Electronic Journal of Statistics
Vol. 14 (2020) 1690–1706
ISSN: 1935-7524
https://doi.org/10.1214/20-EJS1702

A fast MCMC algorithm for the uniform sampling of binary matrices with fixed margins

Guanyang Wang

Department of Mathematics
Stanford University
CA, 94305
e-mail: guanyang@stanford.edu

Abstract: Uniform sampling of binary matrix with fixed margins is an important and difficult problem in statistics, computer science, ecology and so on. The well-known swap algorithm would be inefficient when the size of the matrix becomes large or when the matrix is too sparse/dense. Here we propose the Rectangle Loop algorithm, a Markov chain Monte Carlo algorithm to sample binary matrices with fixed margins uniformly. Theoretically the Rectangle Loop algorithm is better than the swap algorithm in Peskun's order. Empirically studies also demonstrates the Rectangle Loop algorithm is remarkablely more efficient than the swap algorithm.

MSC 2010 subject classifications: Primary 62D05; secondary 62P25.
Keywords and phrases: MCMC, uniform sampling, binary matrix, fixed margins.

Received September 2019.

Contents

Table 1
Occurrence Matrix occurrence matrix of the finches on the Galapagos islands.

	Island																
Finch	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0
3	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0
4	0	0	1	1	1	0	0	1	0	1	0	1	1	0	1	1	1
5	1	1	1	0	1	1	1	1	1	1	0	1	0	1	1	0	0
6	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
7	0	0	1	1	1	1	1	1	1	0	0	1	0	1	1	0	0
8	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	1	1	1	1	1	1	1	1	0	1	0	0	1	0	0
10	0	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0
11	0	0	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0
12	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1. Introduction

The problem of sampling binary matrices with fixed row and column sums has attracted much attention in numerical ecology. In ecological studies, the binary matrix is called *occurrence matrix*. Rows usually correspond to species, the columns, to locations. For example, the binary matrix shown on Table 1 is known as "Darwin's Finch" dataset, which comes from Darwin's studies of the finches on the Galapagos islands (an archipelago in the East Pacific). The matrix represents the presence/absence of 13 species of finches in 17 islands. A "1" or "0" in entry (i, j) indicates the presence or absence of species i at island j. It is clear from Table 1 that some pairs of species tend to occur together (for example, species 9 and 10) while some other pairs tend to be disjoint. Therefore, it is of our interest to investigate whether the cooperation/competition influences the distribution of species on islands, or the patterns found are just by chance.

Assuming different species have independent distributions on islands, then the observed binary matrix is simply a random sample from the uniform distribution of all the binary matrices with fixed margins. Table 2 gives an example of all configurations of 3×3 binary matrices with $[1, 2, 1]$ as both row and column sums. Ideally, if we could list all the binary matrices with arbitrary size, then we could compare the pattern found in the observed matrix with others, to conclude whether the observed matrix is simply by chance. However, enumerating matrices with fixed margins is often impractical both theoretically and computationally for larger matrices. Therefore, sampling such random matrices becomes the natural choice.

The problem of sampling such matrices also occurs in many other fields, with different names. For example, an equivalent formulation is uniformly sampling undirected bipartite graphs with given vertex degrees. A bipartite graph $G = (U, V, E)$ is a graph whose vertices are divided into two disjointed sets, denoted by $U = \{u_1, \cdots, u_m\}$, $V = \{v_1, \cdots, v_n\}$. E is called the edge set where every edge connects one vertex in U to one in V. The binary matrix $M = (m_{i,j})_{m\times n}$

Table 2
All possible 3×3 binary matrices with $[1, 2, 1]$ as both row and column sums

A	B	C	D	E
$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

is often called the bi-adjacency matrix of G and is defined by

$$m_{i,j} = \begin{cases} 1, \text{if there is an edge connecting } u_i \text{ and } v_j, \\ 0, \text{otherwise.} \end{cases}$$

Bipartite graphs are often used in network studies to model the interaction between two objects, for example, customers and products. It is often required to sample graphs with preserved degree sequence in network analysis uniformly. Throughout this paper, we will use 'binary matrix' instead of 'bipartite graph' to avoid confusion, although they are equivalent.

The algorithms of sampling binary matrices with fixed margins are divided into two classes. The first class of algorithms relies on the *rejection sampling* or *importance sampling* techniques, see [16], [10], [3], [7] [8] for examples. Importance sampling usually generates non-uniform distribution, but it can be used to construct estimators to estimate the quantities of interest, such as the number of binary matrices with given margins. Chen et al. [3] introduced a sequential importance sampling (SIS) scheme to test the hypothesis we mentioned at the beginning of this paper on the "Darwin's Finch" dataset.

The second class falls into the *Markov Chain Monte Carlo* (MCMC) category and will be our main focus in this paper. The well-known "swap algorithm" has been used for decades. To the author's best knowledge, it is first introduced by Besag and Clifford [1] in 1989 to solve a statistical testing problem. The swap algorithm has been formally proposed and analyzed by Rao et al. [12] and [9] in the 1990s. A similar question is to sample matrices with nonnegative integer entries, fixed row and column sums. Diaconis and Gangolli have proposed a random walk Metropolis algorithm [4]. Many variations and extensions of this algorithm are described in Diaconis and Sturmfels [5].

The swap method attempts to make a *single* swap in each iteration, but when the matrix is large or is mostly filled (or unfilled), the efficiency of the swap algorithm can be relatively low. In 2008, Verhelst [18] proposed a new MCMC algorithm based on the idea of performing multiple swaps per iteration. In 2014, Strona et al. [17] introduced the "Curveball algorithm", which uses a 'fair trade' operation to replace the 'swap' operation in the swap algorithm, aiming for a faster mixing. The mathematical formulation of the Curveball algorithm is equivalent to Verhelst's algorithm, but with different implementation and reasoning. A nice survey and numerical comparisons of the existing algorithms can be found in a recent dissertation [13]. The class of 'multiple swaps' algorithms tends to improve the mixing time empirically. However, each step of the 'multiple swaps' algorithm is slower than the classical swap algorithm. Meanwhile, it is hard to compare the 'multiple swaps' algorithms and classical swap algorithm theoretically, as the corresponding Markov-chains have complicated behaviors and are therefore hard to analyze mathematically. The only existing result can be found in [2].

In this paper, we introduce a novel algorithm called Rectangle Loop algorithm. The algorithm is based on the classical swap algorithm, with a careful utilization of the matrix structure given by margins. We have also proved the resulting Markov Chain dominates the classical chain used in the swap algorithm in the sense of Peskun's partial ordering [11] and is easy to implement. Section 2 gives a review of the swap algorithm and Curveball algorithm, including the details of both algorithms and a discussion. In Section 3 we introduce our new algorithm – Rectangle Loop algorithm. Section 4 proves the theoretical properties of Rectangle Loop algorithm. Section 5 gives numerical results.

2. Existing methods

2.1. Swap algorithm

The swap algorithm, or equivalently, swap chain is based on the idea of *swapping checkerboard units*. Here a checkerboard unit is a two by two matrix with one of the following forms:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

A swap means changing one checkerboard unit to the other.

Starting from an initial matrix, one chooses two rows and two columns uniformly at random among all rows and columns. If the resulting 2×2 submatrix with entries in the intersection of these rows and columns is a checkerboard unit, it is swapped, otherwise, do nothing.

Algorithm 1 Swap Algorithm

```
Input: initial binary matrix $A_0$, number of iterations $T$
1: for $t = 1, \cdots T$ do
2:     Choose two distinct rows and two distinct columns uniformly at random
3:     If the corresponding $2 \times 2$ submatrix of $A_{t-1}$ is a checkerboard unit, i.e.
           $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ or $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$,
       swap the submatrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ *to* $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ or vice versa.
       Otherwise $A_t \leftarrow A_{t-1}$
4: end for
```

The swap algorithm is a Metropolis-type Markov chain Monte Carlo which converges to the uniform distribution.

2.2. Curveball algorithm

The swap algorithm can often be inefficient, taking Darwin's Finch data as an example, there are $\binom{13}{2}\binom{17}{2} = 10608$ submatrices with size 2×2, however, only about 3% of them are swappable. This means it requires a very large T (the number of iterations) to ensure the generated matrices are close to be uniformly distributed. The Curveball algorithm provides another solution.

Algorithm 2 Curveball Algorithm

```
Input: initial binary matrix $A_0$, number of iterations $T$
1: for $t = 0, \cdots T - 1$ do
2:     Choose two distinct rows $r_a, r_b$ uniformly at random
3:     Determine two disjoint sets
           $$S_{a-b} \doteq \{k : A_t(a,k) = 1, A_t(b,k) = 0\}$$
           $$S_{b-a} \doteq \{l : A_t(a,l) = 0, A_t(b,l) = 1\},$$
   where $A_t(i,j)$ is the $(i,j)$-th entry of matrix $A_t$. Here we assume $|S_{i-j}| \leq |S_{j-i}|$
4:     Choose a subset $V \subset S_{j-i}$ uniformly at random
5:     Set $A_{t+1} = A_t$ except for row $a, b$.
        For row $a$:
           $$A_{t+1}(a,l) = \begin{cases} 1 & \text{if } l \in V \\ 0 & \text{if } l \in S_{a-b} \cup S_{b-a} \setminus V \\ A(a,l) & \text{otherwise} \end{cases}$$
        For row $b$
           $$A_{t+1}(b,k) = \begin{cases} 1 & \text{if } k \in S_{a-b} \cup S_{b-a} \setminus V \\ 0 & \text{if } k \in V \\ A(b,k) & \text{otherwise} \end{cases}$$
6: end for
```

The Curveball algorithm uses 'trade' instead of 'swap' operation in each iteration. Steps 3-5 in Algorithm 2 gives an illustration of trading, it trades elements in column V with elements in column S_{a-b} for row a and row b, preserving their row and column sums. Though seemingly complicated, there is a very intuitive explanation of the Curveball algorithm. We refer the readers to [17] for detailed illustrations.

3. Rectangle loop algorithm

The swap algorithm is proven to converge to the uniform distribution. However, getting stuck at the same configuration is inefficient and thus the convergence could be very slow. For example, numerical experiments suggest that one would expect more than 30 iterations before each successful swap using 'Darwin's Finch' dataset. Assuming the randomly chosen row is mostly filled, such as row 1 in Table 1, the two randomly chosen entries in this row would most likely be $[1,1]$ but the row of a 'checkerboard unit' has to be either $[1,0]$ or $[0,1]$. Therefore swapping rarely happens when the chosen row/column is mostly filled (or equivalently, mostly unfilled).

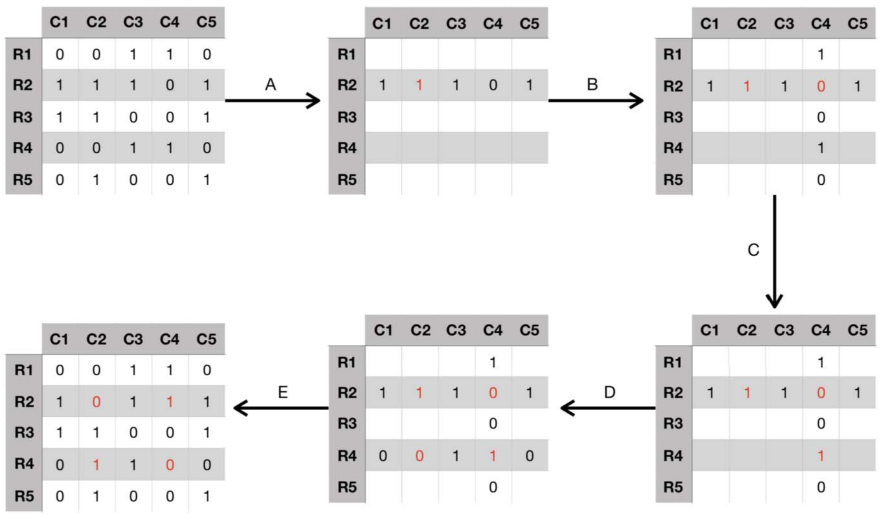

FIG 1. *An illustration of Rectangle Loop algorithm.*

The Rectangle Loop algorithm is designed to increase the chance of swapping. The idea is illustrated in Figure 1. In this example the target matrix is of size 5×5, with row names $R_1, \cdots, R_5$ and column names $C_1, \cdots, C_5$. In each step, we choose one row and one column uniformly at random (Step A). Suppose R_2 and C_2 is chosen, with corresponding entry 1, the red number in the top middle plot of Figure 1. Then we randomly choose a 0 among all the 0s in R_2 (Step B). Since there is only one 0 in R_2, which is at location C_4, this is our only choice. Again, we scan through all the entries in the same column with the 0 just chosen (C_4) and randomly choose a 1 among all 1s (Step C). In our example, the 1s of C_4 are located at R_1 and R_4. Suppose we have chosen (R_4, C_4). Now the three locations $(R_2, C_2), (R_2, C_4), (R_4, C_4)$ altogether give us the fourth one (R_4, C_2), making the four entries a rectangle (Step D). If the fourth entry equals 0, then we swap the submatrix as we did in the swap method (Step E). Otherwise, the fourth entries equals 1 and the original matrix is not changed. After Step A-E is iterated many times, the resulting randomized matrices are used as representatives of uniformly distributed matrix with fixed margins.

The main difference between the Rectangle Loop algorithm and the swap algorithm is the sampling scheme. The Rectangle Loop algorithm is performing 'conditional sampling', making it more efficient than the swap method, which is doing 'unconditional choosing'. For example, suppose both the swap method and Rectangle Loop algorithm have chosen R_2 and C_2, an entry with a value 1. Then R_2 has only one 0 which is in column 4. For the swap algorithm, the probability of correctly choosing C_4 is only $\frac{1}{4}$, as it is uniformly choosing among all columns. The Rectangle Loop algorithm, however, as the mechanism guarantees we only sample from the zero entries, chooses C_4 with probability 1. Therefore it significantly increases the swapping probability, leading to a faster convergence than the swap chain.

The details of the Rectangle Loop algorithm are described in Algorithm 3. Noteworthy, when finding a 0 entry, we sample 1 with the same column as the 0. When finding a 1, we sample 0 with the same row as the 1. This 'symmetric' design ensures the algorithm converges to the correct distribution, as will be proved in Section 4. The paths of sampled entries in each iteration always form a rectangle, that is where the name 'Rectangle Loop algorithm' comes from.

Algorithm 3 Rectangle Loop Algorithm

```
Input: initial binary matrix $A_0$, number of iterations $T$
1: for $t = 1, \cdots T$ do
2:     Choose one row and one column $(r_1, c_1)$ uniformly at random
3:     if $A_{t-1}(r_1, c_1) = 1$ then
4:         Choose one column $c_2$ at random among all the 0 entries in $r_1$
5:         Choose one row $r_2$ at random among all the 1 entries in $c_2$
6:     else   $A_{t-1}(r_1, c_1) = 0$
7:         Choose one row $r_2$ at random among all the 1 entries in $c_1$
8:         Choose one column $c_2$ at random among all the 0 entries in $r_2$
9:     end if
10:    if the submatrix extracted from $r_1, r_2, c_1, c_2$ is a 'checkerboard unit' then
11:        Swap the submatrix
12:    else   $A_t \leftarrow A_{t-1}$
13:    end if
14: end for
```

4. Theoretical results

Given row sums $\mathbf{r} = (r_1, r_2, \cdots, r_m)$ and column sums $\mathbf{c} = (c_1, c_2, \cdots, c_n)$, we define $\Sigma_{\mathbf{r},\mathbf{c}}$ be the set of all matrices with row sums $\mathbf{r}$ and column sums $\mathbf{c}$. The suffcient and necessary condition for $\Sigma_{\mathbf{r},\mathbf{c}}$ not being zero is given by Gale [6], Ryser [15] in 1957. We call two matrices A, B 'swappable' if one can transform to the other via one step swap algorithm. Equivalently, A and B only differs in a 2×2 'checkerboard unit'. For the sake of simplicity, we assume henceforth $0 < r_i < n, 0 < c_j < m$ for any $1 \le i \le m, 1 \le j \le n$, as otherwise we could simply delete that degenerate row/column. The following theorem characterizes the limit distribution and transition probability of the swap chain.

Theorem 1. *Given $\mathbf{r}, \mathbf{c}$ and an initial matrix $A_0 \in \Sigma_{\mathbf{r},\mathbf{c}}$, the swap algorithm defines a Metropolis-type Markov chain with stationary distribution Unif$(\Sigma_{\mathbf{r},\mathbf{c}})$, transition kernel:*

$$\mathbb{P}(A, B) = \begin{cases} \frac{1}{\binom{m}{2}\binom{n}{2}} & \text{If } A \text{ and } B \text{ are swappable,} \\ 1 - \frac{s(A)}{\binom{m}{2}\binom{n}{2}} & \text{If } B = A, s(A) \doteq \#\{C\text{: } C \text{ and } A \text{ are swappable}\} \\ 0 & \text{Otherwise} \end{cases}$$

and acceptance probability 1.

Proof. When A and B are swappable, there exists two rows i_1, i_2 and two columns j_1, j_2 such that A and B only differs in the 2×2 submatrix extracted by row i_1, i_2 and column j_1, j_2. Therefore the probability of swapping A to B equals

$$\frac{1}{\binom{m}{2}\binom{n}{2}}.$$

Recall that in the setting of Metropolis-Hastings, the acceptance probability from A to B is given by

$$\min\{1, \frac{\pi(B)\mathbb{P}(B,A)}{\pi(A)\mathbb{P}(A,B)},\}$$

notice here the stationary distribution π is designed to be $\mathrm{Unif}(\Sigma_{\mathbf{r},\mathbf{c}})$, $\mathbb{P}(A,B) = \mathbb{P}(B,A) = \frac{1}{\binom{m}{2}\binom{n}{2}}$. Hence it is clear that the swap algorithm is a Metropolis-type Markov chain with $\mathrm{Unif}(\Sigma_{\mathbf{r},\mathbf{c}})$ as stationary distribtuion and acceptance probability 1, which justifies the correctness of the swap algorithm. □

The key point in swap algorithm is symmetry. When two different states A, B are swappable, the associated transition probability is symmetric, i.e.,

$$\mathbb{P}(A,B) = \mathbb{P}(B,A),$$

this ensures the chain has acceptance probability 1.

To compare the efficiency of different Markov kernels with the same distribution, Peskun [11] first introduced the following partial-ordering.

Let $\mathbb{P}_1$, $\mathbb{P}_2$ be two Markov transition kernels on the same state space $\mathcal{S}$ with same stationary distribution π, then $\mathbb{P}_1$ *dominates* $\mathbb{P}_2$ *off the diagonal*, $\mathbb{P}_1 \succeq \mathbb{P}_2$, if

$$\mathbb{P}_1(x, A) \geq \mathbb{P}_2(x, A)$$

for all $x \in \mathcal{S}$ and A measurable with $x \notin A$.

When the state space is finite, as in our case, $\mathbb{P}_1 \succeq \mathbb{P}_2$ iff all the off-diagonal entries of $\mathbb{P}_1$ are greater than or equal to the corresponding off-diagonal entries of $\mathbb{P}_2$. This indicates $\mathbb{P}_1$ has lower probability to get stuck in the same state, and is exploring the state space in a more efficient way. The following theorem shows the rectangale loop algorithm also has uniform distribution as stationary distribution, and the corresponding chain dominates the swap chain off the diagonal. For simplicity, we will use $\mathbb{P}_s$ and $\mathbb{P}_r$ to denote transition kernel for the swap chain and rectangle loop chain, respectively, omitting its dependency on $\mathbf{r}, \mathbf{c}, \mathrm{Unif}(\Sigma_{\mathbf{r},\mathbf{c}})$.

Theorem 2. *Given $\mathbf{r}, \mathbf{c}$ and an initial matrix $A_0 \in \Sigma_{\mathbf{r},\mathbf{c}}$, the Rectangale loop algorithm defines a Metropolis-type Markov chain with stationary distribution Unif$(\Sigma_{\mathbf{r},\mathbf{c}})$. The transition kernel $\mathbb{P}_r$ dominates $\mathbb{P}_s$ off the diagonal.*

Proof. Given any two swappable configurations A and B, we are aiming to show $\mathbb{P}_r(A,B) = \mathbb{P}_r(B,A) \geq \mathbb{P}_s(A,B)$.

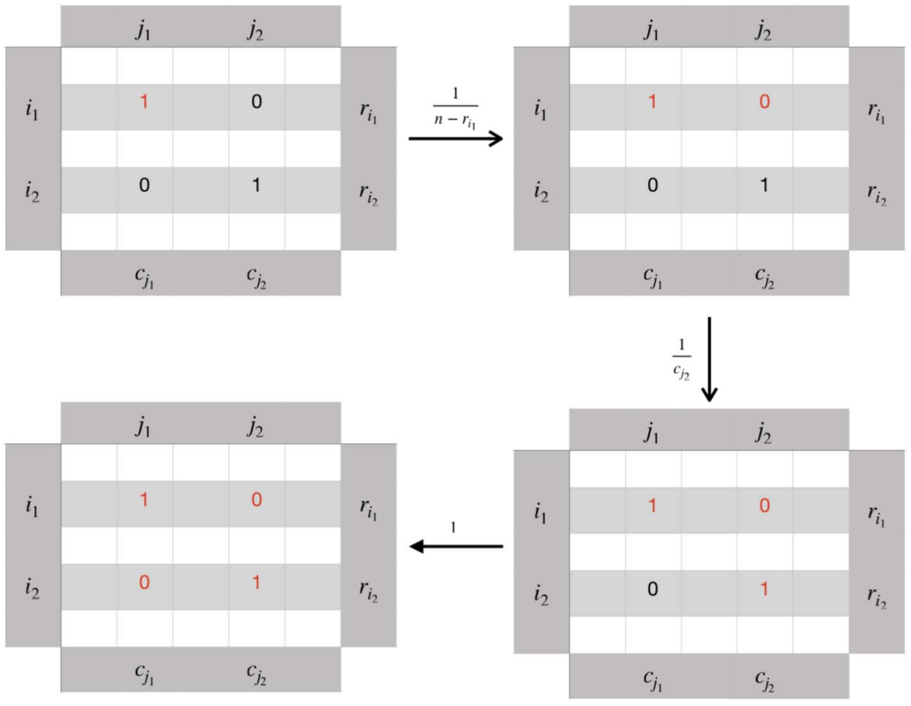

FIG 2. *An illustration of calculating $\mathbb{P}_r(A,B)$ for a single path, starting from vertex (i_1,j_1).*

As A, B are swappable, there exists two rows i_1, i_2 and two columns j_1, j_2 such that A and B only differs in the 2×2 submatrix extracted by row i_1, i_2 and column j_1, j_2. Without loss of generality, we assume the checkerboard unit corresponding to A has the form $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, as shown in Figure 2. Notice that there are four vertices of the 2×2 submatrix and the Rectangle Loop algorithm chooses one arbitrary row and column at its first step. This suggests the probability of transforming A to B equals the summation of four probabilities, each one corresponds to choosing one specific vertex of the ‘checkerboard unit’.

Figure 2 illustrates the calculation of one path, starting from the vertex (i_1, j_1). The possibility of choosing row i_1 and column j_1 is $\frac{1}{mn}$. Then one chooses a 0 among all the 0s in row i_1, and there are $n - r_{i_1}$ of them. Therefore the possibility of choosing column j_2 equals $\frac{1}{n-r_{i_1}}$. Similarly, after choosing j_2, one chooses a 1 among all 1s in column j_2, and there are c_{j_2} of them. Hence the possibility of choosing row i_2 equals $\frac{1}{c_{j_2}}$. The fourth entry is fixed after determining the first three entries thus the last step has probability 1. Multipling the possibilities above altogether, the possibility of transforming A to B, starting with (i_1, j_1), equals

$$\frac{1}{mn} \cdot \frac{1}{n - r_{i_1}} \cdot \frac{1}{c_{j_2}}.$$

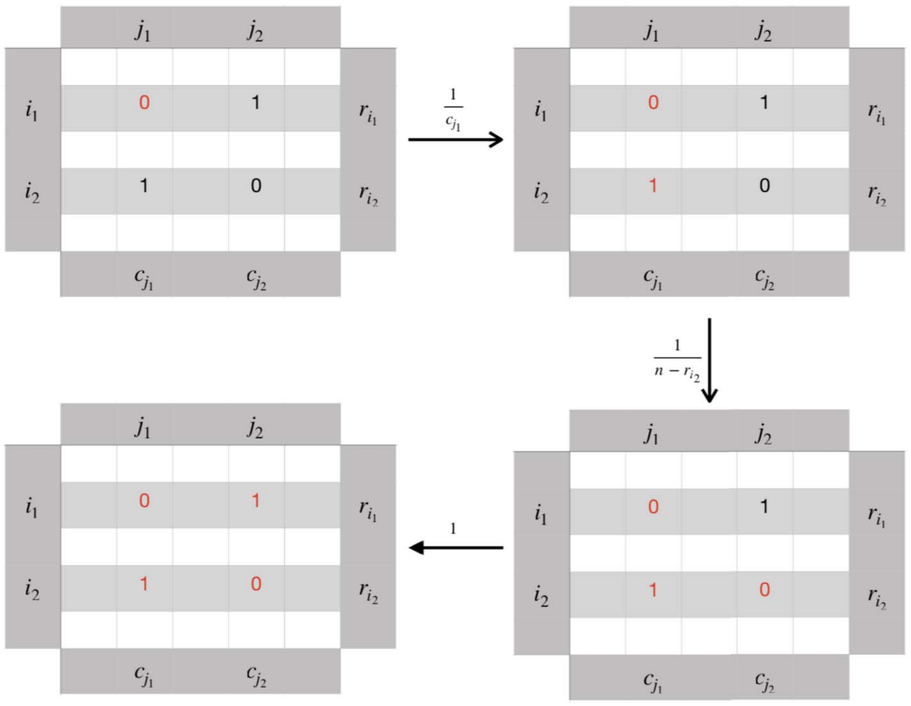

FIG 3. *An illustration of calculating $\mathbb{P}_r(B, A)$ for a single path, starting from vertex (i_1, j_1).*

The probability of transforming A to B with other starting vertex can be calculated accordingly. It turns out $\mathbb{P}_r(A, B)$ can be written as the following summation:

$$\mathbb{P}_r(A,B) = \frac{1}{mn}\left(\frac{1}{n-r_{i_1}}\cdot\frac{1}{c_{j_2}} + \frac{1}{c_{j_2}}\cdot\frac{1}{n-r_{i_2}} + \frac{1}{n-r_{i_2}}\cdot\frac{1}{c_{j_1}} + \frac{1}{c_{j_1}}\cdot\frac{1}{n-r_{i_1}}\right).$$

Following the same strategy, $\mathbb{P}_r(B, A)$ can also be calculated below. Figure 3 illustrates the calculation of $\mathbb{P}_r(B, A)$ starting from (i_1, j_1).

$$\mathbb{P}_r(B,A) = \frac{1}{mn}\left(\frac{1}{c_{j_1}}\cdot\frac{1}{n-r_{i_2}} + \frac{1}{n-r_{i_2}}\cdot\frac{1}{c_{j_2}} + \frac{1}{c_{j_2}}\cdot\frac{1}{n-r_{i_1}} + \frac{1}{n-r_{i_1}}\cdot\frac{1}{c_{j_1}}\right).$$

After matching all the terms of $\mathbb{P}_r(B, A)$ with $\mathbb{P}_r(A, B)$, we conclude that $\mathbb{P}_r(B, A) = \mathbb{P}_r(A, B)$, which justifies the Rectangle Loop algorithm has $\text{Unif}(\Sigma_{\mathbf{r},\mathbf{c}})$ as stationary distribution.

To show $\mathbb{P}_r(A, B) \geq \mathbb{P}_s(A, B)$, notice that

$$\mathbb{P}_s(A,B) = \frac{1}{\binom{m}{2}\binom{n}{2}} = \frac{4}{m(m-1)n(n-1)}$$

and

$$\mathbb{P}_r(A,B) = \frac{1}{mn}\cdot\frac{1}{n-r_{i_1}}\cdot\frac{1}{c_{j_2}} + \frac{1}{mn}\cdot\frac{1}{c_{j_2}}\cdot\frac{1}{n-r_{i_2}} + \frac{1}{mn}\cdot\frac{1}{n-r_{i_2}}\cdot\frac{1}{c_{j_1}} + \frac{1}{mn}\cdot\frac{1}{c_{j_1}}\cdot\frac{1}{n-r_{i_1}}.$$

It is clear that $\mathbb{P}_r(A,B)$ can be written as the summation of four terms. Each term in the summation is greater than or equal to $\frac{1}{m(m-1)n(n-1)}$. Therefore we conclude that $\mathbb{P}_r(A,B) \geq \mathbb{P}_s(A,B)$ for any swappable A, B. This indicates $\mathbb{P}_r \succeq \mathbb{P}_s$, the Rectangle Loop algorithm is exploring the state space in a more efficient way than the swap algorithm. □

5. Simulation results and applications

5.1. A concrete example

Now we use the example used in [17] and [10] to compare the existing algorithms and the Rectangle Loop algorithm. The example below is concrete. The transition matrix can be calculated explicitly and convergence can be assessed analytically.

The five matrices shown in Table 2 are all possible configurations of 3×3 binary matrices with $[1,2,1]$ as both row and column sums. The transition matrices for swap algorithm, Curveball algorithm, and Rectangle loop algorithm are shown in Table 3.

TABLE 3
Transition matrices for swap algorithm (left), Curveball algorithm (middle), Rectangle Loop algorithm (right)

Swap	Curveball	Rectangle Loop
$\begin{bmatrix} \frac{5}{9} & \frac{1}{9} & \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{2}{3} & 0 & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & 0 & \frac{2}{3} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} & \frac{2}{3} & 0 \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} & 0 & \frac{2}{3} \end{bmatrix}$	$\begin{bmatrix} \frac{1}{3} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & 0 & \frac{1}{6} & \frac{1}{3} \\ \frac{1}{6} & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{6} & 0 & \frac{1}{3} \end{bmatrix}$	$\begin{bmatrix} 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{6} & \frac{1}{3} \\ \frac{1}{4} & \frac{1}{3} & \frac{1}{6} & \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{6} & \frac{1}{3} & 0 & \frac{1}{4} \end{bmatrix}$

Figure 4 shows the comparison of the three algorithms. Here we measure the distance between transition kernel $\mathbb{P}$ and the stationary distribution π by total variation distance:

$$\max_{A\in\Sigma_{\mathbf{r},\mathbf{c}}} \|\mathbb{P}^k(A,\cdot)-\pi\|_{\mathrm{TV}} = \frac{1}{2}\max_{A\in\Sigma_{\mathbf{r},\mathbf{c}}} \sum_{B\in\Sigma_{\mathbf{r},\mathbf{c}}} |\mathbb{P}^k(A,B)-\pi(B)|,$$

where k denotes the power. It is clear that all the algorithms converges, but Rectangle Loop algorithm converges faster than the swap algorithm and Curveball algorithm.

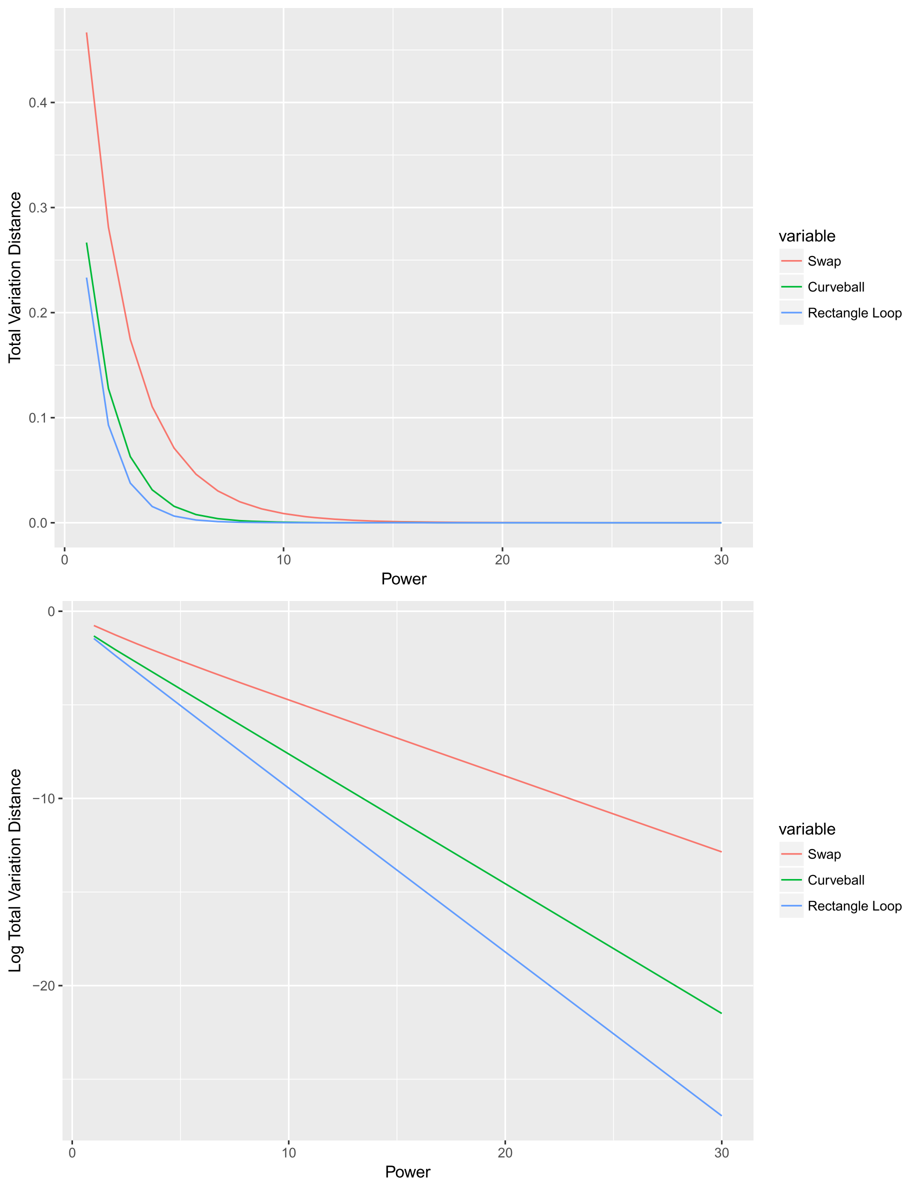

FIG 4. *Comparison between swap algorithm, Curveball algorithm, and Rectangle Loop algorithm. Top: relationship between total variation distance and the power of the transition matrix. Bottom: relationship between the logarithm of total variation distance and the power of transition matrix.*

5.2. Experiments on empirical mixing time

For larger matrices, it is infeasible to calculate the transition matrix theoretically. To provide empirical justification for the advantage of the Rectangle Loop algorithm. We have designed the experiment as follows. For each $p = 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5$, a 100×100 binary matrix is generated for which each entry has probability p to be 1. We ran each algorithm for 10000 iterations and collected the corresponding number of swaps, as shown in Table 4. When the filled portion p is small, the Rectangle Loop algorithm is extremely efficient, producing more than 73 times more swaps than the swap algorithm. For large p, the advantage of Rectangle Loop algorithm is reduced, but still very significant. For $p = 0.5$, the Rectangle Loop still produces 4 times more swaps than the swap algorithm. Noteworthy, the zeros and ones play the symmetric rule in a binary matrix, therefore it is not necessary to generate the random matrix for $p > 0.5$.

TABLE 4
The comparison between swap algorithm and Rectangle Loop algorithm. Each algorithm is implemented 10000 *iterations on* 100×100 *matrices with different filled portions. The third column records the number of successful swaps among the* 10000 *iterations, the last column records the average time per swap, respectively.*

Method	Filled portion	Number of swaps	Time per swap (/s)
Rectangle Loop	1%	586	1.18×10^{-5}
Swap		8	3.67×10^{-4}
Rectangle Loop	5%	977	5.30×10^{-6}
Swap		42	3.52×10^{-5}
Rectangle Loop	10%	1838	3.23×10^{-6}
Swap		156	1.25×10^{-5}
Rectangle Loop	20%	3271	2.64×10^{-6}
Swap		509	5.68×10^{-6}
Rectangle Loop	30%	4222	2.10×10^{-6}
Swap		803	5.06×10^{-6}
Rectangle Loop	40%	4794	1.27×10^{-6}
Swap		1160	4.98×10^{-6}
Rectangle Loop	50%	5080	1.37×10^{-6}
Swap		1271	5.36×10^{-6}

The result above justifies our theoretical result that the Rectangle Loop algorithm converges faster than the swap algorithm. However, the above experiments did not consider the running time for each iteration. In fact, one iteration of the Rectangle Loop algorithm is computationally more expensive than that of the swap algorithm. To investigate this issue, we also record the time per swap for both algorithms, as shown in the last column of Table 4. It turns out that the Rectangle Loop algorithm still has a significant advantage than the swap algorithm after the running time issue is taken into account. For $p = 0.01$, the Rectangle Loop is about 31 times more efficient than the swap algorithm. Even for $p = 0.5$, the Rectangle Loop algorithm is still about 4 times more efficient than the swap algorithm.

We have also used the pertubation score suggested by Strona et al. [17] to access convergence for both algorithm. Pertubation score of a matrix is defined by the fraction of cells differing from the corresponding ones of the initial matrix. It takes several iterations for each algorithm to stabilize around its expectation. It is shown in Figure 5 that it takes less iterations and less time for the Rectangle Loop algorithm to stabilize, suggesting a faster mixing than the swap algorithm.

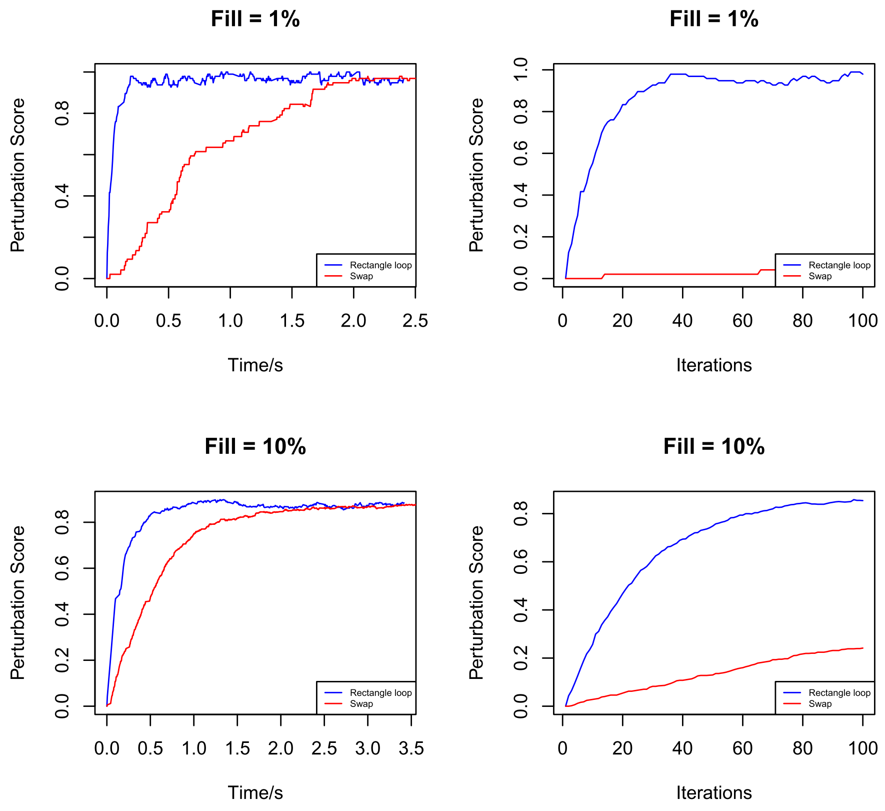

FIG 5. *Comparison between swap algorithm and Rectangle Loop algorithm in mixing* 100×100 *matrices. The left two subplots are the relationship between time and pertubation score for different filled portions. The right two subplots are the relationship between (every 100) iterations and pertubation score for different filled portions. Rectangle Loop algorithm is represented by blue curves and swap algorithm is represented by red curves.*

5.3. Finch data applications

Going back to 'Darwin's Finch' dataset, we use the test statistics $\bar{S}^2$ suggested by Roberts and Stone [14] to compare the three algorithms. $\bar{S}^2$ is defined by

$$\bar{S}^2(A) = \frac{1}{m(m-1)} \sum_{i \neq j} s_{ij}^2,$$

where m is the number of rows of matrix A, $S = (s_{ij}) = AA^T$. For the finch data, $\bar{S}^2 = 45.03$. Suppose this number is too large or too small, comparing with its expectation over all the matrices having the same margins as finch data. We would like to conclude that the cooperation/competition do influence the distribution of species. To investigate this, we implemented the swap algorithm, Curveball algorithm and Rectangle Loop algorithm on the same data for 20000 iterations, using its average as an estimator for $\mathbb{E}(\bar{S}^2)$. The results are shown in Figure 6. After 20000 iterations, Rectangle Loop algorithm gives an estimate of 42.135 with standard deviation 0.537, swap algorithm gives an estimate of 42.126 with standard deviation 0.509, Curveball algorithm gives an estimate of 42.191, with standard deviation 0.590. Therefore the observed data falls outside the three standard deviation boundaries for all three algorithms, suggesting strong evidence that the observed occurrence matrix is not just by chance. Meanwhile, both the swap algorithm and the Rectangle algorithm gives similar estimations and lower standard deviations, which seem to be more accurate than the Curveball algorithm. Lastly, there is a significant pattern in Figure 6 that for both $\bar{S}^2$ and standard deviation estimation, the Rectangle Loop algorithm becomes stabilized much earlier than the swap algorithm, indicating a faster mixing.

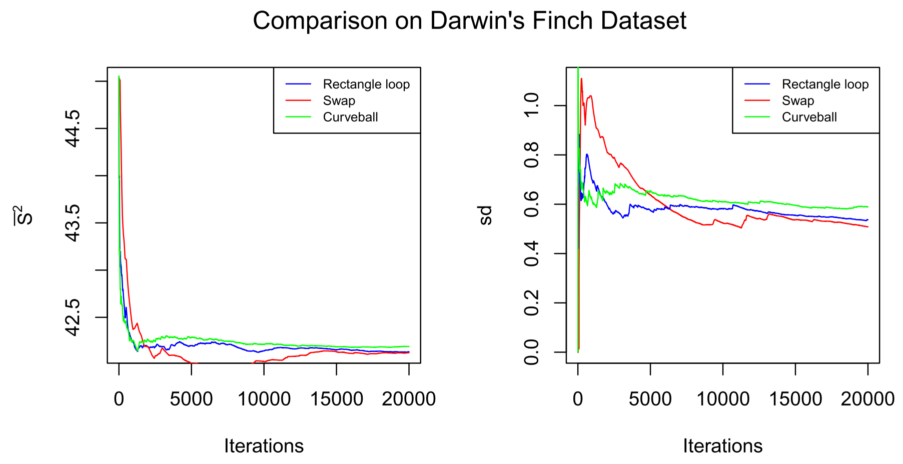

FIG 6. *Comparison between swap algorithm, Curveball algorithm, and Rectangle Loop algorithm. The left subplot is the relationship between the average of $\bar{S}^2$ with the number of iterations. The right subplot is the relationship between the standard deviation of $\bar{S}^2$ with the number of iterations. Blue, red and green curves represent Rectangle Loop, swap and Curveball algorithms respectively.*

6. Discussion

There is a growing tendency to study the behavior of binary matrices with fixed margins in numerous scientific fields, ranging from mathematics to natural science to szocial science. For example, mathematicians and computer scientists are interested in the total number of configurations of given margin sums. Ecologists use the so-called *occurence matrix* to model the presence/absence of species in different locations. Biologists use the binary matrix to model neuronal networks. Social scientists use the binary matrix for studying social network features.

One of the central and difficult problems is uniformly sampling binary matrices with given margins. In this article, we have developed the Rectangle Loop algorithm which is efficient, intuitive and easy to implement. Theoretically, the algorithm is superior to the classical swap algorithm in Peskun's order. In practice, the Rectangle Loop algorithm is notably more efficient than the swap approach. For a fixed number of iterations, the Rectangle Loop algorithm produces 4–73 times more successful swaps than the swap algorithm. For a fixed amount of time, the Rectangle Loop algorithm still produces 4–31 times more successful swaps than the swap algorithm. This suggests the Rectangle Loop algorithm is efficient both statistically and computationally.

Many other problems remain. From a theoretical point of view, it is important to give sharp bounds on the convergence speed of a given Markov chain. However, giving a useful running time estimate is often challenging in practical problems. It would be very interesting if the swap algorithm, Curveball algorithm, and the Rectangle Loop algorithm can be investigated analytically. From an applied point of view, many factors influence the performance of algorithms, such as running time per step (swap algorithm is the fastest, while Curveball algorithm is the slowest), initialization of the matrix, size of the matrix, the ratio between row number and column numbers, filled proportions. Our empirical studies suggest that all the factors have a significant impact on the convergence speed for all the algorithms. It would be beneficial if more numerical experiments are carried out, yielding a complete and comprehensive comparison between all the existing algorithms.

Acknowledgements

The author would like to thank Persi Diaconis for helpful discussions, and David Sichen Wu for helpful suggestions in improving this paper. The authors would also like to thank an anonymous reviewer for helpful comments and suggestions on previous version of this paper.

References

[1] Julian Besag and Peter Clifford. Generalized Monte Carlo significance tests. *Biometrika*, 76(4):633–642, 1989. MR1041408

[2] Corrie Jacobien Carstens and Pieter Kleer. Speeding up switch Markov chains for sampling bipartite graphs with given degree sequence. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. MR3857274

[3] Yuguo Chen, Persi Diaconis, Susan P Holmes, and Jun S Liu. Sequential Monte Carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100(469):109–120, 2005. MR2156822

[4] Persi Diaconis and Anil Gangolli. Rectangular arrays with fixed margins. In *Discrete probability and algorithms*, pages 15–41. Springer, 1995. MR1380519

[5] Persi Diaconis, Bernd Sturmfels, et al. Algebraic algorithms for sampling from conditional distributions. *The Annals of statistics*, 26(1):363–397, 1998. MR1608156

[6] David Gale et al. A theorem on flows in networks. *Pacific J. Math*, 7(2):1073–1082, 1957. MR0091855

[7] Matthew T Harrison and Jeffrey W Miller. Importance sampling for weighted binary random matrices with specified margins. *arXiv preprint* arXiv:1301.3928, 2013.

[8] RB Holmes and LK Jones. On uniform generation of two-way tables with fixed margins and the conditional volume test of diaconis and efron. *The Annals of Statistics*, pages 64–68, 1996. MR1389880

[9] Ravi Kannan, Prasad Tetali, and Santosh Vempala. Simple Markov-chain algorithms for generating bipartite graphs and tournaments. *Random Structures & Algorithms*, 14(4):293–308, 1999. MR1691976

[10] István Miklós and János Podani. Randomization of presence–absence matrices: comments and new algorithms. *Ecology*, 85(1):86–92, 2004.

[11] Peter H Peskun. Optimum Monte-Carlo sampling using Markov chains. *Biometrika*, 60(3):607–612, 1973. MR0362823

[12] A Ramachandra Rao, Rabindranath Jana, and Suraj Bandyopadhyay. A Markov chain Monte Carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 225–242, 1996. MR1662523

[13] Steffen Rechner. Markov chain Monte Carlo algorithms for the uniform sampling of combinatorial objects. 2018.

[14] Alan Roberts and Lewis Stone. Island-sharing by archipelago species. *Oecologia*, 83(4):560–567, 1990.

[15] Herbert J Ryser. Combinatorial properties of matrices of zeros and ones. In *Classic Papers in Combinatorics*, pages 269–275. Springer, 2009.

[16] Tom AB Snijders. Enumeration and simulation methods for 0–1 matrices with given marginals. *Psychometrika*, 56(3):397–417, 1991. MR1131767

[17] Giovanni Strona, Domenico Nappo, Francesco Boccacci, Simone Fattorini, and Jesus San-Miguel-Ayanz. A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. *Nature communications*, 5:4114, 2014.

[18] Norman D Verhelst. An efficient MCMC algorithm to sample binary matrices with fixed marginals. *Psychometrika*, 73(4):705, 2008. MR2469789