First publ. in: Physical Review E 71 (2005), 3, Article 036113

Efficient Generation of Large Random Networks*

Vladimir Batagelj†
Department of Mathematics, University of Ljubljana, Slovenia.

Ulrik Brandes‡
Department of Computer & Information Science, University of Konstanz, Germany.

Random networks are frequently generated, for example, to investigate the effects of model parameters on network properties or to test the performance of algorithms. Recent interest in statistics of large-scale networks sparked a growing demand for network generators that can generate large numbers of large networks quickly. We here present simple and efficient algorithms to randomly generate networks according to the most commonly used models. Their running time and space requirement is linear in the size of the network generated, and they are easily implemented.

I. INTRODUCTION

Recent studies of complex systems such as the Internet, biological networks, river basins, or social networks, have significantly increased the interest in modeling classes of graphs. While random graphs have been studied for a long time, the standard models appear to be inappropriate because they do not share the characteristics observed for those systems. A plethora of new models is therefore being proposed, but many of them are variations of the small-world model [1] or the preferential attachment model [2].

For empirical evaluation of models and algorithms, it is important to be able to quickly generate graphs according to these models. To our surprise we have found that the algorithms used for these generators in software such as BRITE [15], GT-ITM [16], JUNG [17], or LEDA [3] are rather inefficient. Note that superlinear algorithms are sometimes tolerable in the analysis of networks with tens of thousands of nodes, but they are clearly unacceptable for generating large numbers of such graphs.

We here show that standard random graphs, small worlds and preferential attachment graphs can be generated efficiently by presenting generators that are asymptotically optimal both in terms of running time and space requirements, parsimonious in their use of a random number generator, simple to implement, and easily adaptable for model variations. A specific set of implementations is available in Pajek [4].

---

*A preliminary version of this paper was presented at the Sunbelt XXIV Social Network Conference, 12–16 May 2004, Portorož/Slovenia.

†`vladimir.batagelj@uni-lj.si`; Research partially supported by the Ministry of Education, Science and Sport of Slovenia (project J1-8532).

‡Corresponding author, `ulrik.brandes@uni-konstanz.de`; Research partially supported by Deutsche Forschungsgemeinschaft (grant Br 2158/1-2) and the European Commission (FET open project COSIN, grant IST-2001-33555).

II. RANDOM GRAPHS

The now classic models of random graphs were defined to study properties of 'typical' graphs among those with a given number of vertices. The model most commonly used for this purpose was introduced by Gilbert [5]. In Gilbert's model, $\mathcal{G}(n,p)$, the $\frac{n(n-1)}{2}$ potential edges of a simple undirected graph $G(n,p) \in \mathcal{G}(n,p)$ with n vertices are included independently with probability $0 < p < 1$. This edge probability is usually chosen dependent on the number of vertices, i.e. $p = p(n)$.

Note that the number of edges of a graph created according to the $\mathcal{G}(n,p)$ model is not known in advance. The closely related model $\mathcal{G}(n,m)$, in which all simple undirected non-isomorphic graphs with n vertices and *exactly* $0 \le m \le \frac{n(n-1)}{2}$ edges are equiprobable, was introduced by Erdős and Rényi [6]. A variant model, $\mathcal{G}^*(n,m)$, in which parallel edges are allowed, was first studied by Austin et al. [7].

Yet another variant are random graphs with given degree sequence. For a recent discussion of generators for this model we refer the interested reader to a recent article in this journal [8].

A. $\mathcal{G}(n,p)$

Common generators for graphs in the $\mathcal{G}(n,p)$ model draw, independently for each potential edge, a number $r \in [0,1)$ uniformly at random, and insert the edge if $r < p$. Clearly, both the running time and the number of random experiments are in $\Theta(n^2)$, independent of the number of edges actually generated. This method is hence not suitable to generate large sparse graphs.

The number of edges of a graph $G(n,p)$ is binomially distributed with mean $p \cdot \binom{n}{2} = p \cdot \frac{n(n-1)}{2}$. To generate sparse graphs, the edge probability is chosen such that $p(n) \in o(1)$ and therefore most trials of the above generator will be unsuccessful. We hence use the *geometric method* of [9] to skip over potential edges that are not

created. Note that, at any given time during the iteration, the probability of generating the next edge only after k trials is

$$(1-p)^{k-1}p \ ,$$

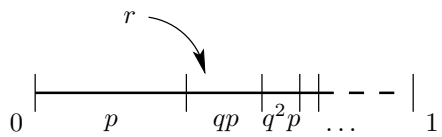
i.e. waiting times are geometrically distributed. In the following, let $q = 1-p$. To sample waiting times, each positive integer k is assigned an interval $I_k \subseteq [0,1)$ of length $q^{k-1}p$ and we have

$$\sum_{k=1}^{\infty} q^{k-1}p \ = \ p \cdot \sum_{k=0}^{\infty} q^k \ = \ p \cdot \frac{1}{1-q} \ = \ 1 \ .$$

If the intervals $I_1, I_2, I_3, \ldots$ are consecutive starting at 0, interval I_k ends at

$$\sum_{i=1}^{k} q^{i-1}p \ = \ p \cdot \sum_{i=0}^{k} q^i \ = \ p \cdot \frac{1-q^k}{1-q} \ = \ 1-q^k \ ,$$

so that waiting times can be sampled by selecting the smallest k for which I_k ends after a randomly chosen $r \in [0,1)$.



Note that

$$r < 1-q^k \iff k > \frac{\log(1-r)}{\log q} \ ,$$

so that we choose $k = 1 + \left\lfloor \frac{\log(1-r)}{\log q} \right\rfloor$.

ALG. 1: $\mathcal{G}(n,p)$

Input: number of vertices n, edge probability $0 < p < 1$
Output: $G = (\{0,\ldots,n-1\}, E) \in \mathcal{G}(n,p)$

```
E ← ∅
v ← 1;   w ← −1
while v < n do
    draw r ∈ [0,1) uniformly at random
    w ← w + 1 + ⌊log(1−r) / log(1−p)⌋
    while w ≥ v and v < n do
        w ← w − v; v ← v + 1
    if v < n then E ← E ∪ {v, w}
```

Pseudo-code for the algorithm is given in Alg. 1. Note that the set of candidate edges is enumerated in lexicographic order, which corresponds to a row-wise traversal of the lower half of the adjacency matrix. An interesting interpretation is that of an evolving graph, since the traversal of each row corresponds to the creation of a new vertex which is assigned edges to already existing vertices.

Observe that each but the last execution of the outer loop yields an edge, and that the total number of executions of the inner loop is equal to the number of vertices, so that Alg. 1 generates a random graph in the $\mathcal{G}(n,p)$ model in $\mathcal{O}(n+m)$ time (where m is the number of edges generated), which is optimal.

The algorithm is easily modified to generate directed, directed acyclic, bipartite, or any other class of graphs for which the set of candidate edges can be indexed so that any edge is obtained from its index in constant time.

B. $\mathcal{G}(n,m)$

In principle, the geometric method can be used to generate graphs according to $\mathcal{G}(n,m)$ as well. However, in this model, skipping probabilities are not independent of the current state of the algorithm. Assume that we have already considered $t-1$ candidate edges, and selected l for inclusion in the graph. Then, the probability of skipping the next $k-1$ candidates is

$$\prod_{i=t}^{t+k-1} \left(1 - \frac{m-l}{\binom{n}{2}-i+1}\right) \frac{m-l}{\binom{n}{2}-t+k} \ .$$

Though fast methods to sample k from this distribution are available [10, 11], they are rather complicated and require more than constant time per edge.

We next describe two different methods for sampling from $\mathcal{G}(n,m)$ that are both simple and efficient. While one is more suitable for sparse graphs, the other should be used when denser graphs need to be generated.

A straightforward approach for sparse graphs is to iteratively pick a random candidate edge and create it or retry if it has been created in a previous step. Using an efficient hashing scheme, the test whether an edge already exists can be carried out in constant expected time. See Alg. 2 for pseudo-code.

ALG. 2: $\mathcal{G}(n,m)$ in linear expected time

Input: number of vertices n
number of edges $0 \le m \le \binom{n}{2}$
Output: $G = (\{0,\ldots,n-1\}, E) \in \mathcal{G}(n,m)$

```
E ← ∅
for i = 0, ..., m − 1 do
    repeat
        draw r ∈ {0, ..., (n choose 2) − 1} uniformly at random
    until e_r ∉ E
    E ← E ∪ {e_r}
```

The running time of this method is probabilistic. After choosing i edges, the probability of sampling a new edge in the next step is $\frac{\binom{n}{2}-i}{\binom{n}{2}}$, so that the expected number of trials until this actually happens is $\frac{\binom{n}{2}}{\binom{n}{2}-i}$. The expected

total running time is therefore

$$\sum_{i=1}^{m} \frac{\binom{n}{2}}{\binom{n}{2}-i} = \binom{n}{2} \cdot \left(\sum_{i=1}^{\binom{n}{2}} \frac{1}{i} - \sum_{i=1}^{\binom{n}{2}-m} \frac{1}{i} \right) \in \Theta\left(\binom{n}{2} \log \frac{\binom{n}{2}}{\binom{n}{2}-m+1} \right)$$

which is $\mathcal{O}(m)$ for $m \leq \frac{1}{2}\binom{n}{2}$. Since this bound gives $\mathcal{O}(m \log m)$ expected running time for $m = \binom{n}{2}$, we rather generate a complete graph with $\binom{n}{2}$ edges and randomly delete $\binom{n}{2} - m + 1$ of them when $m > \frac{1}{2}\binom{n}{2}$. We thus have linear expected running time in all cases.

An alternative eliminating the uncertainty in the number of trials is a virtual Fisher-Yates shuffle. In a standard Fisher-Yates shuffle [12], all candidate elements are stored in an array. After an element is picked at random, it is swapped with the first element, and the interval of elements considered is shortened by one, so that the first one can not be picked again. However, when generating sparse graphs with $m \in o(\binom{n}{2})$, we cannot afford to have an array of size $\binom{n}{2}$.

To implement a virtual shuffle, we number all edges from $1, \ldots, \binom{n}{2}$ and when an edge is created in the ith step, we associate the index of its swapping counterpart with the edge's entry in the hashtable storing the set of created edges. If the edge is later picked again, we create its replacement instead and assign a new replacement edge. Note that the replacement itself is already outside of the random choice interval and will therefore not be picked again. See Alg. 3 for pseudo-code.

ALG. 3: $\mathcal{G}(n, m)$ in linear worst-case time

Input: number of vertices n
number of edges $0 \leq m \leq \binom{n}{2}$
Output: $G = (\{0, \ldots, n-1\}, E) \in \mathcal{G}(n, m)$

$E \leftarrow \emptyset$
for $i = 0, \ldots, m-1$ **do**
  draw $r \in \{i, \ldots, \binom{n}{2} - 1\}$ uniformly at random
  if $e_r \notin E$ **then**
    $E \leftarrow E \cup \{e_r\}$
  else $E \leftarrow E \cup \{e_{\text{replace}[e_r]}\}$
  if $e_i \notin E$ **then**
    replace$[e_r] \leftarrow i$
  else replace$[e_r] \leftarrow$ replace$[e_i]$

Note that the set of candidate edges of a simple undirected graph with vertices $\{0, \ldots, n-1\}$ is easily numbered using the bijection

$$e_i = \{v, w\} \quad \text{with} \quad v = 1 + \left\lfloor -\frac{1}{2} + \sqrt{\frac{1}{4} + 2i} \right\rfloor$$
$$w = i - \frac{v(v-1)}{2}$$

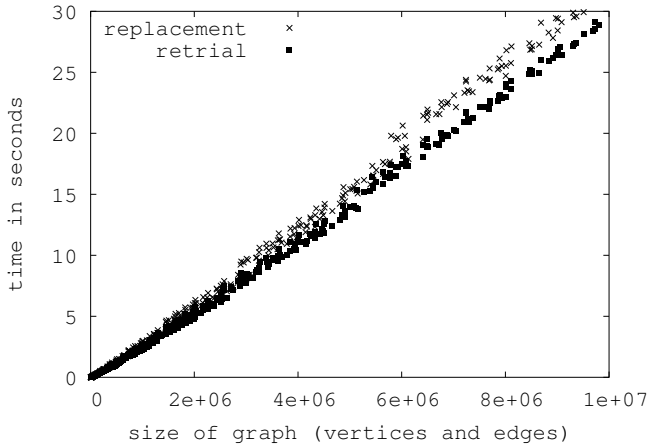


FIG. 1: The simple method is slightly faster in practice

for $i = 0, \ldots, \binom{n}{2} - 1$, i.e. $i = \frac{v(v-1)}{2} + w$ for $0 \leq w < v \leq n-1$.

An experimental evaluation showed, however, that for likely choices of parameters, the simple method based on retrials is slightly better in practice than the replacement method. In Fig. 1, running times are given for the generation of graphs with 1000–20000 vertices and densities varying from 0%–25%. The experiments have been performed on a standard laptop computer.

Again, the methods easily generalize to other classes of graphs such as directed or bipartite graphs.

III. SMALL WORLDS

Watts and Strogatz [1] introduced a popular model for sparse undirected graphs called *small worlds*. The model is designed to exhibit properties found in empirical graphs that are unlikely to be present in random graphs with the same number of vertices and edges. The two defining properties are strong local clustering as measured by the *clustering coefficient*, i.e. the average density of the neighborhood of vertices, and short average distances between pairs of vertices, sometimes called *characteristic path length*.

The basic idea is to start from a fixed graph with local clustering, but large characteristic path length, and to randomly rewire a fraction of the edges to serve as shortcuts. More precisely, we start with a ring of n vertices and connect each of them with its $2d$ nearest neighbors, where $1 \leq d \leq \frac{n-1}{2}$ is an integer constant. Graphs in this family have a relatively high clustering coefficient close to $\frac{3}{4}$, but also a high characteristic path length of about $\frac{n}{2r}$.

A straightforward generator creates, in linear time, the dth power of an n-cycle, and decides for each of the nd edges whether to replace it with a random edge or not. In fact, this algorithm faces to same two obstacles discussed in the previous section: a large number of unsuccessful

random trials and the creation of loops or parallel edges (or the need for retrials). Since we probe only edges that have been created, the first problem does not result in an increased asymptotic running time, and the second problem is less severe, either, since the expected few loops and parallel edges have an insignificant effect on the properties of the graphs generated.

Nevertheless, for demonstration we combine the techniques introduced in the previous section to eliminate both problems. Pseudo-code for a generator that generates small worlds without loops or multiple edges by substituting some edges of the starting graph with random edges is given in Alg. 4, other variants can be obtained from straightforward modifications.

ALG. 4: small worlds

Input: number of vertices n
degree parameter $1 \leq d \leq \lfloor \frac{n-1}{2} \rfloor$
replacement probability $0 \leq p < 1$
Output: small world $G = (\{0, \ldots, n-1\}, E)$

```
E ← ∅
draw r ∈ [0,1) uniformly at random
k ← ⌊log(1−r)/log(1−p)⌋ ; m ← 0
for v = 0,...,n−1 do
    for i = 1,...,d do
        if k > 0 then
            j ← v(v−1)/2 + (v + i mod n)
            E ← E ∪ {e_j}
            k ← k − 1; m ← m + 1
            if e_m ∉ E then
                replace[e_j] ← m
            else
                replace[e_j] ← replace[e_m]
        else
            draw r ∈ [0,1) uniformly at random
            k ← ⌊log(1−r)/log(1−p)⌋
for i = m+1,...,nd do
    draw r ∈ {i,...,(n choose 2)} uniformly at random
    if e_r ∉ E then
        E ← E ∪ {e_r}
    else E ← E ∪ {e_replace[e_r]}
    if e_i ∉ E then
        replace[e_r] ← i
    else replace[e_r] ← replace[e_i]
```

An interesting option for small worlds with only a little randomness is the implicit representation of non-random edges. Note that two vertices are adjacent in the augmented cycle, if and only if their indices differ by at most d. Therefore we do not need to store these edges, but only exceptions caused by random replacement. The adjacency list of a vertex v in a space-efficient small-world data structure thus consist of two parts: an alternating sequence of interval lengths (the number of consecutive neighbors resp. non-neighbors in $v-d, \ldots, v-1, v+1, \ldots, v+d$, all modulo n) and the list of random edges incident to v. Note that the space needed to store all edges is proportional to the number of random edges in the graph.

IV. PREFERENTIAL ATTACHMENT

In random graphs according to $\mathcal{G}(n,p)$, $\mathcal{G}(n,m)$, or the small world model, the degree distribution is sharply concentrated around its mean. In many empirical networks, however, it roughly obeys a power-law, i.e. the number of vertices with degree d is proportional to $d^{-\gamma}$ for some constant $\gamma \geq 1$ [13].

Barabási and Albert [2] describe a process of *preferential attachment* that generates graphs with this property. The graph is created one vertex at a time, and each newly created vertex is attached to a fixed number d of already existing vertices. The probability of selecting a specific neighbor is proportional to its current degree.

We here implement the precise model of Bollobás et al. [14] in which the ambiguity of how to select a set of $d > 1$ neighbors is resolved in the following way. Assume $d = 1$, then the ith vertex is attached to the jth vertex, $j \leq i$, with probability $\frac{d(j)}{m(i)+1}$, if $j < i$, and $\frac{1}{m(i)+1}$, if $i = j$, where $d(j)$ is the current degree of vertex j and $m(i) = \sum_{j=0}^{i-1} d(j)$ is twice the number of edges already created. Note that this process generates a forest. For $d > 1$ the graph evolves as if $d = 1$ until nd vertices have been created, and then intervals of d consecutive vertices are contracted into one. Clearly, this process may yield loops and multiple edges.

Current generators such as BRITE and JUNG recompute prefix sums of degrees to sample neighbors. This is highly inefficient and leads to implementations with at least quadratic running time. Observe that in a list of all edges created thus far, the number of occurrences of a vertex is equal to its degree, so that it can be used as a pool to sample from the degree-skewed distribution in constant time. In Alg. 5 such an edge list is represented by array M in which pairs of vertices represent the edges. Both running time and space requirement are $\mathcal{O}(n+m)$, i.e. linear in the size of the graph generated.

Again, a number of variants is obtained from straightforward modification. Note that we can initialize M with the edges of any seed graph G_0, and that k-partite graphs can be generated in the same way by selecting from different pools. For directed graphs, the neighbor-selection probability can be decomposed into the weighted sum of in-, out- and total degree. In the implementation, we let a three-sided biased dice decide which term is relevant and then sample from odd, even or all positions in M. If the number of edges to add for a new vertex is not fixed, the geometric method of Section II, e.g., can be used to sample neighbors from the prefix of M filled up to now. An example with bipartite preferential attachment is given in Alg. 6.

ALG. 5: preferential attachment

Input: number of vertices n
minimum degree $d \geq 1$

Output: scale-free multigraph
$G = (\{0, \ldots, n-1\}, E)$

M: array of length $2nd$
for $v = 0, \ldots, n-1$ **do**
  for $i = 0, \ldots, d-1$ **do**
    $M[2(vd+i)] \leftarrow v$
    draw $r \in \{0, \ldots, 2(vd+i)\}$ uniformly at random
    $M[2(vd+i)+1] \leftarrow M[r]$
$E \leftarrow \emptyset$
for $i = 0, \ldots, nd-1$ **do**
  $E \leftarrow E \cup \{M[2i], M[2i+1]\}$

ALG. 6: bipartite preferential attachment

Input: number of vertices in each set n
minimum degree $d \geq 1$

Output: bipartite scale-free multigraph
$G = (\{0, \ldots, n-1\} \dot{\cup} \{n, \ldots, 2n-1\}, E)$

M_1, M_2: arrays of length $2nd$
for $v = 0, \ldots, n-1$ **do**
  for $i = 0, \ldots, d-1$ **do**
    $M_1[2(vd+i)] \leftarrow v$
    $M_2[2(vd+i)] \leftarrow n+v$
    draw $r \in \{0, \ldots, 2(vd+i)\}$ uniformly at random
    if r *even* **then** $M_1[2(vd+i)+1] \leftarrow M_2[r]$
    else $M_1[2(vd+i)+1] \leftarrow M_1[r]$
    draw $r \in \{0, \ldots, 2(vd+i)\}$ uniformly at random
    if r *even* **then** $M_2[2(vd+i)+1] \leftarrow M_1[r]$
    else $M_2[2(vd+i)+1] \leftarrow M_2[r]$
$E \leftarrow \emptyset$
for $i = 0, \ldots, nd-1$ **do**
  $E \leftarrow E \cup \{M_1[2i], M_1[2i+1]\} \cup \{M_2[2i], M_2[2i+1]\}$

[1] D. J. Watts and S. H. Strogatz, Nature **393**, 440 (1998).
[2] A.-L. Barabási and R. Albert, Science **286**, 509 (1999).
[3] K. Mehlhorn and S. Näher, *The LEDA Platform of Combinatorial and Geometric Computing* (Cambridge University Press, 1999), URL `http://www.mpi-sb.mpg.de/LEDA/`.
[4] V. Batagelj and A. Mrvar, in *Graph Drawing Software*, edited by M. Jünger and P. Mutzel (Springer, 2003), Mathematics and Visualization, pp. 77–103, URL `http://vlado.fmf.uni-lj.si/pub/networks/progs/random/`.
[5] E. N. Gilbert, Annals of Mathematical Statistics **30**, 1141 (1959).
[6] P. Erdős and A. Rényi, Publicationes Mathematicae Debrecen **6**, 290 (1959).
[7] T. L. Austin, R. E. Fagen, W. E. Penney, and J. Riordan, Annals of Mathematical Statistics **30**, 747 (1959).
[8] R. Milo, N. Kashtan, S. Itzkovitz, M. E. Newman, and U. Alon (2003), cond-mat/0312028v1.
[9] C. Fan, M. E. Muller, and I. Rezucha, Journal of the American Statistical Association **57**, 387 (1962).
[10] J. S. Vitter, ACM Transactions on Mathematical Software **13**, 58 (1987).
[11] K. A. Nair, ACM Transactions on Mathematical Software **16**, 269 (1990).
[12] R. A. Fisher and F. Yates, *Statistical Tables for Biological, Agricultural and Medical Research* (Oliver & Boyd, 1938).
[13] M. Faloutsos, P. Faloutsos, and C. Faloutsos, in *Proceedings of SIGCOMM'99* (1999), pp. 251–262.
[14] B. Bollobás, O. M. Riordan, J. Spencer, and G. Tusnády, Random Structures and Algorithms **18**, 279 (2001).
[15] Boston University Representative Internet Topology Generator, URL `http://cs-www.bu.edu/brite/`
[16] Georgia Tech Internetwork Topology Models, URL `http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html`
[17] Java Universal Network/Graph Framework, URL `http://jung.sourceforge.net/`