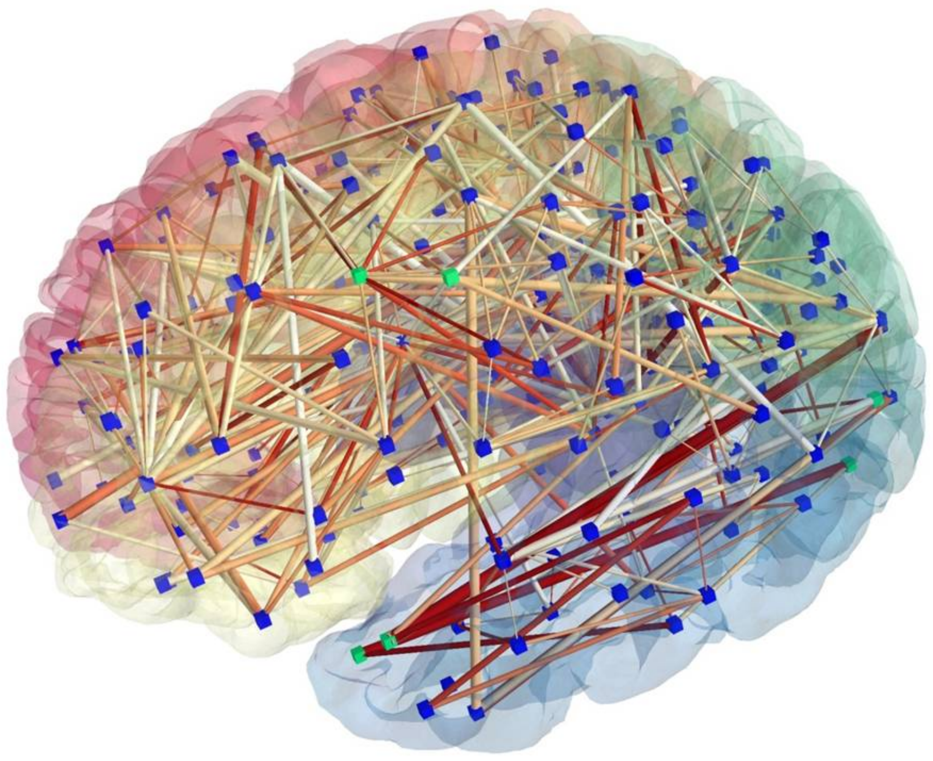
CHALMERS
UNIVERSITY OF TECHNOLOGY



Generation of Random Graphs for Graph Theory Analysis Applied to the Study of Brain Connectivity

Master's thesis in Complex Adaptive Systems

Oleksii Bielikh

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017:NN

Generation of Random Graphs for Graph Theory Analysis Applied to the Study of Brain Connectivity

OLEKSII BIELIKH

Department of Physics
Soft Matter Lab
Chalmers University of Technology
Gothenburg, Sweden 2017

Generation of Random Graphs for Graph Theory Analysis Applied to the Study of Brain Connectivity
OLEKSII BIELIKH

Supervisor: Giovanni Volpe, Department of Physics, University of Gothenburg
Examiner: Giovanni Volpe, Department of Physics, University of Gothenburg

Master's Thesis 2017:NN
Department of Physics
Soft Matter Lab
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The schematic of a brain connectivity graph.
http://integratedlistening.com/blog/2016/02/03/brain-connectivity-a-marker-of-sensory-processing-disorder/

Typeset in LaTeX
Gothenburg, Sweden 2017

Abstract

One of the current frontiers in neurosciences is to understand brain connectivity both in healthy subjects and patients. Recent studies suggest that brain connectivity measured with graph theory is a reliable candidate biomarker of neuronal dysfunction and disease spread in neurodegenerative disorders. Widespread abnormalities in the topology of the cerebral networks in patients correlate with a higher risk of developing dementia and worse prognosis.

In order to recognize such abnormalities, brain network graph measures should be compared with the corresponding measures calculated on random graphs with the same degree distribution. However, creating a random graph with prescribed degree sequence that has number of nodes of magnitude of 10^5 is a recognized problem. Existing algorithms have a variety of shortcomings, among which are slow run-time, non-uniformity of results and divergence of degree distribution with the target one.

The goal of this thesis is to explore the possibility of finding an algorithm that can be used with very large networks. Multiple common algorithms were tested to check their scaling with increasing number of nodes. The results are compared in order to find weaknesses and strengths of particular algorithms, and certain changes are offered that speed up their runtimes and/or correct for the downsides. The degree distributions of the resulting random graphs are compared to those of the target graphs, which are constructed in a way that mimics some of the most common characteristics of brain networks, namely small-worldness and scale-free topology, and it is discussed why some of the models are more appropriate than others in this case. Simulations prove that the majority of algorithms are vastly inefficient in creating random large graphs with necessary limitations on their topology, while some can be adapted to showcase to a certain extent promising results.

Contents

List of Figures

1

Chapter 1

1.1 Brain networks and graph theory

Brain connectivity is an important subject of study for neuroscience. Measured with graph theory, it becomes a biomarker of neuronal dysfunction and spread of neurodegenerative disorders such as Alzheimer's [1], Parkinson's [2], and others. Prognosis becomes worse the more pronounced the abnormalities and disruption in connectivity patterns in patients are.

Brain connectivity, same as social networks, transportation, linguistics etc., can be studied efficiently with the help of graph theory. Representing elements of networks with vertices and connections with edges, a graph is created, various measures of which can be studied later in order to learn more about the underlying system. In case of a brain network, the vertices represent either the individual neurons, or the anatomical or functional areas of a brain, while edges showcase the connections between those (synapses or presentations of statistical dependencies between those areas).

It is useful to inspect the graph by looking at the adjacency matrix, whose elements can be either 0 (meaning the pairs of vertices are adjacent) or 1 (not adjacent). Brain networks are derived from anatomical data, computer simulations or other methods often yield matrices with elements different from 0 and 1, and afterwords thresholded to produce a binary network [3]. The resultant matrices can be symmetric or non-symmetric, representing undirected or directed graphs correspondingly. We are going to mostly focus on undirected graphs which are more important to the model we are investigating. Also, we are going to be working with simple graphs, i.e. such that have no duplicate and self-edges, in other words, no vertex is connected to itself by an edge and has at most 1 edge connecting it to any other vertex.

One of the more commonly used and important measures of graphs is the degree distribution. The degree of a vertex is the number of edges attached to it, and in case of directed graphs there is an in-degree and an out-degree. The degree distribution is the probability distribution of such degrees over the network. This is not to be confused with degree sequence, which might be thought of as a realization of a degree distribution, a non-increasing sequence of degrees of vertices of a specific graph. To find out whether the network has certain irregularities it should be compared to random graphs with the same degree distribution.

Brain networks are complex, which means they display both regularity and randomness. This combination might manifest in different ways [4], and there exist a lot of different random graph models. The Erdős-Rényi model can perhaps be called the base one for creating random graphs. It can be applied to basically any kind of graphs, including directed and non-directed, with self-loops or without them and so on. Within this model, a random graph is picked uniformly and randomly from the set of all graphs with n nodes and M edges ($G(n, M)$). A closely related model, proposed by Edgar Gilbert, involves connecting n nodes with independent probability p between each other ($G(n, p)$). The features that are present in brain networks, however, are inhomogenous degree distribution [5] and a specific connectivity pattern on a local level [6]. These are referred to, correspondingly, as scale-free and small-world networks.

In the case of scale-free networks [7] a small number of vertices have many more connections and are called hubs. They are extremely important to network functioning and should they be removed, the network functionality would be disrupted. At the same time, deletion of less important nodes does not influence the network nearly as much, and this was shown to be true for brain networks. The degree distribution for scale-free networks follows a power law (although it is argued that for the brain it actually is an exponentially truncated power law, which is less vulnerable to attack on specific nodes [8]).

The small-world topology [9] is similar to random networks in terms of degree distribution shape - it resembles bell curve. The network with such topology is inhomogeneous because nodes that are adjacent to a certain node are also highly likely to be adjacent to each other, vastly increasing clustering coefficient (the likelihood of such adjacency) compared to the Erdős-Rényi model. Such architecture allows areas in the brain that are functionally similar to be densely interconnected.

1.2 General idea of the research

Data for creating graphs for brain networks is often gathered from MRI, fMRI, EEG etc. In order to construct a graph from this data we use the BRAPH package [10]. In addition to obtaining the brain connectivity graphs, the package also assess the graph structure and tests for differences between the groups. Finally, it normalizes the network measures by random graphs. However, the networks in question tend to be very large, with 10^5 vertices and more. As it turns out, creating a random graph with a prescribed degree distribution for comparison takes a very long time with naive algorithms and implementations, and creating a random graph according to Erdős-Rényi model is not sufficient because of heterogeneity between the nodes. Hence there is need for creating a suitable algorithm that would not slow down the work of the whole package, while at the same time keeping the degree distribution of created random graph close to the given one (obtained from clinical data) and picking a candidate graph uniformly among all the possible ones, which is the aim of this thesis.

The work is carried out on the following way: first, a graph with a large number of nodes is created, whether completely randomly (e.g. by Erdős-Rényi model), or with scale-free or small-world characteristics in order to resemble the actual brain networks. There are certain mechanisms for creating these, e.g. small-world topology can be created by applying the Watts-Strogatz mechanism, and scale-free network can be implemented by consecutively adding nodes to the graph and using preferential attachment (one of the most basic models is the Barabasi-Albert model). Then, an algorithm is tested, that is supposed to create a random graph with a degree distribution that would be close to the original graph. Runtime is averaged over several runs, and also for different sizes of the original graph in order to learn how it scales with increasing number of nodes. Finally, a degree distribution of the original graph and the created ones are compared to test how different they are.

2

Chapter 2

2.1 Random graphs

The first step is to create a graph that is based on the Erdős-Rényi model, i.e. completely random with exception for the number of nodes n and probability of edge inclusion p. There are quite a few algorithms that aim to create random graphs but the baseline ones scale poorly for large networks, hence many modifications and improvements are proposed, for example, in [11]. We are going to explore some of them, for now focusing on the average run-time and the ways to improve it.

2.1.1 Baseline algorithm (ER)

The most basic idea for creating a $G(n, p)$ graph is to go through every possible edge and include it with probability p, which can be accomplished by creating two loops.

Algorithm 1 ER

```
1: $G(n,p) = \varnothing$
2: for $i = 1$ to $n$ do
3:     for $j = 1$ to $n$ do
4:         Generate uniform random number $\theta \in [0, 1)$
5:         if $\theta < p$ then
6:             $G(n,p) := G(n,p) + (i,j)$
```

This algorithm is really basic and can be improved in many ways, some of which will be demonstrated here. The others will be left out. For example, it is rather simple to use a single loop instead of two without changing the main idea. Also we are deliberately leaving parallel variations out which are out of the scope of this work.

2.1.2 ZER

The central point in this algorithm is skipping the generation of random numbers for some edges. Authors argue [11] this is similar to the process in Z Reservoir algorithm hence the name ZER. Since not all edges are connected and number of skipped edges is geometrically distributed with parameter p, computationally ZER is expected to be faster than the base ER, however due to logarithmical computations this is not always the case. E is maximum number of possible edges in a graph.

Algorithm 2 ZER

```
1: $G(n,p) = \varnothing$
2: $i = -1$
3: while $i < E$ do
4:     Generate uniform random number $\theta \in [0,1)$
5:     Skip value $k = \max(0, \lceil \log_{1-p} \theta \rceil - 1)$
6:     $i := i + k + 1$
7:     $G(n,p) := G(n,p) + e_i$
8: Discard the last edge
```

e_i is an 'encoded' edge. This is done in order to have one loop and one index, thus vastly decreasing computational time. It can be decoded into more usual indices i and j in the following way:

$$\begin{aligned} i &= \lfloor ind/v \rfloor \\ j &= ind \mod v \end{aligned}$$

where ind is the single index and v is the number of edges in the graph.

2.1.3 PreLogZER

This algorithm becomes computationally more efficient than the previous one, ZER, by pre-calculating all the logarithm values. However, the benefits are obvious only to a certain extent; if the random numbers that are used in the process are of higher precision (more than 16-bit) or graphs are not large enough, this variation might actually perform worse. However the idea of pre-calculating certain operations before the main loop might be important in the following work.

Algorithm 3 PreLogZER

```
1: $G(n,p) = \varnothing$
2: $c = \log(1-p)$
3: for $i = 1$ to $RANDMAX$ do
4:     $\log(i) = \log(i/RANDMAX)$
5: $i = -1$
6: while $i < E$ do
7:     Generate uniform random number $\theta \in [0, RANDMAX)$
8:     Skip value $k = \max(0, \lceil \frac{\log(\theta)}{c} \rceil - 1)$
9:     $i := i + k + 1$
10:    $G(n,p) := G(n,p) + e_i$
11: Discard the last edge
```

There is one more version of this algorithm, PreZER, which is based on avoiding logarithmic computation whenever possible at all, by pre-computing the breakpoints of cumulative distribution function of skip value $F(k)$. This lends much better results however since the focus of the work is different we will not go into detail here.

2.2 Random graphs with given degree distribution

Simply generating random graphs as was described in the previous part is not enough. The random graph needs to have the same, or a rather close, degree distribution as the one that is being investigated, and as was mentioned in chapter 1, brain networks tend to exhibit characteristics more complex than simple random graphs of Erdős-Rényi model.

One way to solve the task is to focus on getting the exact (or close) degree sequence d_i. The other approach is to get a correct degree distribution that converges to the one of the original graph as number of nodes goes to infinity [15]. Considering that the graphs we are investigating have thousands of nodes this approach would lead to feasible results. One of the algorithms that follows this idea first creates a directed graph based on the degree distribution of the original graph, and then removes directions.

Algorithm 4 Directed graph with removed directions (DGRD)

```
1: Create a truncated version of the original distribution $G$, $G_n$, where $g_k^n = g_k$ for
   $k = 0, 1, ..., n-2$, $g_k^n = \sum_{k=n-1}^{\infty} g_k$ for $k = n-1$ and $g_k^n = 0$ for $k \geq n$
2: For each vertex $v_i$ get a random variable $Y_i$ from the distribution $G_n$.
3: Pick $Y_i$ random vertices to be hit, independently for each vertex (at this point
   we have a directed graph).
4: Remove directions from the graph by fusing edges together - if at least one
   directed edge is present between two nodes make it undirected.
```

This algorithm shows very fast running times, with mean values being around 1.71s for $p = 0.01$, 2.74s for $p = 0.05$ (for the graph with 10000 nodes). However this algorithm works reliably only for certain distributions, e.g. mixed Poisson. In our tests it also ran fine for random distribution, although for it the 4th step (removal of directions) had to be skipped. It also gives better results than ER or ZER for more complicated original graphs.

2.3 Results

In order to check and compare the performance of various algorithms, first a random graph is created with a degree distribution that will be the target one. The base model again is Erdős-Rényi, meaning each edge is included with predetermined probability. However in order to mimic the brain networks we will introduce two other 'methods' of creating the original, or target graph, related to small-world and scale-free characteristics. Of course, ideally the brain network would have features of all of them simultaneously and we do not know the particular parameters for modelling these, but it still might show which algorithms show more promise than others.

In order to recreate the small-world network behaviour [9] the Watts-Strogatz mechanism can be used. Three parameters need to be specified prior to the creation of such network, namely number of nodes N, average probability of edge inclusion (similar to ER graphs) p or average degree K, and finally a rewiring probability q. At first, a regular lattice is constructed, where each of N nodes is connected to K neighbours. After that, for every node n_i every edge (n_i, n_j) is taken where $i < j$ and it is rewired with a probability q: the edge is changed to (n_i, n_k), with n_k being a random node chosen uniformly so that no multiple or self-edges are created.

For scale-free networks the existing algorithms tend to be too slow once the number of nodes becomes large. The important characteristic of scale-free behaviour however is rather easy to mimic: it is power law distribution of node degrees, i.e. $P(k) = k^{-\gamma}$, where γ is usually picked in range from 2 to 3. Therefore we create a graph that has a degree sequence sampled from power law distribution with appropriate choice of γ.

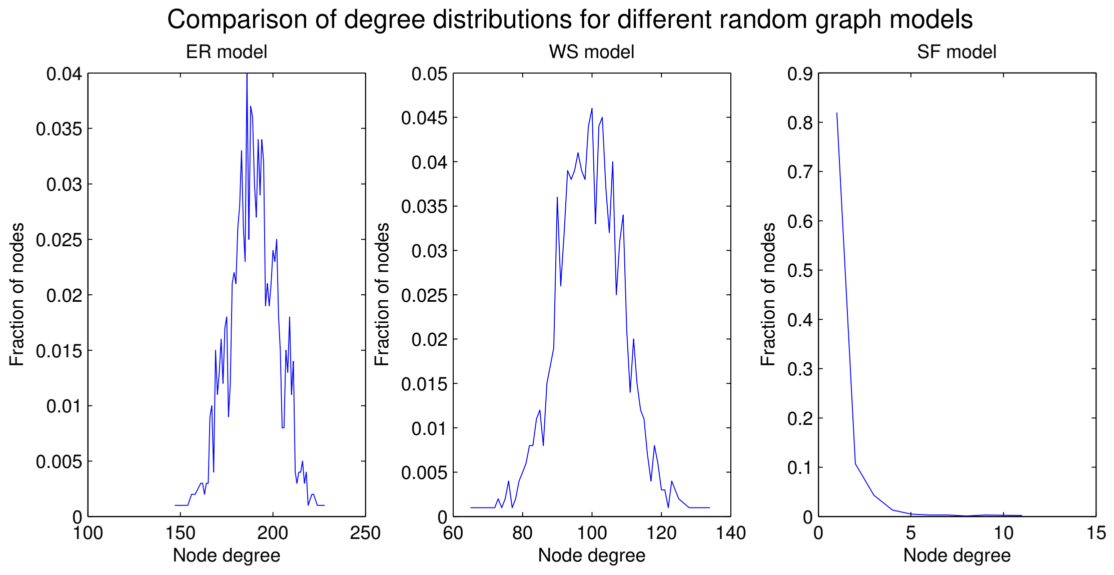


Figure 2.1: Comparison between degree distributions of random graphs produced with various characteristics, left to right: ER model, small-world network, scale-free network. Each network has 1000 nodes, p=0.1 for ER and small-world network, q=0.1 for small-world network. Shape of small-world network degree distribution curve resembles that of ER model, being somewhat sharper

2.3.1 Running times of algorithms

All algorithms showed comparable average running times, with running time increasing almost linearly in log-log representation. There is much more variance in running times for graphs with small number of nodes. Original graph was chosen as ER, but it does not affect average running times of algorithms. On the plot you can see comparison between running times of ER, ZER and DGRD.

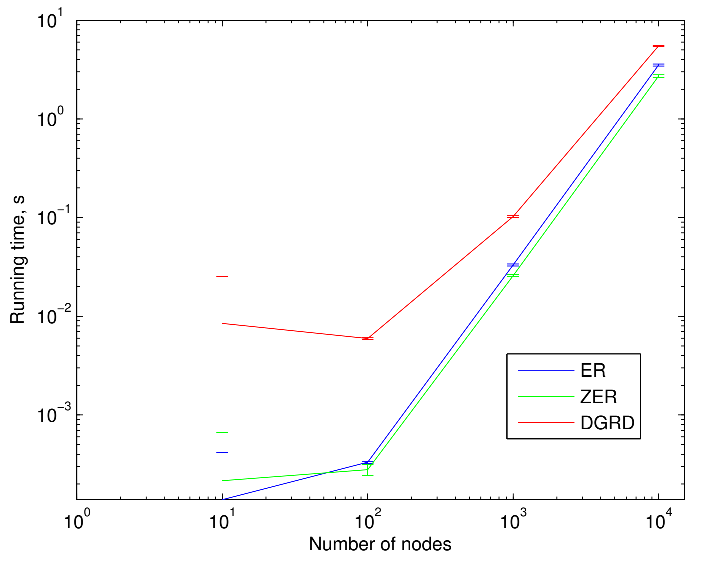


Figure 2.2: Running time of ER, ZER and DGRD algorithms with error bars, for number of nodes 10, 100, 1000 and 10000. Results are averaged over 5 runs. p=0.25

2.3.2 Results of algorithms for original ER graph

Here (and in every following similar plot) the target degree distribution is shown in red. All algorithms perform quite well since the original graph is the simplest to reproduce.

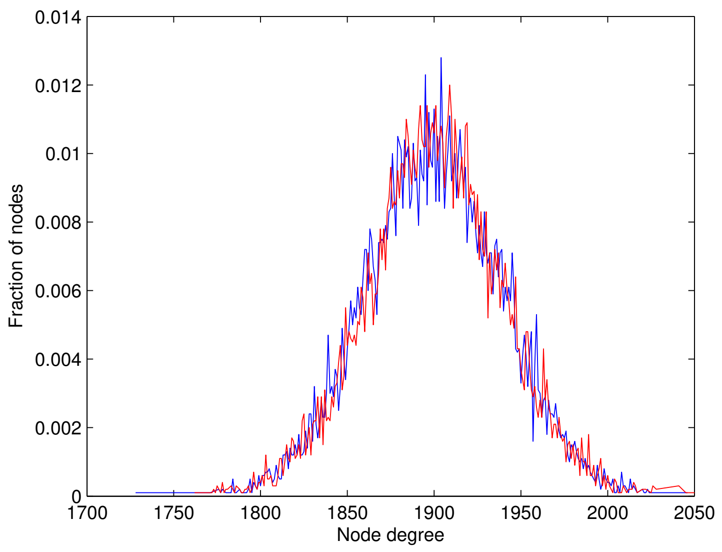


Figure 2.3: Degree distribution for ER, 10000 nodes

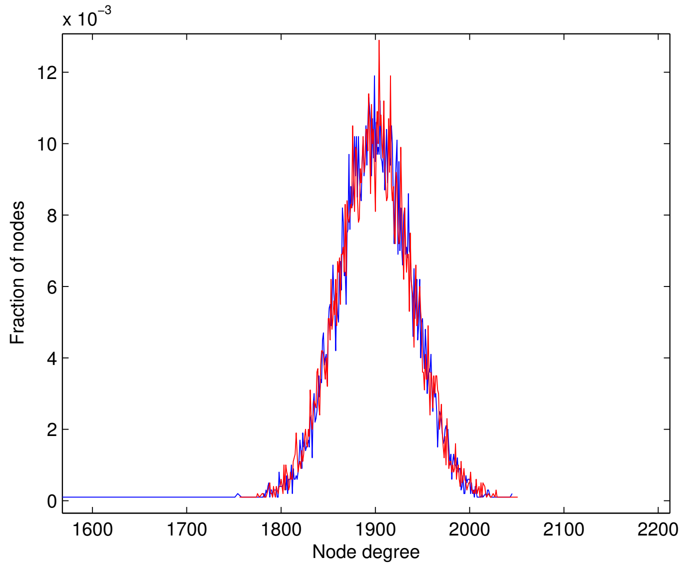


Figure 2.4: Degree distribution for ZER, 10000 nodes

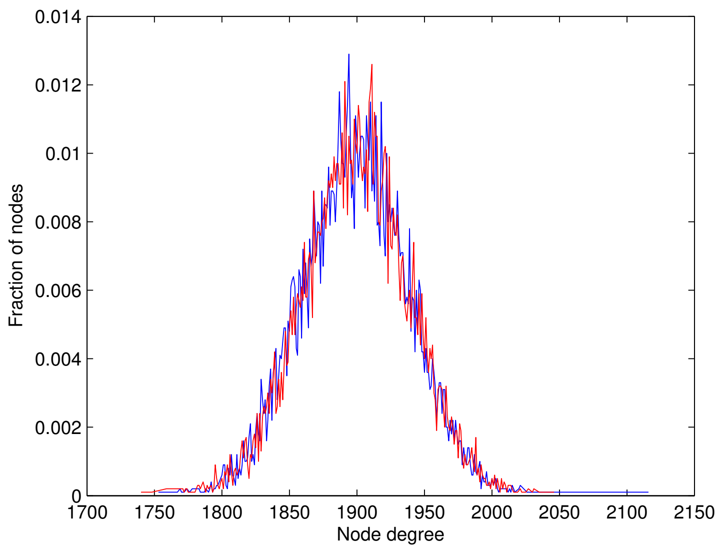


Figure 2.5: Degree distribution for DGRD, 10000 nodes

2.3.3 Results of algorithms for original small-world graph

Shape of the degree distribution for a small-world graph closely resembles one for ER one, and all algorithms again seem to show quite good results.

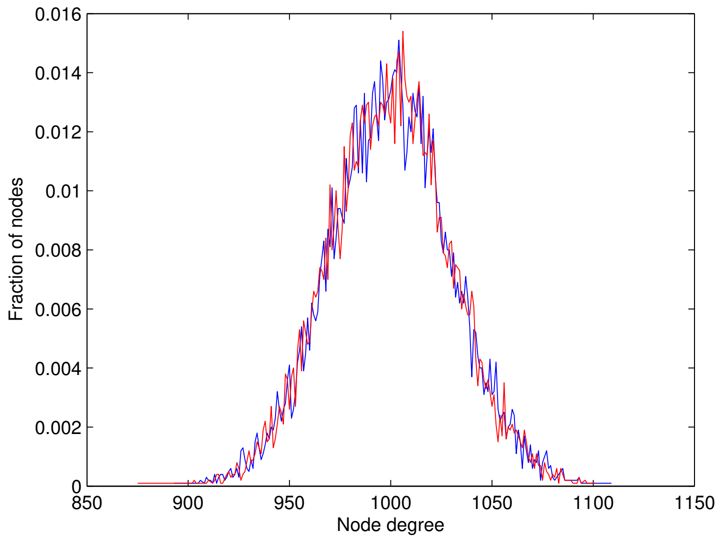


Figure 2.6: Degree distribution for ER, 10000 nodes

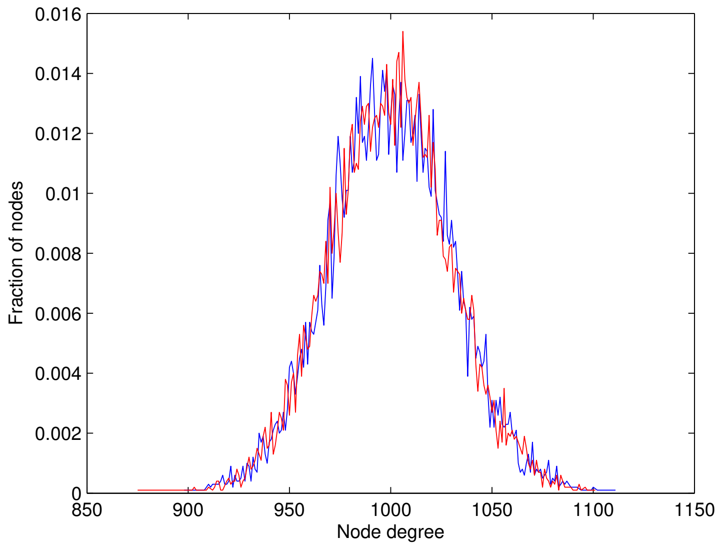


Figure 2.7: Degree distribution for ZER, 10000 nodes

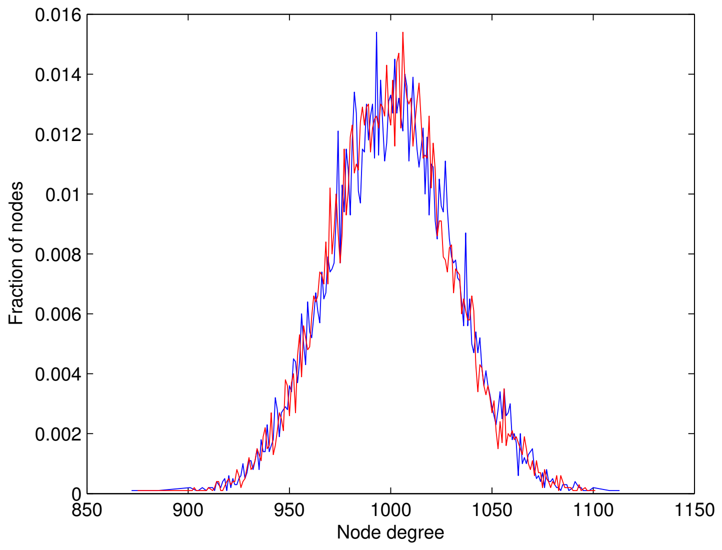


Figure 2.8: Degree distribution for DGRD, 10000 nodes

2.3.4 Results of algorithms for original scale-free graph

In this case we finally can see the difference in performance between the algorithms. Both ER and ZER fail to reproduce the shape of degree distribution curve because they do not take it into account, instead operating on average probability of edge inclusion which is more optimal for previous two models. However DGRD results resemble the original distribution while having approximately similar average run-times. Overall DGRD seems to be a good candidate, repeating degree distribution of various target graphs and having decent average run-times.

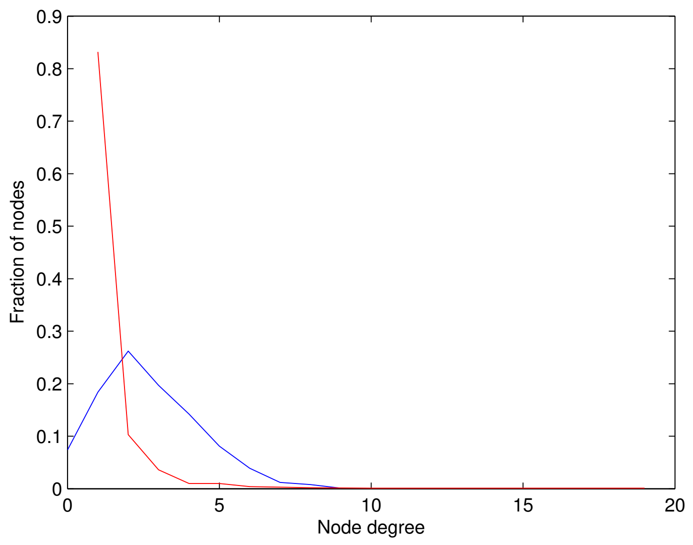


Figure 2.9: Degree distribution for ER, 1000 nodes. We see how a simple model fails to repeat the target degree distribution.

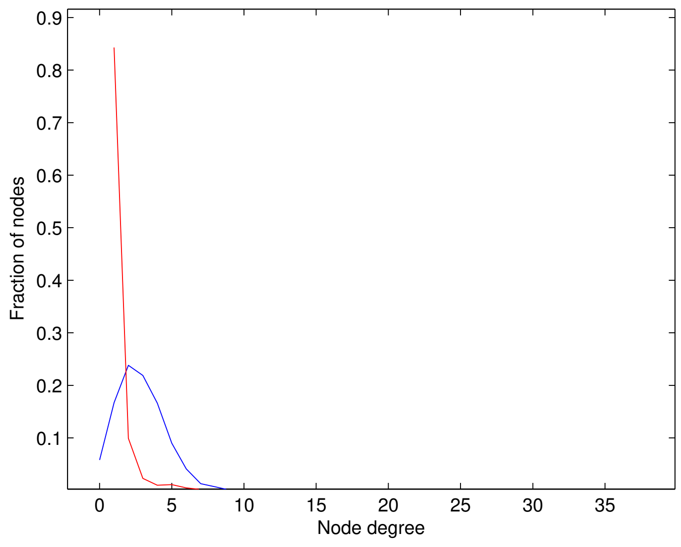


Figure 2.10: Degree distribution for ZER, 1000 nodes. Result is smoother than for ER, however still far from correct.

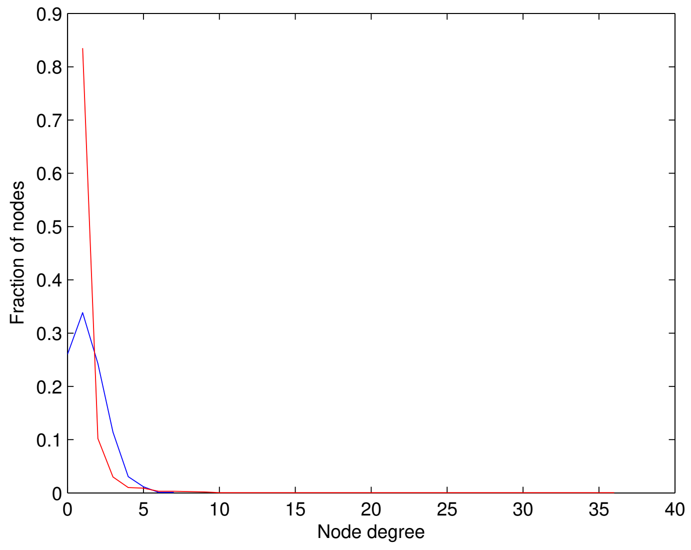


Figure 2.11: Degree distribution for DGRD, 1000 nodes. Result is even better because algorithm does not imply the structure to be almost entirely random as in previous algorithms case, although there is still room for improvement.

3

Chapter 3

3.1 Generation of random graphs with arbitrary degree distribution

The problem with the algorithms from the previous section is that they are related to, or focus on creating graphs with degree distribution that belongs to a certain class (e.g. Poisson distribution). They might work for arbitrary distributions but much slower, or sometimes generate a network that would have a degree distribution that is far from the required one. Therefore there is a need for another, more general approach.

There are two more things to mention here. First, since we are not basing the algorithm on any particular distribution and the algorithms usually build or change a graph step by step, there is a possibility of ending up with a degree sequence that is not graphical, i.e. there is no graph which would have the given sequence as its degree sequence. There are several known algorithms for this problem, the Havel-Hakimi [16] one being perhaps one of the fastest and easiest to implement. Suppose we have a degree sequence $S = d_1, d_2, ..., d_n$ that is non-increasing. It is graphic if and only if another list, $S' = d_2 - 1, d_3 - 1, ..., d_{d_1+1} - 1, d_{d_1+2}, ..., d_n$, has only non-negative integers and is graphic, therefore this algorithm is recursive. The step is applied at most $n - 1$ times, until either the last sequence consists only of zeros, which means that the original sequence was graphic, or the algorithm gets stuck and the current list cannot be reduced accordingly to move on to the next step, which means that the original sequence was not graphic. In most algorithms it is feasible to apply this check at each step to check if the current sequence is graphic or not, and to either retrace the changes one step back or restart the whole random graph creation entirely.

Second, the uniformity requirement might not have to be strict. Even if the results of the algorithm are non-uniform, it is generally possible to compute the probabilities of various outcomes and then simulate a general distribution via importance sampling [17]. Essentially the results are re-weighted from a trial distribution, but this is considered to be a difficult problem, often requiring construction of said trial distribution step by step recursively, which also adds up to the overall run-time of the algorithm. In addition to this, the number of possible random graphs in our case is really high due to large number of nodes, and calculating probabilities of each possible combination appearing as a result of chosen algorithm would be very complicated.

However the results can be non-uniform to a certain extent and this might still satisfy the requirements necessary to check real-world networks. The most important part when it comes to them is to study the appearance of certain subgraphs or motifs [19]. Motifs are statistically significant or recurrent patterns or parts of graph. The ones that researchers are usually interested in appear in the brain networks much more often than would be expected by pure chance, and the majority of algorithms don't give the results that are biased to an extent that would negate an occurence of an important motif. Therefore, while numerically the probabilities of certain random graphs appearing in process of using one of the algorithms we are looking into might be skewed they still would represent the average topology of the graphs and showcase particular motifs. That said, it is still preferable to have non-uniformity being as low as possible.

3.2 Configuration model

Perhaps the most basic idea would be to create a list of 'stubs' for each vertex that is equal to its degree and connect those stubs between each other at random until none are left (a similar approach is described, for example, in [18]). This model has many names, such as configuration model or matching algorithm. However, it has a number of drawbacks, which range from simple and easily avoidable (e.g., the number of stubs has to be even) to more difficult ones. Such algorithms do not work well with distributions with 'heavy tails' such as scale-free, and multiple or self-edges might be created, in which the network usually has to be discarded which considerably slows down the average runtime of the algorithm.

In our tests algorithm (slightly changed to avoid multiple or self-loops) has shown a very fast rise in run-time with increase of number of nodes (the maximum number of nodes tested was 500 which already was very slow), rendering it unfeasible for graphs with 10000 nodes or more. The degree distribution however was rather close, but that was to be expected for this algorithm. In the majority of cases, it reproduced the target degree distribution, and even if there was a difference, it was very small.

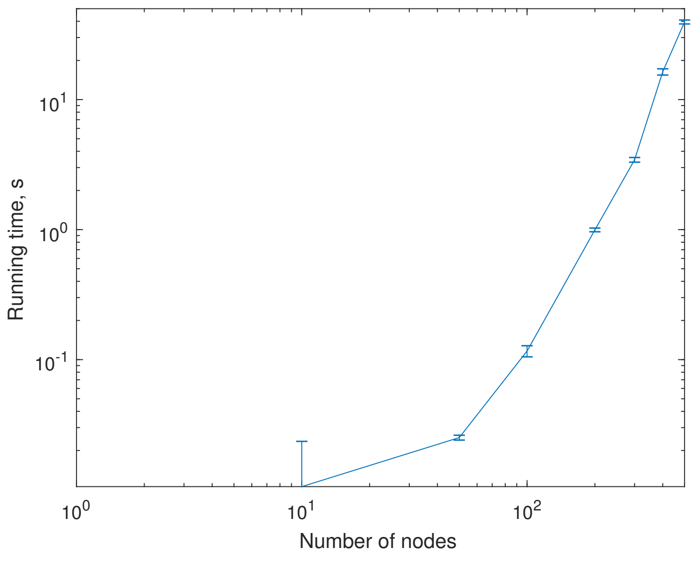


Figure 3.1: Running time of configuration algorithm with error bars, for number of nodes 10, 50, 100, 200, 300, 400 and 500. Results are averaged over 5 runs. p=0.1. The rise is faster than linear in log-log scale.

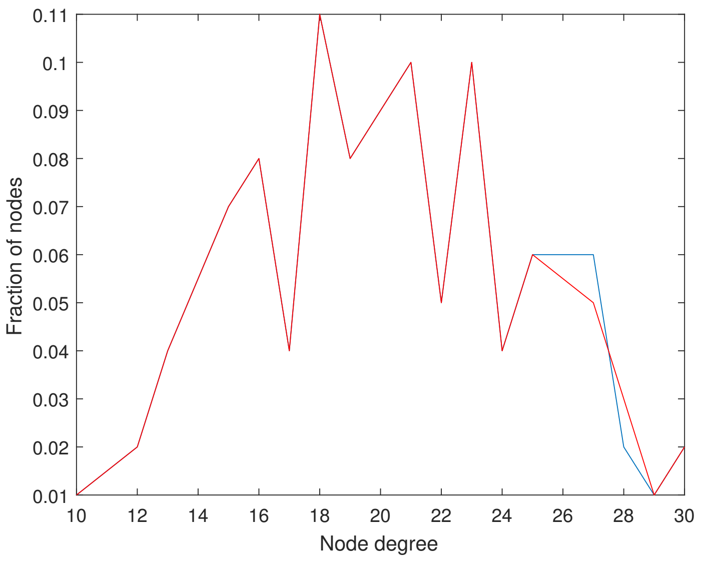


Figure 3.2: Degree distribution for configuration model, 100 nodes, original ER graph, p=0.1. It can be seen that the resultant degree distribution is slightly different from the target one, albeit the difference can almost be neglected.

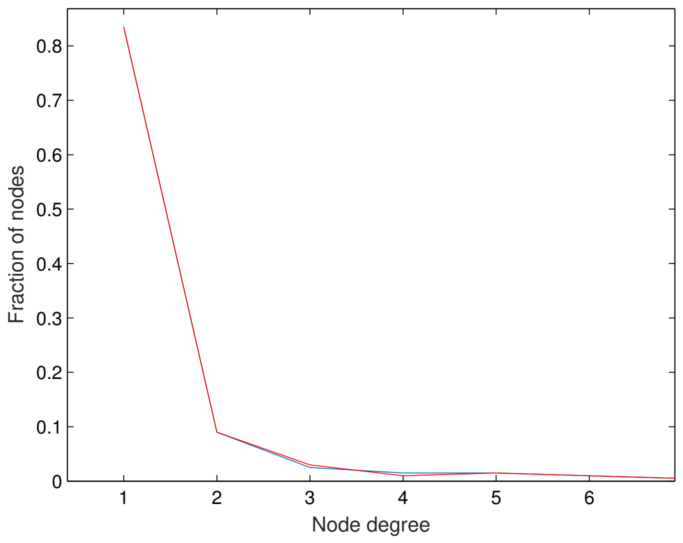


Figure 3.3: Degree distribution for configuration model, 200 nodes, original scale-free graph. The difference is very slight.

3.3 Switching model

The other idea is to use a Markov chain to create the necessary random graph [19]. This algorithm is often called 'switching', because each step a Monte Carlo switching step is performed where a pair of edges is selected at random and then the ends are exchanged. The exchange is carried out only if self-edges or multiple edges were not created in the process. The problem with this algorithm is that it is hard to decide when the the graph is 'shuffled' enough times, and because it is doubtful that the results will be uniform. Some variations, as 'go with the the winners' one, prove to be successful and avoid the last problem however they also tend to become rather slow.

Average running time obviously heavily depends on the number of shuffling steps. By definition of this algorithm the degree distribution remains the same, so this requirement for the algorithm is satisfied by default, but assuming the offered number of shuffling steps to be $4 * (number\ of\ edges)$ [20], increasing the number of nodes in graph vastly increases the running time of algorithm, for 10000 nodes getting an average value of about 464 seconds. Decreasing number of shuffles helps but then there is no guarantee that the algorithm properly randomizes the features of the graph. One of the advantages of both switching and configuration models however is that graphicality check can be avoided as it is possible to simply check for multiple or self-edges at each step and discard the change instantly to avoid it if necessary.

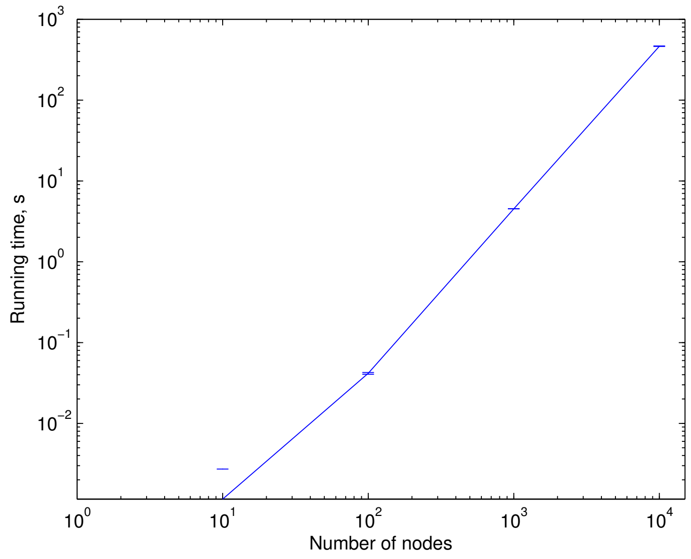


Figure 3.4: Running time of switching algorithm with error bars, for number of nodes 10, 100, 1000 and 10000. Results are averaged over 5 runs. p=0.1. Number of shuffling steps assumed as $4 * (number\ of\ edges)$

3.4 Preferential attachment model

Another group of algorithms [17] suffers from non-uniformity as well. However it never gets stuck or needs a restart because of creation of self- or multiple loops, and it has modifications ([21]) that aim at making the results more uniform. It constructs the network edge by edge, attaching them to nodes preferentially proportionally to current degree of a node. In this algorithm the graphicality check has to be carried out each step for every possible candidate list which significantly increases the runtime. Certain improvements were offered that are said to improve the runtime to $O(nM)$, but the standard model shows runtimes of $O(n^2M)$ ([22]).

One of the best algorithms that is built on this model for getting almost exact degree sequence is reported in [13]. It is claimed to have an almost linear running time ($O(Md_{max})$, where d_{max} is the maximum vertex degree) and to be sampling uniformly from the set of graphs with given degrees, which is also important. The algorithm keeps track of remaining degrees of vertices i and j, $\hat{d}_i$ and $\hat{d}_j$. Once again, using a set of edges L which is initially empty and another set of edges V corresponding to our graph, we can write out the algorithm as follows:

Algorithm 5 Random graph with given degree distribution

```
1: $L = \varnothing$
2: while edges can be added to $E$ do
3:     Choose $v_i\ v_j \in V$ with probability proportional to $\hat{d}_i\hat{d}_j(1 - \frac{d_id_j}{4M})$, $i, (v_i, v_j) \notin E$
4:     $E := E + (v_i, v_j),\ \hat{d}_i := \hat{d}_i - 1,\ \hat{d}_j := \hat{d}_j - 1$
```

The problem however is that whereas this algorithm has incredible runtimes compared to all others mentioned before the limitations and requirements are way too strict. First, it will not always generate a graph altogether, one possible situation being that no edges are allowed to add to the graph but not all necessary edges have been added yet. Second, low probability of this situation occurring holds only for graphs where $d_{max} = O(m^{1-4-\tau})$, where τ is a positive constant, this severely impacting possible application of this algorithm to the real-world networks.

4
Concluding remarks

Creating a random graph for large number of nodes in a matter of seconds while guaranteeing a degree distribution that is close to the target one is still an open problem. While a multitude of different algorithms exist none offer both precision and fast runtime, and it is usually better to focus on one or another. In our tests the DGRD algorithm seems to have shown results that are rather good as it is the only algorithm that showed sub-10 seconds runtime for 10000 nodes while generating a graph with a degree distribution that is somewhat close to the original one.
A lot of other options require significantly more testing and/or updates. A majority of studies focuses on theoretical confirmation of realization of a degree sequence while not paying as much attention to the runtime or in general applicability to a graph that has thousands of nodes.
Perhaps the most perspective algorithm is the one developed by Blitzstein and Diaconis, namely preferential attachment model. While the original offer is very slow, it has many updates and changes in other works that try to adapt it to different requirements, improve its performance and make it faster. The current options have strict limitations but it seems like the solution might be found based on the idea of sequential construction of graph where edge is added preferentially based on the current or leftover degree of nodes. Other models have much more serious drawbacks which probably cannot be accounted for or changed easily; configuration models are exhausted and don't work well for many real-world networks while the switching models have ingrained problem of impossibility of guessing how many shuffling steps need to be performed before the algorithm produces viable results.
It is necessary to test the future algorithms in a way that was followed in this work: both check the runtimes on high number of nodes and compare the degree distributions with the target ones, especially those that resemble real-world networks to a certain extent, for example in case of brain networks having small-world and scale free characteristics. In general, creation of such random graphs is a difficult problem, yet it still is just a small step in overall process of modelling brain networks with help of graph theory.
This master's thesis is part of my studies at Chalmers University of Technology, thanks to a Swedish Institute scholarship.

Bibliography

[1] Jiang Y, Huang H, Abner E, Broster LS, Jicha GA, Schmitt FA, Kryscio R, Andersen A, Powell D, Van Eldik L, Gold BT, Nelson PT, Smith C and Ding M (2016). Alzheimer's Biomarkers are Correlated with Brain Connectivity in Older Adults Differentially during Resting and Task States. Front. Aging Neurosci. 8:15.

[2] Lin-lin Gao and Tao Wu (2016). The study of brain functional connectivity in Parkinson's disease. Translational Neurodegeneration 20165:18.

[3] Bullmore, E., Sporns, O. (2009). Complex brain networks: Graph theoretical analysis of structural and functional systems. Nature Reviews.Neuroscience, 10(3), 186-98.

[4] Prettejohn BJ, Berryman MJ, McDonnell MD. Methods for Generating Complex Networks with Selected Structural Properties for Simulations: A Review and Tutorial for Neuroscientists. Frontiers in Computational Neuroscience. 2011;5:11. doi:10.3389/fncom.2011.00011.

[5] Bassett DS, Bullmore E. (2006). Small-world brain networks. Neuroscientist. 2006 Dec;12(6):512-23.

[6] van den Heuvel MP, Stam CJ, Boersma M, Hulshoff Pol HE. (2008). Small-world and scale-free organization of voxel-based resting-state functional connectivity in the human brain. Neuroimage. 2008 Nov 15;43(3):528-39.

[7] Barabasi AL, Albert R. (1999). Emergence of scaling in random networks. Science. 1999 Oct 15;286(5439):509-12.

[8] Achard S, Salvador R, Whitcher B, Suckling J, Bullmore E. (2006). A resilient, low-frequency, small-world human brain functional network with highly connected association cortical hubs. J Neurosci. 2006 Jan 4;26(1):63-72.

[9] Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. Nature. (1998) Jun 4;393(6684):440-2.

[10] Mijalkov M, Kakaei E, Pereira JB, Westman E, Volpe G, for the Alzheimer's Disease Neuroimaging Initiative (2017) BRAPH: A graph theory software for the analysis of brain connectivity. PLoS ONE 12(8): e0178798.

[11] Nobari SH, Lu X, Karras P, Bressan S (2011). Fast Random Graph Generation. Proceedings of the 14th International Conference on Extending Database Technology. 331-342. 10.1145/1951365.1951406.

[12] Frieze AM, McDiarmid C (1996). Algorithmic Theory of Random graphs. Random Structures Algorithms 10, 5-42 (187)

[13] Bayati, M., Kim, J.H. and Saberi, A. (2010). A Sequential Algorithm for Generating Random Graphs. Algorithmica 58: 860.

[14] Blitzstein J and Diaconis P. (2010). A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees. Internet Math. Volume 6, Number 4 (2010), 489-522.

[15] Britton, T., Deijfen, M. and Martin-Löf (2006). Generating Simple Random Graphs with Prescribed Degree Distribution. A. J Stat Phys 124: 1377.

[16] S. L. Hakimi, "On the realizability of a set of integers as degrees of the vertices of a linear graph," Journal of the Society of Industrial and Applied Mathematics, vol. 10, no. 3, pp. 496–506, 1962.

[17] Blitzstein J and Diaconis P. (2006). A sequential importance sampling algorithm for generating random graphs with prescribed degrees.

[18] Molloy, M. and Reed, B. (1995). A critical point for random graphs with a given degree sequence. Random Struct. Alg., 6: 161–180.

[19] Milo, R.; Kashtan, N.; Itzkovitz, S.; Newman, M. E. J. and Alon, U. (2003). On the uniform generation of random graphs with prescribed degree sequences, Arxiv preprint cond-mat/0312028 .

[20] Maslov S, Sneppen K. (2002). Specificity and stability in topology of protein networks. Science. 2002 May 3;296(5569):910-3.

[21] Obradović D. and Danisch M. (2014). Direct generation of random graphs exactly realising a prescribed degree sequence, 6th International Conference on Computational Aspects of Social Networks, Porto, 2014, pp. 1-6.

[22] Moseman E. (2015). Improving the Computational Efficiency of the Blitzstein-Diaconis Algorithm for Generating Random Graphs of Prescribed Degree.