022. URL: https://en.wikipedia.org/w/index.php?title=Importance_sampling&oldid=1091428930 (visited on 06/04/2022).

[9] R. Kumar et al. “Stochastic models for the Web graph”. In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. 41st Annual Symposium on Foundations of Computer Science. Redondo Beach, CA, USA: IEEE Comput. Soc, 2000, pp. 57–65. ISBN: 978-0-7695-0850-4. DOI: 10.1109/SFCS.2000.892065. URL: http://ieeexplore.ieee.org/document/892065/ (visited on 05/21/2022).

[10] Barton P. Miller, Mengxiao Zhang, and Elisa R. Heymann. “The Relevance of Classic Fuzz Testing: Have We Solved This One?” In: *IEEE Transactions on Software Engineering* (2020), pp. 1–1. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2020.3047766. arXiv: 2008.06537[cs]. URL: http://arxiv.org/abs/2008.06537 (visited on 05/21/2022).

[11] Turán Pálnak. “Gráfok előírt fokú pontokkal”. In: (), p. 11.

[12] *Power law*. In: *Wikipedia*. Page Version ID: 1084603641. Apr. 25, 2022. URL: https://en.wikipedia.org/w/index.php?title=Power_law&oldid=1084603641 (visited on 05/21/2022).

[13] *Preferential attachment*. In: *Wikipedia*. Page Version ID: 1073702575. Feb. 24, 2022. URL: https://en.wikipedia.org/w/index.php?title=Preferential_attachment&oldid=1073702575 (visited on 06/04/2022).

[14] Erzsebet Ravasz and Albert-Laszlo Barabasi. “Hierarchical Organization in Complex Networks”. In: *Physical Review E* 67.2 (Feb. 14, 2003), p. 026112. ISSN: 1063-651X, 1095-3787. DOI: 10.1103/PhysRevE.67.026112. arXiv: cond-mat/0206130. URL: http://arxiv.org/abs/cond-mat/0206130 (visited on 06/04/2022).

[15] Deniz Surmeli. *Random Graph Generators*. original-date: 2022-04-29T10:50:24Z. May 19, 2022. URL: https://github.com/denizsurmeli/random-graph-generator (visited on 05/21/2022).