Direct Generation of Random Graphs Exactly Realising a Prescribed Degree Sequence

Darko Obradović
German Research Center for AI (DFKI)
Kaiserslautern, Germany
Email: darko.obradovic@dfki.de

Maximilien Danisch
Sorbonne Universités, UPMC Univ. Paris 6
Paris, France
CNRS, UMR 7606, LIP6, F-75005 Paris, France.

***Abstract*—This paper intends to extend the possibilites available to researchers for the evaluation of directed networks with the use of randomly generated graphs. The direct generation of a simple network with a prescribed degree sequence still seems to be an open issue, since the prominent configuration model usually does not realise the degree distribution exactly. We propose such an algorithm using a heuristic for node prioritisation. We demonstrate that the algorithm samples approximately uniformly. In comparison to the *switching* Markov Chain Monte Carlo algorithms, the direct generation of edges allows an easy modification of the linking behaviour in the random graph, introducing for example degree correlations, mixing patterns or community structure. That way, more specific random graphs can be generated (non-uniformly) in order to test hypotheses on the question, whether specific network features are due to a specific linking behaviour only. Or it can be used to generate series of synthetic benchmark networks with a specific community structure, including hierarchies and overlaps.**

I. Introduction

Whenever case-studies of social networks are performed and methods and metrics from Social Network Analysis (SNA) [1] are used, the evaluation of the findings is a decisive aspect of the scientific work. One option for such an evaluation is the comparison of the original graph's properties to those of randomly generated graphs with the same degree sequence. Conforming properties can be considered to be trivial, and non-conforming ones indicate a distinctive particular feature or an anomaly of the original graph.

This method had a prominent showcase in a paper of Watts and Strogatz from 1998 [2], in which they showed that the famous "small-world phenomenon" is a common phenomenon in any graph with a small amount of randomness, and thus a trivial property of real-world networks, not a distinctive one. This had a lasting effect on social network research, promoting network evaluation by comparing them with random networks. One important finding was the consideration of degree distribution for these comparisons, as most large real-world networks show a highly heterogeneous power-law distribution [3], opposed to the expected Poisson distribution in the Erdos-Renyi random graph model [4]. Thus, for a sound analysis of properties it is necessary to sample a random graph with the same degree distribution.

In this paper we will take a closer look at existing algorithms for random graph generation and propose a new algorithm, capable of close-to-uniform sampling of random graphs by exactly realising the prescribed degree sequence in a direct way. Thanks to the direct generation, it can be easily tuned to sample networks with specific linking behaviours, which opens new possibilities for a more advanced evaluation of directed social networks and the generation of synthetic benchmark networks.

After a short summarisation of our graph notations in Section II we present related work in the field of random graphs in Section III. Throughout Section IV we present our algorithm and discuss it. Section V gives a brief outlook on our ideas for enhancing network evaluations with our algorithm, before the paper is concluded in Section VI.

II. Notations

First of all, we summarise the terms and notations we will adhere to in this paper. A graph G is defined as $G = (V, E)$ with V being the set of nodes, and $E = (V \times V)$ being the set of edges, directed ones or undirected ones. $n = |V|$ is the number of nodes in the graph, and $m = |E|$ is the number of edges.

Given a directed edge $e = (s, t)$, we call the node s the *source* of the edge and the node t the *target* of the edge. The function $succ(v)$ returns all successors of the node v, the function $pre(v)$ returns all predecessors of v.

With a prescribed degree sequence for all nodes in an undirected graph, we initially give each node the corresponding number of *stubs*, that have to be connected by attaching edges to the node until the prescribed degree is realised. In the process of random graph generation, stubs are replaced by edges. With a prescribed degree sequence for a directed graph we start with *out-stubs* and *in-stubs* for each node. The functions $ost(v)$ and $ist(v)$ return the number of remaining out- and in-stubs for the node v.

III. Related Work

In this section we present related work from the fields of random graph models and random graph generation algorithms, in order to get a better understanding for the benefits of our suggested approach.

A. Random Graph Models

Initiated by Erdos' random graph model [4], the disciplines of mathematics and physics were the first ones to start the study of random graphs and probabilistic random graph models. These studies usually focus on solving the graph with stochastic methods, and investigate global or local graph properties when n is going towards infinity. See [5] for an extensive summary of work in this direction.

The biggest problem with Erdos' random graph in the modelling of social networks is its Poisson degree distribution. Studies have shown that nearly all real-world networks have a highly heterogeneous degree distribution that follows at least asymptotically a power law in most cases. These types of networks are termed "scale-free" [6].

These observations and their practical implications lead to new random graph models, which can be parameterised in order to make a given degree distribution fit this model well [7]. And even models with prescribed arbitrary degree distributions and additional properties exist [8]. These models are very appealing, because they are exactly solvable and hence can give researchers an idea of global and nodal properties of such random graphs in their generalised form.

Following the seminal paper of Watts and Strogatz [2], practitioners in SNA are usually interested into the comparison of real-world network properties with random graph properties, in order to find uncommon differences. As there is hardly any software support, parameterising these models for a given real-world network, or even calculating metrics of interest which are beyond those already solved, are highly non-trivial tasks.

This is most probably the reason why most practical network studies still use explorative and descriptive methods for their evaluation, which might be very helpful in the beginning, but is not strictly conclusive in the end. Using instances of randomly generated graphs and comparing the metrics of real-world and random graphs with methods of descriptive statistics is state-of-the-art in practice and can be considered sufficiently conclusive, given a large enough number of samples.

By no means we want to discourage the use of network models, But recognising that their application is extremely difficult, and that there is still a multitude of properties unsolved for these models, we concentrate on the easier evaluations with randomly generated graphs.

B. Random Graph Generation

First of all we have to distinguish two types of random graph generation. The first one, the generation of instances of models, serves more general purposes, For example the empirical evaluation of model properties or model parameter impacts. For most models, these networks can be generated very efficiently [9] thanks to the exact mathematical properties of their degree distributions. The second type of generation requires a given arbitrary degree distribution of a real-world network to be exactly realised. As mentioned before, this is useful for the evaluation of concrete real-world networks, which is our focus in this paper. This requires a different approach to network generation, which we will look at in the following sections.

Milo et al. give a very good overview of this field in [10]. We will follow their methods and terms for the discussion of our algorithm.

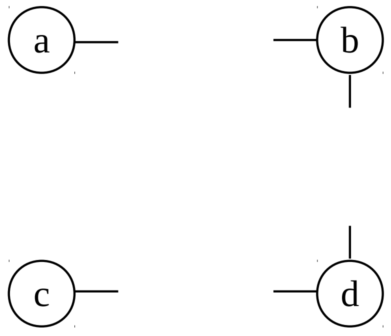

Fig. 1. Example network with stubs for edge generation

1) The Configuration Model: The simplest approach to this is the *configuration model*, which is well summarised by Newman in [11] (see Section IV.B). It is the set of all graphs with a given degree sequence. The generation algorithm is fairly easy, by simply choosing pairs of stubs uniformly at random and connecting them, until all stubs were replaced by edge endpoints. This algorithm is the default generating algorithm in most network libraries that offer generation by degree sequence, e.g. NetworkX[1] for Python, while other packages do not even include this one, e.g. JUNG[2] for Java.

However, it has one serious drawback for practical use cases. It's not restricted to simple graphs, it includes graphs with loops and parallel edges. In real-world networks, these are often forbidden properties, and hence an evaluation with this model is not fully accurate anymore. Figure 1 shows an undirected graph with a given degree sequence, for which we want to create edges by random. With the configuration model, any two stubs are chosen by random, which allows eight different connections in the first step. If you are however restricted to simple graphs, there are only five legal connections left, because (b, b) and (d, d) would directly violate the simplicity criteria, while (a, c) would inevitably lead to a violation in the next steps.

This discrepancy decreases with higher n, but when using the configuration model for evaluations, you nevertheless will have to discard loops and parallel edges afterwards at the price of a more or less different degree sequence than initially prescribed. A usable algorithm based on such a modification is evaluated by Milo et. al. [10] under the name *matching algorithm*. Creations of parallel edges do not stop the generation, but are just rejected. This increases the chances to get closer to the prescribed degree sequence, but will not realise it exactly in most larger networks. This algorithm has a noticeable bias in the uniformness of its samples. Viger and Latapy [12] have empirically demonstrated that this introduces a bias into network properties. On the other hand, Milo et al. argue that the consequences appear to be relatively small in their experiments. Still they suggest to use *Markov Chain*

[1]http://networkx.lanl.gov/
[2]http://jung.sourceforge.net/

Monte Carlo Algorithms instead.

2) Markov Chain Monte Carlo Algorithms: As claimed by Viger and Latapy in [12]:

> Although it has been widely investigated, it is still an open problem to directly generate such a random graph, or even to enumerate them in polynomial time [...]

This enumeration has been accomplished by Snijders [13], but because of the resulting exponential runtime complexities, most researchers turned towards *Monte Carlo* methods for random graph generation.

According to Milo et al. [10], the fastest of these algorithms are *Markov Chain Monte Carlo* (MCMC) algorithms. They have the additional benefit to also enable the creation of *connected* simple graphs if desired, at the price of a higher runtime complexity. These algorithms do not directly create random graphs, but proceed in the following three steps.

1) Generate a simple graph realising the prescribed degree sequence.
2) Connect it with edge swaps, if connectivity is desired.
3) Perform a series of edge swaps, until the graph appears to be a random one. This is called *shuffling* the graph.

Step 1 can be done with with a Havel-Hakimi algorithm [14], which exactly realises a degree sequence in a deterministic way. When connectivity is not enforced, Viger and Latapy [12] validate empirically that $O(m)$ edge swaps are sufficient for nearly perfect uniform sampling, but a proof was still an open issue. Milo et al. [10] estimate the constant factor of this bound to be around 100. Furthermore they describe that a naive implementation with guaranteed connectivity has a runtime complexity within $O(m^2)$. This naive algorithm is called *switching algorithm.* Viger and Latapy [12] propose a speed-up to a runtime complexity of $O(m \cdot log(m))$ for undirected graphs, based on a corrolary that also has the issue of a missing proof, but is backed up with a thorough empirical validation. The suitability of the speed-up for directed graphs is not discussed.

While our proposed algorithm will not be in a better complexity class than the optimised MCMC algorithm, and will also not generate connected networks only, we still see two advantages of our proposed algorithm over MCMC algorithms. First, our algorithm is much simpler to implement, and might have a better chance to replace *matching algorithms* in software packages. Second, it is much easier for researchers to introduce a specific linking behaviour with a directly generating algorithm than with edge swaps, where the legal pairs of edges to be swapped are additionally constrained in order to prevent non-simple graphs. For these reasons, we will pursue a direct generation algorithm for our random graph generations.

3) Sequential Sampling: Another algorithm with a sequential sampling is proposed by Blitzstein and Diaconis [15]. Like our proposed algorithm, it generates a network sequentially by enforcing an order for one side of newly created edges, and a random selection of the other side. In consequence, it also does not sample strictly uniformly at random. We see two major points why our proposed algorithm is a viable alternative. First, their algorithm is currently only described and proved for undirected networks, though an adaptation for directed networks might be possible in the future. Second, their algorithm relies on checking at each step if setting a new edge will lead to a realisable degree sequence, slowing down the algorithm to a running time of $O(m \cdot n^2)$, which is inferior to the other methods.

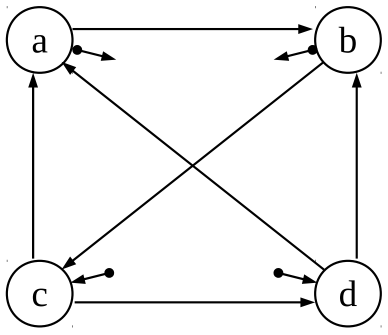

Fig. 2. Example network in the process of being generated

IV. Prioritisation of the Most Constrained Nodes

A. Principle Idea

It is our goal to have a graph generation algorithm that creates edges in such an order, that the graph will definitely become a simple graph, and that can be easily tuned to incorporate specific linking behaviours between nodes. This last requirement can be easily accomplished by having each stub select a *peer* stub at random, while it is allowed to prefer certain stubs over others.

Concerning the guarantee to end up with a simple graph, we will impose a certain order on the nodes that select their peers, which will avoid to run into unresolvable situations during the generation process. Figure 2 shows a network in the process of being generated, with four stubs left to be connected. Obviously, all four nodes are structurally equivalent, thus a more sophisticated measure for prioritisation than the number of stubs is required.

B. Nodal Degrees of Freedom

The key for solving the prioritisation problem is the number of remaining peer nodes for connecting the stubs of a given node. We therefore define for each node v two sets of *candidate nodes*, target candidates $tc(v)$ for v's out-stubs and source candidates $sc(v)$ for v's in-stubs, if v has corresponding stubs left that need to be connected.

$$tc(v) = \{w \in V | w \neq v \land w \notin succ(v) \land ist(w) > 0\} \quad (1)$$

$$sc(v) = \{w \in V | w \neq v \land w \notin pre(v) \land ost(w) > 0\} \quad (2)$$

	a	b	c	d
idf	∞	∞	0	1
odf	1	0	∞	∞
df	1	0	0	1

TABLE I
NODAL DEGREES OF FREEDOM FOR THE EXAMPLE NETWORK FROM FIGURE 2

With these candidate lists we can now compute for each node v its degree of freedom with respect to potential peer nodes. This is done separately for outbound edges with $odf(v)$ and for inbound edges with $idf(v)$ as follows.

$$odf(v) = \begin{cases} |tc(v)| - sst(v) & , ost(v) > 0 \\ \infty & , else \end{cases} \tag{3}$$

$$idf(v) = \begin{cases} |sc(v)| - tst(v) & , ist(v) > 0 \\ \infty & , else \end{cases} \tag{4}$$

With these two values, we can decide the node order by prioritising the most constrained nodes (MCN), i.e. those with the lowest nodal degree of freedom in any of the two cases. This is expressed by the function value $df(v)$.

$$df(v) = min(odf(v), idf(v)) \tag{5}$$

Table I lists the nodal degrees of freedom for the example network from Figure 2 as defined by our formulas. With these values it is apparent that either node b or node c have to be prioritised in selecting a target peer or a source peer respectively, which will lead to the new edge (b, d) or (a, c) to be created, and avoids edges that would not allow the creation of a simple graph. This principle leads to the algorithm described in the next Subsection.

C. The Algorithm

In this subsection we describe an initial naive implementation of our algorithm. We use square brackets to distinguish the two different cases that can occur in the loop. So in each iteration, either the left part or the right part are applicable throughout the loop.

```
1) calculate initial idf and odf values for all nodes
2) repeat m times:
    a) randomly use a node with the lowest occuring df
       value as mcn
    b) distinguish [ odf(mcn) < idf(mcn) / else ]
    c) create candidates list for mcn, using all nodes
       with [in/out]_stubs > 0 except mcn and [
       succ(mcn) / pre(mcn) ]
    d) select peer at random from candidates
    e) create the edge mcn [ -> / <- ] peer
    f) correct idf of all nodes if [ mcn / peer ] used
       its last source stub
    g) correct odf of all nodes if [ peer / mcn ] used
       its last target stub
```

We have implemented a slightly optimised version of the described algorithm in our open source network analysis package `SNA::Network`, which can be found at Perl's central package platform CPAN[3].

D. Space and Time Complexity

The complexity for storage space is linear, i. e. $O(n + m)$, as we are operating on a single graph instance, without multi-dimensional data structures.

For runtime complexity, step 1 can be performed in $O(n)$, by first counting all possible source and target nodes, and then iterating over all nodes and setting the idf and odf values. Steps (f) and (g) are executed at most n times each, independently from the loop in step 2, as each node can only use its last out- or in-stub once. The resulting correction of idf and odf values requires the same cost as in step 1 in each such case. Thus the cost for these two steps is in $O(n^2)$. The loop in step 2 is executed exactly m times, and steps (b), (d) and (e) can all be performed in $O(1)$. Step (a) requires a linear search in the node list, which can be done in $O(n)$. Step (c) requires an iteration over all nodes, with resulting costs in $O(n)$. For the complete loop the summed up runtime cost is within $O(m \cdot n)$, and the total runtime complexity for the complete algorithm applied on networks without isolated nodes is thus in:

$$O(n + n^2 + m \cdot n) = O(m \cdot n) \tag{6}$$

E. Uniformness

By restricting the random order in which nodes are selected for connections, we violate the requirement for uniformness. This is more a problem in the initial phase of the algorithm, when df values are highly heterogeneous. These values will balance out over time and then allow a close to real random selection.

We evaluate the uniformness of the samples with exactly the same experiment as performed by Milo et al. [10]. Figure 3 displays the two possible topologies for a simple toy network consisting of an out-hub with ten outgoing edges, an in-hub with ten incoming edges, and ten nodes with one incoming and one outgoing edge each. There is only one way to form the topology in (a), but 90 different ways to form the topology in (b). A uniformly sampling algorithm would thus sample each of the 91 configurations in 1.099% of all cases.

Using our algorithm we sampled 1,000,000 networks with this degree sequence and counted how often each of the configurations occured. Restricting the node selection to the two hub nodes for the first edge creations introduces a measurable bias in the sampling frequency of the configuration (a), which was sampled in 1.45% of all cases. This is an oversampling of about 32%. The 90 configurations of the topology type (b) are sampled uniformly. The oversampling can be explained by the prioritisation of the two hub nodes, which does not give the inner nodes equal chance to connect early with each other,

[3]http://www.cpan.org/

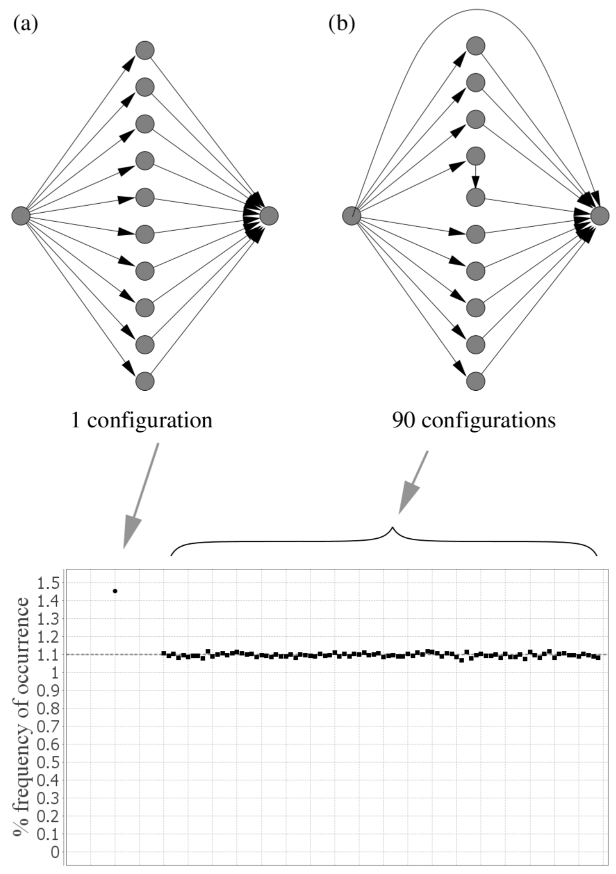

Fig. 3. Test of the uniformness of our proposed algorithm (compare with [10] p.3)

but only later, when they reach equally low *df* values as the two hubs.

The *matching algorithm* evaluated by Milo et al. sampled the configuration (a) in under 0.3% of all cases [10], which is an undersampling of more than 70%. But even for this algorithm, the measurable bias in this extreme degree sequence does usually not result in a heavy bias for large real-world networks. In the tests we conducted, we also did not see noticeable discrepancies for our algorithm.

F. Connectivity

Just like the matching algorithm, our algorithm does not guarantee weak connectivity for the graph. But the set of all connected configurations is also uniformly sampled when ignoring disconnected samples and repeating the sample.

In evaluations with descriptive statistics, for which a specific number of connected random graph samples is requested, we can partially work around this problem. The probabilities for a uniformly sampled random graph to be connected can be exactly calculated, as shown by Molloy and Reed [16]. Having an a priori estimation on how many iterations are required. such a set of connected random graphs can be created by iteratively sampling until the requested number of connected samples is generated. This highly depends on the resulting probabilities for the given degree sequence though.

G. Missing Proof for Correctness

Our suggested algorithm uses a heuristic for the order of edge generation, that guarantees the realisation of the degree sequence if it is realisable. However, we have no formal proof for this claim. All our empirical tests on real-world networks were successful and we were also not able to construct a graph in which obeying to the MCN prioritisation heuristic leads to an unrealisable graph state. Instead, all such problematic edges, determined by Snijders [13] as definitive 0 entries in the matrix, are effectively avoided, as both corresponding nodes have relatively high *df* values in the beginning.

V. Non-Uniform Random Peer Selection

The initial motivation for this work was the desire to bring statistical evaluations of social networks with randomly generated graphs one step forward. As shown in Section III, state-of-the art network generation algorithms for prescribed degree sequences all work in a non-deterministic way, with back-tracking, reverse steps, etc. And also with indirect edge generation, like the Markov Chain class of algorithms.

This makes it quite hard to introduce a specific bias in uniformness in an exactly controlled way, which is required to simulate real-world linking behaviour. With a direct selection of peers for a given node, this becomes a lot easier, as different classes of nodes can be weighted differently in the random selection of step (2d) of our algorithm (see Section IV-C).

In this section, we want to give a brief outlook on the applicability and benefits of this idea, by focusing on two problems in the evaluation of networks.

A. Assortative Mixing in Political Blogs

A first example of application originates from a study by Adamic and Glance [17], in which they analysed the linking patterns in the 2004 U.S. political blogosphere. They aggregated a blogroll network of over 1,000 blogs, partly conservative ones, partly liberal ones. They observed that the two groups each form a subcommunity by preferring blogs of their own class by roughly 90%.

This phenomenon is called a *mixing pattern* in network theory [18], and *assortative mixing* in social networks. The preference weights in link selection can be derived from the *mixing matrix* and will generate a random network with an *assortativity coefficient* expected to be the same as in the original network.

That way, the political blog network's properties could be compared statistically to random networks with the same degree sequence and the same assortativity coefficient, and possibly reveal structural differences that would have remained unnoticed otherwise, or prove properties to be a trivial consequence of the mixing pattern.

B. Synthetic Community Structure

A second example of application is the generation of sample networks with an a priori defined synthetic community structure. A community is generally modeled by a group of nodes that are more probably linked to one-another than to the outside. The community structure can be seen as a partition, as hierarchical or as overlapping. Being able to generate benchmark graphs with community structure, while keeping other common complex networks' properties like the exact power-law degree distribution is necessary for the evaluation of community detection algorithms.

However we could find only one method to generate such networks with a non-overlapping community structure [19]. This method consists in applying the configuration model with the additional constraint of connecting nodes preferably to nodes belonging to the same community rather than to nodes belonging to other communities. The method was then extended to directed networks with an overlapping community structure in [20]. However in this model most nodes should belong to only one community and the rest of the nodes should all belong to the same number of communities. While these methods are the only ones being able to generate networks with communities while keeping a prescribed degree sequence and are widely used, they are limited by the fact that they are using the configuration model with additional constraints. They thus only roughly keep the prescribed degree sequence. They are also limited by the constraints on the number of communities nodes can belong to and by the overlap of the communities, which is not a tunable parameter. On the contrary, our algorithm, not suffering from these defects, can be used to generate all combinations of plain, hierarchical and overlapping communities that can be described in blocks and linking preferences between them.

VI. Conclusion

In this paper we presented a fast and very simple algorithm to directly create random graphs exactly realising a prescribed degree sequence nearly uniformly, which was an open issue before. The matching algorithm has issues with the exact realisation, and the switching algorithm works indirectly, which makes changes in the linking behaviour much more difficult.

Using descriptive statistics, such a tunable linking behaviour can be used to compare real-world networks to random networks with specific linking behaviour, and provide network analysts with new insights. In contrast to network models, an evaluation with random graph samples can be directly applied to any metric that is computable on a given network instance. Hence, we see a huge potential for practitioners in this methodology.

The algorithm is also well-suited for the generation of networks with synthetic community structure of any kind that you can describe in blocks and linking preferences. This opens new applications for benchmarking.

We have two urgent points for future work on this subject, apart from the missing proof for the determinism. Although related work indicates that the bias in uniformity should be no issue in real-world degree sequences, an extensive evaluation similar to the one described by Milo et al. [10] still has to be performed. Concerning efficiency, we see room for improvement by exploiting the monotony and relative stability of df values with more sophisticated data structures than plain arrays. This however would happen at the price of a more complicated implementation.

Acknowledgements

This reasearch was funded by the German Ministry of Education and Research (BMBF) in the NEXUS project, grant 01IW11001.

References

[1] S. Wasserman, K. Faust, and D. Iacobucci, *Social Network Analysis : Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, 1994.

[2] D. Watts and S. Strogatz, "Collective dynamics of small-world networks," *Nature*, no. 393, pp. 440–442, 1998.

[3] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 1999, pp. 251–262.

[4] P. Erdos and A. Renyi, "On random graphs," *Publ. Math. Debrecen*, vol. 6, p. 290, 1959.

[5] B. Bollobas, *Random Graphs*. London: Academic Press, 1985.

[6] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[7] S. Wasserman and G. L. Robins, "An introduction to random graphs, dependence graphs, and p*," in *Models and methods in social network analysis*, P. J. Carrington, J. Scott, and S. Wasserman, Eds. Cambridge University Press, 2005, pp. 148–161.

[8] M. Newman, D. Watts, and S. Strogatz, "Random graph models of social networks," *Proceedings of the National Academy of Sciences USA*, vol. 99, pp. 2566–2572, 2002.

[9] V. Batagelj and U. Brandes, "Efficient generation of large random networks," *Physical Review E*, vol. 71, no. 3, p. 036113, March 2005.

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

[11] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, pp. 167–256, 2003.

[12] F. Viger and M. Latapy, "Efficient and simple generation of random simple connected graphs with prescribed degree sequence." in *Proceedings of the 11th international conference on Computing and Combinatorics*, ser. LNCS, vol. 3595. Springer, 2005, pp. 440–449.

[13] T. Snijders, "Enumeration and simulation methods for 0-1 matrices with given marginals," *Psychometrika*, vol. 56, no. 3, pp. 397–417, September 1991.

[14] 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.

[15] J. Blitzstein and P. Diaconis, "A sequential importance sampling algorithm for generating random graphs with prescribed degrees," *Internet Mathematics*, vol. 6, no. 4, pp. 489–522, 2011.

[16] M. Molloy and B. Reed, "A critical point for random graphs with a given degree sequence," *Random Structures and Algorithms*, vol. 6, pp. 161–179, 1995.

[17] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 u.s. election: divided they blog," in *Proceedings of the 3rd international workshop on Link discovery (LinkKDD)*, 2005, pp. 36–43.

[18] M. E. J. Newman, "Mixing patterns in networks," *Physical Review E*, vol. 67, 2003.

[19] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Review E*, vol. 78, no. 4, p. 046110, 2008.

[20] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical Review E*, vol. 80, no. 1, p. 016118, 2009.