Journal of Complex Networks (2019) **0000**, 1–17
doi: 10.1093/comnet/cnz011

Generating connected random graphs

CAITLIN GRAY†, LEWIS MITCHELL AND MATTHEW ROUGHAN

ARC Centre of Excellence for Mathematical and Statistical Frontiers, School of Mathematical Sciences, University of Adelaide, North Terrace, South Australia 5005, Australia
†Corresponding author. Email: caitlin.gray@adelaide.edu.au

Edited by: James Gleeson

[Received on 3 August 2018; editorial decision on 13 February 2019; accepted on 11 March 2019]

Sampling random graphs is essential in many applications, and often algorithms use Markov chain Monte Carlo methods to sample uniformly from the space of graphs. However, often there is a need to sample graphs with some property that we are unable, or it is too inefficient, to sample using standard approaches. In this article, we are interested in sampling graphs from a conditional ensemble of the underlying graph model. We present an algorithm to generate samples from an ensemble of connected random graphs using a Metropolis–Hastings framework. The algorithm extends to a general framework for sampling from a known distribution of graphs, conditioned on a desired property. We demonstrate the method to generate connected spatially embedded random graphs, specifically the well-known Waxman network, and illustrate the convergence and practicalities of the algorithm.

Keywords: random graphs; MCMC; network sampling; connected networks.

1. Introduction

Random graphs are commonly used as underlying models in many fields, such as computer networking, biology, social sciences and physics [1–5]. The ability to generate random graphs with desired properties is crucial, as they may be used in conjunction with complex models, for instance a routing protocol in computer networking [6].

Real-world networks come with countless properties that one may consider modelling, for example degree distributions, clustering levels, etc. Most random graph models focus on one of these properties to model an observed network. However, many current methods for generating random graphs result in networks with some undesirable properties for a particular applications.

For instance,

- the graphs may not be connected, for example the Gilbert–Erdős–Rényi (GER) model or spatial Waxman graph [6]; or
- the graphs may not be *simple*, that is they might have multi-edges or self-loops, for example the configuration model.

While one might argue that this is a modelling problem, there are nevertheless many instances in the literature where a model matches enough properties of the real networks in question that it is useful, except for one deficiency such as noted above.

Examples include:

- using the Waxman graph to model physical networks that are inherently connected, for example router networks; and
- using the configuration model that generates graphs with self-loops and multi-edges to model simple networks.

Generating connected graphs with a given degree sequence has been discussed at length in the literature using Markov chain Monte Carlo (MCMC) methods [7–10]. The existing MCMC algorithms use 'edge swaps' to give a uniform sample over the graph space. While this may be useful when requiring only a graph with the desired property, the natural question remains of how to sample graphs while ensuring we maintain the conditional ensemble of the underlying graph model. This is essential in many applications; for example, when estimating parameters, or in applications of Approximate Bayesian Computation where the ensemble encompasses prior knowledge of the system.

We present an algorithm to produce random graphs from a known ensemble conditioned on an extra desired property of the network. Our algorithm uses MCMC methods to sample from the ensemble of interest. In particular, we focus here on generating connected networks. We show the algorithm samples graphs from the desired distribution and demonstrate the algorithm on spatially embedded random networks (SERNs), in particular the Waxman random graph [6]. We show that the algorithm is $\mathrm{O}(K)$ for K iterations on sparse graphs. Although we cannot theoretically bound the number of iterations required for convergence, empirical results show convergence scales like $\mathrm{O}(N^2)$ in the number of nodes in the graph.

The algorithm not only has practical applications in that one can generate connected graphs for use in various applications, but also, such a simulation algorithm could be used to estimate the probability of such graphs in an ensemble.

2. Background

2.1 *Mathematical formalities*

A graph (or network), $G = (V, E)$, consists of a set of N nodes, which we label $V = \{1, 2, \ldots, N\}$. The graph has edges (or links) $E \subset V \times V$.

We say that a link exists between two nodes i and j if $(i, j) \in E$. We are primarily concerned here with undirected graphs (though much work on random graphs is easy to generalize to directed graphs), and so if $(i, j) \in E$ then $(j, i) \in E$. We focus on simple graphs with no self-edges (i.e. edge (i, i) does not exist).

We say that two nodes are connected if a path (a sequence of edges) exists between the two nodes. The graph is connected if all pairs of nodes (i, j) are connected.

The well-known GER random graph, $G_{n,p}$ of n nodes is constructed by assigning each edge (i, j) to be in E independently, with fixed probability p [11, 12].

SERNs stem from the notion that often longer links are more expensive to build or maintain. Often real-world networks display this spatial structure, and so various types of SERNs are used in social, infrastructure and epidemiological modelling [13–16]. Here, we create a SERN by placing N nodes uniformly at random within some defined region R of a metric space Ω with distance metric $d(x, y)$. Each pair of nodes is made adjacent independently, with probability p_{ij}, which is a function of $d(x_i, x_j)$. In the

Waxman case,

$$p_{ij} = qe^{-sd_{ij}}, \tag{2.1}$$

for $q, s \geq 0$, and the Euclidean distance d_{ij}. The parameter s controls the extent to which spatial structure is incorporated into the graph. Note that when $s = 0$ we recover the GER random graph, with edge probability q. In general, the q value controls the overall edge density in the graph. Note that the parametrization in (2.1) differs from much of the literature on Waxman graphs. We chose to do this as unfortunately, the parameters (α, β) used traditionally have become confused by frequent reversal.

The basic properties of the Waxman graph can be derived. For instance, it is shown [17] that the average node degree is given by

$$\bar{z} = (n-1)q\tilde{G}(s), \tag{2.2}$$

where $\tilde{G}(s)$ is the Laplace transform of the probability density function between a pair of random points (the Line-Picking Problem), see references for further details [17, 18]. The Waxman is just one example of a SERN, and we use it here to provide a simple and clear example. Results generalize to other SERNs.

2.2 *Markov chain Monte Carlo*

MCMC methods are widely used to sample from complex probability distributions that are difficult to generate directly. These approaches generate Markov chains that converge to the distribution of interest.

Specifically, we use the Metropolis–Hastings (M–H) algorithm [19, 20], given in Algorithm 1, to draw samples from our distribution of interest, namely, the distribution of networks with our desired property.

Consider the target distribution $\pi(\theta)$ we wish to sample from. We use the M–H algorithm to create a Markov chain $\theta^{(1)}, \theta^{(2)}, \ldots$, using the proposal distribution $Q(\theta'|\theta)$ to propose the next candidate θ' from the current state θ. The proposal distribution must be able to explore the entire space in a finite number of steps [21].

The proposed parameter value θ' is accepted with some probability given by, in the case of M–H, the acceptance probability

$$\alpha = \min\left(1, \frac{\pi(\theta')Q(\theta|\theta')}{\pi(\theta)Q(\theta'|\theta)}\right).$$

If the proposal distribution is symmetric then

$$\alpha = \min\left(1, \frac{\pi(\theta')}{\pi(\theta)}\right).$$

The chain is generated from the proposed parameter θ' as follows

$$\theta^{(t+1)} = \begin{cases} \theta', & \text{if accepted,} \\ \theta^{(t)}, & \text{otherwise,} \end{cases}$$

where θ' is generated from $Q(\theta'|\theta^{(t)})$.

Algorithm 1 General Metropolis–Hastings algorithm [21]

1. Set $\theta^{(0)}$
2. **for** $t = 1...K$ **do**
3. Generate $\theta' \sim Q(\theta'|\theta^{(t-1)})$
4. Take $\theta^{(t)} = \begin{cases} \theta', & \text{with probability } \alpha \\ \theta^{(t-1)}, & \text{with probability } 1-\alpha. \end{cases}$
   where $\alpha = \min\left(1, \frac{\pi(\theta')Q(\theta|\theta')}{\pi(\theta)Q(\theta'|\theta)}\right)$
5. **end for**

Markov chain traversals of graphs have been used to sample from a variety of spaces [22]. MCMC methods are also widely used to sample exponential random graphs [23], and there has been much focus on generating networks that have a desired degree sequence [24–26]. This is achieved through the use of an 'edge swaps' proposal distribution that preserves the degree sequence of the network throughout the MCMC process. Much of this work focusses on the configuration model; that is, the uniform sampling of networks with a given degree sequence. These have applications when using the configuration model directly or as null models [24]. Other works sample uniformly from graphs with power-law distributions in a similar manner [27]. Uniform sampling can be useful in some situations; however, we are often interested in sampling from a model with a more complicated underlying distribution, and in ensuring we do not oversample rare graphs. Therefore, here we focus on sampling from spaces of graphs that have a non-uniform distribution. Recently, the 'edge-switch' proposal in MCMC methods have been used to sample bipartite graphs with only expected degrees that provide a framework to study partially observed networks [8], and the extension of the double swap to a triple swap to allow sampling of 'loopy' graphs [7]. Another related work, [10], uses link switches to generate synthetic networks preserving properties of a real graph input with privacy and significance testing applications.

2.3 *Connectedness*

We present our algorithm in the context of generating connected random networks. The property of connectedness is often observed in physical networks, such as a telecommunications network, where there is the requirement that a path exist between all nodes. Other physical examples include the Internet routing network. It is also important in the application of social networks. In general, each individual may not be connected to all others through some path. However, in the application of epidemics and information diffusion, there is particular interest in the network over which information propagates. To participate in a cascade, the individual must have come into contact with the contagion; therefore, there is necessarily a path between all individuals in the network over which the cascade is observed.

Many random graph generators do not consider connectivity and simply take the giant component of the resulting graphs or prove properties like the distribution of connected component size in the asymptotic limit. However, in many applications we are interested in generating connected networks of fixed size from our distribution.

Rejection sampling is commonly used to generate networks that display a desired property by simply rejecting graphs that do not display this property. While appropriate in some cases, there are many situations in which this method is extremely slow. For example, rejection sampling of simple graphs from the configuration model may take exponential time in the size of the graph for some degree sequences [22]. For connectedness, the probability of all nodes being connected can be very low even for quite

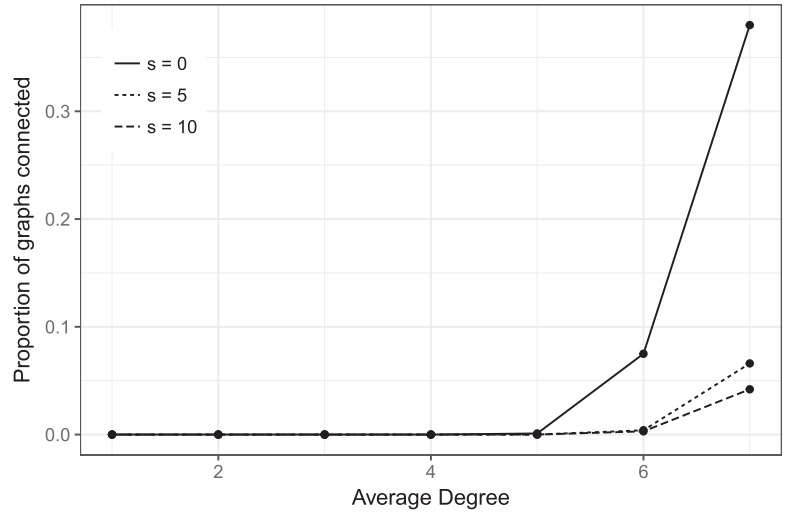

FIG. 1. Proportion of connected networks in 1000 samples of a Waxman network with $N = 1000$.

reasonable parameter values, and so rejection sampling is often not practical. While the probability of connectedness has not been found analytically for Waxman graphs, simple simulations can show that connected graphs are often unlikely. Figure 1 shows the proportion of Waxman graphs that are connected after samples, for a variety of parameters, and we can see that as the dependence on distance becomes stronger (s parameter increasing) the probability of connectedness decreases. Additionally, the traditional $O(N^2)$ sampler makes running even a few hundred samples of the Waxman expensive.

Markov chain methods have been used to produce connected random networks with a prescribed degree sequence [9, 22], with a particular focus on with a networks in peer-to-peer applications [25].

3. Generating connected graphs

We assume a random graph model that generates an ensemble of sometimes unconnected graphs and that the model provides a probability distribution across the ensemble, that is the probability $P(G)$ for each graph G. Even if we assume that this probability is calculable, direct simulation from the distribution is usually intractable due to the size of the ensemble. Usually, there is an algorithm to generate graphs from the ensemble.

Given the model, we would like to generate connected graphs with the same conditional probability distribution as the model of interest, that is we would like to generate connected graphs G with probabilities

$$P\{G|G \text{ is connected}\} = \frac{P\{G \text{ and } G \text{ is connected}\}}{P\{G \text{ is connected}\}},$$

where the numerator is given by:

$$P\{G \text{ and } G \text{ is connected}\} = \begin{cases} P(G), & \text{for } G \text{ connected} \\ 0, & \text{otherwise.} \end{cases}$$

The required connected random graphs are samples from the unknown conditional probability distribution $P\{G|G \text{ is connected}\}$. This leads naturally to the use of well known MCMC methods as the basis for the sampling algorithm.

We implement the M–H method to generate a Markov chain that will result in samples from the desired distribution. The algorithm produces a new graph $G' = (V, E')$ based on the old graph G. The two main components are a symmetric proposal distribution that can explore the entire space and a tractable acceptance ratio.

We initialize the algorithm using the underlying model to create a random graph, $G^{(-1)}$, with $P(G^{(-1)}) > 0$. This network is connected by adding arbitrary links. The graph need not be necessarily chosen with the correct probability, so in this case almost any procedure to obtain connectivity is adequate. Whichever connectivity procedure is used leads to a connected random graph $G^{(0)}$ used as the input to the M–H algorithm.

The process described in detail below.

Step 1—Proposal: The probability density $Q(G'|G)$, is the proposal distribution that gives the next candidate for the algorithm. An advantageous feature of Q for the M–H algorithm is that it is symmetric, that is $Q(G'|G) = Q(G|G')$, as this simplifies the acceptance ratio.

Here, we perform the algorithm link by link. At each step, we select a node pair (i, j) at random, and consider adding or removing a link to obtain the new network. In practice, we choose two distinct nodes at random and consider the possible link between them.

Mathematically,

(1) if $(i,j) \in E$ then $E' = E \backslash \{(i,j)\}$,

(2) if $(i,j) \notin E$ then $E' = E \cup \{(i,j)\}$.

All node pairs are chosen with equal probability, so $Q(G'|G) = 2/(N(N-1))$ for all G and G' that differ by one link. Therefore, the transition is symmetric.

This proposal has been used in graph sampling previously, notably in applications related to sampling exponential random graphs, for example [23], and there is no consideration of connectivity in this step.

Step 2—Acceptance: The M–H acceptance ratio (the probability of accepting the proposed transition) given that the proposal is symmetric is given by

$$\alpha = \min\left\{1, \frac{P\{G'|G' \text{ is connected}\}}{P\{G|G \text{ is connected}\}}\right\}. \tag{3.1}$$

If the proposed graph has a higher probability than the previous graph, we accept the move. If not, we accept with some probability dependent on the ratio of the two graph probabilities. However, the ratio is intractable in this form, as we cannot calculate $P\{G|G \text{ is connected}\}$.

To determine a tractable acceptance ratio, we consider the connectivity of each proposed graph. Recall, we start with a valid connected graph $G^{(0)}$. If G' is unconnected, then $P\{G'|G' \text{ is connected}\} = 0$, so unconnected graphs will never be accepted; therefore, we remain in the space of connected graphs.

We use this to establish a tractable ratio. When G and G' are connected, the conditionals can be dropped from the probabilities, as $P\{G \text{ is connected}\}$ is constant over the ensemble.

This gives

$$\alpha = \min\left\{1, \frac{P(G')}{P(G)}\right\}, \tag{3.2}$$

for connected graphs G and G'. The ratio is tractable in many cases where we can calculate the ratio of the probability distributions. If all edges are independent then this can be calculated very quickly.

The process is iterated a number of times until the Markov chain converges and the networks are being sampled from the stationary distribution of interest.

To implement this algorithm, we must check the connectivity of the graph when removing a link. There are a variety of algorithms for checking connectivity [28]. We use a simple breadth first search with complexity $\mathrm{O}(N + |E|)$. As we are interested in sparse graphs with $|E| \sim \mathrm{O}(N)$, meaning the search is $\mathrm{O}(N)$. After removing a link (i,j) the graph remains connected if and only if a path still remains between i and j. Therefore, determining if the graph still has a path between i and j, although still $\mathrm{O}(N)$, is likely to be faster than the worst case, especially on spatial graphs.

This algorithm will work well on networks where each edge exists independently of any other, for example the GER graph, inhomogenous random graphs [29] or SERNs. In these cases, the calculation of $P(G')/P(G)$ is a simple ratio of edge probabilities. In principle, this algorithm can be applied to any model in which every graph has positive probability prior to adding that extra constraint, although $P(G')/P(G)$ may be hard to calculate. Note also that the algorithm, as described here, will work only for graph models that assign positive probability to graphs with a different number of edges. For example, the configuration model network has a fixed degree sequence, hence a fixed number of edges. Therefore, the proposal of adding or removing a single edge will be inappropriate as it will break the degree sequence. A simple change of proposal distribution allows for sampling from these networks [9, 22].

4. Theoretical convergence

THEOREM 4.1 Algorithm 2 generates samples from the random graph ensemble with probability distribution $P\{G|G \text{ is connected}\}$.

Proof. Theorem (7.4) of Robert and Casella [21] states that the chain produced by the M–H algorithm (Algorithm 1) converges to the stationary distribution π if:

(1) it is irreducible, and
(2) it is aperiodic.

Consider the Markov chain produced by Algorithm 2. We show there exists a sequence of a finite number of steps with positive probability from any connected graph H to any connected H', that is $P(H \to H') > 0$. We must ensure that the graph remains connected in all steps. Therefore, consider adding all edges not in H to create a clique. Then remove the edges in subsequent steps to reach H'.

$$P(H \to H_{\text{clique}} \to H') = P(H \to H_{\text{clique}})P(H_{\text{clique}} \to H').$$

If every connected graph in the ensemble has non-zero probability, both terms on the RHS have positive probability. Therefore, the chain is *irreducible*.

A sufficient condition for the Markov chain to be aperiodic is to choose Q such that the probability of the event $\left\{X^{(t+1)} = X^{(t)}\right\}$ is non-zero for some state. If the removal of an edge destroys connectivity the transition is rejected and the chain remains in the current state. Therefore, the chain is *aperiodic*.

Algorithm 2 Algorithm to generate connected random graphs

1. Generate $G^{(-1)}$ from the model
2. Connect $G^{(-1)}$ to get $G^{(0)}$
3. **for** k=1..K **do**
4. Generate a random node pair (i,j)
5. **if** $(i,j) \in E$ **then**
6. Remove the edge: $E' = E \setminus \{(i,j)\}$
7. **if** G' is connected **then**
8. accept G' with probability $P(G')/P(G)$
9. **else**
10. reject G'
11. **end if**
12. **else**
13. Add edge: $E' = E \cup \{(i,j)\}$
14. accept G' with probability $P(G')/P(G)$
15. **end if**
16. **end for**

Note that the acceptance probability construction ensures $\pi = P\{G|G \text{ is connected}\}$. Hence, by Theorem (7.4) of Robert and Casella, Algorithm 2, with acceptance probability α (3.2) converges to the distribution of interest.

□

Unfortunately, this result only assures us that after infinite time the process will be sampling from the distribution of interest. We show evidence for convergence in finite time in Section 7.

5. Complexity

THEOREM 5.1 Algorithm 2 with K iterations has computational complexity $O(K)$, independent of the size of the graph, for sparse graphs.

Proof. We use a neighbourhood list stored in a hash map to describe the edges in the graph. This results in expectedly $O(1)$ operations to check edges for existence and add/remove edges at each iteration. We check for connectivity when edge removal is proposed. The breadth first search algorithm is $O(N)$ for a sparse network with N nodes. For a sparse graph the number of edges is $O(N)$, and so the probability of selecting an edge to delete is $O(1/N)$. That is, for large N

$$P\{\text{edge } (i,j) \text{ exists}\} \sim \frac{1}{N}.$$

So, the probability there exists an edge between the two chosen nodes, requiring the $O(N)$ connectedness routine, decreases like $1/N$. Therefore, each iteration is on average $O(1)$, and overall the algorithm is $O(K)$ in the number of iterations. □

6. SERN example

Here, we present the example of spatially embedded networks to demonstrate the algorithm.

Edges in a SERN are independent (conditional on distance), and hence the probability distribution of a SERN is given by

$$P(G) = \prod_{(i,j)\in E} p_{ij} \prod_{(i,j)\notin E} (1 - p_{ij}), \tag{6.1}$$

where p_{ij} is the probability of an edge for the specific SERN of interest. For example, in the case of a Waxman network the edge (i,j) is given by

$$p_{ij} = qe^{-sd_{ij}},$$

for nodes separated by distance d. In the Waxman formulation, d is calculated by the Euclidean distance.

Using (6.1) above, the acceptance probability when adding a link (i,j) becomes

$$\frac{P(G')}{P(G)} = \frac{p_{ij}}{1 - p_{ij}},$$

and for removing a link is

$$\frac{P(G')}{P(G)} = \frac{1 - p_{ij}}{p_{ij}}.$$

While the probability distribution of the ensemble is known, it is often difficult in practice to determine the value of $P(G)$ explicitly. Here, we only require the ratio of the probabilities between each pair of graphs, a much easier calculation.

Often, we assume that we are dealing with sparse graphs. Dense graphs are more likely to be connected and so would not require this algorithm. Additionally, in physical networks there exists a cost constraint of constructing links, resulting in many sparse real-world networks.

6.1 *Single link Markov chain: general Waxman graph*

Theorem 4.1 guarantees convergence in infinite time; however, to be practical we would like it to mix in a reasonable number of steps. We would like estimate the number of iterations K required to have evidence that the Markov chain has sufficiently converged to the stationary distribution.

In this section, we introduce the mixing time of MCMC processes on graphs in the case of the standard Waxman graph (i.e. not necessarily connected). The mixing time of the algorithm will depend on the number of nodes in the graph, and the type of graph we target in the posterior. We begin here with the simplest case: an illustrative example demonstrating that the standard Waxman graph with independent links has a mixing time of $O(N^2)$ using our algorithm. We then introduce the connectivity constraint and show empirically that this extra constraint does not slow mixing in practice.

Let us consider a standard Waxman graph with independent links. Our method is performed by proposing a change to a single node pair in each step. Therefore, let us consider that we choose a node pair (i,j) in the graph G with probability δ. In this case, we choose all node pairs with equal probability,

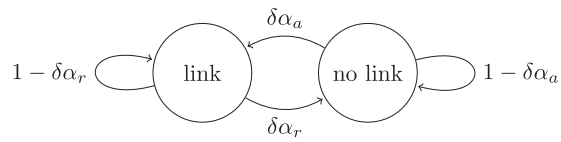

FIG. 2. Single link in the Markov chain.

that is $\delta = 2/(N(N-1))$. While we will analyse one node pair, by choosing a link in the graph with probability δ we are considering the graph as a whole.

Figure 2 shows the transition probabilities of one node pair. Note that the probability of remaining in the state is through two processes; either not choosing the node pair or choosing the link and not accepting the change. That is $1-\delta+\delta(1-\alpha) = 1-\delta\alpha$. For the node pair chosen the probability of accepting a change is

$$\alpha_{\mathrm{a}} = \min\left(1, \frac{p_{ij}}{1-p_{ij}}\right) \text{ if adding,}$$
$$\alpha_{\mathrm{r}} = \min\left(1, \frac{1-p_{ij}}{p_{ij}}\right) \text{ if removing.}$$

Combining the probability of choosing the node pair (i,j) and the transition probabilities, the transition matrix of node pair (i,j) is

$$P = \begin{array}{c} \begin{array}{cc} \text{link} & \text{no link} \end{array} \\ \begin{pmatrix} 1-\delta\alpha_r & \delta\alpha_r \\ \delta\alpha_a & 1-\delta\alpha_a \end{pmatrix} \end{array} \begin{array}{c} \text{link} \\ \text{no link} \end{array}. \tag{6.2}$$

In the limit, this converges to the stationary probability of a link between nodes i and j

$$\begin{aligned} P(\text{link}) &= \frac{\alpha_a}{\alpha_a+\alpha_r}, \\ &= \frac{\min\left(1, \frac{p_{ij}}{1-p_{ij}}\right)}{\min\left(1, \frac{p_{ij}}{1-p_{ij}}\right) + \min\left(1, \frac{1-p_{ij}}{p_{ij}}\right)}, \\ &= \begin{cases} \frac{\frac{p_{ij}}{1-p_{ij}}}{\frac{p_{ij}}{1-p_{ij}}+1}, & \text{if } p_{ij} < 0.5, \\ \frac{1}{1+\frac{1-p_{ij}}{p_{ij}}} & \text{if } p_{ij} \geq 0.5, \end{cases} \\ &= p_{ij}. \end{aligned}$$

Hence, the MCMC process will produce a Waxman network with the required link probability.

The mixing of the Markov chain is important in the application of the algorithm in finite time. The spectral gap controls the rate of exponential decay to equilibrium and the relaxation time gives an indication of how fast the chain converges. The two eigenvalues of the transition matrix (6.2) are $\lambda_1 = 1$

and $\lambda_2 = 1 - \delta\alpha_r - \delta\alpha_a$, giving a spectral gap of $\gamma^* = \delta(\alpha_r + \alpha_a)$. Note that we select edge (i,j) with probability $\delta \sim 1/N^2$ and $\alpha_r + \alpha_a$ is constant for any given link.

The relaxation time is given by,

$$\begin{aligned} t_{\text{rel}} &= \frac{1}{\gamma^*}, \\ &= \frac{1}{\delta(\alpha_r + \alpha_a)}, \end{aligned}$$

where

$$\begin{aligned} \alpha_r + \alpha_a &= \begin{cases} \frac{1}{1-p_{ij}}, & \text{if } p_{ij} < 0.5, \\ \frac{1}{p_{ij}} & \text{if } p_{ij} \geq 0.5, \end{cases} \\ &\in [1, 2]. \end{aligned}$$

Therefore, in general, $t_{\text{rel}} \sim N^2$ for the graph, and we expect that $K \sim \mathrm{O}(N^2)$ for the algorithm to converge.

Above we assume that node pair transitions are independent, as they would be in the standard Waxman graph. However, when considering connectedness, the presence or absence of other edges may prevent a particular edge being removed. This will increase the mixing time of the chain as it is possible that the most likely path from one graph to another travels through some unconnected graph.

Due to the complicated dependence structure that connectedness introduces between edges, the transition matrix does not simplify as above. A theoretical analysis on the mixing time of the connected case of sampling from $P\{G|G$ is connected$\}$ requires a more complex handling of the graph of the underlying Markov chain. Canonical path methods and bounding Markov chain properties like 'conductance' have been used to investigate the mixing time of Markov chains on graphs in different applications, such as bipartite and regular graphs [25, 30, 31]. These methods can prove upper bounds on the mixing times of chains and show mixing is polynomial in the size of the network. Bounds using these methods are at present very loose, sometimes as high as $\mathrm{O}(N^{45})$. We therefore turn to numerical simulation to explore the rate of convergence in connected Waxman graphs. Experimental evidence is often used to provide support for faster mixing chains [30], and we use this approach below in Section 7.

Nevertheless, the above analysis gives us a lower bound on and an intuition about the mixing time of our algorithm. To investigate the real mixing time, we next turn to the practicalities of implementing the algorithm and investigate the convergence.

7. Connected Waxman graphs

Section 4 showed that Algorithm 2 will converge to the distribution of connected Waxman graphs in infinite time, but we expect approximate convergence in $K \sim \mathrm{O}(N^2)$ steps. The critical question becomes, how long is required in practice?

We implement the algorithm described above using the NetworkX package in Python 2.7.13 [32] to produce connected SERNs, and provide the code.[1]

[1] https://github.com/caitlin-gray/ConnectedGraphGen

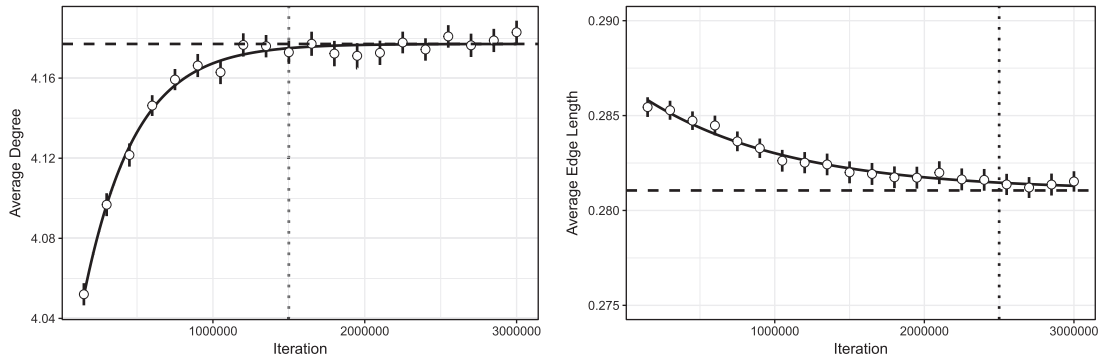

FIG. 3. Summary statistics over the MCMC process for a Waxman network with $N = 1000$ nodes. The mean of the average degree and average edge length over 200 chains are shown with 95% confidence intervals. The solid fitted regression curve is shown, and the dashed line represents the fitted asymptote. Note that there is evidence for convergence at approximately 1.5 and 2.5 million iterations for average degree and average edge length, respectively.

In order to simulate networks in finite time, we must provide evidence for the convergence of the chain. Many applications of MCMC use visual means to determine when the chain seems to have converged. Here, we use a heuristic that uses statistics of the graph.

Summary statistics are often used to describe network ensembles. Here, we utilize the distributions of two summary statistics over the ensemble to determine convergence, the distribution of average degrees and average path length. When we condition on connectedness, we expect a slight increase in average degree to allow for connectedness. This results in a shift in the distribution of the average degree over the ensemble. Conversely, we expect the average path length to decrease as the starting graph $G^{(0)}$ will have longer links than a typical Waxman graph (as we added random links to connect the graph). Note that the average edge length has particular significance in SERNs, and with the average number of edges (closely related to average degree) creates a minimal set of sufficient statistics for the parameters of the Waxman graph [17]. After convergence, we expect no change in the distribution of summary statistics of the network as they are being drawn from the same underlying distribution. We investigate the change in these statistics to provide evidence for convergence.

Figure 3 shows the confidence intervals of average degree in 200 chains of the MCMC process, that is values at intervals along the process in 200 runs of the algorithm. This demonstrates a steady increase in average degree as the algorithm progresses. We suggest that there is no significant change in average degree after 1.5 million iterations, and we have reached the average degree of $P\{G|G$ is connected$\}$. The average edge length, changes significantly but the magnitude of the change is much smaller. Additionally, it appears to converge slightly slower than the average degree, reaching within 99.9% of the fitted asymptote at $\sim$2.5 million iterations. Therefore, we have evidence that the system has converged, and we are sampling from the posterior distribution of connected Waxman networks.

7.1 *Iterations until convergence*

To estimate K, the number of steps required until convergence, we must investigate how the number of iterations to convergence scales with the number of nodes in the network. Therefore, determining convergence by eye is insufficient. We develop a framework to automate the process and give estimates of the required iterations to convergence. First, we use non-linear least squares in R [33] to fit an exponential

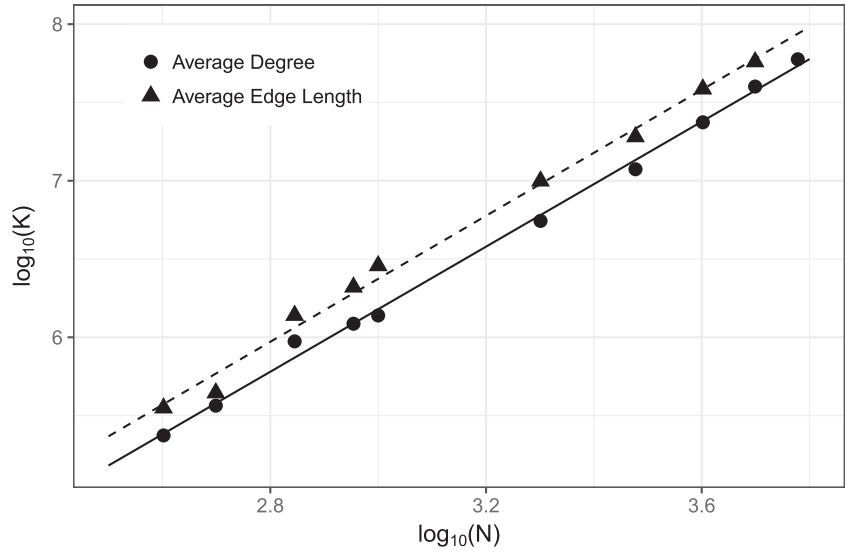

FIG. 4. Log–log plot of the iterations to convergence of the algorithm for varying size networks using average degree (circles) and average edge length (triangles) as the summary statistic. The slope of the fitted line for average degree (solid) is 1.99 ± 0.04, and for average edge length (dashed) is 2.01 ± 0.06. This supports the $O(N^2)$ mixing time expected over the edges in a network.

function to the average degree over the iterations and determine when the average degree distribution is no longer changing. The function, of the form

$$f(x) = C + Ae^{-Bx},$$

is fitted to the full data (not just the means) to determine the parameter C. This fitted parameter is used as the average degree of the target ensemble $P\{G|G$ is connected$\}$ after convergence, see Fig. 3. We define strong evidence for convergence to be when the fitted values are within 0.1% of this value.

We apply this framework to the MCMC process for varying N to determine the scaling of convergence. From the results in Fig. 4, we note that the line of best fit is a power-law with an exponent of 1.99 ± 0.04. We conclude that the mixing time of this algorithm (number of iterations to convergence) is approximately $O(N^2)$. This agrees with the theoretical analysis in Section 6.1. We note that we see the same results when fitting other functions, for example a logistic curve.

To provide further evidence for this $O(N^2)$ complexity, we conduct a similar analysis with the average edge length. We again fit an exponential model and as before the parameter C is the asymptote taken to be the average edge length of the target ensemble. The iterations until convergence, as calculated by the average edge length is shown in Fig. 4 (triangles). The average edge length converges more slowly than the average degree. This is expected as some links cannot be removed until other links provide new paths through the network. It displays the same scaling, with the exponent of the line of best fit of 2.01 ± 0.06, providing further evidence for convergence $O(N^2)$.

Combining with results from Section 5, we demonstrate the convergence of these statistics in $O(N^2)$, giving evidence that we have converged to the distribution of interest.

The practical implications of this analysis is to determine how much 'burn-in' is appropriate for graphs of size N. Burn in is the number of samples discarded from the chain to ensure we are taking

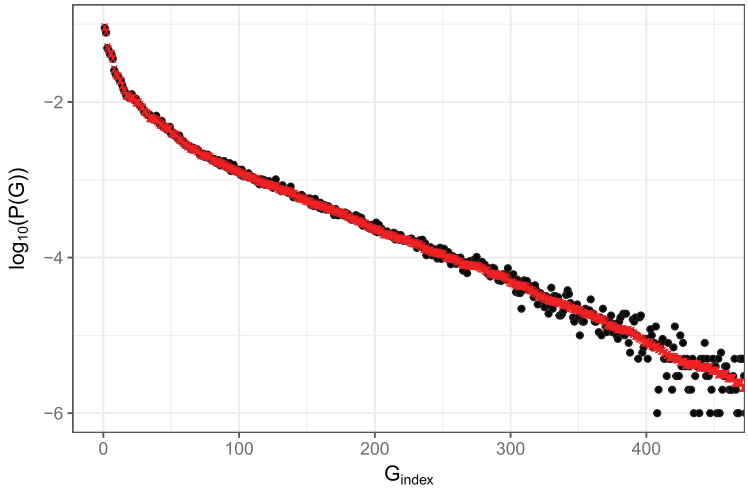

FIG. 5. Probability distribution $P(G)$ for the 450 most probable networks for a Waxman graph with $N = 5$, $s = 5$ and average degree $z = 3$ using the accept–reject method (red) and the MCMC algorithm (black). The graphs are ordered by frequency in the accept–reject algorithm.

dependent samples from the distribution of interest. Our analysis shows that the two statistics of the graph have different coefficients of convergence time, and care should be taken to determine sufficient burn-in time to have convergence of the graph itself. Often, burn-in is conservative and our analysis in the above section can be used to determine an appropriate number of iterations for the chain [34]. For example, from Fig. 3 we may choose to implement 3 or 4 million iterations before taking a sample. Note that despite this the scaling remains $O(N^2)$.

7.2 *Small network analysis*

In the above discussion, we have used graph statistics to determine the scaling of the convergence. Ideally, we would like to observe the entire ensemble of graphs and determine when we observe convergence for a variety of graph sizes. There are many reasons this is not feasible. We would have to compare our distribution with the accept–reject algorithm which is prohibitively slow and was the reason for the MCMC process in the first place. Also, the number of graphs in the ensemble is $O(2^{N^2})$, making sampling enough graphs to get a good estimate of the ensemble infeasible for even reasonable N.

To give evidence that the entire ensemble, not just some statistics, converges, we restrict the network to $N = 5$. Here we only have $2^{10} = 1024$ graphs in the ensemble. We perform accept–reject sampling to determine the empirical distribution of each connected graph. We then run one million MCMC chains to sample from the distribution after $3N^2$ iterations—using the intercept of 3 from the above analysis. We plot the distribution of networks in Fig. 5 and samples after $3N^2$ iterations produced indistinguishable distributions to that shown. This gives evidence that the convergence time for the graph statistics are also reasonable for the entire graph ensemble.

8. Discussion

We have introduced our algorithm in the context of generating connected networks. However, this method generalizes to generate networks from the probability distribution given by

$$P(G|G \text{ has some properties}),$$

assuming the properties can be tested. For example, generating a network without self loops or multi-edges would be easily implemented as above. Although we only condition on connectedness here, the process is not restricted to a single property, a set of properties can be used.

It is worth noting that the proposal distribution can affect the properties that can be tested. In this implementation, the proposal considers individual node pairs, and each step changes a single link. If we were to fix the number of triangles or exact degree sequence (i.e. the configuration model) in the network, our proposal distribution would need to facilitate this. In these cases, an 'edge swap' proposal in which the number of links remains constant, for example [9, 27], would be an appropriate choice. There are many other constraints this method could be applied to in this form, or by changing the proposal. Other types of connectivity (k-connectivity) and using other models are natural extensions, and this type of method has applications in modelling many real-world networks, for example ancestries where the relationships between animals must satisfy a variety of conditions.

The above algorithm assumes that the probability distribution of the network has the form in (6.1). However, other probability distributions, for example that of exponential random graphs [23] can easily be used. Note that we must be able to calculate the ratio of probabilities of graphs that differ by one link.

We initialize the algorithm by simulating a graph from the model of interest; for example, the Waxman network and connecting arbitrarily. However, any connected network can be used in this step as the MCMC process by design forgets the initial point of the Markov chain. This is particularly useful where the generation of the graph of interest is computationally expensive. However, starting 'further' from the distribution of interest may increase time to convergence.

We have also focussed on the simulation of a single graph, assuming that multiple graphs can be sampled by running multiple instances. However, we can sample multiple graphs from the same chain. Thinning of the chain will need to be employed to create independent samples. We expect the number of iterations until independent samples to be of the same order (not necessarily the same time) of mixing time, $O(N^2)$. This is intuitive as each node pair must have the opportunity to change to create independent graphs.

A speed up heuristic, proposed by Gkantsidis *et al.* [27] on a simple Markov Chain, attempts to reduce the requirement of checking connectedness by only running the check after T 'flip' transitions and rejecting if disconnected. This produces a concatenation of Markov chains that maintain the required stationary distribution. This speed up factor could easily be applied here to the single link M–H method. However, we only check for connectedness when the proposal removes a link, compared to every step. Rejecting all T transitions (both link additions and removals) if the graph becomes disconnected would slow mixing. Hence, it is unlikely that this speed up method would produce the same dramatic increase in complexity observed in [9]. Alternative connectivity algorithms present opportunities for improving complexity. Eppstein *et al.* [28] present a dynamic connectivity check in $O(\sqrt{N})$ per change in the graph. This is promising; however this is required at every addition or deletion of an edge, rather than only at deletion, so would not improve overall performance. These types of dynamic algorithms provide an opportunity to allow sampling of graphs with other properties, for example k-connectivity.

9. Conclusion and future work

This article describes an algorithm to create random networks from a known ensemble conditioned on an extra desired property. We use a Bayesian framework, implemented with MCMC, to generate connected random networks. This implementation can be extended to include other desired properties

of a network. We demonstrate the time complexity is $O(N^2)$ with strong evidence of convergence to the desired ensemble. Future work includes applying this algorithm to other constraints and networks, and improving the efficiency of the algorithm. Extensions of the M–H method, such as importance sampling, aim to improve mixing and complexity of convergence could also be investigated in this context [35].

Funding

Data to Decisions Cooperative Research Centre (D2D CRC); the ARC Center of Excellence for Mathematical and Statistical Frontiers (ACEMS); and an Australian Government Research Training Program (RTP) Scholarship.

References

1. Bascompte, J. & Jordano, P. (2007) Plant-animal mutualistic networks: the architecture of biodiversity. *Annu. Rev. Ecol. Evol. Syst.*, **38**, 567–593.
2. Crucitti, P., Latora, V. & Marchiori, M. (2004) Model for cascading failures in complex networks. *Phys. Rev. E*, **69**, 045104.
3. Gray, C., Mitchell, L. & Roughan, M. (2018) Super-blockers and the effect of network structure on information cascades. *Companion Proceedings of the Web Conference 2018*, WWW'18, Switzerland, 2018. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-5640-4. doi: 10.1145/3184558.3191590. pp. 1435–1441.
4. Pastor-Satorras, R., Castellano, C., Van Mieghem, P. & Vespignani, A. (2015) Epidemic processes in complex networks. *Rev. Mod. Phys.*, **87**, 925–979.
5. Payne, J. L., Dodds, P. S. & Eppstein, M. J. (2009) Information cascades on degree-correlated random networks. *Phys. Rev. E*, **80**, 026125.
6. Waxman, B. M. (1988) Routing of multipoint connections. *IEEE J. Sel. Areas Commun.*, **6**, 1617–1622.
7. Nishimura, J. (2018) The connectivity of graphs of graphs with self-loops and a given degree sequence. *J. Complex Netw.*, **6**, 927–947.
8. Rechner, S., Strowick, L. & Müller-Hannemann, M. (2017) Uniform sampling of bipartite graphs with degrees in prescribed intervals. *J. Complex Netw.*, **6**, 838–858.
9. Viger, F. & Latapy, M. (2005) Efficient and simple generation of random simple connected graphs with prescribed degree sequence. *Computing and Combinatorics* (L. Wang ed.). Berlin, Heidelberg: Springer, pp. 440–449.
10. Ying, X. & Wu, X. (2009) Graph generation with prescribed feature constraints. *Proceedings of the 2009 SIAM International Conference on Data Mining* (C. Apte, H. Park, K.e Wang & M. J. Zaki eds). Miami, Florida: SIAM, pp. 966–977, doi: 10.1137/1.9781611972795.83.
11. Erdös, P. & Rényi, A. (1959) On random graphs, I. *Publ. Math. (Debrecen)*, **6**, 290–297,
12. Gilbert, E. (1959) Random graphs. *Ann. Math. Stat.*, **30**, 1141–1144.
13. Barthélemy, M. (2011) Spatial networks. *Phys. Rep.*, **499**, 1–101.
14. Bringmann, K., Keusch, R. & Lengler, J. (2019) Geometric inhomogeneous random graphs. *Theor. Comput. Sci.*, **760**, 35–54.
15. Crucitti, P., Latora, V. & Porta, S. (2006) Centrality measures in spatial networks of urban streets. *Phys. Rev. E*, **73**, 036125.
16. Lang, J. C., De Sterck, H., Kaiser, J. L. & Miller, J. C. (2018) Analytic models for SIR disease spread on random spatial networks. *J. Complex Netw.*, **6**, 948–970.
17. Roughan, M., Tuke, J. & Parsonage, E. (2015) Estimating the parameters of the Waxman random graph. arXiv preprint: 1506.07974.
18. Ghosh, B. (1951) Random distance within a rectangle and between two rectangles. *Bull. Calcutta Math. Soc.*, **43**, 17–24.

19. HASTINGS, W. K. (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**, 97–109

20. METROPOLIS, N. & ULAM, S. (1949) The Monte Carlo method. *J. Am. Stat. Assoc.*, **44**, 335–341.

21. ROBERT, C. P. & CASELLA, G. (2005) *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Secaucus, NJ, USA: Springer New York, Inc.

22. FOSDICK, B., LARREMORE, D., NISHIMURA, J. & UGANDER, J. (2018) Configuring random graph models with fixed degree sequences. *SIAM Rev.*, **60**, 315–355.

23. LUSHER, D., KOSKINEN, J. & ROBINS, G. (2012) *Exponential Random Graph Models for Social Networks: Theory, Methods, and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 2012. doi: 10.1017/CBO9780511894701.

24. ARTZY-RANDRUP Y. & STONE, L. (2015) Generating uniformly distributed random networks. *Phys. Rev. E*, **72**, 056708.

25. COOPER, C., DYER, M. & GREENHILL, C. (2007) Sampling regular graphs and a peer-to-peer network. *Comb. Probab. Comput.*, **16**, 557–593.

26. TABOURIER, L., ROTH, C. & COINTET, J.-P. (2011) Generating constrained random graphs using multiple edge switches. *J. Exp. Algorithmics*, **16**, 1.7:1.1–1.7:1.15.

27. GKANTSIDIS, C., MIHAIL, M. & ZEGURA, E. (2003) The Markov chain simulation method for generating connected power law random graphs. *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments (ALENEX)* (R. Ladner ed.). Baltimore, Maryland: SIAM, 2003.

28. EPPSTEIN, D., GALIL, Z., ITALIANO, G. F. & NISSENZWEIG, A. (1997) Sparsification—a technique for speeding up dynamic graph algorithms. *J. ACM*, **44**, 669–696.

29. BOLLOBÁS, B., JANSON, S. & RIORDAN, O. (2007) The phase transition in inhomogeneous random graphs. *Random Struct. Algorithms*, **31**, 3–122.

30. COOPER, C., DYER, M., GREENHILL, C. & HANDLEY, A. (2019) The flip Markov chain for connected regular graphs. *Discrete Appl. Math.*, **254**, 56–79.

31. FEDER, T., GUETZ, A., MIHAIL, M. & SABER, A. (2006) A local switch Markov chain on given degree graphs with application in connectivity of peer-to-peer networks. *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science.* Washington DC: IEEE computer society, 2006.

32. HAGBERG, A., SCHULT, D. A. & SWART, P. (2008) Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conferences (SciPy 2008)* (G. Varoquaux, T. Vaught & J. Millman, eds). Pasadena, CA, USA, pp. 11–15.

33. R CORE TEAM. (2017) *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2017. URL: https://www.R-project.org/.

34. RAFTERY, A. E. & LEWIS, S. M. (1992) [Practical Markov chain Monte Carlo]: Comment: one long run with diagnostics: implementation strategies for Markov chain Monte Carlo. *Stat. Sci.*, **7**, 493–497.

35. KROESE, D. P., TAIMRE, T. & BOTEV, Z. I. (2011) *Handbook of Monte Carlo Methods*. Wiley Series in Probability and Statistics. New York: John Wiley & Sons, 2011.