Efficient Generation of Networks with Given Expected Degrees

Joel C. Miller[1,2] and Aric Hagberg[3]

[1] Center for Communicable Disease Dynamics, Harvard School of Public Health, Boston, MA 02115, USA
[2] Fogarty International Center, National Institute of Health, Bethesda, MD 20892, USA
joel.c.miller.research@gmail.com
[3] Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA
hagberg@lanl.gov

Abstract. We present an efficient algorithm to generate random graphs with a given sequence of expected degrees. Existing algorithms run in $\mathcal{O}(N^2)$ time where N is the number of nodes. We prove that our algorithm runs in $\mathcal{O}(N+M)$ expected time where M is the expected number of edges. If the expected degrees are chosen from a distribution with finite mean, this is $\mathcal{O}(N)$ as $N \to \infty$.

1 Introduction

Random graph models are regularly used for studying random processes on networks. Such processes include rumor spreading or the spread of epidemics on social networks [15,13,9], and web searchers seeking information on the World Wide Web [4,14]. Capturing the degree distribution is one of the most important goals in creating a model and some well-understood graph models have been developed where the degree distribution is controlled. In this paper we focus on the model of Chung and Lu [5] which generates a random graph with a given sequence of expected degrees.

Because application networks are often very large, efficient random graph generation is important to evaluate the models and processes. Although many theoretical results are known about the Chung-Lu model [2,12,5,6,7,8], the algorithms used for generating such graphs are inefficient. In this paper we introduce a new algorithm which generates Chung-Lu random graphs with expected runtime that is $\mathcal{O}(N + M)$ where N the number of nodes and M is the expected number of edges. For sequences with finite average degree $M = \mathcal{O}(N)$ and the expected runtime is $\mathcal{O}(N)$, a significant improvement over previous algorithms that require $\mathcal{O}(N^2)$ runtime.

We begin by showing a fast algorithm for generating Erdős–Rényi graphs. This approach is by itself not a significant result (indeed closely related algorithms exist already [1]), but it serves as an example to motivate our algorithm for generating Chung-Lu graphs which uses an additional rejection sampling step.

A. Frieze, P. Horn, and P. Prałat (Eds.): WAW 2011, LNCS 6732, pp. 115–126, 2011.

Once we introduce the main algorithm, we prove that it generates Chung-Lu graphs and that its expected runtime is $\mathcal{O}(N+M)$. We conclude with a discussion of other applications and implications of this algorithm.

1.1 Model Description

The basic random graph model is the Erdős–Rényi graph $G(N,p)$ with N nodes $0, 1, \ldots, N-1$. Each pair of nodes has an edge with probability p, and edges between a pair of nodes are assigned independently of one another. The expected degree of a node is $\kappa = p(N-1)$. Typically we consider graphs for which κ is $\mathcal{O}(1)$ as $N \to \infty$, so p is small and the graph is sparse. Under such conditions, the expected degree approaches pN as $N \to \infty$. The obvious algorithm to generate $G(N,p)$ graphs requires considering each of the $\mathcal{O}(N^2)$ pairs of nodes independently and assigning an edge with probability p. However, it has been shown that the runtime can be reduced to $\mathcal{O}(N+M)$ where N is the number of nodes and M the number of edges [1].

The degree distribution resulting from the Erdős–Rényi model is binomial, and in the large N, constant pN limit it approaches a Poisson distribution with mean pN. Many real world graphs have much more pronounced heterogeneity in degrees [16,2]. This has led to models which attempt to incorporate more heterogeneity. Some of these models retain independence of edges but allow nodes to have different expected degrees.

Of these models the most prominent was introduced by Chung and Lu [6]. In the Chung-Lu model each node u is assigned a weight w_u which we can assume is chosen from a distribution with density ρ. We do not need to restrict w_u to be an integer. We define $\overline{w} = \sum_u w_u/N$ to be the average weight. Two nodes u and v with weights w_u and w_v are then joined by an edge with probability $p_{u,v} = w_u w_v/N\overline{w}$. Looking at all nodes $v \neq u$, we anticipate that the expected degree of u is $\sum_{v\neq u} w_u w_v/N\overline{w} = w_u - w_u^2/N\overline{w}$. In a large graph this typically converges to w_u. So the weight w_u represents the expected degree of node u in a large graph. A number of theoretical results are known about this model [2,12,6,8,7]. For a Chung-Lu model graph, we will use $M = N\overline{w}/2$ as our estimate for the expected number of edges. This is an upper bound, but usually this is only a small effect.

We note that it is possible that $w_u w_v/N\overline{w} > 1$. In this case, we set $p_{u,v} = 1$, and the expected degree of u will be less than $w_u - w_u^2/N\overline{w}$. Other approaches have been introduced to produce related graphs that avoid this difficulty. For example, we could define $p_{u,v} = 1 - \exp(-w_u w_v/N\overline{w})$ or $p_{u,v} = w_u w_v/(N\overline{w} + w_u w_v)$ [17,3,12]. Typically as N grows the difference between these approaches is negligible because $w_u w_v/N \to 0$ and thus the leading order terms for $p_{u,v}$ are the same.[1] We will focus our attention on the Chung-Lu graphs, though our algorithm can be easily translated to the others. A fast algorithm to generate approximate Chung-Lu graphs in the bipartite case appears in Ref. [10].

[1] In distributions for which the average squared weight is infinite, but the average weight is finite, $w_u w_v/N$ may not tend to zero for the highest weight nodes, in which case these models differ.

2 Erdős–Rényi Case

We begin by describing our algorithm in a simpler context. The Erdős–Rényi network is a special case of Chung-Lu networks in which all nodes have the same weight w and therefore $p_{u,v} = p = w/N$ for all pairs u and v. We first describe the obvious inefficient $\mathcal{O}(N^2)$ algorithm, and show how it can be naturally sped up to an algorithm that is $\mathcal{O}(N + M)$.

Let the the nodes be numbered $0, 1, \ldots, N-1$ and begin by setting $u = 0$. Then for each $v = 1, 2, \ldots, N-1$ we generate a random number r. If $r < p$, then we place an edge from u to v. Once all possible choices for v have been considered, we set $u = 1$, and then consider each $v = 2, 3, \ldots, N-1$. We continue this process until all possible choices for u have been considered. This algorithm is $\mathcal{O}(N^2)$ because it considers each pair of nodes separately.

This algorithm is slow because considerable effort is spent on node pairs which never form edges. The algorithm can be sped up by skipping these pairs. Returning to the first pass through the algorithm described above with $u = 0$, let v_1 be the first neighbor with which u forms an edge. The value of v_1 is $u + 1 + \delta$ where δ is the number of pairs considered that do not form edges before the first successful edge formation. Similarly, the second neighbor v_2 is $v_1 + 1 + \delta$ where δ is the new number of pairs that do not form edges. The probability of a particular value of δ is $(1-p)^\delta p$ (in fact, δ is geometrically distributed). Thus, rather than considering every node after u as above, we can find the next neighbor in a single step by choosing r uniformly in $(0, 1)$ and setting $\delta = \lfloor \ln r / \ln(1-p) \rfloor$, taking $\delta = 0$ if $p = 1$. The full procedure for $p < 1$ is presented in Algorithm 1.

Algorithm 1. $G(N, p)$ Graph

Input: number of nodes N, and probability $0 < p < 1$
Output: $G(N, p)$ graph $G(V, E)$ with $V = \{0, \ldots, N-1\}$

```
E ← ∅
for u = 0 to N − 2 do
    v ← u + 1
    while v < N do
        choose r ∈ (0, 1) uniformly at random
        v ← v + ⌊log(r) / log(1 − p)⌋
        if v < N then
            E ← E ∪ {u, v}
            v ← v + 1
```

Theorem 1. *Algorithm 1 runs in $\mathcal{O}(N + M)$ time where M is the number of edges in the output graph and N the number of nodes.*

Proof. The proof of this theorem is relatively straightforward. We simply count the number of times that each loop executes.

The outer loop is executed $N-1$ times. To calculate the number of executions of the inner loop we separate those "successful" iterations where the new v is at most $N-1$ from those "unsuccessful" iterations with $v \geq N$. In each pass through the outer loop, the inner loop executes once unsuccessfully. The total number of successful executions of the inner loop is exactly the number of edges that are generated which is $\mathcal{O}(M)$. Combining the total number of passes through the outer loop with the number of successful and unsuccessful passes through the inner loop the runtime is $\mathcal{O}(M+N)$. □

This runtime is exact in the sense that the time taken to generate a given graph is $\mathcal{O}(M+N)$ where M is the actual number of edges in the graph produced: in a worse-case scenario the bound above remains correct, but M is large. For the Chung-Lu graphs our results are less precise: we bound the expected runtime in terms of the expected number of edges.

An algorithm similar to Algorithm 1 is described in [1] which avoids the unsuccessful iterations of the inner loop. However, our approach lends itself to the generalization we describe below.

3 Chung-Lu Case

Having set the framework with the Erdős–Rényi case we now describe an algorithm to create Chung-Lu networks where not all nodes have the same weight. As before, we want to skip as many nodes as possible, but this is more difficult because the probabilities that any pair of nodes have an edge are not fixed. To simplify this, we assume that we have a list W of the weights in the network, and that this list is sorted in descending order.

The obvious $\mathcal{O}(N^2)$ algorithm considers each pair u and v and assigns an edge with probability $p_{u,v} = w_u w_v / N\overline{w}$. We present a slightly different $\mathcal{O}(N^2)$ algorithm, which is easily modified the same way we altered the Erdős–Rényi algorithm to create an $\mathcal{O}(M+N)$ algorithm.

Starting with $u = 0$, we consider every $v = 1, 2, \ldots, N-1$ in turn. As v increases, $p_{u,v}$ decreases monotonically, so we can avoid recalculating p for each v by setting $p = p_{u,u+1} = w_u w_{u+1} / N\overline{w}$ initially and discarding each v with probability $1-p$. When we arrive at the first node v_1 which is not discarded, we call v_1 a *potential neighbor*. We have selected v_1 with probability p, but the probability of an edge between v_1 and u is actually $q \leq p$. We calculate $q = p_{u,v_1}$, and then assign an edge with probability q/p. We then set $p = q$ and continue on, discarding nodes with probability $1-p$ until we have considered all possible nodes. We then increase u by 1 and repeat. This algorithm is $\mathcal{O}(N^2)$.

In the algorithm just described, p is fixed at each step until a potential neighbor is identified. The same method used in the Erdős–Rényi approach can quickly identify the potential neighbors v_i without considering each intermediate v in turn. Starting with $u = 0$ and using $p = p_{u,u+1}$, we choose a random number r uniformly in $(0,1)$ and find the first potential neighbor $v_1 = u + 1 + \delta$ where $\delta = \lfloor \ln r / \ln(1-p) \rfloor$. If $p = 1$, we take $\delta = 0$. Once v_1 is identified, we assign an edge between u and v_1 with probability q/p where $q = p_{u,v_1}$. We then set $p = q$

and continue, jumping immediately to the next potential neighbor v_2, possibly placing an edge. Again resetting p, we continue until there are no more nodes to consider. We then increase u by 1 and repeat. Ultimately, u takes all possible values, and the set of edges is complete. Note that for given u the value of p decreases monotonically, so the expected value of δ increases monotonically.

The Chung-Lu graph generating procedure is presented in Algorithm 2.

Algorithm 2. Chung-Lu Graph

Input: list of N weights, $W = w_0, \ldots, w_{N-1}$, sorted in decreasing order
Output: Chung-Lu graph $G(V, E)$ with $V = \{0, \ldots, N-1\}$

```
E ← ∅
S ← Σ_u w_u
for u = 0 to N − 2 do
    v ← u + 1
    p ← min(w_u w_v / S, 1)
    while v < N and p > 0 do
        if p ≠ 1 then
            choose r ∈ (0, 1) uniformly at random
            v ← v + ⌊log(r) / log(1 − p)⌋
        if v < N then
            q ← min(w_u w_v / S, 1)
            choose r ∈ (0, 1) uniformly at random
            if r < q/p then
                E ← E ∪ {u, v}
            p ← q
            v ← v + 1
```

Proposition 1. *Algorithm 2 generates Chung-Lu graphs.*

Proof. We need to prove that an edge is assigned from node u to node v with probability $p_{u,v}$ that is independent of other edges.

Consider a given u and $v > u$. Let $\hat{p}$ represent the value of p when the inner loop reaches (or passes) v while assigning edges for u. This is the probability with which v is selected as a potential neighbor. The value of $\hat{p}$ is influenced by other edges which have been assigned to u. Our goal is to show that regardless of the value of $\hat{p}$, an edge is placed between u and v with probability $p_{u,v}$.

Because of the ordering of the weights we know that $\hat{p} \geq p_{u,v}$. If v is selected as a potential neighbor, then with probability $q = p_{u,v}/\hat{p} \leq 1$, v will become an actual neighbor of u. Thus the probability that an edge exists between u and v is $\hat{p}q = p_{u,v}$. This is independent of $\hat{p}$, so it is independent of any previous edges that have been assigned. By similar argument it has no influence on what later edges will be assigned. □

3.1 Efficiency

For the Chung-Lu algorithm it is more difficult to bound the number of steps because of the rejection sampling: in some cases a potential neighbor v_i is

identified, but upon closer inspection no edge is placed to v_i. We refer to these as *excess potential neighbors*. Let m be the number of actual neighbors and l the number of excess potential neighbors generated in a particular realization. Let L be the expected value of l. We define $M = N\overline{w}/2$, and because each node's expected degree is at most its weight we know that the expected value of m is at most M. For many relevant weight distributions, the expected value of m approaches M as $N \to \infty$. L is more difficult to bound. We will prove that $L = \mathcal{O}(N + M)$, and so the algorithm executes in $\mathcal{O}(N + M)$ expected time.

Theorem 2. *Algorithm 2 executes in $\mathcal{O}(N + M)$ expected time where $M = N\overline{w}/2$ and N is the number of nodes.*

Proof. We follow a similar argument to the Erdős–Rényi case, and conclude that the runtime is $\mathcal{O}(N+m+l)$ where m is the actual number of edges created and l the number of excess potential neighbors. We will show that the expected value of l is $L = \mathcal{O}(N + M)$, and since the expectation of m is at most M, the expected runtime is $\mathcal{O}(N + M)$. However, the calculation of L is considerably more technical, and is the focus of our proof. We will bound the probability that each node is selected as an excess neighbor of another node. Summing this gives a bound on L.

Consider a given u and $v > u$. Let $\rho_{u,v}(\hat{p})$ be the *a priori* probability that the value of p is $\hat{p}$ when the inner loop reaches (or passes) v while assigning edges for u. Define $P_{u,v} = \sum_{\hat{p}} \rho_{u,v}(\hat{p})\hat{p}$, the *a priori* probability that v will be chosen as a potential neighbor of u. The probability that v will be selected as an excess potential neighbor is $P_{u,v} - p_{u,v}$. We seek to calculate $P_{u,v}$.

We know that $P_{u,u+1} = p_{u,u+1}$. We look to find $P_{u,v+1} - P_{u,v}$ for all v. This requires calculating the change in $\rho_{u,v}(\hat{p})$ from v to $v + 1$. If v is not chosen as a potential neighbor, there is no change to p, but if v is chosen, then p changes from $\hat{p}$ to $p_{u,v}$ and so the change in p is $p_{u,v} - \hat{p}$. This occurs with probability p. So the change in $P_{u,v}$ is the expected change in p, which we define to be $\Delta P_{u,v} = P_{u,v+1} - P_{u,v}$. To make $\Delta P_{u,N-1}$ defined, it is convenient to set $P_{u,N}$ to be the value P would take for node N if the node list were extended by adding an additional node with weight 0. Note that $\Delta P_{u,v} \leq 0$. We have

$$\begin{aligned}
\Delta P_{u,v} &= P_{u,v+1} - P_{u,v} \,, \\
&= \sum_{\hat{p}} \rho_{u,v}(\hat{p})\hat{p}\,(p_{u,v} - \hat{p}) \,, \\
&= P_{u,v}p_{u,v} - \sum_{\hat{p}} \rho_{u,v}(\hat{p})\hat{p}^2 \,, \\
&\leq P_{u,v}\,(p_{u,v} - P_{u,v}) \,, \qquad (1)
\end{aligned}$$

using Jensen's inequality to say that the expectation of p^2 is at least the square of the expectation of p. Note that $P_{u,v}$ decreases monotonically with v and that $P_{u,v}$ cannot be less than $p_{u,v}$.

We now define $\zeta(u)$ to be the number of excess potential neighbors node u is expected to have,

$$\zeta(u) = \sum_{v=u+1}^{N-1} P_{u,v} - p_{u,v} \,. \tag{2}$$

From our bound (1) for $\Delta P_{u,v}$, we can bound $\zeta(u)$ as

$$\zeta(u) \leq \sum_{v=u+1}^{N-1} \frac{-\Delta P_{u,v}}{P_{u,v}} \,.$$

By analogy with the integral $-\int_a^b \phi'(x)/\phi(x)\,\mathrm{d}x$, we anticipate that this summation behaves like a logarithm, and we use this to bound the sum. We use the inequality for $x > -1$ that $\ln(1+x) \leq x$, implying $-x \leq -\ln(1+x)$. For simplicity, we extend the range to include $x = -1$ by allowing for infinity. Then

$$\begin{aligned}
\frac{-\Delta P_{u,v}}{P_{u,v}} &\leq -\ln\left(1 + \frac{\Delta P_{u,v}}{P_{u,v}}\right) \,, \\
&\leq -\ln \frac{P_{u,v} + \Delta P_{u,v}}{P_{u,v}} \,, \\
&\leq -\ln \frac{P_{u,v+1}}{P_{u,v}} \,, \\
&\leq \ln P_{u,v} - \ln P_{u,v+1} \,.
\end{aligned}$$

So $\zeta(u)$ is bounded by a telescoping summation,

$$\zeta(u) \leq \ln P_{u,u+1} - \ln P_{u,N} \,.$$

It is difficult to bound $P_{u,N}$ away from zero. So instead we break the sum in Eq. (2) into terms for which P can be easily bounded away from zero and those for which it is more difficult. Set x to be the first node such that $w_x < 1$, so $w_{x-1} \geq 1$. Assume for now $u < x$. We have

$$\zeta(u) \leq \sum_{v=u+1}^{x-1} \frac{-\Delta P_{u,v}}{P_{u,v}} + \sum_{v=x}^{N-1} P_{u,v} - p_{u,v} \,.$$

The first summation is at most

$$\ln P_{u,u+1} - \ln P_{u,x} = \ln[P_{u,u+1}/P_{u,x}] \,.$$

We have

$$P_{u,u+1} = p_{u,u+1} = \min(w_u w_{u+1}/N\overline{w}, 1) \leq w_u w_{u+1}/N\overline{w} \,,$$

while

$$P_{u,x} \geq \min(w_u w_{x-1}/N\overline{w}, 1) \geq w_u/N\overline{w} \,.$$

Thus $P_{u,u+1}/P_{u,x} \leq w_{u+1}$. The first summation is at most $\ln w_{u+1}$, which in turn is at most $\ln w_u$.

The second summation can be bounded by observing that the expected number of excess potential neighbors in $[x, N-1]$ is at most the expected number of potential neighbors in $[x, N-1]$. Assuming that u has at least one potential neighbor $v \geq x$, the probability for any later node to be a potential neighbor is at most $w_u w_v / N\overline{w} < w_u / N\overline{w}$. There are $N-1-x$ nodes in this region, which is bounded by N, so u has at most $1 + N w_u / N\overline{w}$ expected further potential neighbors in $[x, N-1]$. This gives an upper bound on the second sum. So if $u < x$, $\zeta(u) \leq 1 + \ln w_u + w_u/\overline{w}$.

If $u \geq x$, then the approach used above for the second summation gives an upper bound of $\zeta(u) \leq 1 + w_u N / N\overline{w}$.

Both bounds for $u < x$ and $u \geq x$ can be replaced by

$$\zeta(u) \leq 1 + w_u + \frac{w_u}{\overline{w}}\,,$$

for any u. This is a significant overestimate, but will not weaken our final bound, and we discuss it further in the next section. We sum $\zeta(u)$ over all nodes and get

$$\begin{aligned} L &= \sum_{u=0}^{N-1} \zeta(u)\,, \\ &\leq \sum_{u=0}^{N-1} 1 + w_u + \frac{w_u}{\overline{w}}\,, \\ &\leq N + N\overline{w} + \frac{N\overline{w}}{\overline{w}}\,, \\ &\leq 2M + 2N\,. \end{aligned}$$

Therefore $L = \mathcal{O}(N+M)$, and we have that $\mathcal{O}(N+M+L) = \mathcal{O}(N+M)$. So the algorithm executes in $\mathcal{O}(N+M)$ expected time. □

4 Examples

We demonstrate the runtime of Algorithm 2 using three different weight distributions. The first has weights chosen uniformly in $(1, 50)$, giving an average of 25.5. The second has all weights equal to 25. The third has weights chosen from a power law distribution with exponent $\gamma = -2.1$, and every weight above 100 is set to 100, giving an average of about 4.7. We generate graphs on N nodes where the weights are chosen from each of these distributions. In Fig. 1 we show that the execution time is $\mathcal{O}(M+N)$. In contrast, the standard algorithm for generating these scales like $\mathcal{O}(N^2)$.

The proof of Theorem 2 shows that the total expected excess neighbors is $L = \mathcal{O}(M+N)$. Reducing L reduces the runtime so we now look at L more closely. The bound used for L is fairly crude. It was derived by replacing $\ln w_{u+1}$ with w_u and assuming that every node would have at least one neighbor of weight $w < 1$.

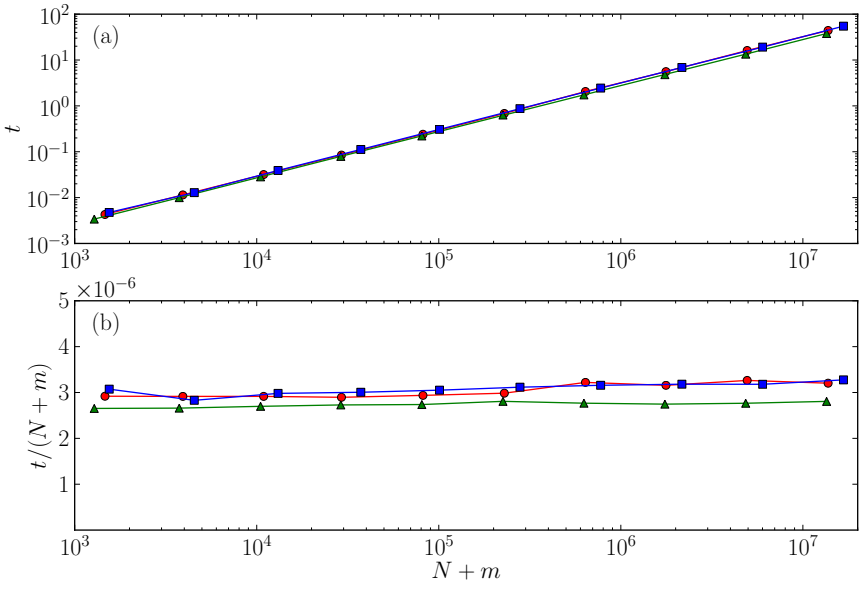

Fig. 1. Performance of Algorithm 2 showing linear scaling in the number of nodes and edges $N+m$. The 3 curves are data for weights chosen from the following distributions (red circles) uniform on (1,50); (green triangles) constant $w = 25$; and (blue squares) power law with exponent $\gamma = -2.1$. (a) Running time vs number of nodes and edges. (b) Estimate of the coefficient.

Analyzing the algorithm closer, we see that to have larger L, there is a competition: a node is more likely to be an excess potential neighbor if the weights before it are large, but its weight is small. However, such a node reduces the probability that nodes after it become excess potential neighbors. Thus heterogeneity in weights plays a role in determining the number of excess potential neighbors.

To investigate this further, we show the total number of excess potential neighbors generated by the algorithm in Fig. 2. In Fig. 2(a) we consider the same three distributions as before. We find that L/M, the number of excess potential neighbors created for each edge is largely independent of the number of nodes, and that if all nodes have the same weight $L = 0$. Although we have proven that $L \leq 2M + 2N$, the numerical experiments suggest that stronger bounds are possible and depend on the heterogeneity in weights.

In Fig. 2(b) we show the interplay of heterogeneity and average weight more closely. We considered two classes of distributions. In one, w is chosen uniformly in $(\overline{w}-5, \overline{w}+5)$, and in the other w is chosen uniformly in $(1, 2\overline{w}-1)$. We see that the number of excess potential neighbors generated per node L/N decreases in the first case and increases slowly in the second case. In both cases M increaseslinearly with $\overline{w}$, so L/M decreases to zero. The excess potential neighbors become a negligible contribution to runtime as graph size increases.

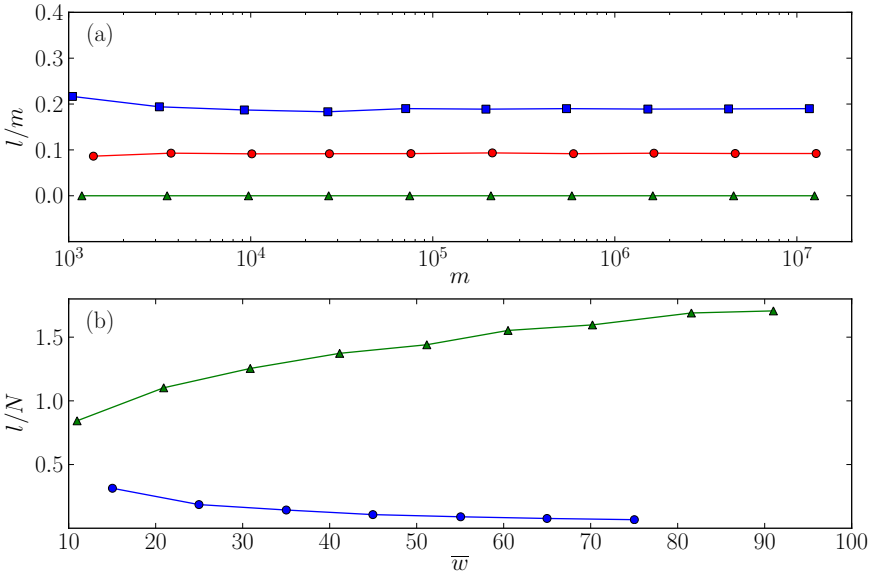

Fig. 2. Fraction of "excess potential neighbors" l/m generated by Algorithm 2. (a) The fraction l/m vs total number of edges in the graph m for weights from the three distributions of Fig. 1. (b) The fraction l/m for uniformly distributed sequences with fixed variance: w chosen uniformly from $(\overline{w}-5, \overline{w}+5)$ (blue circles), and with fixed coefficient of variance $\mathrm{std}(W)/\overline{w}$: w chosen uniformly from $(1, 2\overline{w}-1)$ (green triangles).

5 Discussion

We have developed an algorithm for creating Chung-Lu random graphs in $\mathcal{O}(M+N)$ expected runtime. Our algorithm may be generalized to other contexts. In particular, it may also be used to generate the random graphs introduced in Refs. [17,3]. This algorithm requires first that there is a single parameter w assigned to each node which determines the probability that any two nodes share an edge, and second that the nodes may be ordered in such a way that if u appears before v_1 which appears before v_2, then the probability that u has an edge with v_1 is at least the probability that u has an edge with v_2. It is possible to generate many other graph models in this manner, including models which have assortative mixing (nodes with similar weights are preferentially connected).

Some graph models, such as the configuration model, do not assign edges independently, and so have somewhat different generation algorithms. In the configuration model each node is assigned a degree *a priori*, and then nodes are wired together subject to the assigned degrees as a constraint. An efficient algorithm to do this begins by placing nodes into a list once for each edge the node will have. The list is then shuffled, and adjacent nodes are joined by an edge. Consequently edges may be repeated, nodes may have edges to themselves, and if the sum of degrees is odd, a node is left unpaired. Typically the number

of such edges is small compared to the number of nodes, so these are simply discarded. It is possible to avoid these cases, but even the most efficient known algorithms that produce true graphs with the imposed degree distribution are substantially slower than $\mathcal{O}(N + M)$ [11].

Acknowledgments

JCM was supported by 1) the RAPIDD program of the Science and Technology Directorate, Department of Homeland Security and the Fogarty International Center, National Institutes of Health and 2) the Center for Communicable Disease Dynamics, Department of Epidemiology, Harvard School of Public Health under Award Number U54GM088558 from the National Institute Of General Medical Sciences. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institute Of General Medical Sciences or the National Institutes of Health. Part of this work was funded by the Department of Energy at Los Alamos National Laboratory under contract DE-AC52-06NA25396, and the DOE Office of Science Advanced Computing Research (ASCR) program in Applied Mathematics.

References

1. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Physical Review E 71, 036113 (2005)
2. Bollobás, B., Janson, S., Riordan, O.: The phase transition in inhomogeneous random graphs. Random Structures and Algorithms 31(1), 3 (2007)
3. Britton, T., Deijfen, M., Martin-Löf, A.: Generating simple random graphs with prescribed degree distribution. Journal of Statistical Physics 124(6), 1377–1397 (2006)
4. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the Web. In: WWW9: 9th International World Wide Web Conference, vol. 33, pp. 309–320 (2000)
5. Chung, F., Lu, L.: Connected components in random graphs with given expected degree sequences. Annals of Combinatorics 6(2), 125–145 (2002)
6. Chung, F., Lu, L.: The average distance in a random graph with given expected degrees. Internet Mathematics 1(1), 91–113 (2004)
7. Chung, F., Lu, L.: The volume of the giant component of a random graph with given expected degrees. SIAM Journal on Discrete Mathematics 20(2), 395–411 (2007)
8. Chung, F., Lu, L., Vu, V.: The spectra of random graphs with given expected degrees. Internet Mathematics 1(3), 257–275 (2004)
9. Draief, M., Ganesh, A.: A random walk model for infection on graphs: spread of epidemics & rumours with mobile agents. Discrete Event Dynamic Systems 21, 41–61 (2011)
10. Eubank, S., Kumar, V.S., Marathe, M.V., Srinivasan, A., Wang, N.: Structural and algorithmic aspects of massive social networks. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 718–727 (2004)

11. del Genio, C.I., Kim, H., Toroczkai, Z., Bassler, K.E.: Efficient and exact sampling of simple graphs with given arbitrary degree sequence. PloS ONE 5(4), e10012 (2010)
12. van der Hofstad, R.: Critical behavior in inhomogeneous random graphs, arXiv:0902.0216v2 [math.PR] (2010)
13. Keeling, M.J., Eames, K.T.D.: Networks and epidemic models. Journal of The Royal Society Interface 2(4), 295–307 (2005), doi:10.1098/rsif.2005.0051
14. Kleinberg, J.: Complex networks and decentralized search algorithms. In: Proceedings of the International Congress of Mathematicians, ICM (2006)
15. Newman, M.E.J.: Spread of epidemic disease on networks. Physical Review E 66(1), 016128 (2002)
16. Newman, M.E.J.: The structure and function of complex networks. SIAM Review 45, 167–256 (2003)
17. Norros, I., Reittu, H.: On a conditionally Poissonian graph process. Advances in Applied Probability 38(1), 59–75 (2006)