Generating random networks from a given distribution

Nathan Carter, Charles Hadlock, Dominique Haughton*

Mathematical Sciences Department, Bentley College, Waltham, MA 02452, United States

Received 31 October 2007; accepted 27 January 2008
Available online 7 February 2008

Abstract

Several variations are given for an algorithm that generates random networks approximately respecting the probabilities given by any likelihood function, such as from a $p^*$ social network model. A novel use of the genetic algorithm is incorporated in these methods, which improves its applicability to the degenerate distributions that can arise with $p^*$ models. Our approach includes a convenient way to find the high-probability items of an arbitrary network distribution function.

1. Introduction

We discuss our method primarily in the context of the $p^*$ model, since this is currently the main approach for building statistical models of social networks. The $p^*$ network model, as reviewed by Wasserman and Pattison (1996), is a generalization of Markov random graphs discussed by Frank and Strauss (1986), and allows social network analysts to model networks in which holistic characteristics play a large role. In contrast to $p_1$ models, in which all dyads in the network are assumed to be independent, $p^*$ models allow the inclusion of graph measures such as hierarchialization, various types of centralization, and other quantities that measure the network as a whole (Wasserman and Pattison, 1996). We explain both models briefly in later sections.

The issue of randomly generating networks from a particular $p^*$ model has attracted considerable recent attention, for at least two reasons. One reason is that sampling can give an idea of whether a given network is “typical” among networks described by a certain $p^*$ model. Another reason is that almost all recent attempts to solve the difficult problem of the maximum likelihood estimation of the parameters of a $p^*$ model involve iterative methods which at each step simulate networks from another $p^*$ model. We briefly review this past work, and then state where our contribution fits in this stream of research.

1.1. Summary of past work

When $p^*$ models were first introduced, the proposed method of estimating parameters was pseudo-likelihood estimation, which amounted to fitting a logistic regression model, using the $p^*$ model equation but acting as if links between pairs were independent. This method had the advantage of convenience and rapidity, but unfortunately...
pseudo-likelihood estimates are not always accurate and standard errors for the parameters are not always correct (e.g., Robins et al. (2007), Faust and Skvoretz (2002, p. 277), and Robins et al. (2007)). A recent special issue of Social Networks (Volume 29, 2007) summarized developments related to the estimation and simulation of \( p^* \) models; an overview of those papers can be found in the editorial for that issue (Robins and Morris, 2007).

The best techniques available to date for simulating and estimating \( p^* \) models are those based on Monte Carlo Markov Chain (MCMC) methods; efforts in that direction are active and widely discussed. Snijders (2002) introduced an algorithm based on MCMC simulation and the Robbins–Monro algorithm (an MCMC version of the Newton–Raphson algorithm). As \( p^* \) models are part of an exponential family, their likelihood estimators can be found as solutions of moments equations. Snijders’ algorithm approximates such solutions; it was implemented in the software package SIENA, which is part of Stocnet (Stocnet, 2007).

Another approach also relies on the fact that \( p^* \) models are part of an exponential family, but uses a different algorithm, one inspired by work by Geyer and Thompson (1992); it is presented in Hunter (2007) and Hunter and Handcock (2006). This algorithm approximates the log-likelihood and then uses MCMC techniques to seek maxima of that approximated log-likelihood. It is implemented by Handcock, Hunter and colleagues in the R package Statnet (Statnet, 2007). MCMC methods are also used in the package Pnet by Peng Wang, Garry Robins and Philippa Pattison (Pnet, 2007). As pointed out in Snijders (2002, p. 25), which of these two MCMC approaches one uses is mainly a matter of convenience.

Several authors report that MCMC methods can fit a number of \( p^* \) models, but not all. Part of the reason is that the Markov assumption (discussed below) is not satisfied by many holistic graph measures (and thus many \( p^* \) models). Problems other authors encountered include convergence problems (the algorithm has trouble reaching a solution) and degeneracy problems, where the distribution on networks implied by a \( p^* \) model equation places high probability on a small number of networks and very small probability on all other networks (e.g., Robins et al. (2007)). Snijders (2002, p. 29) reports that experience with the MCMC Robbins–Monro algorithm has been rather mixed because of these problems.

The Markov assumption for networks requires that the conditional – on the rest of the network – probability of a link between a pair \((i, j)\) depend only on other ties involving \(i\) and/or \(j\). The obvious limitations of this assumption are discussed in Robins et al. (2007), Robins and Morris (2007), and elsewhere. These considerations have motivated the introduction by Snijders and colleagues (Snijders et al., 2006) of new specifications for \( p^* \) models that yield non-Markov models; see also Robins et al. (2007). However these authors report that the MCMC fitting of such models still presents some challenges and that there are situations for which MCMC methods are simply not applicable. On the other hand, success stories are also reported, as for example in Goodreau (2007) where a \( p^* \) model is fitted with Statnet to a network with over 1000 nodes. Further work on similar data is also presented in Hunter et al. (2008).

1.2. The contribution of this paper

We propose in this paper a non-MCMC method for simulating networks from any likelihood function, and in particular when that likelihood function comes from a \( p^* \) model. Our technique is centered around a simple algorithm for sampling items from any space while approximately respecting a distribution function. As our algorithm (like all other known algorithms for this purpose) suffers in accuracy when the distribution is degenerate, we introduce and illustrate two augmentations that address this problem: in the \( p^* \) case, using \( p_1 \) approximations to the \( p^* \) model, and, in the general case, using genetic algorithms. Section 2 introduces our algorithm, Section 3 covers the augmentation by \( p_1 \) approximations, and Section 4 covers the genetic algorithm, including a general technique for finding the high-probability items in a distribution of networks.

2. Sampling from large sample spaces

2.1. The problem

We aim to sample networks from the sample space \( G_n \), the set of all graphs on \( n \) vertices, with associated probabilities specified by a \( p^* \) model. Our approach is not limited to \( p^* \) models, but we focus on that case for the reasons given in the introduction. We require a likelihood function that we can evaluate for any given element of \( G_n \). The \( p^* \) model provides such a function \( L \) that maps each network to a likelihood (an element of \((0, \infty))\) (Wasserman
From such a likelihood function, it is natural to create the following naïve algorithm for choosing a network from a \( p^* \) model. First, create a probability assignment \( f \) from \( L \) by

\[
f(g) = \frac{L(g)}{\sum_{h \in G_n} L(h)}.
\]

Then order the elements of \( G_n \), say \( g_1, g_2, \ldots, g_{|G_n|} \), and compute the cumulative probability assignment \( C \) corresponding to \( f \) by

\[
C(g_i) = \sum_{j=1}^{i} f(g_j).
\]

Let \( X \) be a uniform random variable on the interval \([0, 1]\) and select a random value \( x \) from \( X \). Then choose the corresponding network \( g_i \in G_n \) by taking the smallest \( i \) such that \( C(g_i) \geq x \).

This algorithm would respect the given probabilities perfectly, but unfortunately requires computing a list of \( |G_n| \) likelihoods, as well as all the partial sums of such a list. As the number of elements in \( G_n \) is \( 2^{n(n-1)/2} \), for even relatively small values of \( n \) the sample space gets quite large. For instance, ten actors constitute a relatively small social network, yet \( |G_{10}| \approx 3.5 \times 10^{13} \). Social networks can easily have 30 actors or more, and \( |G_{30}| > 10^{130} \). Therefore this naïve algorithm requires computations that are not feasible on current computing hardware for most useful values of \( n \).

In the sections that follow, we propose various modifications to the naïve technique described above that yield practical methods for selecting random networks from \( p^* \) and other distributions and we discuss their general applicability.

### 2.2. Approximated sampling

We require a way to take a sample space \( S \) and a likelihood function \( L \) on \( S \) and from it produce a random element, reasonably respecting \( L \) but without evaluating \( L \) on very many elements of the sample space. (For now, we speak of sample spaces in general. We will return to the case when \( S = G_n \) in Section 3.) Obviously the likelihood function can at best only be approximately respected if we are not permitted to inspect all its values.

In the technique we introduce below, three factors make this approximation acceptable. First, the accuracy of the approximation is dependent on the amount of computing time one allocates to the algorithm, so that greater accuracy can be achieved if needed. Second, the amount of time needed to get a reasonably good approximation is generally a miniscule fraction of the vast time the naïve algorithm would consume. The third reason applies particularly to \( p^* \) models, which in some cases have fairly degenerate probability distributions that are problematic for both our and others’ methods. Even in such cases, where the algorithm we give below may result in a poor approximation of the actual probability assignment, the modifications we introduce in Sections 3 and 4 can substantially improve the quality of the approximation.

Assume that \( S \) is a finite sample space and \( L : S \rightarrow (0, \infty) \) is a likelihood function. Then the following algorithm draws elements from \( S \) in such a way that the probability of each \( s \in S \) being drawn approximates the probability assignment

\[
P(s) = \frac{L(s)}{\sum_{t \in S} L(t)}.
\]

The sample-weight-draw algorithm. To draw \( k \) items from a sample space \( S \) with likelihood function \( L \), use the following four-step process.

1. Randomly select a subset \( S' \subset S \) with \( k \ll |S'| = N < M = |S| \). (Below we discuss how to determine \( N \).) Choose this subset without reference to \( L \); let all elements have equal probabilities of being chosen.
2. Let \( P' \) be a probability assignment on \( S' \) defined by

\[
P'(s) = \frac{L(s)}{\sum_{t \in S'} L(t)}
\]
and compute a cumulative probability assignment $C$ for $P'$ (assuming some ordering of the elements of $S'$, say $s_1, s_2, \ldots, s_{|S'|}$).

3. Pick a random value $x$ from a uniform random variable $X$ on $[0, 1]$. The first element chosen from $S$ is then $s_i$ where $i$ is the smallest number satisfying $C(s_i) \geq x$.

4. Repeat step 3 until $k$ elements have been drawn.

Thus we are choosing an initial sample according to a uniform distribution, reweighting the elements according to their likelihood values, and choosing the final sample of $k$ items by treating the reweighted members of $S'$ as a new sample space. To maximize accuracy, one should choose as large a value of $N$ as the available computing resources allow. As discussed below, tests indicate that the ratio of $N$ to $M$ does not need to be very large for many common types of likelihood functions.

Note first the added complexity here both in the multi-step process and in the interpretation of experimental results—we now have three probability assignments to consider. We are given a likelihood function from which to sample, and we will call the corresponding probability assignment $P$ the “original”. The sample-weight-draw algorithm approximates that probability assignment, and because it is only an approximation, the probability assignment describing the output of the sample-weight-draw algorithm will be different from the original; call it “the algorithm’s probability assignment”. Finally, we run simulations to help us visualize the algorithm’s probability assignment. But because these simulations are yet another level of approximation, we have a third probability assignment, which we will call “the simulated probability assignment”. So our tests actually compute an approximation of an approximation, and compare that to the original probability assignment.

The two graphs in Fig. 1 illustrate this technique on the sample spaces $S = \{1, \ldots, 1000\}$ and $S = \{1, \ldots, 100000\}$. (Refer to the caption for details.) Note that much of the jaggedness of the simulated probability assignments in Fig. 1 is not due to the algorithm’s probability assignment, which they are approximating, but rather to the simulation; the increasing number of iterations smooths the jaggedness.

The algorithm’s probability assignment is slightly low for high-probability values, and slightly high for low-probability values; these miniscule effects come from the first step of the sample-weight-draw algorithm, which involves a uniform distribution. Although this effect is very slight in these simulations, for a degenerate distribution (one in which most of the probability mass is concentrated in a very small fraction of the elements of the sample space)
the effect can be significant enough to cause problems. In such a case, the likelihood of $S'$ containing an adequate fraction of the elements of relatively high probability is small, and yet without them any sampling from $S'$ will not be representative of the whole space. In short, distributions without much variation in the probabilities (such as those in Fig. 1) work very well with this sampling technique, but distributions that are degenerate may be ill-served by it, just as was the case with MCMC techniques. We now suggest two ways to address the problem of degeneracy for our algorithm, one in each of the following two sections.

3. Improving accuracy by $p_1$ approximations

Although the sample-weight-draw algorithm applies to more situations than just social network analysis, the case we focus on herein is to sample networks from models in the $p^*$ family. This section shows how in that specific case we can use some properties of $p^*$ functions to improve the algorithm’s accuracy. We begin by introducing both $p_1$ and $p^*$ models; more detailed information can be obtained from Wasserman and Pattison (1996).

3.1. $p^*$ models

A $p^*$ model is defined by a vector function $\vec{z}$ and a vector $\vec{\theta}$ of weights. The function $\vec{z} : G_n \rightarrow R^k$ is constructed from $k$ different network measures $z_i : G_n \rightarrow R$ by $\vec{z}(X) = (z_1(X), \ldots, z_k(X))$. The vector $\vec{\theta}$ assigns relative weights to the network measures $z_i$ that constitute $\vec{z}$.

The components of $\vec{\theta}$ are named to correspond with the components of $\vec{z}$ according to the convention in Table 1 (a more complete version of which appears in Wasserman and Pattison (1996)). For instance, if the two network measures in a model were reciprocity and cyclicity, then $z_1 = M$, $z_2 = T_c$, $\theta_1 = \rho$, and $\theta_2 = \tau_c$. Furthermore, if a $p^*$ model is introduced as having $\rho = 1$ and $\tau_c = -2.5$, no more information needs to be given; the reader can infer what $\vec{z}$ and $\vec{\theta}$ must be.

From $\vec{z}$ and $\vec{\theta}$, the $p^*$ model is then defined as an assignment of a probability to any network $X$ in a space $S$, using the following formulas. First a likelihood function $L$ on $S$ is defined by

$$L(X) = \exp(\vec{\theta} \cdot \vec{z}(X)),$$

and from it a probability assignment can be constructed by normalizing.

$$P(X) = \frac{L(X)}{\sum_{Y \in S} L(Y)}.$$

This normalization is what can be staggeringly expensive to compute, and thus earlier we stated that from a $p^*$ model all one can truly use is the likelihood function $L$.

We have said that we are particularly concerned with degenerate probability distributions. Unfortunately, the exponential nature of the $p^*$ model can easily make $L$ degenerate. Slight changes in the configuration of a network may change its scores on the measures in $\vec{z}$ only moderately, but that may change the likelihood by an order of magnitude. Fig. 2 shows an example of such an $L$. Respecting such a probability assignment when sampling would require the leftmost element to be chosen nearly half the time, and yet the chance of that one element appearing in an $S'$ with $|S'| \ll |S|$ is very small.

In such situations, we need a simple and effective way to skew Step 1 of the sample-weight-draw algorithm toward the high-probability elements of $S$. Such a skewing technique need not approximate the likelihood function faithfully – that is the original problem we are trying to solve – but it must give us a simple way to pick the high-probability items of $S$. We can use such a technique in place of Step 1’s uniform sampling approach so that items that ought to frequently be chosen when respecting the original probability assignment have a higher probability of becoming members of $S'$, from which they are also likely to be chosen.

3.2. $p_1$ models

Fortunately, for every $p^*$ model there exists a corresponding model from which it is easy to draw samples, and whose probability assignment often correlates highly with that of the $p^*$ model. The model in question is the restriction...
Fig. 2. Example probability assignment for a $p^*$ model on all graphs with five nodes, showing the exponential nature of the likelihood function, yielding a degenerate distribution. This $p^*$ model used two parameters, $\theta = 1$, $\tau_T = 5$.

Table 1
The $p^*$ measures and corresponding parameter symbols used in this paper

<table>
<thead>
<tr>
<th>Measure</th>
<th>Symbol</th>
<th>Computation</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choice</td>
<td>$L$</td>
<td>Number of ties</td>
<td>$\theta$</td>
</tr>
<tr>
<td>Reciprocity</td>
<td>$M$</td>
<td>Number of two-way ties</td>
<td>$\rho$</td>
</tr>
<tr>
<td>Cyclicity</td>
<td>$T_c$</td>
<td>Number of cyclic triads</td>
<td>$\tau_c$</td>
</tr>
<tr>
<td>Transitivity</td>
<td>$T_T$</td>
<td>Number of transitive triads</td>
<td>$\tau_T$</td>
</tr>
<tr>
<td>Intransitivity</td>
<td>$T_1$</td>
<td>Number of intransitive triads</td>
<td>$\tau_1$</td>
</tr>
<tr>
<td>Degree centralization</td>
<td>$C_D$</td>
<td>Described in Wasserman and Faust (1999)</td>
<td>$\phi_5$</td>
</tr>
<tr>
<td>Betweenness centralization</td>
<td>$C_B$</td>
<td>Described in Wasserman and Faust (1999)</td>
<td>$\phi_6$</td>
</tr>
<tr>
<td>Krackhardt’s hierarchicalization</td>
<td>$K$</td>
<td>Introduced in Krackhardt (1994)</td>
<td>$k$</td>
</tr>
</tbody>
</table>

Much of this terminology comes from Wasserman and Pattison (1996), from which a more complete coverage is available (especially see pages 415–416). That paper does not cover $K$, which is introduced in Krackhardt (1994).

of the $p^*$ model to a simpler family of social network models, the $p_1$ family. We introduce $p_1$ models briefly here, but a more thorough treatment can be found in Wasserman and Pattison (1996).

The $p_1$ family of models assumes independence of dyads in the network. Each dyad in the network can have either no ties between the two actors, ties in both directions, or one tie in either of two directions. Assume that $i, j$ are arbitrary actors in a network and use the notation $i \rightarrow j$ when $i$ has a tie to $j$, and $i \nrightarrow j$ when it does not. A $p_1$ model assigns probabilities to each of these four events for every pair of actors $i, j$ in the network. The defining equations for the most general $p_1$ model include the variables $\theta$ for choice, $\alpha_i$ for the expansiveness of actor $i$, $\beta_i$ for the attractiveness of actor $i$, and $\rho$ for reciprocity.

Of course, not all the network measures that appear in a $p^*$ model’s $\vec{z}$ function can be used in a $p_1$ model, because not all allow for the assumption of independence of network ties. But network size and density alone, which are both part of the $p_1$ model, have been shown to have powerful effects on other graph-level indices (Anderson et al., 1999), and therefore those parameters available in $p_1$ models can be an excellent approximation of a more robust $p^*$ model. For instance, the $p^*$ model shown in Fig. 2 has $\vec{z} = (\theta, \tau_T)$. The first of these two is usable in a $p_1$ context, and thus a $p_1$ approximation to this $p^*$ model would use only the $\theta$ parameter, keeping the same weight for that parameter as in the original $p^*$ model.
Table 2
Analyses of the same \( p^* \) models (and their \( p_1 \) restrictions) depicted in earlier figures

<table>
<thead>
<tr>
<th>( \theta )</th>
<th>( \rho )</th>
<th>( \tau_T )</th>
<th>( C_B )</th>
<th>( 95% )</th>
<th>( \theta )</th>
<th>( \tau_1 )</th>
<th>( C_D )</th>
<th>( 95% )</th>
<th>( \rho )</th>
<th>( k )</th>
<th>( 95% )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.5</td>
<td>5</td>
<td>5</td>
<td>90.19</td>
<td>-0.5</td>
<td>-5</td>
<td>4.58</td>
<td>-1</td>
<td>5</td>
<td>90.88</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-0.5</td>
<td>5</td>
<td>10</td>
<td>79.40</td>
<td>-0.5</td>
<td>-5</td>
<td>5</td>
<td>88.85</td>
<td>-1</td>
<td>-5</td>
<td>90.88</td>
</tr>
<tr>
<td>1</td>
<td>-0.5</td>
<td>10</td>
<td>5</td>
<td>84.28</td>
<td>-0.5</td>
<td>5</td>
<td>73.70</td>
<td>1</td>
<td>5</td>
<td>95.01</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-0.5</td>
<td>10</td>
<td>10</td>
<td>76.72</td>
<td>-0.5</td>
<td>5</td>
<td>77.84</td>
<td>1</td>
<td>-5</td>
<td>95.01</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>5</td>
<td>5</td>
<td>92.32</td>
<td>0.5</td>
<td>-5</td>
<td>65.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>10</td>
<td>10</td>
<td>98.59</td>
<td>0.5</td>
<td>5</td>
<td>77.36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>10</td>
<td>5</td>
<td>85.79</td>
<td>0.5</td>
<td>5</td>
<td>89.07</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>10</td>
<td>10</td>
<td>91.19</td>
<td>0.5</td>
<td>5</td>
<td>96.41</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parameter values have been varied (unlike in the graphs) to show that over a range of parameter space, the \( p_1 \) approximation is good at picking out the high-\( p^* \)-probability items from the space of networks. All the analyses that went into computing these values were done over the space of \( G_5 \), of size 1024.

Fig. 3. Examples of \( p_1 \) restrictions of \( p^* \) models, one that correlates well with the original \( p^* \) model (on the left, the model \( \mu = 1, k = -5 \)) and one that does not (on the right, the model \( \theta = 1, \tau_T = 5, C_B = 10 \)). In each graph, the thick gray line represents the logarithm of the \( p^* \) model probability assignment and the thin black line represents the logarithm of the \( p_1 \) restriction thereof.

3.3. Accuracy of \( p_1 \) approximations

Fig. 3 shows two examples, one of a \( p^* \) model probability assignment that correlates strongly with its corresponding \( p_1 \) restriction and one that does not correlate. Further tests not shown here indicate that the correlation happens with considerable frequency and for reasons that can be understood in terms of the particular parameters.

Before using a \( p_1 \) restriction as an approximation for a \( p^* \) model, one should run experiments such as those represented by Fig. 3 to determine whether the correlation exists for that particular case. Obviously such experiments will need to be done on spaces \( G_n \) for small \( n \). A selection of experiments is shown in Table 2. In order to know whether a \( p_1 \) model assigns high probabilities to the same elements that the \( p^* \) model does, we choose the smallest value of \( n \) such that the \( n \) highest-\( p^* \)-probability elements of the sample space account for at least 95% of the total probability. Then we ask what percentage of the total \( p_1 \) probability those same \( n \) items account for. If the \( p_1 \) model gives high probabilities when the \( p^* \) model does, this value should also be around 95%. When this is so, the \( p_1 \) model provides a reliable skewing of probabilities toward the high-\( p^* \)-probability items; when it does not, this technique is less accurate. Note that most of the values in Table 2 are at least 85%. Thus this simple and efficient modification of the algorithm is often sufficient to combat problems caused by degeneracy.

4. Improving selection using the genetic algorithm

As the solution in the previous section does not always apply, we need another. We have found that by applying a genetic algorithm (Eiben and Smith, 2003) we can improve upon the total probability of the set \( S^\prime \) initially sampled from \( S \), which we shall henceforth refer to as a “pool” of networks. We apply to it a genetic algorithm which evolves \( S^\prime \) into a pool containing the high-probability items from the space \( S \).
4.1. The mechanics of the genetic algorithm in this context

We begin with a pool $S_1$ of $N$ networks of size $n$ that we have generated by some simulation technique, such as random selection from $G_{n}^{*}$, with all being equally likely to be chosen (as in the choice of $S'$ in Step 1 of the sample-weight-draw algorithm). We shall evolve this set into a sequence of subsequent pools $S_2, S_3, \ldots$ of the same size but that over the long run will represent a more likely sample of networks chosen from a population governed by a particular $p^*$ model. Thus this problem may be regarded as an optimization problem where the objective function, called the “pool fitness” in this context, is the total likelihood of the set according to the $p^*$ model and where the independent “variables” are the $N$ networks in each $S_i$, each with its own individual likelihood or fitness value. We intend it to be inserted between Steps 1 and 2 of the sample-weight-draw algorithm, so that sets $S'$ that are not representative of degenerate distributions are first evolved into ones that are representative, before they are used.

Genetic algorithms proceed from one pool or generation $S_i$ to the next by applying rough mathematical analogs to the biological genetic processes of crossover (recombination) and mutation (of chromosomes and genes). In particular, the members of $S_i$ will “mate” in pairs, and each pair will produce a single offspring whose structure is based on the structure of the two parents, through a genetic crossover transformation. In addition, individual mutations to network structure may also be applied to the results of the crossover process. A new expanded pool consisting of the original pool plus these offspring must then be cut down by a “natural selection” process to yield a pool of size $N$, which is the desired next stage pool $S_{i+1}$. There are a number of options for how to carry out each of these steps, as discussed below. Generally one should try different variations on a particular problem, and thus there is both art and science in using this technique.

Central to this application is the mathematical representation of an arbitrary network $X$, which must provide a structure in which the evolutionary processes make reasonable sense. For networks of size $n$, the typical representation of such a network is an $n \times n$ sociomatrix of 0’s and 1’s, where a 1 in row $i$ and column $j$ means that there is a link from node $i$ to node $j$; otherwise the value is 0. We do not consider multiple links from one node to another, nor do we include links from a node to itself, so the principal diagonal is composed of 0’s.

**Selection of mating pairs from within $S_i$.** First we select a number $m$ indicating how many mating pairs there will be at each stage, each pair to produce one offspring, and then we choose the actual members of each mating pair. There are many possible approaches to this step, such as choosing elements of $S_i$ with probabilities in proportion to their fitness values, or by selecting a subset of $S_i$ with higher fitness values and then making up pairs by choosing randomly from this subset. One does not want to concentrate exclusively on only the very highest fitness values for fear of losing network characteristics that may in the long run, with other evolving changes, contribute to an even more fit pool. Mathematically, this corresponds to trying to avoid getting stuck at a local maximum.

**The crossover process for obtaining offspring from each mating pair.** Herein, we adopt the simple case of a so-called “one-point” crossover, although there are obvious extensions to more complex crossover processes. In network terms, we arbitrarily label the parents as “first” and “second,” and we partition the set of nodes randomly into two subsets, $U$ and $V$. We create an offspring with all its links copied from $U$, except that any links that are entirely within $V$ are copied from $V$. If we listed the nodes sequentially, first from $U$ and then from $V$, and formed the corresponding sociomatrix, then the block in the lower right of the offspring’s sociomatrix would thus be a copy of the same block from the sociomatrix of $V$ and the other three blocks would be copied from the sociomatrix of $U$.

**Applying the mutation process to the offspring.** Each offspring is also subject to a random mutation process (at birth), which looks at every possible link (represented by the off diagonal entries of the sociomatrix) and changes it to its opposite value according to a preset, and generally very small, mutation probability. Thus multiple mutations are possible in the same offspring, although they are less likely. Mutations also serve to let the evolving system explore alternative network structures and thus avoid getting stuck too soon at a local maximum, and thus the mutation rate is one of the control mechanisms one has available in genetic algorithm applications.

**Selection of the members of the new pool $S_{i+1}$.** Once the set of offspring has been determined, their individual fitness values are calculated using the likelihood assignment from the $p^*$ model. We now look at the combined set of networks consisting of both $S_i$ and the set of offspring, and use the fitness function in some way to select $N$ members for $S_{i+1}$. The simplest way is to select the elements with the highest individual fitnesses, which is what we have done in our tests below. Under some circumstances the need to avoid local maxima makes it desirable to pick from a slightly more diverse range.
Fig. 4. Application of the genetic algorithm to evolving a pool of networks, each with 10 nodes, over 400 iterations. The fitness function is given by the likelihood corresponding to the $p^*$ model with coefficient $C_B = 1$. This logarithmically scaled graph of total pool fitness shows that only about 200 iterations were necessary to reach a plateau.

As a first example, we consider a $p^*$ model containing only one measure, betweenness centralization $C_B$, introduced by Freeman (1977) and discussed further in Freeman (1979) and Wasserman and Faust (1999, p. 191). This particular $p^*$ model has been chosen for its theoretical utility in illustrating our approach rather than for its applied significance in social network theory. This is a non-Markov model which is not readily amenable to the MCMC approach. Fig. 4 illustrates the evolution over time of the sum of the $p^*$ likelihoods for all networks in the evolving pool, using a logarithmic scale. Note how the evolution plateaus, having found a maximum after about 200 iterations. This is further substantiated by the fact that the betweenness centralization of every network in the final pool is 0.986111; the theoretical maximum for betweenness centralization is 1.0. This first example was done for networks of 10 actors, which is fairly small. Increasing the size to 20 actors slowed the convergence (800 iterations), but the plateau was still reached, and in fact the betweenness centralization scores of all networks in the final pool were 1.0.

Our second example points out that the genetic algorithm technique, although more computationally intensive than the $p_1$ approximation technique from the previous section, functions when the $p_1$ approximation technique does not. Recall the $p^*$ model from the right of Fig. 3 with coefficients $\theta = 1$, $\tau_T = 5$, $C_B = 10$; it did not correlate well with its $p_1$ restriction, making that technique unusable. Fig. 5 shows the application of the genetic algorithm to finding a pool of high-probability items from this $p^*$ model, so that a strong $S'$ can be created from which to sample. Again, note that the objective function plateaus, indicating convergence to an optimal solution. All values of the $p^*$ log-likelihood function for the networks in the final pool lay between 103.9 and 105.

4.2. Using the genetic algorithm together with the sample-weight-draw algorithm

A natural approach for taking advantage of the genetic algorithm in combination with the sample-weight-draw algorithm is to use it to upgrade the fitness of the sample $S'$ before drawing the $k$ representative networks. However, the “inbreeding” process inherent in the generation of successive pools of networks may differentially affect graph-level indices that co-vary with the fitness that is being increased. While this is an as yet unexplored theoretical issue that no doubt depends on the particular parameters in the model, we can avoid this concern by choosing $k = 1$ in the sample-weight-draw algorithm and beginning with a separate pool $S'$, subsequently improved by the genetic algorithm, for each additional sample network that we wish to generate. We do this because many sampling applications use the sample to test hypotheses about the distribution of graph-level indices not built into the original model, and thus we must avoid computational steps that can change this relationship.
Fig. 5. Application of the genetic algorithm to evolving a pool of networks, each with 10 nodes, over 600 iterations. The fitness function is given by the likelihood corresponding to the $p^*$ model with coefficients $\theta = 1$, $\tau_T = 5$, $C_B = 10$. This logarithmically scaled graph of total pool fitness shows that only about 300 iterations were necessary to reach the maximum.

5. Conclusion

This paper makes three contributions related to sampling networks from $p^*$ models. The first appeared in Section 2, where we introduced a simple but very flexible algorithm for sampling from large sample spaces while approximately respecting the probabilities of an arbitrary distribution function, which need not be normalized. Though this algorithm has applications beyond sampling networks from $p^*$ models, that was the application to which the rest of the paper put it. The remainder of the paper addressed the degeneracy issues common to all network sampling techniques, the difficulty of finding the highest-probability elements in a space where they are a very small percentage of the total number of elements.

The second contribution of this paper is in Section 3, which relies on the fact that the graph-level indices available in $p_1$ models have a strong impact on other graph-level indices (Anderson et al., 1999). A natural consequence of this fact is that the restriction of a $p^*$ model to its $p_1$ portion often has enough correlation to be helpful in finding the high-$p^*$-probability elements of a space $G_n$. As it is both easy and fast to sample networks from a $p_1$ distribution, this technique, when applicable, is a very efficient way to make our naive sampling algorithm perform well.

As the $p_1$ restriction of a $p^*$ model does not always correlate well with the original $p^*$ model, we needed a more general technique. The more general technique suggested here is the application of genetic algorithms in Section 4, the final contribution of this paper. That approach has been shown in the examples above to be a powerful way to hone in on high-$p^*$-probability items in a space $G_n$. Although we intend this algorithm for finding high-probability items to be used for improving accuracy when sampling from $p^*$ distributions (as stated in Section 4), it is neither limited to the case of sampling, nor to distributions from the $p^*$ family. The $p_1$ restriction technique remains useful when it applies, because it is more efficient than the genetic algorithm.

The method of the genetic algorithm introduced in Section 4 contrasts with the existing MCMC techniques in a few notable ways. Existing MCMC techniques treat the space $G_n$ as a state space for a Markov chain, with transitions that only change one edge in a network at a time; the combination method for our genetic algorithm is much more holistic, in that it “mates” two networks, keeping large parts of each. Our tests have shown that this seems to work well in the $p^*$ context, where models are built from holistic measures that violate the Markov assumption. Furthermore, because our combination and mutation methods create networks that, from the Markov chain point of view, are far removed from their parents (in that many individual ties would need to be flipped to walk from a parent to a child in the state space $G_n$), and therefore one can think of them as exploring a greater diversity of the state space in fewer
iterations than a walk that flips only one matrix entry at a time. These two differences give some intuition about why the seemingly undirected progress of the genetic algorithm can be successful in situations where MCMC techniques fail.

Acknowledgments

We are grateful to two anonymous referees of an earlier version of this manuscript for connecting us to part of the literature of which we had been unaware and thus for encouraging us to further develop and to focus on the distinct contribution of this work.

References