Efficient and simple generation of random simple connected graphs with prescribed degree sequence

Fabien Viger[1,2], Matthieu Latapy[2]
{fabien,latapy}@liafa.jussieu.fr

Abstract. We address here the problem of generating random graphs uniformly from the set of simple connected graphs having a prescribed degree sequence. Our goal is to provide an algorithm designed for practical use both because of its ability to generate very large graphs (efficiency) and because it is easy to implement (simplicity).
We focus on a family of heuristics for which we prove optimality conditions, and show how this optimality can be reached in practice. We then propose a different approach, specifically designed for typical real-world degree distributions, which outperforms the first one. Assuming a conjecture, we finally obtain an $O(n \log n)$ algorithm, which, in spite of being very simple, improves the best known complexity.

1 Introduction

Recently, it appeared that the degree distribution of most real-world complex networks is well approximated by a power law, and that this unexpected feature has a crucial impact on many phenomena of interest [5]. Since then, many models have been introduced to capture this feature. In particular, the Molloy and Reed model [13], on which we will focus, generates a random graph with prescribed degree sequence in linear time. However, this model produces graphs that are neither *simple*[3] nor *connected*. To bypass this problem, one generally simply removes multiple edges and loops, and then keeps only the largest connected component. Apart from the expected size of this component [14,2], very little is known about the impact of these removals on the obtained graphs, on their degree distribution and on the simulations processed using them.
The problem we address here is the following: given a degree sequence, we want to generate a random *simple connected* graph having exactly this degree sequence. Moreover, we want to be able to generate very large such graphs, typically with more than one million vertices, as often needed in simulations.
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, even without the connectivity requirement [11,12].
In this paper, we will first present the best solution proposed so far [6,12], discussing both theoretical and practical considerations. We will then

[1] LIP6, University Pierre and Marie Curie, 4 place Jussieu, 75005 Paris
[2] LIAFA, University Denis Diderot, 2 place Jussieu, 75005 Paris
[3] A simple graph has neither multiple edges, *i.e.* several edges binding the same pair of vertices, nor loops, *i.e.* edges binding a vertex to itself.

deepen the study of this algorithm, which will lead us to an improvement that makes it optimal among its family. Furthermore, we will propose a new approach solving the problem in $O(n \log n)$ time, and being very simple to implement.

2 Context

The Markov chain Monte-Carlo algorithm

Several techniques have been proposed to solve the problem we address. We will focus here on the Markov chain Monte-Carlo algorithm [6], pointed out recently by an extensive study [12] as the most efficient one.
The generation process is composed of three main steps:

1. **Realize the sequence**: generate a simple graph that matches the degree sequence,
2. **Connect** this graph, without changing its degrees, and
3. **Shuffle** the edges to make it random, while keeping it connected and simple.

The Havel-Hakimi algorithm [8,7] solves the first step in linear time and space. A result of Erdös and Gallai [4] shows that this algorithm succeeds if and only if the degree sequence is realizable.
The second step is achieved by swapping edges to merge separated connected components into a single connected component, following a well-known graph theory algorithm [3,15]. Its time and space complexities are also linear.

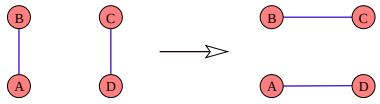

Fig. 1. Edge swap

The third step is achieved by randomly swapping edges of the graph, checking at each step that we keep the graph simple and connected. Given the graph G_t at some step t, we pick two edges at random, and then we swap them as shown in Figure 1, obtaining another graph G' with the same degrees. If G' is still simple and connected, we consider the swap as *valid*: $G_{t+1} = G'$. Otherwise, we reject the swap: $G_{t+1} = G_t$
This algorithm is a Markov chain where the **space** S is the set of all simple connected graphs with the given degree sequence, the **initial state** G_0 is the graph obtained by the first two steps, and the **transition** $G_i \to G_j$ has probability $\frac{1}{m(m-1)}$ if there exists an edge swap that transforms G_i in G_j. If there are no such swap, this transition has probability

0 (note that the probability of the transition $G_i \rightarrow G_i$ is given by the number of invalid swaps on G_i divided by $m(m-1)$).
We will use the following known results:

Theorem 1 *This Markov chain is irreducible [15], symmetric [6], and aperiodic [6].*

Corollary 2 *The Markov chain converges to the uniform distribution on every states of its space,* i.e. *all graphs having the wanted properties.*

These results show that, in order to generate a random graph, it is sufficient to do *enough* transitions. No formal result is known about the convergence speed of the Markov chain, *i.e.* the required number of transitions. However, massive experiments [6,12] applied the shuffle process with an extremely biased G_0 and showed clearly that $O(m)$ edge swaps are sufficient, by comparing a large set of non-trivial metrics (such as the diameter, the flow, and so on) over the sampled graphs and random graphs. Moreover, we proved[4] that for any *non-ill shaped*[4] degree distribution, the ratio of *valid* edge swaps is greater than some positive constant, so that $O(m)$ transitions are sufficient to ensure $\Omega(m)$ swaps to be done. Therefore, we will assume the following:

Empirical Result 1 *[12,6] The Markov chain converges after $O(m)$ transitions.*

Complexity

The first two steps of the random generation (realization of the degree sequence and connection of the graph) are done in $O(m)$ time and space. Using hash tables for the adjacency lists, each transition may be done in $O(1)$ time, to which we must add the connectivity tests that take $O(m)$ time per transition. Thus, the total time complexity for the shuffle is quadratic:

$$C_{naive} = O(m^2) \tag{1}$$

Using the structures described in [9,10,17] to maintain connectivity in dynamic graphs, one may reduce this complexity to the much smaller :

$$C_{dynamic} = O\left(m \log n (\log \log n)^3\right) \tag{2}$$

Notice however that these structures are quite intricate, and that the constants are large for both time and space complexities. The naive algorithm, despite the fact that it runs in $O(m^2)$ time, is therefore generally used in practice since it has the advantage of being extremely easy to implement. Our contribution in this paper will be to show how it can be significantly improved while keeping it very simple, and that it can even outperform the dynamical algorithm.

[4] All the proofs, and more details may be found in the full version[18]

Speed-up and the Gkantsidis et al. heuristics

Gkantsidis et al. proposed a simple way to speed-up the naive implementation [6]: instead of running a connectivity test for *each* transition, they do it every T transitions, for some integer $T \geq 1$ called the *speed-up window*. If the graph obtained after these T transitions is not connected anymore, the T transitions are cancelled.
They proved that this process still converges to the uniform distribution, although it is no longer composed of a single Markov chain but of a concatenation of Markov chains [6]. The global time complexity of connectivity tests C_{conn} is reduced by a factor T, but at the same time the swaps are more likely to get cancelled: with T swaps in a row, the graph has more chances to get disconnected than with a single one. Let us introduce the following quantity:

Definition 1 (Success rate) *The success rate $r(T)$ of the speed-up at a given step is the probability that the graph obtained after T swaps is still connected.*

The shuffle process now requires $O(m/r(T))$ transitions. The time complexity therefore becomes:

$$C_{Gkan} = O\left(r(T)^{-1}\left(m + \frac{m^2}{T}\right)\right) \tag{3}$$

Notice that there is a trade-off between the idea of reducing the connectivity test complexity and the increase of the required number of transitions. To bypass this problem, Gkantsidis et al. used the following heuristics:

Heuristics 1 (Gkantsidis et al. heuristics) `IF` *the graph got disconnected after T swaps* `THEN` $T \leftarrow T/2$ `ELSE` $T \leftarrow T + 1$

3 More from the Gkantsidis et al. heuristics

The problem we address now is to estimate the efficiency of the Gkantsidis heuristics. First, we introduce a framework to evaluate the ideal value for the window T. Then, we analyze the behavior of the Gkantsidis et al. heuristics, and get an estimation of the difference between the speed-up factor they obtain and the optimal speed-up factor. We finally propose an improvement of this heuristics which reaches the optimal. We also provide experimental evidences for the obtained performance.

The optimal window problem

We introduce the following quantity:

Definition 2 (Disconnection probability) *Given some graph G, the* disconnection probability *p is the probability that the graph becomes disconnected after a random edge swap.*

Hypothesis 1 *The disconnection probability p is constant during T consecutive swaps*

Hypothesis 2 *The probability that a disconnected graph gets connected with a random edge swap, called the* reconnection probability*, is equal to zero.*

These hypothesis are reasonable approximations in our context and will actually be confirmed in the following. Thanks to them, we get the following expression for the success rate $r(T)$, which is the probability that the graph stays connected after T swaps:

$$r(T) = (1-p)^T \tag{4}$$

Definition 3 (Speed-up factor) *The* speed-up factor $\theta(T) = T \cdot r(T)$ *is the expectation of the number of swaps actually performed (not counting cancelled swaps) between two connectivity tests.*

The speed-up factor $\theta(T)$ represents the *actual* gain induced by the speed-up for the total complexity of the connectivity tests C_{conn}.

Now, given a graph G with disconnection probability p, the *best* window T is the window that maximizes the speed-up factor $\theta(T)$. We find an optimal value $T = -1/\ln(1-p)$, which corresponds to a success rate $r(T) = 1/e$. Finally, we obtain the following theorem:

Theorem 3 *The speed-up factor θ_{max} is reached if and only if one of the equivalent conditions is satisfied:*

$$\textbf{(i)}\ T = (-\ln(1-p))^{-1} \qquad \textbf{(ii)}\ r(T) = e^{-1}$$

The value of θ_{max} depends only on p and is given by

$$\theta_{max} = (-\ln(1-p)\cdot e)^{-1} \quad \sim_{p\to 0} \quad (p\cdot e)^{-1}$$

Analysis of the heuristics

Knowing the optimality condition, we tried to estimate the performance of the Gkantsidis et al. heuristics. Considering p as asymptotically small, we obtained[4] the following:

Theorem 4 *The speed-up factor $\theta_{Gkan}(p)$ obtained with the Gkantsidis heuristics verifies:*

$$\forall \epsilon > 0, \qquad \theta_{Gkan} = o\left((\theta_{max})^{1/2+\epsilon}\right) \quad when \quad p \to 0$$

More intuitively, this comes from the fact that the Gkantsidis et al. heuristics is too pessimistic: when the graph gets disconnected, the decrease of T is too strong; conversely, when the graph stays connected, T grows too slowly. By doing so, one obtains a very high success rate (very close to 1), which is not the optimal (see Theorem 3).

An optimal dynamics

To improve the Gkantsidis et al. heuristics we propose the following one (with two parameters q^- and q^+) :

Heuristics 2 `IF` *the graph got disconnected after T swaps* `THEN` $T \leftarrow T \cdot (1 - q^-)$ `ELSE` $T \leftarrow T \cdot (1 + q^+)$

The main idea was to avoid the linear increase in T, which is too slow, and to allow more flexibility between the two factors $1 - q^-$ and $1 + q^+$. We proved[4] the following:

Theorem 5 *With this heuristics, a constant p, and for q^+, q^- close enough to 0, the window T converges to the optimal value and stays arbitrarily close to it with arbitrarily high probability if and only if*

$$q^+/q^- = e - 1 \tag{5}$$

Experimental evaluation of the new heuristics

To evaluate the relevance of these results, based on Hypothesis 1 and 2, we will now compare empirically the speed-up factors θ_{Gkan}, θ_{new} and θ_{best} respectively obtained with the three following heuristics:

1. The Gkantsidis et al. heuristics (Heuristics 1)
2. Our new heuristics (Heuristics 2)
3. The *optimal* heuristics: at every step, we compute the window T giving the maximal speed-up factor θ_{best}.[5]

We generated random graphs with various heavy tailed[6] degree sequences, using a wide set of parameters, and all the results were consistent with our analysis: θ_{Gkan} behaved asymptotycally like $\sqrt{\theta_{best}}$, and our average speed-up factor θ_{new} always reached at least 90% of the optimal θ_{best}. Some typical results are shown below.

[5] The heavy cost of this prohibits its use, as a heuristics. It only serves as a reference.

[6] We used power-law like distributions: $P(X = k) = (k + \mu)^{-\alpha}$, where α represents the "heavy tail" behavior, while μ can be tuned to obtain the desired average z.

These experiments show that our new heuristics is very close to the optimal. Thus, despite the fact that p actually varies during the shuffle, our heuristics react fast enough (in regard to the variations of p) to get a good, if not optimal, window T. We therefore obtain a success rate $r(T)$ in a close range around e^{-1}.

These empirical evidences confirm the validity of our formal approach. We obtained a total complexity $C_{new} = O(m + \mathbf{p} \cdot m^2)$, instead of the already improved $C_{Gkan} = O(m + \sqrt{\mathbf{p}} \cdot m^2)$. Despite the fact that it is asymptotically still outperformed by the complexity of the dynamic connectivity algorithm $C_{dynamic}$ (see Eq. 2), C_{new} may be smaller in practice if p is small enough. For many graph topologies corresponding to real-world networks, especially the dense ones (like social relations, word co-occurences, WWW), and therefore a low disconnection probability, our algorithm represents an alternative that may behave faster, and which implementation is much easier.

4 A log-linear algorithm ?

We will now show that in the particular case of heavy-tailed degree distributions like the ones met in practice [5], one may reduce the disconnection probability p at logarithmic cost, thus reducing dramatically the complexity of the connectivity tests.

Guiding principle

In a graph with a heavy-tailed degree distribution, most vertices have a low degree. This means in particular that, by swapping random edges, one may quite easily create very small isolated component. Conversely, the non-negligible number of vertex of high degree form a robust core, so that it is very unlikely that a random swap creates two large disjoint components.

Definition 4 (Isolation test) *An isolation test of width K on vertex v tests wether this vertex belongs to a connected component of size lower than or equal to K.*

$\alpha = 2.5$

z	θ_{Gkan}	θ_{new}	θ_{best}
2.1	0.79	0.88	0.90
3	3.00	5.00	5.19
6	20.9	112	117
12	341	35800	37000

$\alpha = 3$

z	θ_{Gkan}	θ_{new}	θ_{best}
2.1	1.03	1.20	1.26
3	5.94	12.3	12.4
6	32.1	216	234
12	578	89800	91000

Table 1. Average speed-up factors for various values of the average degree z, and for graphs of size $n = 10^4$

To avoid the disconnection, we will now perform an isolation test after every transition. If this isolation test returns `true`, we cancel the swap rightaway. This way, we detect, at low cost $O(K)$, a significant part of the disconnections.
The disconnection probability p is now the probability that after T swaps *which passed the isolation test*, the graph gets disconnected. It is straightforward to see that p is decreasing with K; more precisely, strong empirical evidences and formal arguments[4] led us to the following conjecture:

Conjecture 1 *The disconnection probability p for random connected graphs with heavy-tailed degree distributions decreases* **exponentially** *with K: $p(K) = O(e^{-\lambda K})$ for some positive constant λ depending on the distribution, and not on the size of the graph.*

The final algorithm

Let us introduce the following quantity:

Definition 5 (Characteristic isolation width) *The characteristic isolation width K_G of a graph G having m edges is the minimal* isolation test width *K such that the disconnection probability $p(K)$ verifies $p(K) < 1/m$.*

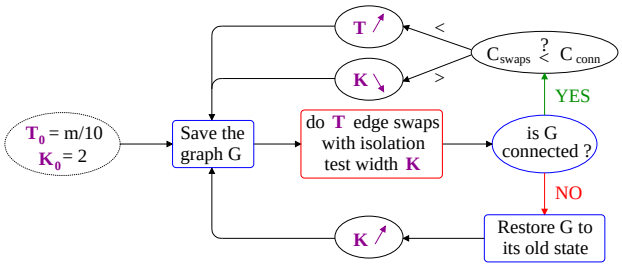

Fig. 2. Our final heuristics used to adjust K and T

Now, we can apply the shuffle process, as seen before, but with a window $T = \Theta(m)$, and an isolation test width K equal to K_G. From Conjecture 1 and the definition of K_G, we deduce that this process will perform $\Omega(m)$ swaps in $O(m \log n)$ time. The difficulty might be to guess K_G. We showed[4] that the heuristics shown in Figure 2 solves this: it aims at equilibrating C_{swaps} and C_{conn} by dynamically adjusting K and T, looking for a high success rate $r(K, T)$ and keeping a large window $T = \Omega(m)$.
We compare in Table 2 typical running times with the naive algorithm, the Gkantsidis et al. heuristics, our improved version of this heuristics, and our final algorithm. Implementations are provided at [18].

m	Naive	Gkan. heur.	Heuristics 2	Final algo.
10^3	0.51s	0.02s	0.02s	0.02s
10^4	26.9s	1.15s	0.47s	0.08s
10^5	3200s	142s	48s	1.1s
10^6	$\approx 4{\cdot}10^5$s	$\approx 3{\cdot}10^4$s	10600s	25.9s
10^7	$\approx 4{\cdot}10^7$s	$\approx 3{\cdot}10^6$s	$\approx 10^6$s	420s

Table 2. Average time for the generation of graphs of various sizes with the same heavy-tailed degree distribution ($\alpha = 2.5$, $z = 6.7$) on a Centrino 1.5GHz with 512MB RAM.

Towards a $O(m \log\log n)$ algorithm ?

The isolation tests are typically breadth- or depth-first searches that stop when they have visited $K+1$ vertices, or when then have explored a component of size S lower than K. In the latter case, Conjecture 1 ensures[4] that the expectation of S is $< S >= O(1)$, so that the average complexity of the isolation test was also $O(1)$. Taking advantage of the heavy-tailed degree distribution, we may be able to reduce as well the complexity of the isolation tests that do not detect a disconnection.
The idea is simple: if the search meets a vertex of degree greater than K, it can stop, because it means that the component's size is also greater than K. Several recent results indicate that searching a vertex of degree at least K in an heavy-tailed network takes $O(\log K)$ steps in average [16,1], if the search is a depth-first search that always goes to the unvisited neighbour of highest degree. Thus, running an isolation test would be done in $O(\log K)$ average time instead of $O(K)$. Finally, the global complexity would become $O(m \log\log n)$ time.

5 Conclusion

Focusing on the speed-up method introduced by Gkantsidis et al. for the Markov chain Monte Carlo algorithm, we introduced a formal background allowing us to show that this heuristics is not optimal in its own family. We improved it in order to reach the optimal, and empirically confirmed the results.
Going further, we then took advantage of the characteristics of real-world networks to introduce an original method allowing the generation of random simple connected graphs with heavy-tailed degree distributions in $O(m \log n)$ time and $O(m)$ space. It outperforms the previous best known methods, and has the advantage of being extremly easy to implement. We also have pointed directions for further enhancements to reach a complexity of $O(m \log\log n)$ time. The empirical measurement of the performances of our methods show that it yields significant progress. We provide an implementation of this last algorithm [18].

Notice however that the last results rely on a conjecture, for which we have several arguments and strong empirical evidences, but were unable to prove.

References

1. Adamic, Lukose, Puniyani, and Huberman. Search in power-law networks. *Phys. Rev. E*, 64(046135), 2001.
2. Aiello, Chung, and Lu. A random graph model for massive graphs. *Proc. of the 32nd ACM STOC*, pages 171–180, 2000.
3. C. Berge. *Graphs and Hypergraphs*. North-Holland, 1973.
4. Erdos and Gallai. Graphs with prescribed degree of vertices. *Mat. Lapok*, 11:264–274, 1960.
5. Faloutsos, Faloutsos, and Faloutsos. On power-law relationships of the internet topology. *Proc. ACM SIGCOMM*, 29:251–262, 1999.
6. Gkantsidis, Mihail, and Zegura. The markov chain simulation method for generating connected power law random graphs. *in proc. ALENEX*, 2003.
7. S. L. Hakimi. On the realizability of a set of integers as degrees of the vertices of a linear graph. *SIAM Journal*, 10(3):496–506, 1962.
8. V. Havel. A remark on the existence of finite graphs. *Caposis Pest. Mat.*, 80:496–506, 1955.
9. Henzinger and King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. of ACM*, 46(4), 1999.
10. Holm, de Lichtenberg, and Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *STOC'98*, pages 79–89, 1998.
11. J. M. Roberts Jr. Simple methods for simulating sociomatrices with given marginal totals. *Social Networks*, 22:273–283, 2000.
12. Milo, Kashtan, Itzkovitz, Newman, and Alon. Uniform generation of random graphs with arbitrary degree seq. *sub. Phys. Rev. E*, 2001.
13. Molloy and Reed. A critical point for random graphs with a given degree sequence. *Random Struct. and Algo.*, pages 161–179, 1995.
14. Molloy and Reed. The size of the giant component of a random graph with a given degree sequence. *Comb., Prob. and Comp.*, 7:295, 1998.
15. R.Taylor. Constrained switchings in graphs. *Comb. Mat. 8*, 1980.
16. Sarshar, Boykin, and Roychowdhury. Scalable percolation search in power law networks. *P2P'04*, pages 2–9.
17. M. Thorup. Near-optimal fully-dynamic graph connectivity. *Proc. of the 32nd ACM STOC*, pages 343–350, 2000.
18. www.liafa.jussieu.fr/∼fabien/generation.