*Network Science* **4** (1): 117–139, 2016.  
doi:10.1017/nws.2015.35

# *An improved MCMC algorithm for generating random graphs from constrained distributions*

TREVOR TAO

*Defence Science and Technology Organisation, Edinburgh, South Australia 5111, Australia*
*(e-mail:* trevor.tao@dsto.defence.gov.au*)*

**Abstract**

We consider the problem of generating uniformly random graphs from a constrained distribution. A graph is valid if it obeys certain constraints such as a given number of nodes, edges, $k$-stars or degree sequence, and each graph must occur with equal probability. A typical application is to confirm the correctness of a model by repeated sampling and comparing statistical properties against empirical data. Markov Chain Monte Carlo (MCMC) algorithms are often used, but have certain difficulties such as the inability to search the space of all possible valid graphs. We propose an improved algorithm which overcomes these difficulties. Although each individual iteration of the MCMC algorithm takes longer, we obtain better coverage of the search space in the same amount of time. This leads to better estimates of various quantities such as the expected number of transitive triads given the constraints. The algorithm should be of general interest with many possible applications, including the world wide web, biological, and social networks.

**Keywords:** *MCMC, adjacency matrix, graph theory, constrained distribution, social network, triad census*

## 1 Introduction

An important and interesting mathematical problem is sampling random graphs from a constrained distribution. By constrained we mean that the graph must satisfy certain properties such as a fixed number of nodes or edges, or fixed in-degree and out-degree sequence. A typical application is to confirm the correctness of a model by testing it against empirical data. For example, we might postulate that a social network can have $N$ nodes and $M$ edges for some $N, M$. We could generate a large number of random networks with the correct values of $N$ and $M$ and observe some statistics, such as the size of the largest connected component (Chung & Lu, 2002) or the number of transitive triads (Lusher *et al*., 2012). We might find that the observed statistics do not match that of empirical social networks, and we would conclude that the above model is not valid for social networks.

The above model is the famous Erdös–Rényi model (Erdös & Rényi, 1959). This is a conditionally uniform model (Snijders, 2011) in the sense that any graph with the correct number of nodes and edges occurs with equal probability and any other graph has probability zero. Clearly, generating from a uniform distribution is desirable so we can ensure there is no bias in a large sample of random networks. Note that a constrained uniform distribution is different to an Exponential Random

Graph Model (ERGM) (Lusher *et al.*, 2012; Holland & Leinhardt, 1981) in the following sense: if two graphs have the same statistical properties as specified by the ERGM then they are equally probable, but any deviation from the "correct" statistics does not imply a probability of zero.

One can consider more complex models: for example we might postulate that each node has a specified in-degree or out-degree (Blitzstein & Diaconis, 2010). Or we might specify a certain number of reciprocated links, transitive triads or $k$-stars (Lusher *et al.*, 2012) for the network as a whole. Simple problems such as the Erdös–Rényi model have been extensively analyzed (see e.g. Lusher *et al.* (2012) and references therein). However, progress on more complex problems has been limited. The crux of the problem is that obtaining the uniform distribution over all valid graphs is non-trivial. At the most basic level, one can generate random graphs by repeatedly sampling among all graphs with $N$ nodes and discarding the graphs that violate the constraints. But this is extremely inefficient since most graphs will be discarded. Conversely, one can generate valid graphs very quickly, but at the cost of giving up the uniform distribution. To illustrate these difficulties, consider the problem of placing a rook, knight, bishop, and queen on a chessboard such that no piece attacks any other. We wish to assign a probability of $1/k$ to each solution, where $k$ is the number of solutions. It turns out that placing the pieces on "safe squares" one at a time will not yield the uniform distribution. Moreover, the distribution depends on the order in which the pieces are placed. The uniform distribution cannot be obtained unless all four pieces are placed simultaneously. Similar problems occur when constructing valid graphs (or the corresponding adjacency matrices) satisfying given constraints.

One of the simplest non-trivial problems is that of generating a uniformly random matrix $A$ with given row and column marginals (i.e. row and column sums, or the number of ones in the case of a binary matrix), which we will refer to as Problem URC. Note that this is equivalent to sampling bipartite graphs with a given degree sequence (Erdös *et al.*, 2013). This problem has been studied in the context of real-world networks including the world wide web, social networks, and biological networks (Blitzstein & Diaconis, 2010; Newman, 2003). One can also constrain the main diagonal to be all zeros, which we will refer to as Problem URC0. This corresponds to forbidding self-loops between nodes in a network.

Some early researchers have attempted to use brute force to solve Problem URC or URC0 by filling in "forced moves" or making random guesses if no forced moves are available. This leads to the same difficulties as the chessboard problem discussed earlier, and their methods are infeasible even for moderate datasets (Rao *et al.*, 1996). Rao *et al.* (1996) proposed a simple MCMC algorithm to ensure the distribution is uniform. For details we refer the reader to Rao *et al.* (1996) and references therein.

A more challenging problem is to sample graphs with a fixed number of mutual dyads, where two nodes in a graph have links to each other. This corresponds to two 1s in an adjacency matrix being mirror opposites with respect to the main diagonal. Thus one can consider an extended version of the above problem, where the row/column marginals and number of mutual dyads are fixed, which we will refer to as Problem URCM (and similarly for Problem URCM0). McDonald *et al.* (2007) proposed an MCMC algorithm for the URCM0 problem, but were unable to

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

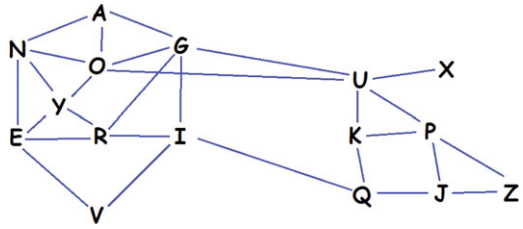



Fig. 1. Markov chain Monte Carlo example. (Color online)

conclusively show that their algorithm produced an irreducible Markov chain (c.f. Section 2). In this report, we propose an improved version of the MCMC algorithm. Section 2 describes MCMC methods. Section 3 describes previous algorithms. Section 4 describes the proposed algorithm. Section 5 describes the experiments, and Section 6 is the conclusion.

We emphasize that the proposed algorithm is not tied to any particular application and should be of general interest. For instance it is not hard to extend this algorithm so that the number of transitive triads or $k$-stars remain constant, as will become clear in Section 4. This is of particular relevance to social networks (however, it remains an open question if the resulting algorithm will be efficient enough to be practical).

## 2 Markov chain Monte Carlo algorithms

MCMC is an algorithm for generating samples from a set of possible states. An MCMC is defined by specifying the probability of moving from any state to any other state (including itself). MCMC algorithms are useful because it is often easy to calculate a change in energy potential (or other function such as log-energy or probability) when moving from one state to a "nearby" state, but impractical to calculate the actual energy potential (for the URC and URCM problems even the uniform distribution is hard because one cannot easily enumerate all legal states).

An example is shown in Figure 1. States are represented by letters and adjacent states are indicated by the solid lines. Adjacent means that the probability of going from one state to another is strictly positive. In this example, it is assumed that the adjacency property is symmetric, (i.e. A is adjacent to B if and only if B is adjacent to A, but the transition probabilities need not be equal) and reflexive (i.e. each state is adjacent to itself). These properties are often desirable, but not necessary for a Markov chain. An example path would be N-E-V-E-R-G-O-N-N-A-G-I-V-E-Y-O-U-U-P.

The purpose of generating such a path is to enable estimation of the distribution of states. For instance, after a large number of iterations, one might find that states E,A,Q occur 12.7% 8.2%, and 0.01% of the time respectively. One could then use these values as estimated values of $\pi(E), \pi(A), \pi(Q)$. Similarly one can estimate a distribution over some function of states. For example, suppose that each state represents an adjacency matrix. We could count the number of transitive triads $T(s)$ for every state $s$, tabulate the results as shown in Table 1 and estimate the distribution of $T$.

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

Table 1. *Number of transitive triads for each state* $s \to T(s)$.

| | | | |
|---|---|---|---|
| $A \to 1$ | $E \to 1$ | $G \to 2$ | $I \to 1$ |
| $J \to 8$ | $K \to 5$ | $N \to 1$ | $O \to 1$ |
| $P \to 3$ | $Q \to 10$ | $R \to 1$ | $U \to 1$ |
| $V \to 4$ | $X \to 8$ | $Y \to 4$ | $Z \to 10$ |

Observe that the adjacency graph of Figure 1 contains two components which are "thinly" connected via three links namely, GU, OU, and IQ. The path N-E-V-E-R-G-O-N-N-A-G-I-V-E-Y-O-U-U-P starts in the left component and only reaches the right component in the final three states. This illustrates a typical problem with Markov chains in that a random walk can often get stuck in one component for some period of time and there is no easy way of detecting this. The ideal situation occurs when all possible states lie in a single component, but this can't always be guaranteed. In mathematical terms, this problem occurs if it is only possible to reach $B$ from $A$ by following a low-probability link from one component to another. Or worse still, the components may not even be connected, in which case one will never reach $B$ for any path length. A more subtle (though less important) problem occurs when it is only possible to reach a certain state at periodic intervals, e.g. only after $10k$ iterations for some integer $k$. In practice this is rarely a problem and is only mentioned for sake of completeness.

One can summarize the above discussion using the concepts of irreducibility, positive recurrent, and aperiodicity. Irreducibility means that any state can be reached from any other state (i.e. one cannot separate the Markov Chain into multiple components). Positive recurrence means that if one starts at state $i$, one must eventually return to state $i$. Moreover, the expected number of iterations required to return to state $i$ must be finite[1]. Aperiodicity means that there is no period $k > 1$ such that one can return to state $i$ only if the number of iterations is a multiple of $k$. The importance of this resides in the following well-known theorem:

*Theorem 1*
If a Markov chain is positive recurrent, irreducible, and aperiodic, then there exists a steady state distribution $\pi$ such that

$$P[X_n = j | X_0 = i] \to \pi(j) \quad \forall i, j \quad as \quad n \to \infty \tag{1}$$

where $P$ is the transition matrix. Also, the expected recurrence time for state $i$ is finite.

This theorem implies that one can perform a random walk on the possible states, and after a sufficiently large number of iterations one can count the number of occurrences of each state and get a reasonable approximation to the actual distribution. We should mention that in practice there is no easy way of determining how many iterations are "sufficiently large". We do not attempt to address this problem in this report.

[1] This last caveat is only of concern when dealing with an infinite number of states, so will not concern us here.

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

### 2.1 Metropolis–Hastings algorithm

A popular variant of MCMC algorithms is the Metropolis-Hastings algorithm. One property of the MCMC algorithm is that it is legal to move from a state with high probability to one with lower probability; this is obviously necessary to avoid the path terminating at a globally or locally optimum state. The MH algorithm makes this explicit by defining proposal and acceptance probabilities. Given an old state $S_{old}$ a new state $S_{new}$ is proposed with probability $p_{prop}(S_{new}|S_{old})$. If the proposed new state has higher probability than the old state, the probability of acceptance $p_{acc}(S_{new}|S_{old})$ is 1; if the new state has lower probability then $p_{acc}(S_{new}|S_{old})$ is a ratio of probabilities of old and new states. If the proposed new state is rejected, then $S_{new} = S_{old}$. For more details see Rubinstein *et al.* (2013).

## 3 Previous algorithms

We consider the problem of generating a random adjacency matrix that is "compatible" with a given matrix $A$. We say that two matrices are compatible if they have the same row and column marginals and mutual dyads (in the case of URCM0).

### 3.1 Problem URC0

Recall that Problem URC0 means that the row and marginal columns are constant and the diagonal entries must be zero. In this case, the diagonal zeros are called structural zeros (Rao *et al.*, 1996).

Given an adjacency matrix $A$ a simple algorithm for computing the next iteration $A'$ is:

- Select a random tetrad, defined by the intersection of two columns and two rows, avoiding any structural zeros (as shown in Figure 2(a).
- If the tetrad is $[1,0;0,1]$ or $[0,1;1,0]$, then invert it to obtain a new matrix $A'$. Otherwise set $A' = A$.

This ensures that row and column marginals are constant, but Rao *et al.* (1996) proved that the Markov chain is not necessarily irreducible. They proved that the Markov chain becomes irreducible if symmetric hexad swaps were allowed as well as tetrad swaps. A symmetric hexad swap is obtained by inverting six entries of the form $A(i,j)$, $A(i,k)$, $A(j,k)$, $A(j,i)$, $A(k,i)$, $A(k,j)$ as shown in Figure 2(b).

### 3.2 Problem URCM0

Problem URCM0 is significantly harder than URC0, since the algorithm of Rao et al. does not work for problem URCM0. Other researchers have proposed adding operations such as swapping non-symmetric hexads (McDonald *et al.*, 2007) and symmetric opposite tetrads (Roberts, 2000) as shown in Figure 2(c), (d). For discussion of these algorithms we refer the reader to McDonald *et al.* (2007) and references therein.

McDonald *et al.* (2007) proposed an open question as to whether these extensions implied an irreducible Markov Chain. Unfortunately we have a simple counterexample to show this is not so.

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

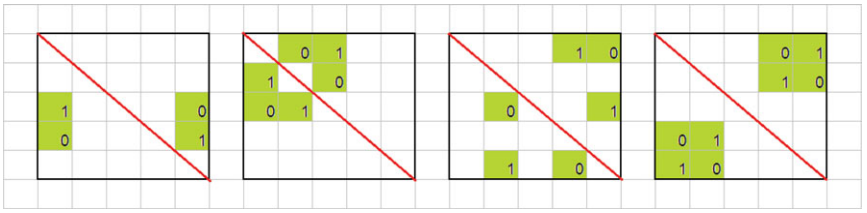


Fig. 2. (a) Tetrad, (b) symmetric hexad, (c) asymmetric hexad, and (d) symmetrically opposite tetrads. (Color online)

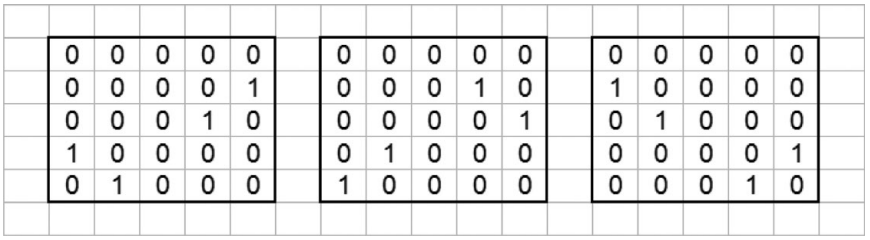


Fig. 3. Counterexample to McDonald et al.'s hypothesis.

In Figure 3, there are three compatible adjacency matrices. The reader can draw the networks corresponding to these matrices and convince him- or herself there is no other $5 \times 5$ adjacency matrix with the same row/column marginals and one mutual dyad. But these matrices do not differ from each other by single tetrad swap, symmetric hexad, asymmetric tetrad, or symmetrically opposite tetrads (although the matrices are isomorphic, permuting rows/columns is not a valid operation). Thus the conjecture is false.

It will be noted that symmetric hexad, asymmetric hexad or symmetrically opposite tetrad updates are composed of two tetrad updates (for hexads, the two tetrad swaps will "intersect" at one entry). This suggests that a natural extension would be to allow any two tetrad swaps in a row. Thus the first tetrad swap is allowed to change the number of mutual dyads (or insert a one on the main diagonal in the case of symmetric hexads), provided it is corrected by the second swap. Indeed, this would solve the counterexample in Figure 3, so one could reasonably conjecture that double tetrad swaps will ensure the chain is irreducible. Unfortunately, a second counterexample shows even this conjecture is false. Figure 4 shows two $12 \times 12$ compatible adjacency matrices, but it is not possible to obtain one from the other using double tetrad swaps. Using trial and error, one can verify that most of the matrix entries are "fixed" in the sense that they cannot be inverted even if one ignores the mutual dyads constraint. Further experimentation with the remaining entries shows that it is not possible to transition from one adjacency matrix to the other if any double-tetrad swaps are allowed.

If the Markov chain is reducible (or one cannot prove it is irreducible), then a common strategy is to extend the set of allowable states $S$ to a new set $S'$ to ensure all states are connected. This is illustrated in Figure 5.

In the left half of Figure 5 one has 15 states in 3 unconnected components so the Markov Chain is reducible. In the right half, we have added some extra states (unshaded) to connect these components into one large component. The obvious

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 0 | 0 | 0 | x | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | x | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | x | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | x | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | x | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | x | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | x | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | x | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | x |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | x | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | x | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | x | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | x | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | x | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | x | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | x | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | x | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | x |

Fig. 4. A second counterexample.

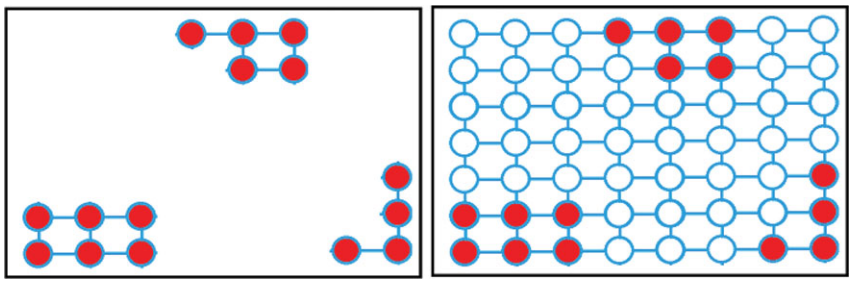


Fig. 5. Adding states to a Markoc chain to make it irreducible. (Color online)

disadvantage of this approach is that one must reject states in $S' \setminus S$ in the final path, so one needs more iterations in the Markov chain to obtain a good estimate of the correct distribution. One can tweak the Markov Chain so that states in $S$ have more probability than states in $S' \setminus S$. This results in a trade-off between "rejection" and "coverage". If too much weight is placed on $S$, then the Markov Chain will probably become trapped in an isolated component. Thus one will need a large number of iterations to obtain a good coverage of $S$. If not enough weight is placed on $S$ then one will reject too many states in the final path. Note that all states in $S' \setminus S$ need not have the same probability. If an adjacency matrix is such that the number of mutual dyads is very far from the correct value, one can assign a lower probability than a matrix whose number of mutual dyads is only slightly incorrect. Also note that tweaking the Markov Chain implies that the distribution on all states is no longer uniform. Thus the Metropolis–Hastings algorithm described in Section 2.1 can be used.

This strategy was used by McDonald *et al.* (2007). A matrix is a valid state if the marginals $x_{i+}, x_{+j}$ are correct, regardless of $M$. Since one is only interested in matrices with the correct value of $M$, one should only retain these matrices in the final path. From Rao *et al.* (1996), we know that the Markov Chain is irreducible. The tradeoff between coverage and rejection is achieved by introducing a parameter $\beta$. More specifically, given $A$, the new matrix $A'$ is accepted with probability

$$\min[\exp(-\beta(|m_0 - M(A')| - |m_0 - M(A)|)), 1] \tag{2}$$

where $m_0$ is the desired number of mutual dyads and $M(A), M(A')$ is the number of mutual dyads for matrices $A, A'$. One can easily show that a transition from state $A$

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

to $B$ will have acceptance probability less than one if and only if the new state $B$ has a mutual dyad count further from $m_0$ from that of $A$.

If $\beta$ is large, most of the matrices will be retained but at the risk of missing some isolated component of compatible matrices as illustrated in the left panel of Figure 5. Conversely, if $\beta$ is small there is a danger of discarding too many matrices, particularly if the algorithm is "lost at sea" exploring a random area of the solution space with $M$ very far from the correct value. Unfortunately there is no obvious method of choosing $\beta$.

## 4 Proposed algorithm

We take a completely different approach. Roughly speaking, we want the ability to update any number of entries at once, instead of being restricted to tetrad or double-tetrad swaps (which can only invert at most eight entries). This will trivially imply the Markov chain is irreducible. But this poses some obvious problems, since for an $N \times N$ matrix, there are $\mathcal{O}(2^{N^2})$ possible adjacency matrices to consider. We essentially want a fast algorithm that can choose a new adjacency matrix whilst ensuring the Markov chain is irreducible.

### *4.1 Preliminaries*

*Definition 1*
An alternating cycle is a sequence of the form $(i_1, i_2) \to (i_3, i_2) \to (i_3, i_4) \to \cdots \to (i_{2n+1}, j_{2n})$ with $i_{2n+1} = i_1$ and $i_k \neq i_{k+1}$ for all $k$, such that the entries of the matrix $A(i_1, i_2), A(i_3, i_2), \ldots$ are alternating 1s and 0s.

This means that the alternating cycle consists of a sequence $1, 0, 1, 0, 1, 0 \ldots$ tracing alternating horizontal and vertical line segments on the adjacency matrix $A$, and not intersecting with the main diagonal. Clearly, the length of an alternating cycle is even and at least four. If the length is exactly four, the cycle is called an alternating rectangle (Rao *et al.*, 1996).

*Lemma 1*
If $A$ and $B$ are two matrices with the same row and column marginals, then the entries where $A$ and $B$ differ are a disjoint union of 0 or more alternating cycles.

*Proof*
If $A = B$, then the entries where $A$ and $B$ differ are a disjoint union of 0 alternating cycles. Suppose $A \neq B$. Without loss of generality choose an element $(i, j)$ such that $A(i, j) = 1$ and $B(i, j) = 0$. Since $A$ and $B$ must have the same marginal total for row $i$ there must be a $j'$ such that $A(i, j') = 0$ and $B(i, j') = 1$. We can apply the same argument to column $j'$ to show there must be an $i'$ such that $A(i', j') = 1$ and $B(i', j') = 0$. Continuing this argument one must eventually obtain an alternating cycle (since $A$ is finite). Now define $A_1$ by inverting all elements on the alternating cycle. Then, $A_1$ is closer to $B$ than the original $A$. By repeating this procedure we can obtain $A_2, A_3 \ldots$ where each $A_i$ is closer to $B$ than the previous $A_{i-1}$. Eventually we must obtain $A_n = B$ for some finite $n$. Then the entries where $A, B$ differ must be a disjoint union of $n$ alternating cycles. □

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

We note that Rao *et al.* (1996) had a similar proof for problem URC0, but only considered swapping alternating rectangles instead of entire alternating cycles of length more than four.

This suggests the following algorithm for computing the next iteration $A_{new} = f(A_{old})$

- set $A_{temp} = A_{old}$
- LOOP
  - — choose a random alternating cycle that doesn't intersect with a previous alternating cycle.
  - — Invert all elements of $A_{temp}$ on the alternating cycle
  - — if $M(A_{temp}) = M(A_{old})$ then return $A_{new} = A_{temp}$.
  - — if there are no more cycles return $A_{new} = A_{old}$.
- END LOOP

Note that, this algorithm is somewhat similar to the Metropolis–Hastings algorithm in that we propose a new matrix $A_{temp}$ and accept it if it is compatible with $A_{old}$. However, one difference is that if $A_{temp}$ is rejected, then we do not immediately set $A_{new} = A_{old}$ since we still have another chance to obtain $A_{new} \neq A_{old}$ for every new alternating cycle.

As an aside, we note that checking that there are no more cycles may be awkward, so we will "cheat" by picking randomly and if an old cycle is repeated then we will simply return $A_{new} = A_{old}$ immediately.

We now answer the problem of choosing a "random alternating cycle".

### *4.2 Selection of random alternating cycles*

The proof of Lemma 1 is based on finding alternating cycles in the adjacency matrix using arbitrary "pointers" to entries in the same row/column, and we can do something similar. Given an adjacency matrix, we consider each entry $A_{ij}$. If $A_{ij} = 1$, then assign a pointer to a random zero on the same column, but not the structural zero. If $A_{ij} = 0$, then assign a pointer to a random 1 on the same row. If no 0 or 1 is available, then point to the structural zero on the same row or column.

To choose a random alternating cycle, simply pick a "starting seed" $(i, j)$ at random and follow the pointers. There are three possible outcomes: (i) one hits a new cycle, (ii) one repeats an old cycle, (iii) one hits a structural zero. In cases (ii) or (iii), set $A_{new} = A_{old}$. In case (i) a success occurs if the number of mutual dyads $M$ is the correct value, otherwise pick a new starting seed and repeat.

To be more specific, the algorithm is as follows:

Problem: determine the next iteration $A_{new}$ given $A_{old}$.

- Set $A_{temp} = A_{old}$.
- Assign pointers: For each $A_{temp}(i, j)$
  - — if $i = j$, then set $P(i, j) =$ `NULL` else,
  - — if $A_{temp}(i, j) = 0$, then choose a random $j' \neq i$ such that $A_{temp}(i, j') = 1$. If no $j'$ exists, then $j' = i$. Set $P(i, j) = (i, j')$.
  - — if $A_{temp}(i, j) = 1$, then choose a random $i' \neq j$ such that $A_{temp}(i', j) = 0$. If no $i'$ exists, then $i' = j$. Set $P(i, j) = (i', j)$.

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

- Assign a "seed order". Let $R$ be a random permutation of the numbers $1 \dots N^2$ arranged in a square matrix.
- set $k = 1$.
- LOOP
  - — Find $i, j$ such that $R(i, j) = k$. Start with $A_{temp}(i, j)$ and follow the pointers in $P$ until you hit a structural zero or an alternating cycle.
  - — If you hit a structural zero, then return $A_{new} = A_{old}$
  - — If you hit an alternating cycle that is marked `VISITED`, then return $A_{new} = A_{old}$
  - — If you hit an alternating cycle that is not marked `VISITED`, then
    - Invert all entries of $A_{temp}$ corresponding to the alternating cycle
    - If $M(A_{temp}) = M(A_{old})$, then return $A_{new} = A_{temp}$
    - Otherwise mark the alternating cycle as `VISITED`, increment $k$ and NEXT LOOP
- END LOOP

*Remark 1*
In the introduction, we alluded to the fact that the algorithm could be extended to add more constraints such as fixed number of transitive triads or $k$-stars. This is obtained by replacing $M$ with a vector e.g. $\Theta = (M, T, k_3)$, where $T$ is the number of transitive triads and $k_3$ is the number of 3-stars. Obviously the extra constraints will imply a higher probability of $A_{new} = A_{old}$, but the basic algorithm remains the same.

We need to prove that this algorithm does indeed produce samples from $U|x_{i+}, x_{+j}, M$.

Suppose that $A$ and $B$ are compatible. From Lemma 1, it is possible to move from $A$ to $B$ in a finite number of iterations of the Markov chain. All that is required is to select the correct alternating cycles, since Lemma 1 guarantees they exist. The number of iterations required to move from $A$ to $B$ will depend on how many times the correct number of mutual dyads is obtained when inverting the alternating cycles one at a time. For instance, suppose that four alternating cycles were inverted to obtain $A^{(6)} \to A_1^{(6)} \to A_2^{(7)} \to A_3^{(8)} \to A_4^{(6)} = B^{(6)}$, where the superscripts indicate the number of mutual dyads. Then it would take two iterations to change $A$ to $B$, i.e. $x_0 = A, x_1 = A_1, x_2 = A_4 = B$, where $x_0, x_1, \dots$ is the Markov chain. Since this argument works for any compatible matrices $A, B$ and any sequence $A_1, A_2, \dots$, the Markov chain is irreducible. There are only a finite number of possible pointers for each entry, and the number of ways to select starting seeds is also finite. Therefore, we have positive recurrence. To prove aperiodicity, we only need ensure $p(A|A) > 0$ for some $A$ i.e. we can reach $A$ from $A$ in one step. The simplest solution is to allow the first starting seed to be a structural zero in which case we automatically have $A_{new} = A_{old}$.

It only remains to prove we get the correct distribution $U|x_{i+}, x_{+j}, M$. A sufficient (but not necessary) condition is detailed balance i.e.

$$\pi(A)p(B|A) = \pi(B)p(A|B) \tag{3}$$

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

where $A, B$ are matrices and $\pi$ is the steady state distribution. Since we want the uniform distribution, it suffices to show that $p(B|A) = p(A|B)$.

Note that, the algorithm requires us to generate random matrices $P, R$ to generate the next iteration. In other words, $B$ is a function of $A, P, R$. Let us denote this as $\phi(A, P, R) = B$. Before we can prove detailed balance, we need an algorithm for determining an inverse $P$ and $R$, i.e. given $A, P^{\uparrow}, R^{\uparrow}, B$ such that $\phi(A, P^{\uparrow}, R^{\uparrow}) = B$ determine $P_{\downarrow}, R_{\downarrow}$ such that $\phi(B, P_{\downarrow}, R_{\downarrow}) = A$. The algorithm is given below:

- Given: $A_{old}, P^{\uparrow}, R^{\uparrow}$. Required: $P_{\downarrow}, R_{\downarrow}$.
- Set $A_{new} = \phi(A_{old}, P^{\uparrow}, R^{\uparrow})$
- If $A_{new} = A_{old}$, then set $c = 0$. Otherwise let $c$ be the number of alternating cycles inverted to obtain $A_{new}$.
- Find the entries $1, 2, \ldots, c$ in $R^{\uparrow}$ and replace with $c, c-1, \ldots, 1$ to obtain $R_{\downarrow}$.
- Set $P_{\downarrow} = P^{\uparrow}$
- Define the active cycles to be the cycles of $A_{old}$ that are inverted. Each cell within an active cycle is called an active cell.
- For each active cycle, adjust the entries in the pointer matrix $P_{\downarrow}$ to point in the "opposite" direction of the cycle. This means that if $P^{\uparrow}(i, j) = (i' j')$, then $P_{\downarrow}(i', j') = (i, j)$.
- For each *inactive* cell $(i, j)$ pointing to an active cell $(i', j')$, set $P_{\downarrow}(i, j) = P_{\downarrow}(i', j')$ (note that each $P_{\downarrow}(i', j')$ was reversed in the previous step).

*Lemma 2*
The above algorithm satisfies $A_{old} = \phi(A_{new}, P_{\downarrow}, R_{\downarrow})$

*Proof*
Let the number of alternating cycles swapped be $c$. If $c = 0$, then $A_{new} = A_{old}, P_{\downarrow} = P^{\uparrow}, R_{\downarrow} = R^{\uparrow}$ and the proof is trivial. Hence we can assume $c > 0$. The above algorithm generates $A_{old} \to A_1 \to \ldots \to A_c = A_{new}$ such that each $A_i$ differs from the $A_{i-1}$ by inversion of a single alternating cycle. Moreover, it is true that $M(A_{old}) \neq M(A_i)$ for all $1 \leqslant i < c$. If one starts with $A_{new}, P_{\downarrow}, R_{\downarrow}$ then the algorithm will generate $A_{new} \to A_{c-1} \to \ldots \to A_1 \to A_0 = A_{old}$ since $M(A_{old}) = M(A_{new})$. But it is also true that $M(A_{new}) \neq M(A_i)$ for all $1 \leqslant i < c$, so this prevents $\phi(A_{new}, P_{\downarrow}, R_{\downarrow}) = A_i$ for some $1 \leqslant i < c$. Therefore, $A_{old} = \phi(A_{new}, P_{\downarrow}, R_{\downarrow})$. □

*Remark 2*
Note that, this proof would not work if we simply set $R_{\downarrow} = R^{\uparrow}$ since we would end up with a new sequence $A_{new} \to B_1 \to B_2 \to \ldots$ and we might have an $i$ such that $B_i \neq A_j$ for all $j$. Hence it might be possible that $M(B_i) = M(A_{old}) = M(A_{new})$ and therefore $\phi(A_{new}, P_{\downarrow}, R_{\downarrow}) = B_i \neq A_{old}$.

*Lemma 3*
Suppose that $\phi(A_{old}, P^{\uparrow}, R^{\uparrow}) = A_{new}$ and $P_{\downarrow}, R_{\downarrow}$ are determined as above. Then
$p(A_{new}, P^{\uparrow}, R^{\uparrow}|A_{old}) = p(A_{old}, P_{\downarrow}, R_{\downarrow}|A_{new})$

*Remark 3*
$p(A_{new}, P^{\uparrow}, R^{\uparrow}|A_{old})$ refers to the probability that given $A_{old}$ we obtain $A_{new}$ as well as the correct values of $P^{\uparrow}, R^{\uparrow}$. One can think of $P^{\uparrow}, R^{\uparrow}$ of playing a similar role to e.g. latent variables in a hidden Markov model Rabiner & Juang (1986). One can therefore use equations such as e.g. $p(A_{new}|A_{old}) = \sum p(A_{new}, P^{\uparrow}, R^{\uparrow}|A_{old})$ with summation over all $P^{\uparrow}, R^{\uparrow}$.

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

*Proof*
Let us calculate $p(A_{new}, P^{\uparrow}, R^{\uparrow}|A_{old})$.

Since $P^{\uparrow}, R^{\uparrow}$ are independent of each other and do not depend on $A_{new}$, it is true that

$$p(A_{new}, P^{\uparrow}, R^{\uparrow}|A_{old}) = p(P^{\uparrow}|A_{old})p(R^{\uparrow}|A_{old})p(A_{new}|A_{old}, R^{\uparrow}, P^{\uparrow}) \tag{4}$$

There are $N^2$ entries and therefore $(N^2)!$ possible permutations of the matrix $R$, so $p(R^{\uparrow}|A_{old}) = 1/(N^2)!$. The term $p(A_{new}|A_{old}, R^{\uparrow}, P^{\uparrow})$ is 1 if $\phi(A_{old}, R^{\uparrow}, P^{\uparrow}) = A_{new}$ and 0 otherwise. It only remains to calculate $p(P^{\uparrow}|A_{old})$.

For each zero element $A(i, j)$ observe that there are $x_{i+}$ 1s in the $i$th row. If $x_{i+} > 0$, then the probability of $P(i, j)$ pointing to a particular $(i, j')$ is $1/x_{i+}$. If $x_{i+} = 0$, then $P(i, j)$ will point to the structural zero in row $i$ with probability one. Since there are $N - 1 - x_{i+}$ (non-structural) zeros in row $i$ and each entry of $P$ is independent, this contributes a factor of

$$\left(\frac{1}{\max(1, x_{i+})}\right)^{N-1-x_{i+}} \tag{5}$$

Similarly, all the 1s in column $j$ contribute a factor of

$$\left(\frac{1}{\max(N-1-x_{+j}, 1)}\right)^{x_{+j}} \tag{6}$$

Putting everything together yields a probability of $p(A_{new}, P^{\uparrow}, R^{\uparrow}|A_{old}) = f_1 f_2 f_3 f_4$ where

$$f_1 = \frac{1}{(N^2)!} \tag{7}$$

$$f_2 = \prod_i \left(\frac{1}{\max(1, x_{i+})}\right)^{N-1-x_{i+}} \tag{8}$$

$$f_3 = \prod_j \left(\frac{1}{\max(N-1-x_{+j}, 1)}\right)^{x_{+j}} \tag{9}$$

$$f_4 = \mathbf{1}_{\phi(A_{old}, P^{\uparrow}, R^{\uparrow}) = A_{new}} = \mathbf{1}_{\phi(A_{new}, P^{\downarrow}, R^{\downarrow}) = A_{old}} \tag{10}$$

A similar argument yields the same expression for $p(A_{old}, P_{\downarrow}, R_{\downarrow}|A_{new})$. Since $A_{old}$ and $A_{new}$ are compatible, they have the same values of $x_{i+}, x_{+j}$ and therefore $p(A_{new}, P^{\uparrow}, R^{\uparrow}|A_{old}) = p(A_{old}, P_{\downarrow}, R_{\downarrow}|A_{new})$ □

*Theorem 2*
Let $A$ and $B$ be compatible matrices. Then $p(B|A) = p(A|B)$

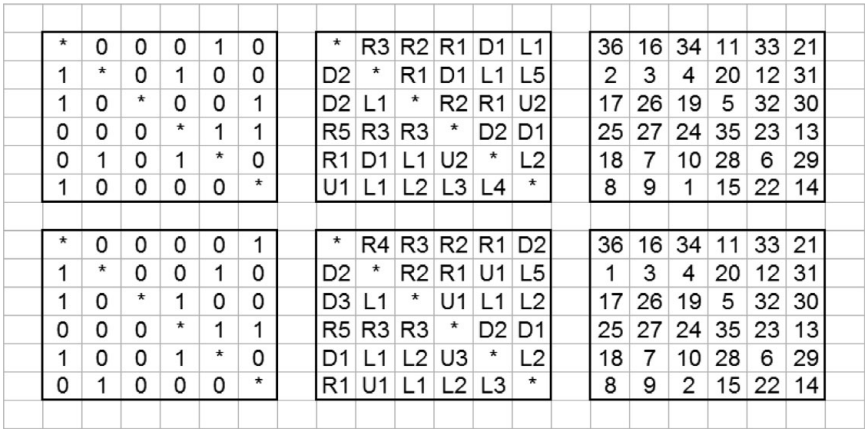
*Proof*
It is true that

$$p(B|A) = \sum_{P,R} p(B, P, R|A) = \sum_{P,R} p(P|A)p(R|A)\mathbf{1}_{\phi(A,P,R)=B} \tag{11}$$

and also

$$p(A|B) = \sum_{P,R} p(A, P, R|B) = \sum_{P,R} p(P|B)p(R|B)\mathbf{1}_{\phi(B,P,R)=A} \tag{12}$$

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press



| | | | | | |
|---|---|---|---|---|---|
| * | 0 | 0 | 0 | 1 | 0 |
| 1 | * | 0 | 1 | 0 | 0 |
| 1 | 0 | * | 0 | 0 | 1 |
| 0 | 0 | 0 | * | 1 | 1 |
| 0 | 1 | 0 | 1 | * | 0 |
| 1 | 0 | 0 | 0 | 0 | * |

| | | | | | |
|---|---|---|---|---|---|
| * | R3 | R2 | R1 | D1 | L1 |
| D2 | * | R1 | D1 | L1 | L5 |
| D2 | L1 | * | R2 | R1 | U2 |
| R5 | R3 | R3 | * | D2 | D1 |
| R1 | D1 | L1 | U2 | * | L2 |
| U1 | L1 | L2 | L3 | L4 | * |

| | | | | | |
|---|---|---|---|---|---|
| 36 | 16 | 34 | 11 | 33 | 21 |
| 2 | 3 | 4 | 20 | 12 | 31 |
| 17 | 26 | 19 | 5 | 32 | 30 |
| 25 | 27 | 24 | 35 | 23 | 13 |
| 18 | 7 | 10 | 28 | 6 | 29 |
| 8 | 9 | 1 | 15 | 22 | 14 |

| | | | | | |
|---|---|---|---|---|---|
| * | 0 | 0 | 0 | 0 | 1 |
| 1 | * | 0 | 0 | 1 | 0 |
| 1 | 0 | * | 1 | 0 | 0 |
| 0 | 0 | 0 | * | 1 | 1 |
| 1 | 0 | 0 | 1 | * | 0 |
| 0 | 1 | 0 | 0 | 0 | * |

| | | | | | |
|---|---|---|---|---|---|
| * | R4 | R3 | R2 | R1 | D2 |
| D2 | * | R2 | R1 | U1 | L5 |
| D3 | L1 | * | U1 | L1 | L2 |
| R5 | R3 | R3 | * | D2 | D1 |
| D1 | L1 | L2 | U3 | * | L2 |
| R1 | U1 | L1 | L2 | L3 | * |

| | | | | | |
|---|---|---|---|---|---|
| 36 | 16 | 34 | 11 | 33 | 21 |
| 1 | 3 | 4 | 20 | 12 | 31 |
| 17 | 26 | 19 | 5 | 32 | 30 |
| 25 | 27 | 24 | 35 | 23 | 13 |
| 18 | 7 | 10 | 28 | 6 | 29 |
| 8 | 9 | 2 | 15 | 22 | 14 |

Fig. 6. Example adjacency, pointer, and starting seed matrices.

There are two cases to consider: $A = B$ and $A \neq B$. In either case Lemmas 2 and 3 imply that each $P, R$ satisfying $\phi(A, P, R) = B$ can be put in 1-1 correspondence with each $P', R'$ satisfying $\phi(B, P', R') = A$ and also $p(P|A) = p(P'|B), p(R|A) = p(R'|B)$. Therefore

$$p(B|A) = p(A|B) \tag{13}$$

as claimed □

Therefore, we have proved detailed balance, i.e. the proposed algorithm does obtain samples from $U|x_{i+}, x_{+j}, M$.

An example is shown in Figure 6. Let $A$ and $B$ be adjacency matrices (top/bottom resp. of Figure 6, left) and let the corresponding pointer matrices $P_A, P_B$ and starting seed matrices $R_A, R_B$ be shown. The notation R4 means move four spaces to the right, and L U D similarly mean left up or down.

For matrix $A$, the start-seed matrix has a 1 at element (6,3), so one would start at $A(6, 3)$, follow the pointers and find a 4-cycle at $(6, 1) \to (5, 1) \to (5, 2) \to (6, 2) \to (6, 1)$. A swap will increase the number of mutual dyads $M$ by 1, so look for another cycle. The next element is $A(2, 1)$ since $R_A(2, 1) = 2$. Following the pointers results in a 6-cycle at $(3, 6) \to (1, 6) \to (1, 5) \to (2, 5) \to (2, 4) \to (3, 4) \to (3, 6)$. It turns out that swapping both the above 4-cycle and 6-cycle results in no change in $M$, so the swap is accepted, and the new matrix is $B$. Similarly, one can start with $B, P_B, R_B$ and obtain $A$. One can easily verify that $A, R_A, P_A$ and $B, P_B, R_B$ form a 1-1 correspondence as described above, and that

$$p(B, P_A, R_A|A) = p(A, P_B, R_B|B) = \frac{1}{36!} \times 1^8 \times \left(\frac{1}{2}\right)^{15} \times \left(\frac{1}{3}\right)^6 \times \left(\frac{1}{4}\right)^1 \tag{14}$$

*Remark 4*

It is worth noting that the proof does not require a closed formula for $p(A|B)$ or $p(B|A)$ - it is only necessary to show they are equal. This differs from the algorithms of Rao *et al.* (1996) or McDonald *et al.* (2007), where the updates were trivial. If for instance, one only allowed tetrad swaps in Problem URC, then the probability of any change is $p(B|A) = (2/n(n-1))^2$ if $B$ is compatible with $A$. If tetrads intersecting the main diagonal were not allowed, or if hexads were allowed, then it is easy to

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

modify the formula for $p(B|A)$ accordingly. With the proposed algorithm, it is easy to calculate $P(B, P, R|A)$ for any given $P, R$, but there are many different values of $P, R$ that would cause $A$ to be updated to $B$, and it is not feasible to count them. This explains why the computation of $p(B|A)$ is infeasible.

*Remark 5*
The algorithm for selecting random alternating cycles is fairly complex, but it can be optimized somewhat. In practice, one would not construct the entire pointer and starting-seed matrices explicitly. This would be extremely inefficient if, for example, the first starting seed in a $100 \times 100$ matrix happened to yield a simple alternating rectangle with the correct value of $M$. Instead, one would simply compute each element "on demand". Also, if one updates alternating cycles, one can also update $M$ incrementally instead of having to recompute it from scratch. Again this is more efficient for a large e.g. $100 \times 100$ matrix.

### *4.3 Discussion*

Recall that McDonald et al.'s algorithm requires $\beta$ as a parameter. If $\beta$ is too low, a large proportion of matrices will have the wrong value of $M$. If $\beta$ is too high, then the algorithm may get trapped in an isolated component of all possible graphs. The proposed algorithm actually avoids this trade-off. To see this, observe that this algorithm would temporarily allow matrices with the wrong value of $M$, so there is no danger of getting stuck in an isolated component. But if one keeps inverting random alternating cycles, eventually they must repeat because the matrix is finite. By construction, once we repeat a cycle, the new matrix $A'$ is immediately rejected, so $A_{new} = A$ which is known to have the correct $M$. This avoids the problem where the algorithm is "lost at sea" spending a large amount of time exploring random adjacency matrices with the number of mutual dyads very far from the correct value. Hence we avoid the above trade-off as claimed.

## 5 Experiments

### *5.1 Synthetic data*

McDonald et al. propose to measure the rejection rate (i.e. how many matrices must be discarded from the MH algorithm) for problem URMC0. The problem is this doesn't tell us if one has found the whole set of admissible matrices or whether the algorithm is stuck in a isolated component. Instead, we wish to count the number of different matrices directly, (i.e. the number of matrices that occur at least once). If for example, the number of different matrices is increasing linearly, then there is a strong indication the algorithm is not repeating an isolated component and there are several matrices that haven't been found yet. If the number of different matrices starts to flatline then there is a good indication the algorithm is repeating a component or found all the matrices.

Counting the number of different matrices obtained in any MCMC algorithm is not trivial. A brute force search requires $\mathcal{O}(n^2)$ comparisons, where $n$ is the length of the sequence. This is awkward if $A$ is large. Instead we use a hash trick. Let $R$ be a random matrix of the same size as $A$. We then pretend $R$ and $A$ are vectors and

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

compute a dot product, thus

$$h_R(A) = \sum_{i,j} A(i,j)R(i,j) = tr(R'A) = tr(A'R) \tag{15}$$

where $tr$ is the trace of a matrix. Assuming no hash collisions, then $h_R(A) = h_R(B)$ if and only if $A = B$. The hash converts every matrix into a real number. It is relatively simple to count the number of different real numbers obtained, e.g. by sorting the numbers and discarding repeats.

Intuitively, hash collisions shouldn't be a serious problem, but can we rigorously prove this? One might be tempted to use a power sequence for $R$, i.e. let $R$ be a permutation of $1, 2, 4, 8, \ldots, 2^{N^2}$ where $N \times N$ is the size of the matrix $A$ or $B$. Then it is easy to show that $h_R(A) = h_R(B)$ if and only if $A = B$, since $A, B$ are binary matrices. Unfortunately this only works theoretically, and one may encounter numerical problems such as underflow/overflow for large $N$. Thus it is necessary to estimate the probability of hash collisions.

Let us assume that $R$ is a random matrix with entries between 0 and 1. Let us "generously" assume two hash values are equal if the fractional part is equal (i.e. the difference is a whole number). With the above assumption it is reasonable to treat $h_R(A)$ as a uniformly random number between 0 and 1. If one has $n$ distinct matrices and $k$ possible hash values, then one can show that the expected number of collisions $C$ is $E[C] = n - k + k(1 - 1/k)^n$, as shown in the Appendix.

If $k$ is large relative to $n$, then

$$\begin{aligned} E[C] &= n - k + k\left(1 - \frac{1}{k}\right)^n \\ &= n - k + k\left(1 - \binom{n}{1}\frac{1}{k} + \binom{n}{2}\frac{1}{k^2} - \cdots + \binom{n}{n}\frac{(-1)^n}{k^n}\right) \\ &= \binom{n}{2}\frac{1}{k} - \binom{n}{3}\frac{1}{k^2} + \binom{n}{4}\frac{1}{k^3} - \cdots \approx \frac{n^2}{2k} \end{aligned}$$

Assume the Markov chain stops after $n$ iterations. In practice, this will yield an upper bound for the number of hash collisions for two reasons: (i) we assumed $h(A) = h(B)$ if the difference is a whole number, not necessarily 0, (ii) the $n$ iterations are not necessarily distinct matrices.

A typical value of "machine epsilon" is $\epsilon = \mathcal{O}(10^{-16})$, so we will take $k = 10^{16}$. Therefore, hash collisions should not really be an issue for reasonable $n$. In the following experiments, we have $n < 10^8$ which would imply we should expect less than one collision on average.

McDonald et al.'s algorithm is simple to implement in the sense that swapping a tetrad is trivial, whereas the proposed algorithm is much more complex. In other words, each iteration of McDonald et al.'s algorithm is cheaper than the proposed algorithm in terms of time. Thus we decided to compute various quantities of interest as a function of time, not number of iterations.

In Section 2, we remarked that it can be useful to estimate the distribution of some function of states (i.e. adjacency matrix) in the Markov chain. In this case, we take each adjacency matrix $A$ in the Markov chain and measure the transitivity (Wasserman, 1977) $\tau$ by counting the number of ordered triples $p, q, r$ satisfying $p \to q, q \to r, p \to r$. Note that, the relationships $q \to p, r \to q, r \to p$

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

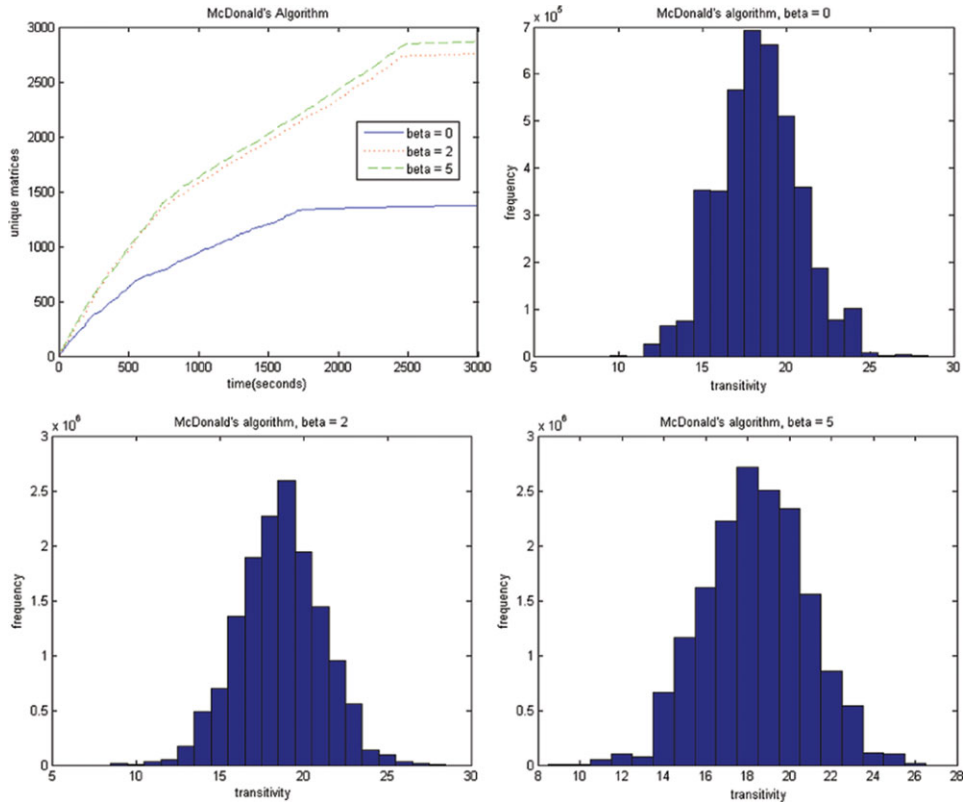



Fig. 7. McDonald et al.'s algorithm with $G = G(10, 0.3)$. (Color online)

are irrelevant and we also do not subtract 1 for any intransitive triples satisfying $p \to q, q \to r, \neg(p \to r)$.

Figure 7 shows the result of McDonald et al.'s algorithm and the proposed algorithm on a randomly generated matrix $A$ of size 10 and density 0.3, with both algorithms running for 3,000 seconds or 50 minutes, using Matlab version 7.5 on an Intel i7-2600 desktop machine with 16 Gb RAM, running Windows 7. (In Figure 7, the notation $G = G(10, 0.3)$ means that $G$ is a given $10 \times 10$ adjacency matrix with 30% of entries equal to 1, and $A$ must be compatible with $G$). For $\beta = 0, 2, 5$ the number of iterations was 1.6e+07, 1.3e+07, 1.2e+07 respectively. The left panel shows that McDonald et al.'s algorithm only achieves 2,800 unique matrices (even with the best possible parameter $\beta = 5$). The other panels show the distribution of transitive triads which look "erratic". In contrast, Figure 8 shows the proposed algorithm has $1.4 \times 10^5$ unique matrices, and the distribution of transitive triads is much smoother. The proposed algorithm only achieved 7.5e+06 iterations, but that is a small price to pay for the huge improvement over McDonald et al.'s algorithm. This suggests that the proposed algorithm has made much more progress in estimating the true distribution of $\tau$ than McDonald et al.'s algorithm.

Similar results were obtained for matrices of the form $G = G(20, 0.3)$ and $G = G(30, 0.3)$. Figures 9 and 10 show the result for $G = G(20, 0.3)$ again with both algorithms running for 50 minutes. McDonald et al.'s algorithm obtains between 1,100 and 4,500 unique matrices, compared to $1.5 \times 10^5$ unique matrices for the

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

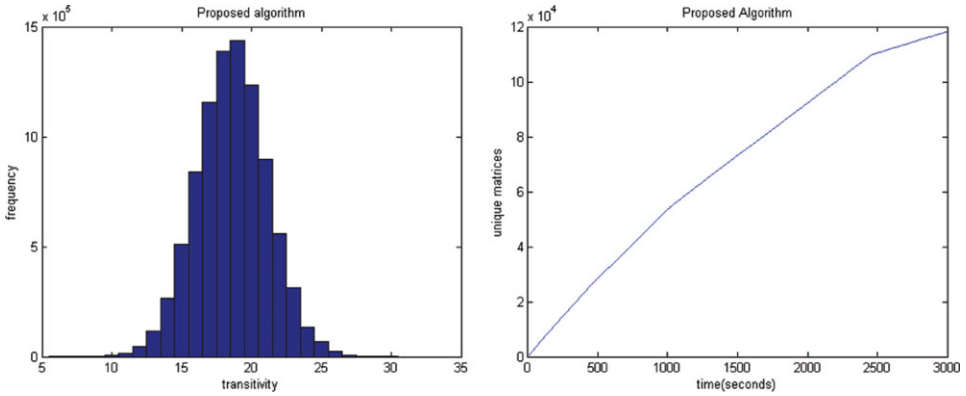



Fig. 8. Proposed algorithm with $G = G(10, 0.3)$. (Color online)

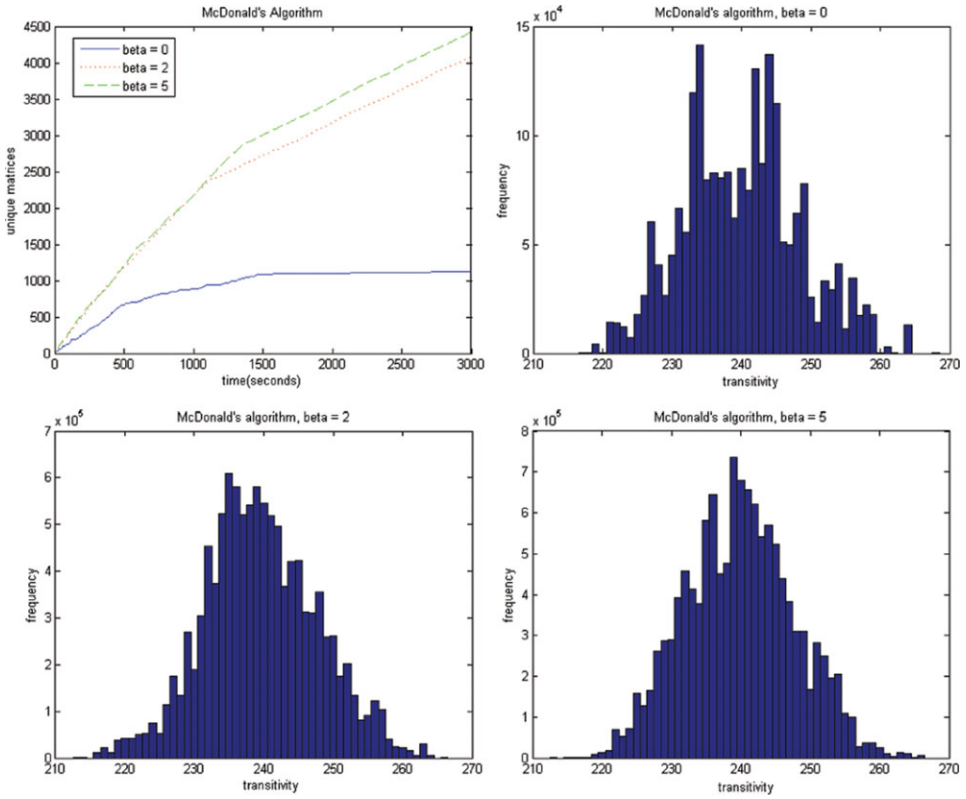



Fig. 9. McDonald et al.'s algorithm with $G = G(20, 0.3)$. (Color online)

proposed algorithm, and the distribution of the triad count is again much smoother for the proposed algorithm than McDonald et al's algorithm. The same result is obtained at $G = G(30, 0.3)$ as shown in Figures 11 and 12. McDonald et al.'s algorithm has less than 8,000 unique matrices compared to $1.5 \times 10^5$ for the proposed algorithm.

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

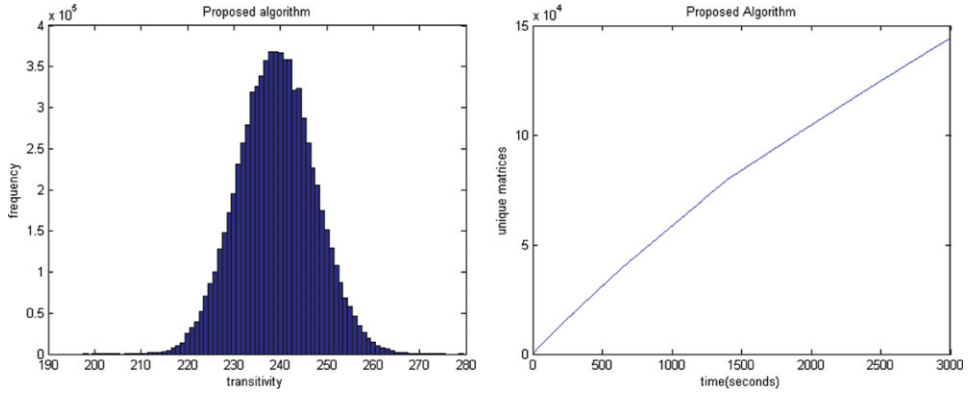



Fig. 10. Proposed algorithm with $G = G(20, 0.3)$. (Color online)

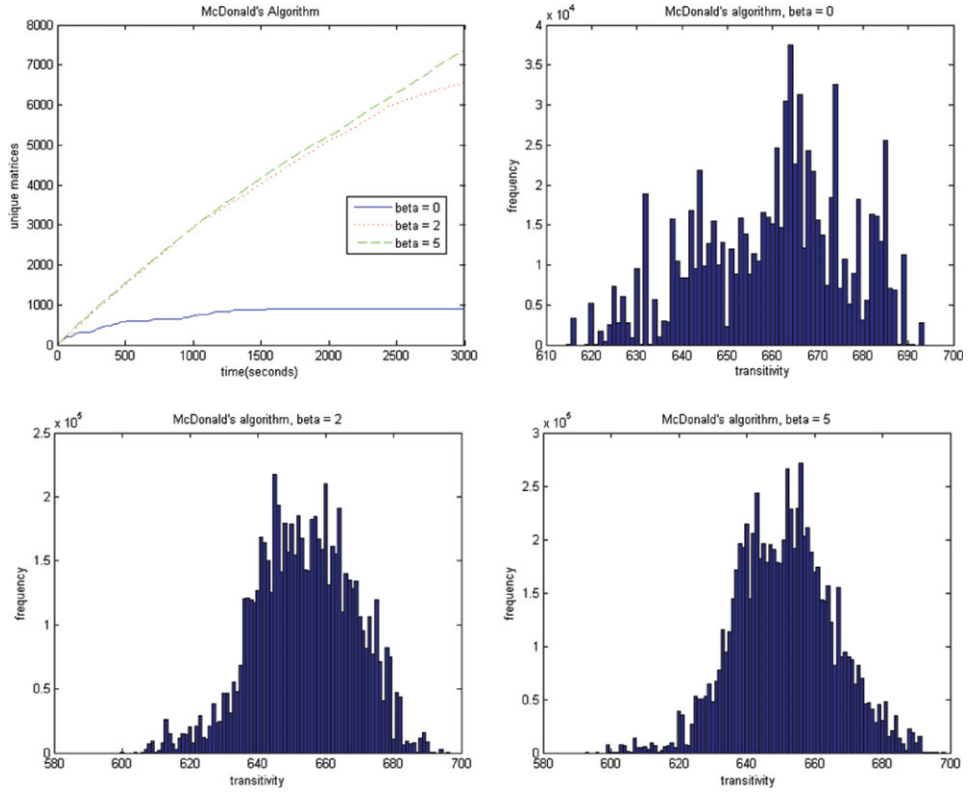



Fig. 11. McDonald et al.'s algorithm with $G = G(30, 0.3)$. (Color online)

### *5.2 Classroom experiment—Triad census*

The triad census (Wasserman, 1977) is a set of summary statistics describing the local structure in a network. For any triad of three nodes, there are six possible relationships between ordered pairs of distinct nodes. If one allows each such relationship to be "on" or "off", then there are 16 possible triads, up to isomorphism (Wasserman, 1977). The triad census is therefore a 16-element vector and is a function of an adjacency matrix (or corresponding network). Hence it carries much

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

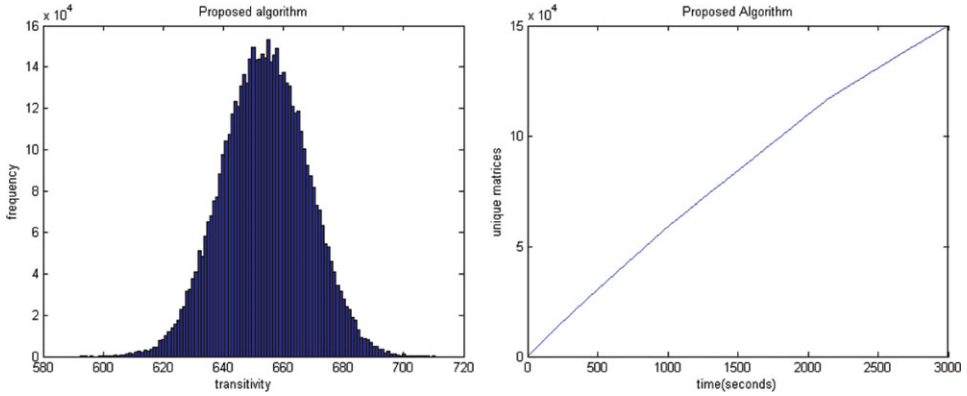



Fig. 12. Proposed algorithm with $G = G(30, 0.3)$. (Color online)

more information than the transitivity $\tau$ in the previous section. If each element of the vector is treated separately, one can do similar calculations to the transitivity in the previous experiment, such as estimating the distribution or calculating moments etc.

McKinney (1948) analyzed the preferences of 29 ninth-grade classroom students regarding their acceptance/rejection preferences of each other and presented the adjacency matrix for a subset of 10 students. Wasserman (1977) and McDonald *et al.* (2007) attempted to estimate the expected value of the triad census under certain conditions. More specifically, matrices compatible with $A$ are randomly drawn from the uniform distribution, and the expected values of the triad census can be estimated as described in Section 2. The adjacency matrix $A$ is given below:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

The first column of Table 2 represents the standard names given to all possible triads using the MAN labeling (Wasserman & Faust, 1994) such as 003 for the empty triad, 012 for a single asymmetric arc, and so on. For each triad type, the next three columns are the observed number of triads in $A$, expected number of triads under $U|MAN, B_{in}, B_{out}$ and expected number of triads under $U|X_{i+}, B_{in}, m$ respectively. These results are due to Wasserman (1977) and copied by McDonald *et al.* (2007). The fifth column is the expected number of triads under $U|M, X_{i+}, X_{+j}$ (i.e. the model of interest to us) due to McDonald *et al.* (2007). The last column is the expected number of trials using the proposed algorithm on $U|M, X_{i+}, X_{+j}$.

Here $MAN$ refers to the number of mutual, asymmetric and null dyads in an adjacency matrix, and $B_{in}, B_{out}$ refer to the number of 2-instars and 2-outstars in

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

Table 2. *Triad census and expected values for relationships in ninth-grade classroom.*

| Triad | Observed | $U\|MAN, B_{in}, B_{out}$ | $U\|X_{i+}, B_{in}, m$ | $U\|M, X_{i+}, X_{+j}$ | Proposed |
|---|---|---|---|---|---|
| 003 | 8 | 7.00 | 5.23 | 7.31 | 7.25 |
| 012 | 19 | 20.50 | 19.12 | 19.97 | 20.31 |
| 102 | 7 | 8.77 | 10.23 | 7.65 | 7.50 |
| 021D | 5(?) | 4.91(?) | 5.83(?) | 4.40(?) | 4.27 |
| 021U | 5(?) | 7.95(?) | 5.71(?) | 8.19(?) | 8.05 |
| 021C | 11 | 7.07 | 9.20 | 7.90 | 7.78 |
| 111D | 18(?) | 14.94(?) | 13.07(?) | 14.87(?) | 14.88 |
| 111U | 7(?) | 7.07(?) | 12.70(?) | 8.53(?) | 8.44 |
| 030T | 4 | 6.62 | 5.29 | 6.74 | 6.96 |
| 030C | 0 | 0.28 | 0.75 | 0.44 | 0.47 |
| 201 | 6 | 6.52 | 7.08 | 6.94 | 7.19 |
| 120D | 9(?) | 5.79(?) | 4.00(?) | 6.18(?) | 6.12 |
| 120U | 7(?) | 3.70(?) | 4.26(?) | 3.31(?) | 3.33 |
| 120C | 4 | 5.51 | 6.94 | 4.60 | 4.53 |
| 210 | 6 | 10.92 | 9.08 | 9.89 | 9.85 |
| 300 | 4 | 2.44(?) | 1.49 | 3.06 | 3.03 |

the graph. Thus $U|MAN, B_{in}, B_{out}$ means the uniform distribution over all graphs with $M$ mutual dyads, $A$ asymmetric dyads, $N$ null dyads, $B_{in}$ 2-instars, and $B_{out}$ 2-outstars, for given values of $M, A, N, B_{in}, B_{out}$.

There are a number of discrepancies in Table 2 indicated by a question mark(?). Experimentation on the matrix $A$ suggests that Wasserman has inadvertently swapped row 021D with 021U, and similarly for 111D/111U and 120D/120U. Also, McDonald et al. had the incorrect value of 1.41 instead of 2.44 in the last row, which was obviously copied from the wrong column of Wasserman's table. If these suggestions are valid, then Table 2 is correct, including all entries indicated with a question mark(?).

McDonald et al. compares the columns labeled $U|MAN, B_{in}, B_{out}$, $U|X_{i+}, B_{in}, m$ and $U|M, X_{i+}, X_{+j}$. McDonald et al. correctly claim that of the first two the former is closer to the column labeled $U|M, X_{i+}, X_{+j}$ than the latter. This can be verified by inspection; or if one wanted to verify it numerically, then one can calculate:

$$\sum_{i=1}^{16} [|b_i - d_i|, |c_i - d_i|] = [9.11, 26.44]$$

$$\sum_{i=1}^{16} [(b_i - d_i)^2, (c_i - d_i)^2] = [7.5685, 58.6892]$$

where each $1 \leqslant i \leqslant 16$ corresponds to a triad type and $b, c, d$ represent the columns labeled $U|MAN, B_{in}, B_{out}$, $U|X_{i+}, B_{in}, m$, $U|M, X_{i+}, X_{+j}$.

The entries for the proposed algorithm are very close to those obtained by McDonald et al's algorithm. This suggests the set of matrices compatible with $A$ is "well-behaved" e.g. there are no problems associated with isolated components of the Markov chain, hence there is no reason to expect the output of the proposed algorithm to differ from that of McDonald et al.

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

One can also compare all the columns of Table 2 and verify that

$$\sum_{i=1}^{16}[|b_i - a_i|, |c_i - a_i|, |d_i - a_i|, |e_i - a_i|] = [32.29, 39.48, 29.92, 30.64]$$

$$\sum_{i=1}^{16}[(b_i - a_i)^2, (c_i - a_i)^2, (d_i - a_i)^2, (e_i - a_i)^2] = [97.2343, 139.6788, 80.6524, 82.7466]$$

where $a, b, c, d, e$ correspond to columns 2–6 in Table 2. We see that the proposed algorithm and Mcdonald's algorithm produce similar deviations from the triad census of $A$. The other algorithms (corresponding to columns 3 and 4) have larger deviations.

## 6 Conclusion

The problem of generating uniformly random graphs (or adjacency matrices) from a constrained distribution is important and has many applications, including the world wide web, social, and biological networks. The MCMC algorithm is a popular approach to this problem. One of the main difficulties with existing MCMC algorithms such as McDonald *et al.* (2007) is that one cannot guarantee the Markov chain is irreducible, and hence one cannot guarantee a reasonable coverage of all possible networks of interest. McDonald et al. conjectured that if certain elementary operations were allowed such as swapping tetrads hexads or alternating tetrads, then the chain becomes irreducible. We showed that this conjecture is false.

We have proposed a new improved MCMC algorithm for the problem of generating random graphs (or adjacency matrices) with fixed in-degree, out-degree, and mutual dyads. The basic idea is to allow global instead of local changes that allow many entries of the adjacency matrix to be inverted in a single iteration. This trivially guarantees the Markov chain is irreducible. We showed there is a fast algorithm for obtaining these global changes, and proved that the uniform distribution is attained. Although each individual MCMC iteration of the proposed algorithm costs more time than existing algorithms, we obtain better coverage of the search space and smoother distributions regarding the transitivity (number of triples satisfying $p \to q, q \to r, p \to r$) if both algorithms are run for the same amount of time. The proposed algorithm produces a similar tetrad census to McDonald et al.'s algorithm for ninth-grade classroom data due to McKinney (1948).

Future work may include adding further constraints such as the number of in-stars or out-stars in the graph, or indeed any quantity in the triad census. This would yield more realistic models for certain applications such as social network analysis, although it remains an open question if the resulting algorithms will be feasible in practice. Also, it is worth investigating if the algorithm is scalable for larger size graphs, i.e. does computation time becomes an issue for larger networks (e.g. 300 or 1,000 nodes) and if so, whether the algorithm can be optimized to mitigate these difficulties.

## Acknowledgements

The author would like to thank the editor and anonymous reviewers for helpful suggestions.

## References

Blitzstein, J., & Diaconis, P. (2010). A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics*, **6**(4), 489–522.

Chung, F., & Lu, L. (2002). Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, **6**(2), 125–145.

Erdös, P., & Rényi, A. (1959). On random graphs. *Publicationes Mathematicae*, **6**, 290–297.

Erdös, P. L., Miklós, I., & Soukup, L. (2013). Towards random uniform sampling of bipartite graphs with given degree sequence. *Electronic Journal of Combinatorics*, **20**(1).

Holland, P., & Leinhardt, S. (1981). An exponential family of probability distributions for directed graphs. *Journal of the American Statistical Association*, **76**(373), 33–50.

Lusher, D., Koskinen, J., & Robins, G. (eds). (2012). *Exponential random graph models for social networks, theory, methods, and applications*. New York, NY, USA: Cambridge University Press.

McDonald, J., Smith, P., & Forster, J. (2007). Markov chain Monte Carlo exact inference for social networks. *Social Networks*, **29**(1), 127–136.

McKinney, J. C. (1948). An educational application of a two-dimensional sociometric test. *Sociometry*, **11**(4), 358–367.

Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Review*, **45**(2), 167–256.

Rabiner, L. R., & Juang, B. H. (1986). An introduction to hidden Markov models. *IEEE ASSP Magazine*, **3**(1), 4–16.

Rao, A., Jana, R., & Bandyopadhyay, S. (1996). A Markov chain Monte Carlo method for generating random 0-1 matrices with given marginals. *Sankhya*, **Series A, 58**(2), 225–242.

Roberts, J. (2000). Simple methods for simulating sociomatrices with given marginal totals. *Social Networks*, **22**(3), 273–283.

Rubinstein, R. Y., Ridder, A., & Vaisman, R. (2013). *Introduction to Monte Carlo methods* (pp. 1–5). Hoboken, NJ, USA: John Wiley and Sons, Inc.

Snijders, T. (2011). Statistical models for social networks. *Annual Review of Sociology*, **37**, 131–53.

Wasserman, S., & Faust, K. (1994). *Social network analysis: methods and applications*. Cambridge, UK: Cambridge University Press.

Wasserman, S. S. (1977). Random directed graph distributions and the triad census in social networks. *Journal of Mathematical Sociology*, **5**(1), 61–86.

## A expected number of collisions

I wish to prove that if a hash function $h(\cdot)$ maps $n$ objects onto $k$ "slots", and the distribution is i.i.d. uniform for each $h(z)$ then the expected number of collisions $E[C]$ is given by

$$E[C] = n - k + k\left(1 - \frac{1}{k}\right)^n \tag{A 1}$$

*Proof*

I start by considering the probability that a given slot is empty. Let $\phi_i$ represent the event that slot $i$ is empty for $1 \leqslant i \leqslant k$. Also let $\mathbf{1}_{\phi_i}$ denote the indicator function, equal to 1 if $\phi_i$ occurs and 0 otherwise. The probability that state $i$ is empty is given by

$$E[\mathbf{1}_{\phi_i}] = \left(\frac{k-1}{k}\right)^n = \left(1 - \frac{1}{k}\right)^n \tag{A 2}$$

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press

for any $1 \leqslant i \leqslant k$ since the distribution of each hash $h(z)$ is i.i.d. uniform for $1 \leqslant z \leqslant n$. The expected number of empty slots is therefore given by

$$E\left[\sum_{i=1}^{k} \mathbf{1}_{\phi_i}\right] = \sum_{i=1}^{k} E[\mathbf{1}_{\phi_i}] \tag{A 3}$$

$$= \sum_{i=1}^{k} \left(1 - \frac{1}{k}\right)^n \tag{A 4}$$

$$= k\left(1 - \frac{1}{k}\right)^n \tag{A 5}$$

If there are $m$ empty slots, there must be $k - m$ non-empty slots and hence $n - (k - m)$ collisions. Therefore, the expected number of collisions is

$$E[C] = n - k + k\left(1 - \frac{1}{k}\right)^n \tag{A 6}$$

as required □

https://doi.org/10.1017/nws.2015.35 Published online by Cambridge University Press