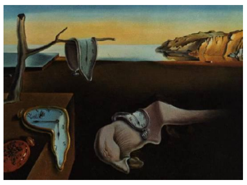


Journal of Statistical Software

February 2008, Volume 24, Issue 8. *http://www.jstatsoft.org/*

networksis: A Package to Simulate Bipartite Graphs with Fixed Marginals Through Sequential Importance Sampling

Ryan Admiraal
University of Washington

Mark S. Handcock
University of Washington

Abstract

The ability to simulate graphs with given properties is important for the analysis of social networks. Sequential importance sampling has been shown to be particularly effective in estimating the number of graphs adhering to fixed marginals and in estimating the null distribution of graph statistics. This paper describes the **networksis** package for R and how its `simulate` and `simulate_sis` functions can be used to address both of these tasks as well as generate initial graphs for Markov chain Monte Carlo simulations.

Keywords: networks, social network analysis, R, statnet, bipartite network, Markov chain Monte Carlo, graph counting.

1. Introduction

A bipartite graph is a graph for nodes of two distinct types with the relation defined to be between nodes of different types. The set of nodes can be represented by two subsets $\mathcal{R}$ and $\mathcal{C}$ for which nodes in $\mathcal{R}$ only have ties to nodes in $\mathcal{C}$, and nodes in $\mathcal{C}$ only have ties to nodes in $\mathcal{R}$. One of the more common types of bipartite graphs is an affiliation network for which the two types can be called "actor" and "event" and the relation indicates the affiliation of the actor with the given event. As an example of an affiliation network, we will later examine the occurrence of finch species on different islands. Bipartite graphs can be represented by a matrix $A = a_{ij}$, where $i \in \mathcal{R}$, $j \in \mathcal{C}$, and $a_{ij} = 1$ if there is a relational tie from i to j and 0 otherwise. In the case of an affiliation matrix, $a_{ij} = 1$ would correspond to actor i being affiliated with event j. In this paper, we demonstrate how bipartite graphs with fixed marginals can be simulated through sequential importance sampling, using the **networksis** package for R (R Development Core Team 2007). By doing this, we can approximate the size of a graph space, estimate the null distribution of a graph statistic, or produce initial graphs for Markov chain Monte Carlo simulations.

In the analysis of bipartite graphs, most research has focused on analytic methods of analyzing properties of the graph or calculating graph statistics. Where analytic methods are impractical, research has generally turned to approximation methods. The approximation method most commonly utilized is Markov chain Monte Carlo (MCMC), which often provides a means to representatively sample the graph space. The standard MCMC algorithm is that developed by Snijders (1991) and extended by Rao, Jana, and Bandyopadhyay (1996). While useful in solving many difficult problems, MCMC has its limitations, and alternative Monte Carlo methods are of interest. Recent research has brought to light the effectiveness of sequential importance sampling (SIS) in solving certain problems for which analytic methods and current MCMC algorithms do not provide good solutions or any solution at all.

SIS has proved particularly useful in counting bipartite graphs with given marginals. For many graphs, exact enumeration of all graphs adhering to marginal constraints simply is not practical. To illustrate the issues, consider the bipartite graph in Table 1, which consists of only 13 rows, 17 columns, and 122 ties. For this graph, a graph of moderate size, the total number of unique graphs matching the row and column sums is more than 6.7×10^{16}. Several elaborate algorithms can perform such tasks of exact graph counting, but the amount of time required to directly compute the number of graphs meeting the marginal constraints generally makes such a computation impractical. One such algorithm was developed by Good and Crook (1977). If we let $\boldsymbol{r} = (r_1, r_2, \ldots, r_m)$ represent the row sums and $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)$ represent the column sums, the coefficient of the generating function

$$\prod_{i=1}^{m} \prod_{j=1}^{n} (1 - x_i y_j)$$

corresponding to the polynomial expansion $\prod_{i=1}^{m} x_i^{r_i} \prod_{j=1}^{n} y_j^{c_j}$ gives us the number of 0–1 tables matching the row sums and column sums given by $\boldsymbol{r}$ and $\boldsymbol{c}$.

Attempts to approximate the number of graphs meeting marginal constraints by means of MCMC are more practical in terms of computing time, but, for graphs similar in size to Table 1, they still require an exorbitant run time for a low degree of accuracy. SIS, on the other hand, requires relatively few sampled graphs and minimal computing time to produce a highly accurate estimate of the number of graphs meeting the marginal constraints. It does this by sampling columns of the graph sequentially, ensuring that new graphs meet the column and row sum constraints of the observed graph. In each column, sampling of rows to receive 1's is done according to specified inclusion probabilities to ensure that the new graph comes from a distribution that is relative uniform (and known). Chen, Diaconis, Holmes, and Liu (2005)(CDHL) provide details of the algorithm, which we expand on in the appendix.

Another application for which SIS has proved useful is in approximating the null distribution of a graph statistic. Such capabilities are important in significance testing. Graphs produced by sequential importance sampling constitute a probability sample, so, by weighting the graphs according to the inverse graph probabilities, the null distribution is the uniform distribution over all graphs. Weighting the graph statistics for each of these graphs according to the inverse graph probabilities, we obtain the null distributions for the graph statistics of interest. For the null distribution of most graph statistics for a graph of moderate size, there are no known analytic approaches that can accurately approximate the distribution. This is primarily

	Island																
Finch	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
Large ground finch	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
Medium ground finch	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0
Small ground finch	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0
Sharp-beaked ground finch	0	0	1	1	1	0	0	1	0	1	0	1	1	0	1	1	1
Cactus ground finch	1	1	1	0	1	1	1	1	1	1	0	1	0	1	1	0	0
Large cactus ground finch	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
Large tree finch	0	0	1	1	1	1	1	1	1	0	0	1	0	1	1	0	0
Medium tree finch	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Small tree finch	0	0	1	1	1	1	1	1	1	1	0	1	0	0	1	0	0
Vegetarian finch	0	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0
Woodpecker finch	0	0	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0
Mangrove finch	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Warbler finch	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 1: Darwin's finch data.

because such algorithms must be modified for the graph statistic under consideration, leading to even more complex algorithms than those used in the graph counting problem. MCMC also appears to be an inferior approach, as Chen *et al.* (2005) provide numerous examples in which MCMC algorithms are consistently less efficient than SIS. CDHL suggests that this is to be expected, as the problems of counting graphs and approximating the null distribution of a graph statistic are both easily solved when sampling graphs uniformly or nearly uniformly, and this is exactly what SIS attempts to do. However, the validity of CDHL's assessment is an empirical question and will depend on the specifics of the problem addressed.

A third application for which SIS can be beneficial is in providing an initial graph from which to start MCMC simulations. Since SIS can quickly generate graphs from different regions of the graph space, it can assist in MCMC diagnostics by providing various starting points, enabling the researcher to observe whether or not the Markov chains converge to the same maximum likelihood estimators (MLEs) for the distributions of interest. This also removes the need for a burn-in period.

In this paper, we describe the **networksis** package for R and how it can be used for each of the three applications we have previously described. The `simulate` and `simulate_sis` functions in the **networksis** package provide a means to simulate bipartite graphs with fixed marginals through sequential importance sampling. They are intended to be used in conjunction with the **statnet** (Handcock, Hunter, Butts, Goodreau, and Morris 2003b) suite of packages. These allow users to create, manipulate, and perform inference on social network and bipartite network objects.

In Section 2, we demonstrate how the `simulate` function in the `networksis` package can be used to estimate the size of the graph space for a bipartite graph. In Section 3, we provide examples of how `simulate` and `simulate_sis` can produce the null distribution of a graph statistic. In Section 4, we describe how bipartite graphs produced by `simulate` can be used by the **ergm** package (Handcock, Hunter, Butts, Goodreau, and Morris 2003a) as part of the **statnet** suite of packages to provide initial graphs for MCMC simulations. Throughout, we

1	1	0	0	2
0	0	1	1	2
1	1	1	0	3
2	2	2	1	

Table 2: Example of a bipartite graph with three nodes of one type and four nodes of another.

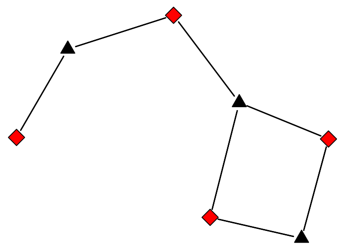


Figure 1: Bipartite graph for our example. Black triangle are nodes corresponding to the rows, and red square are nodes corresponding to the columns.

will use Darwin's finch data to illustrate the varied uses of `simulate` and `simulate_sis`.

2. Enumerating a graph space

Estimating the size of a graph space is important not only in the field of statistics but also other disciplines. For example, Sanderson (2000) recounts how a null space that a well-known ecologist had thought consisted of ten graphs in fact consisted of thousands of graphs, completely changing how the observed graph might be perceived in relation to other graphs in the graph space. Of course, with the understanding that a graph space is much larger than previously thought also comes the realization that the uniform probability of any particular graph is substantially lower, meaning that the observed graph may have been far less likely than originally thought.

We focus specifically on enumerating graph spaces where the marginals are to be held fixed. To briefly illustrate why enumerating these graph spaces can be difficult, consider the bipartite graph given in Table 2. To create a bipartite network object and plot it, we use the following R code:

```
R> library("networksis")
R> bipartite.graph <- c(1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0)
R> bipartite.graph <- matrix(bipartite.graph, nrow = 3, byrow = TRUE)
R> example.net <- network(bipartite.graph)
R> example.net %v% "set" <- c(rep(1, 3), rep(2, 4))
R> color <- shape <- example.net %v% "set"
R> plot(example.net, vertex.col = color, vertex.sides = 2 + shape,
+      vertex.cex = 2.5)
```

The key function here is the `network` function, which, along with the other primary functions in this code, is inherited from the **network** package (Butts, Handcock, and Hunter 2007). This function receives as input the matrix corresponding to the bipartite graph, and produces

Probability = 0.095

$$\begin{array}{|cccc|}\hline 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ \hline\end{array}$$

$$\begin{array}{|cccc|}\hline 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ \hline\end{array}$$

Probability = 0.071

$$\begin{array}{|cccc|}\hline 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ \hline\end{array}$$

$$\begin{array}{|cccc|}\hline 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ \hline\end{array} \quad \begin{array}{|cccc|}\hline 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ \hline\end{array}$$

Table 3: Graphs and corresponding probabilities for graphs matching the marginals of Table 2.

a `network` object. If the matrix is not square, the function infers the network is bipartite. Otherwise, it presumes it is unipartite. You can ensure a bipartite network for square matrices with the optional `bipartite = TRUE` argument. The plot of the bipartite graph is given in Figure 1, where the black triangles are the nodes corresponding to the rows and the red squares are the nodes corresponding to the columns.

For this graph, the number of graphs adhering to the same marginals is 12. The set of possible graphs and corresponding graph probabilities are shown in Table 3. In attempting to simulate these graphs adhering to the marginals, simply ensuring that the column constraints are met is not sufficient, as sampling $(1\ 1\ 0)^\top$ for each of the first two columns ensures that the column marginals are satisfied but creates an inconsistency with the row marginals. This should be clear in that the third row must sample three 1's yet, but only two columns remain. Thus, the number of graphs consistent with the marginals is not simply a function of the number of graphs consistent with the column marginals or the number of graphs consistent with the row marginals. While algorithms for computing the graph space size exactly are not so impractical for bipartite graphs similar in dimension to the one given in Table 2, it quickly becomes impractical to consider such algorithms as the size of the bipartite graph grows. Consequently, simulation methods like MCMC or SIS become useful.

The advantage of SIS over MCMC in enumerating the graph space is that sampling probabilities are known for each simulated graph, and graphs are generated independently, so estimating the size of the graph space is easy. Let $\mathcal{A}$ be the graph space consisting of all bipartite graphs with marginals consistent with the observed bipartite graph A. If we sample N graphs from $\mathcal{A}$ with probabilities $p_1, p_2, \ldots, p_N$, then we can estimate the graph space size by the Hansen-Hurwitz estimator described in Thompson (2002), given by

$$\widehat{|\mathcal{A}|} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{p_i}.$$

The corresponding unbiased estimator of the variance is given by

$$\widehat{\mathbb{V}}\left(\widehat{|\mathcal{A}|}\right) = \frac{1}{N(N-1)}\sum_{i=1}^{N}\left(\frac{1}{p_i} - \widehat{|\mathcal{A}|}\right)^2.$$

If we want to sample 1,000 graphs using SIS and estimate the graph space size and corresponding standard error for the bipartite graph of Table 2, we use the following code:

```
R> sim <- simulate(example.net, nsim = 1000)
```

This code illustrates the two inputs required by the `simulate` function. The first input must be a bipartite network object. The second input must be the number of graphs to sample. To determine the (logarithm of the) size of the graph space and corresponding (log) standard error, we need only extract the `log.graphspace.size` or `log.graphspace.SE` attributes of the `network` object `sim`. Doing so produces the following estimate and standard error:

```
R> exp(sim %n% "log.graphspace.size")

[1] 11.9875

R> exp(sim %n% "log.graphspace.SE")

[1] 0.05474112
```

This provides a close approximation to the known graph space size of 12. Increasing the number of sampled graphs will result in greater accuracy and precision.

For a more compelling example, we use Darwin's finch data, data that have prompted much discussion in the field of ecology in regard to evolution and competition between species. The data can be found in Table 1. Charles Darwin compiled this data for thirteen finch species found on the Galapagos Islands during the voyage of the H.M.S. *Beagle* from 1831–1836. For each finch type, he recorded on which of seventeen islands that finch could be found. Sulloway (1982) explains that the finch was of greater interest than other birds found on the islands because the sheer quantity of finch species present in this set of islands far exceeded that of any other type of bird. This suggests that these birds had a greater evolutionary complexity and were probably one of the the earliest winged inhabitants of the island chain. It is theorized that, due to prolonged isolation on the different islands, the thirteen finch species developed. Later, some of the species began to migrate to different islands, leading to competition among finch species that were similar and producing the groupings of finch species that Darwin observed on these islands. These groupings represented in Table 1 can also be seen in Figure 2, which was generated using the following code:

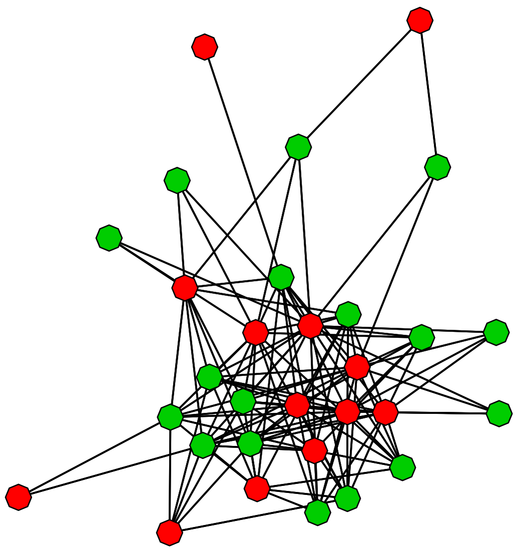


Figure 2: Finch species and the Galapagos Islands they inhabit, as reported by Charles Darwin on the voyage of the H.M.S. *Beagle*. (Islands are red, and finch species are green.)

```
R> data("finch")
R> plot(finch, vertex.col = c(rep(2, 13), rep(3, 17)), vertex.cex = 2.5)
```

In Figure 2, we notice that most of the finches and islands are clustered together, representing those islands for which most of the finch species are present. On the outskirts of the graph are those islands for which only several finch species may be found, and on the fringes of the cluster of islands and finches are those finches which occupy those less-populated islands.

In examining island biogeography, Sanderson (2000) argues that it is important to condition on the number of islands and species in order to sample from the appropriate null space, so graphs sampled from the null distribution of the observed graph should have the same marginals. CDHL report the number of graphs matching the marginal constraints of Darwin's finch data to be 67,149,106,137, 567,626. Using a sequential importance sample of size 10,000, we can estimate the total number of such graphs using the following code:

```
R> sim <- simulate(finch, nsim = 10000)
R> exp(sim %n% "log.graphspace.size")
[1] 6.839628e+16
R> exp(sim %n% "log.graphspace.SE")
[1] 7.215396e+14
```

A sequential importance sample of size 10,000 estimates the total number of such graphs to be 6.840×10^{16} with a standard error of 7.2×10^{14}, estimates which closely match the results from CDHL's SIS algorithm. To observe the distribution of the importance weights, we plot a histogram of the inverse graph probabilities. This can be done using the following code:

```
R> importance.weights <- 1 / exp(sim %n% "log.prob")
R> hist(importance.weights, breaks = 75,
+       xlab = "Inverse Graph Probability", main = "")
```

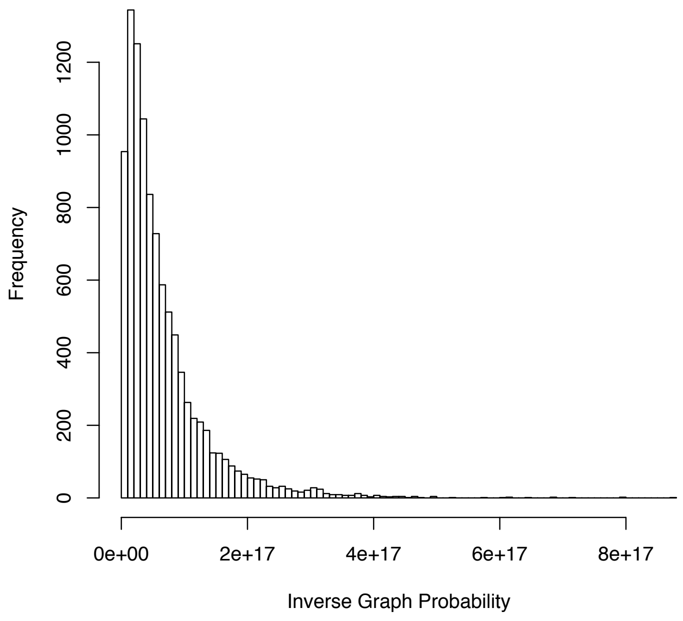


Figure 3: Histogram of finch data importance weights ($1,000$ weights).

This produces the histogram in Figure 3, which closely resembles CDHL's histogram of importance weights. Note the skewness of the weights. If SIS produced a simple random sample from the space of graphs, the weights would be approximately equal (and equal to about 6.715×10^{16}). Since SIS produces a probability sample, this is not the case, and we observe a substantial proportion of graphs with weights more than three times this level. We note that the sample weights are approximately log-normal and the expectation of the distribution is the size of the graph space. This suggests estimating the graph space size by the UMVU estimator of the log-normal mean given by Finney (1941). The (log) estimate and the corresponding (log) estimate of the SE are given by the the `log.graphspace.size.lne` or `log.graphspace.SE.lne` attributes of the resultant `network` object. The efficiency of the regular estimates are above 93% if the variance of the log probabilities is less than 0.7. It decreases as the variance of the log probabilities increases.

3. Approximating the null distribution of a graph statistic

A second application for which SIS is useful is in approximating the null distribution of a graph statistic. This can be done in two ways, using the `simulate` and `simulate_sis` functions. The first way is to use `simulate` to generate new graphs and use those graphs to calculate the statistics of interest. Since `simulate` only retains the last graph sampled, this requires the user to specify the number of simulations to be '1' and then extract the sampled graph using the `as.matrix.network` command. This allows the user to save the graph in matrix form and perform necessary operations on the matrix to compute the statistics of interest. The second way is to use `simulate_sis` and pass to it as arguments bipartite graph statistics already incorporated in the **ergm** package. This allows the user to sample the number of desired graphs all at once and then extract the statistics using the `samplestatistics` attribute. In order to do this, the user must specify the graph statistics of interest on the right-hand side of the $Y \sim X$ formula. Multiple graph statistics can be computed simultaneously, simply by separating graph statistic names with a '+'. Of course, this second approach will only work if the graph statistics of interest are ones included in the **ergm** package. Custom graph statistics will require the first approach.

In considering approximations to the null distribution of a test statistic, CDHL again consider Darwin's finch data. In particular, they address the question of whether the observed grouping of finch species on islands happened by random chance or if it was the result of a struggle in which only species which depended on different food sources could coexist on an island. To test this hypothesis, CDHL consider the test statistic proposed by Roberts and Stone (1990), given by

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

where m is the number of finch species, $S = (s_{ij}) = AA^\top$, and $A = (a_{ij})$ is the bipartite graph in Table 1. Here, s_{ij} is simply the number of islands on which finch species i and j coexist for $i \neq j$. The test statistic $\bar{S}^2$ gives one of many possible measures of finch species coexistence, others of which have been touched on by Sanderson, Moulton, and Selfridge (1998) and Sanderson (2000). For instance, Sanderson (2000) considered the relatively simple graph statistic given by the number of instances where two different finch species live on the same island. In the case of the graph statistic proposed by Roberts and Stone (1990), it may seem natural to consider the first moment, $\bar{S}$, but this will be the same for all graphs that meet the marginal constraints of the observed graph. Consequently, they suggest considering the second moment as a measure of coexistence. If there were no competition among the finch species, we would expect finches to share nearly equal numbers of islands with each different type of finch. If competition exists, however, we would expect that a finch will share a larger number of islands with non-competitive finch species and a smaller number of islands with competitive finch species. Since $\bar{S}^2$ is minimized for equal s_{ij} and maximized for values of s_{ij} that are furthest from $\bar{S}$, a large value of $\bar{S}^2$ would be consistent with competition among certain finch species and cooperation among others.

For the finch data, $\bar{S}^2$ has a value of 53.115. We would expect that, if the observed pattern happened by random chance, its value of $\bar{S}^2$ would not be an extreme value when compared with the values of $\bar{S}^2$ produced by randomly sampled graphs. Since $\bar{S}^2$ is not a commonly used graph statistic and is unavailable in the **ergm** package, we generate 100,000 graphs and compute $\bar{S}^2$ separately for each of these graphs using the following code:

```
R> nsim <- 100000
R> prob.vec <- rep(0, nsim)
R> s.bar.squared.vec <- rep(0, nsim)
R> for(i in 1:nsim)
+ {
+ sim <- simulate(finch, nsim = 1)
+ new.graph <- as.matrix.network(sim)
+ s.bar.squared <- (sum((new.graph %*% t(new.graph)) ^ 2) -
+                    sum(diag((new.graph %*% t(new.graph)) ^ 2))) /
+                   (13 * 12)
+ s.bar.squared.vec[i] <- s.bar.squared
```

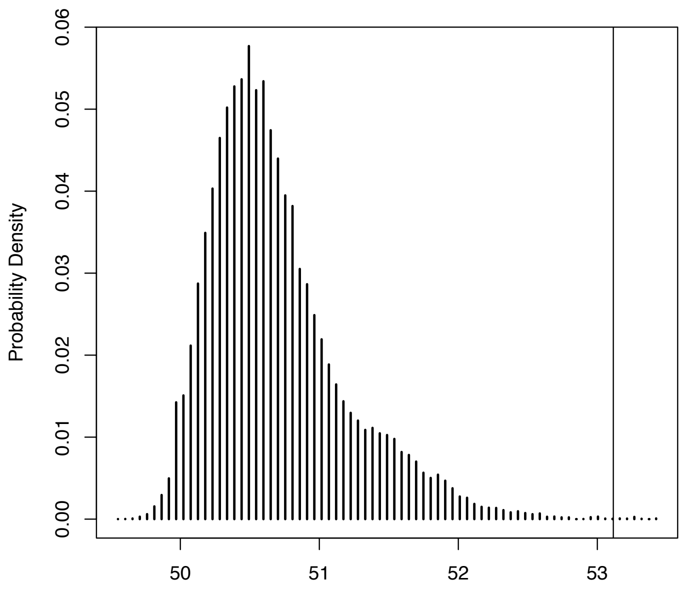


Figure 4: Null distribution of the test statistic $\bar{S}^2$. The vertical line represents the observed value of S^2.

```
+ prob.vec[i] <- exp(sim %n% "log.prob")
+ }
```

It is important that we record not only the graph statistic but also the corresponding graph probability for each generated graph. The reason for this is that the graphs are not a random sample but a probability sample, as demonstrated in Figure 3. Consequently, we must consider the weighted distribution of the statistics in order to determine the null distribution. To plot the null distribution, we use the following code:

```
R> nbreaks <- 75
R> w <- (1 / prob.vec) / (sum(1 / prob.vec))
R> intervals <- cut(s.bar.squared.vec, nbreaks, include.lowest = TRUE)
R> weights <- sapply(split(w, f = intervals), sum)
R> x <- seq(min(s.bar.squared.vec), max(s.bar.squared.vec),
+          length.out = nbreaks)
R> plot(x, weights, type = "h", xlab = "", ylab = "Probability Density",
+      lwd = 2)
R> abline(v = 53.115)
```

This produces the null distribution seen in Figure 4. Only seven graphs produced values of $\bar{S}^2$ larger than the observed value of 53.115 (represented by the vertical line), providing strong evidence that it was unlikely that the grouping of finch species on islands was due to random chance and making it plausible that what we observe is the result of competition and cooperation among the finch species.

While $\bar{S}^2$ may prove useful in determining competition and cooperation among finch species, simply comparing how many finch pairs share a specified number of islands for both the observed graph and simulated graphs (in the vein of Sanderson (2000)) may better demonstrate the competition among certain species and cooperation among other species. We can do this using the `coincidences` term included in the **ergm** package, which counts the number of nodes of the first type (i.e., the row type) that are tied to the same node of the second type (i.e., the column type). For instance, if we wanted to count the number of finches that shared exactly one island in common, we would specify our formula to be `finch ~ coincidences(1)`. If we wanted to count the number of finches that shared exactly two islands in common, we would specify our formula to be `finch ~ coincidences(2)`. We are specifically interested in the number of finches that share no islands in common (i.e., `coincidences(0)`), as these would suggest that these finches are in direct competition with each other. However, we will consider all possible `coincidences` terms to illustrate the ability of the package to easily calculate a number of graph statistics for simulated graphs. The specification we described allows us to simulate all graphs and statistics immediately without having to separately extract the new graphs and compute the statistics ourselves. To do this for 10,000 graphs, we use the `simulate_sis` function and the code:

```
R> sim <- simulate_sis(finch ~ coincidences(0:17), nsim = 10000)
```

This produces the distributions seen in Figure 5, which plots the mean number of finch pairs sharing x islands, $x = 0, 1, ..., 17$, from 10,000 simulated graphs. These means are represented by a circle, and the trend is represented by a dashed line. Vertical lines for each possible value of x show the 95% confidence intervals for the mean number of finches sharing x islands, $x = 0, 1, ..., 17$. For the sake of comparison, the number of finch pairs sharing x islands from the observed graph are also represented in Figure 5 by a red $\times$, and the trend is represented by a solid line. This plot was generated using the **Hmisc** package (Harrell Jr. 2007) and the following code:

```
R> observed.stats <- summary(finch ~ coincidences(0:17))
R> sampled.stats <- sim %n% "samplestatistics"
R> library("Hmisc")
R> p <- exp(sim %n% "log.prob")
R> p <- p / sum(p)
R> maxs <- apply(sampled.stats, 2, wtd.quantile,
+                weights = p, probs = 0.975, normwt = TRUE)
R> mins <- apply(sampled.stats, 2, wtd.quantile,
+                weights = p, probs = 0.025, normwt = TRUE)
R> means <- apply(sampled.stats, 2, wtd.mean, weights = p)
R> plot(0:17, means, type = "b", ylim = c(0, 24.5), lwd = 3, lty = 3,
+       xlab = "Number of Islands", ylab = "Pairs of Finches")
R> for(i in 1:18)
+ {
+ points(rep(i - 1, 2), c(maxs[i], mins[i]), type = "l", lwd = 2)
+ }
R> points(0:17, observed.stats, type = "b", pch = 4, lwd = 3)
```

The mean trend for the simulated graphs is fairly similar to that of the observed graph, possibly suggesting that the observed graph statistics are not extreme. However, the vertical bars corresponding to the 95% confidence intervals tell a much different story. In particular,

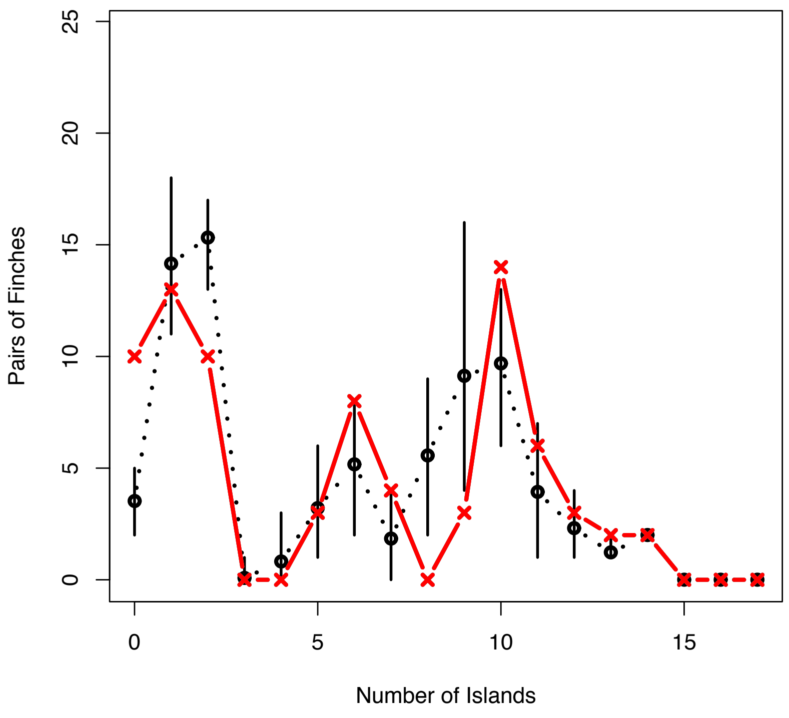


Figure 5: Number of pairs of finches sharing x islands, $x = 0, 1, ..., 17$. Black circles represent the mean numbers of pairs of finches sharing x islands produced from the sampled graphs, and vertical bars correspond to 95% confidence intervals. Red $\times$'s denote the observed numbers of paris of finches sharing x islands.

it seems highly unlikely that the observed number of finches sharing 0 islands happened by random chance. To determine the p-value corresponding to this observed value of interest, we use the following code:

```
R> r0 <- (p %*% sweep(sampled.stats, 2, observed.stats, "<"))[1, ]
R> r1 <- (p %*% sweep(sampled.stats, 2, observed.stats, ">"))[1, ]
R> round(apply(cbind(r0, r1), 1, min), digits = 8)[1]
coincidences0
   0.00083007
```

The p-value of less than 0.001 provides strong evidence that the observed value did not happen randomly, so the claim of competition among certain finch species may possibly explain what we observe. The user should note that, had we been interested in testing more than one `coincidences` terms, a Bonferroni correction for multiple comparisons would have been necessary to assess the significance of the p-value.

4. Generating initial graphs for MCMC simulations

The third application we consider for SIS consists of using SIS as a diagnostic tool for Monte Carlo maximum likelihood estimation. The `ergm` function in the **ergm** package provides a means by which parameters corresponding to graph statistics can be estimated through maximum likelihood or maximum pseudo-likelihood techniques. This can be done for bipartite graphs and social networks. In using MCMC to simulate graphs for the purpose of maximum likelihood estimation, it is possible that certain regions of the graph space will not be sampled or that the chain will remain in certain regions of the space for extended periods of time, leading to incorrect inference (Gelman 1996). Consequently, graph diagnostics may be important to determine whether or not the MLE produced by the Markov chain is in fact a global maximum and not strictly a local maximum. This is typically done by considering multiple runs of a Markov chain, specifying different initial estimates of the parameters to increase the likelihood that all regions of the graph space are explored.

In addition to or in place of changing the initial parameter estimates, we might consider using SIS to allow us to freely move about the graph space and start a Markov chain in a region of the graph space with low probability. To do this, we can sample bipartite graphs using SIS until we obtain one with a probability below a certain threshold. Then this graph can be used as the start graph for a Markov chain. In addition, for users who specify a burn-in phase for the Markov chain, this serves as a substitute for the burn-in. If we again consider the graph statistics represented by `coincidences` and consider a probability threshold of 10^{-18}, we can choose an appropriate start graph and implement it using the `control` option in the `ergm` function. To do this, we use the following code:

```
R> threshold <- 10 ^ (-18)
R> breakloop <- 0
R> while(breakloop == 0)
+ {
+ sissim <- simulate(finch, nsim = 1)
+ if(sissim %n% "log.prob" < log(threshold)){breakloop <- 1}
+ }
R> sim <- ergm(finch ~ coincidences(0), constraints = ~degrees,
+             theta0 = 0.6108, MCMCsamplesize = 10000,
+             control = control.ergm(initial.network = sissim))
```

The user may not always be interested in choosing a start graph that is of low sampling probability but may instead be interested in start graphs for which the graph statistics are of certain values. In this case, the formula may take on significance, as this may be used to produce graph statistics.

5. Discussion

The **networksis** package provides a means to simulate bipartite graphs with fixed marginals through sequential importance sampling. It leverages the facilities of the **statnet** (Handcock *et al.* 2003b) suite of packages to allow a range of significance tests for a wide range of network hypotheses where the null distribution is the space of graphs with fixed marginals.

The package has been parallelized using the **snow** package to use either PVM or MPI (Tierney, Rossini, Li, and Sevcikova 2007). This can be used by specifying the optional `control` argument. For example,

```
R> sim <- simulate_sis(finch ~ coincidences(0:17),
+                          control = simulate.control(parallel = 4)),
+                          nsim = 10000)
```

simulates 10,000 networks using 4 processors. This is particularly useful on a single machine with a multi-core CPU where all cores can be used simultaneously.

It should be noted that the size of bipartite graphs for which sequential importance sampling is effective is determined in part by the available memory in R and the sparsity of the bipartite graph. Sparse graphs will typically require less memory and be much easier to simulate. It should also be noted that this package cannot generate social networks or other 0–1 tables with structural zeros. Chen (2007) provides a sequential importance sampling algorithm for which this can be accomplished, but that algorithm is not implemented in the **networksis** package.

References

Butts CT, Handcock MS, Hunter DR (2007). ***network**: Classes for Relational Data.* Statnet Project `http://statnetproject.org/`, Seattle, WA. R package version 1.3, URL `http://CRAN.R-project.org/package=network`.

Chen XH, Dempster AP, Liu JS (1994). "Weighted Finite Population Sampling to Maximize Entropy." *Biometrika*, **81**, 457–469.

Chen Y (2007). "Conditional Inference on Tables with Structural Zeros." *Journal of Computational and Graphical Statistics*, **16**, 445–467.

Chen Y, Diaconis P, Holmes SP, Liu JS (2005). "Sequential Monte Carlo Methods for Statistical Analysis of Tables." *Journal of the American Statistical Association*, **100**, 109–120.

Finney DJ (1941). "On the Distribution of a Variable Whose Logarithm is Normally Distributed." *Journal of the Royal Statistical Society B*, **7**, 155–161.

Gelman A (1996). "Inference and Monitoring Convergence." In WR Gilks, S Richardson, DJ Spiegelhalter (eds.), "Markov Chain Monte Carlo in Practice," pp. 131–144. New York, Chapman & Hall/CRC.

Good I, Crook J (1977). "The Enumeration of Arrays and a Generalization Related to Contingency Tables." *Discrete Mathematics*, **19**, 23–45.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2003a). ***ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks.* Statnet Project `http://statnetproject.org/`, Seattle, WA. R package version 2.0, URL `http://CRAN.R-project.org/package=ergm`.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2003b). ***statnet**: Software Tools for the Statistical Modeling of Network Data.* Statnet Project `http://statnetproject.org/`, Seattle, WA. R package version 2.0, URL `http://CRAN.R-project.org/package=statnet`.

Harrell Jr FE (2007). ***Hmisc**: Harrell Miscellaneous.* R package version 3.4-3., URL http://CRAN.R-project.org/package=Hmisc.

Kong A, Liu J, Wong W (1994). "Sequential Imputations and Bayesian Missing Data Problems." *Journal of the American Statistical Association*, **89**, 278–288.

Rao AR, Jana R, Bandyopadhyay S (1996). "A Markov Chain Monte Carlo Method for Generating Random $(0, 1)$ Matrices with Given Marginals." *Sankhya A*, **58**, 225–242.

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, Version 2.6.1, URL http://www.R-project.org/.

Roberts A, Stone L (1990). "Island-Sharing by Archipelago Species." *Oecologia*, **83**, 560–567.

Sanderson JG (2000). "Testing Ecological Patterns." *American Scientist*, **88**, 332–339.

Sanderson JG, Moulton MP, Selfridge RG (1998). "Null Matrices and the Analysis of Species Co-occurrences." *Oecologia*, **116**, 275–283.

Snijders TAB (1991). "Enumeration And Simulation Methods For 0-1 Matrices with Given Marginals." *Psychometrika*, **56**, 397–417.

Sulloway FJ (1982). "Darwin and His Finches: The Evolution of a Legend." *Journal of the History of Biology*, **15**, 1–53.

Thompson SK (2002). *Sampling.* John Wiley & Sons, Inc.

Tierney L, Rossini AJ, Li N, Sevcikova H (2007). ***snow**: Simple Network of Workstations.* R package version 0.2-9, URL http://CRAN.R-project.org/package=snow.

A. Sequential importance sampling for bipartite graphs

Let $\boldsymbol{r} = (r_1, r_2, ..., r_m)$ denote the row sums and $\boldsymbol{c} = (c_1, c_2, ..., c_n)$ the column sums of an $m \times n$ bipartite graph. Then we will denote the space of all graphs with column sums $\boldsymbol{c}$ by $\mathcal{A}_{\boldsymbol{c}}$, and we will denote the space of all graphs with row sums $\boldsymbol{r}$ and column sums $\boldsymbol{c}$ by $\mathcal{A}_{\boldsymbol{rc}}$. For most significance tests and for likelihood inference, we must generate new graphs that have the same marginals as the observed graph. Consequently, our focus will be on generating graphs in $\mathcal{A}_{\boldsymbol{rc}}$. For the graph in Table 4, $\mathcal{A}_{\boldsymbol{c}}$ is the space of all graphs with column sums $(3, 3, 3, 1)$, and $\mathcal{A}_{\boldsymbol{rc}}$ is the space of all graphs with row sums $(3, 3, 2, 2)$ and column sums $(3, 3, 3, 1)$. Suppose we want to generate new graphs with the same row and column sums as this graph. If we simply ensure that we sample according to the column sum constraints and the algorithm samples $(0\ 1\ 1\ 1)^\top$ for the first column, sampling $(0\ 1\ 1\ 1)^\top$ for the second column could produce a valid graph for $\mathcal{A}_{\boldsymbol{c}}$ but not for $\mathcal{A}_{\boldsymbol{rc}}$. This should be clear in that only two columns remain to be sampled, and yet we still need to sample three 1's for the first row in order to obtain a valid graph in $\mathcal{A}_{\boldsymbol{rc}}$. Since our goal is to sample new graphs in $\mathcal{A}_{\boldsymbol{rc}}$, we will need to address this problem.

A.1. Conjugate sequences

One possible remedy for this problem is to generate graphs in $\mathcal{A}_{\boldsymbol{c}}$ and simply discard those graphs that are not in $\mathcal{A}_{\boldsymbol{rc}}$. This is highly inefficient, however, so we consider a different approach and implement a forward-looking step to ensure that all sampled graphs are valid. To do this, we define the conjugate sequence of column sums $c_1, c_2, ..., c_n$ by $C_i^{(0)} = \#\{c_j : c_j \geq i\}$, $i = 1, 2, ..., m$, $j = 1, 2, ..., n$. Thus, for any given column sum c_j, we will increment $C_i^{(0)}$ by 1 for each j satisfying $1 \leq C_i^{(0)} \leq c_j$. This means that $C_1^{(0)}$ counts the number of non-zero column sums, $C_2^{(0)}$ counts the number of column sums that are at least two, $C_3^{(0)}$ counts the number of column sums that are at least three, and so on.

In general, we will let $\boldsymbol{C}^{(j)} = \left(C_1^{(j)}, ..., C_m^{(j)}\right)$ denote the conjugate sequence of $c_{j+1}, ..., c_n$. This represents the conjugate sequence after the first j columns have been sampled. For example, for the graph in Table 4, $\boldsymbol{C}^{(0)} = (4, 3, 3, 0)$, $\boldsymbol{C}^{(1)} = (3, 2, 2, 0)$, $\boldsymbol{C}^{(2)} = (2, 1, 1, 0)$, $\boldsymbol{C}^{(3)} = (1, 0, 0, 0)$, and $\boldsymbol{C}^{(4)} = (0, 0, 0, 0)$. By construction, $\boldsymbol{C}^{(0)}$, $\boldsymbol{C}^{(1)}$, ..., $\boldsymbol{C}^{(n)}$, are independent of the sampling mechanism for the columns, so all conjugate sequences can be computed prior to sampling any of the n columns. Also by construction, $C_i^{(0)} \geq C_i^{(1)} \geq \cdots \geq C_i^{(n)}$ for all i.

A.2. Knots and corresponding restrictions

For a given conjugate sequence, we can determine the maximum number of ones that can be

1	1	1	0	3
1	1	0	1	3
1	0	1	0	2
0	1	1	0	2
3	3	3	1	

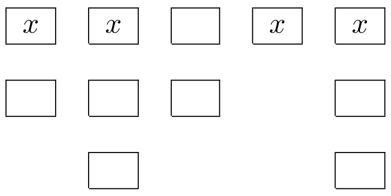
Table 4: First Example.

sampled in a specified set of columns for a given number of rows by computing partial sums from the conjugate sequence. In particular, $C_1^{(j)}$ gives the maximum number of 1's that can be sampled in columns $j+1$ to n for any one row, $\sum_{i=1}^{2} C_i^{(j)}$ gives the maximum number of 1's that can be sampled in columns $j+1$ to n for any two rows, and $\sum_{i=1}^{t} C_i^{(j)}$ gives the maximum number of 1's that can be sampled in columns $j+1$ to n for any t rows. Thus, for the graph in Table 4, $\boldsymbol{C}^{(1)} = (3, 2, 2, 0)$ tells us that, in the last three columns, we can sample no more than three 1's for any given row, no more than five 1's for any two rows, no more than seven 1's for any three rows, and no more than seven 1's for any four rows.

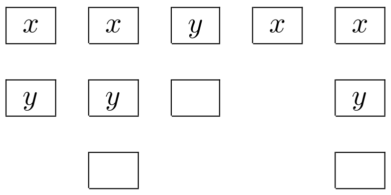
Likewise, the row sums tell us how many 1's must be sampled for a specific set of rows. Since rows with larger row sums will have greater restrictions in the sampling process, we rearrange the rows so that row sums are ordered from largest to smallest. Then $\sum_{i=1}^{t} r_i$ tells us the total number of 1's that must be sampled for the first t rows. Note that $\sum_{i=1}^{m} C_i^{(0)} = \sum_{j=1}^{n} c_j = \sum_{i=1}^{m} r_i$. Then, since $C_i^{(0)} \geq C_i^{(1)}$ for all i, we have $\sum_{i=1}^{m} r_i \geq \sum_{i=1}^{m} C_i^{(1)}$. In particular, if $\sum_{i=1}^{t} r_i > \sum_{i=1}^{t} C_i^{(1)}$, then at least $\sum_{i=1}^{t} r_i - C_i^{(1)}$ ones must be sampled in rows 1 to t of the first column. Failure to do so will result in a graph that fails to meet our marginal constraints, as the first t rows of the graph will require more ones than afforded by the remaining column sums of size t or less.

Motivation

To illustrate this, suppose $C_1^{(1)} = 5$, $C_2^{(1)} = 4$, and $C_3^{(1)} = 2$; and suppose $r_1 = 4$, $r_2 = 4$, and $r_3 = 4$. Then $\sum_{i=1}^{3} r_i > \sum_{i=1}^{3} C_i^{(1)}$, and $\sum_{i=1}^{3} r_i - C_i^{(1)} = 1$. Let x denote instances of a 1 being sampled for the first row, y denote instances of a 1 being sampled for the second row, and z denote instances of a 1 being sampled for the third row. If we fail to sample a 1 in the first column for any of these three rows, satisfying the row sum requirements for the first row requires that we sample 1's for four of the five remaining columns with non-zero column sums (i.e. $c_2, c_3, ..., c_m$), leaving unsampled only one of the columns with a non-zero column sum.



Satisfying the row sum requirements for the second row requires that we sample four 1's from either the column with a non-zero column sum that has yet to be sampled or the columns with column sums of at least two which have already been sampled once. This results in all columns with non-zero column sums being sampled at least once and all but one of the columns with column sums of at least two being sampled twice.



Finally, satisfying the row sum requirements for the third row requires that we sample four

1's for either the column with a column sum of at least two which has been sampled only once or the columns with column sums of at least three which have already been sampled twice. Here, we encounter a problem, as there are not enough column sums of at least three to accommodate the number of 1's required by this third row sum. Of course, we cannot sample a column twice for the same row either, so it becomes clear that the only solution is to sample a 1 for at least one of the first three rows in the first column.

$$\begin{array}{cccccc}
\boxed{x} & \boxed{x} & \boxed{y} & \boxed{x} & \boxed{x} & \\
\boxed{y} & \boxed{y} & \boxed{z} & & \boxed{y} & \\
 & \boxed{z} & & & \boxed{z} & z
\end{array}$$

Determination of knots and corresponding restrictions

To determine how many 1's must be sampled in certain rows for a given column, we record the row number t whenever $\sum_{i=1}^{t} r_i > \sum_{i=1}^{t} C_i^{(1)}$, and we also record the corresponding difference $\sum_{i=1}^{t} r_i - C_i^{(1)}$. Let $k_1, k_2, ...$ (which we will refer to as *knots*) take on the values of t for instances where $\sum_{i=1}^{t} r_i > \sum_{i=1}^{t} C_i^{(1)}$, and let $v_1, v_2, ...$ take on the corresponding differences $\sum_{i=1}^{t} r_i - C_i^{(1)}$. Thus, v_i tells us how many ones must be sampled by row k_i. If $v_j \leq v_i$ for some $j > i$, we will remove k_j and v_j, as the restrictions placed on our sampling through k_i and v_i ensure that the restrictions placed on our sampling through k_j and v_j are met. Likewise, if $v_j - v_i \geq k_j - k_i$ for any $j > i$, then we will remove k_i and v_i, as the restrictions placed on our sampling through k_j and v_j ensure that the restrictions placed on our sampling through k_i and v_i are met.

For the first column of the graph given in Table 4, we initially record the knots and corresponding restrictions However, k_1 and v_1 ensure that the restrictions stipulated by k_2 and v_2

$$\begin{aligned}
k_1 &= 2, \quad v_1 = 1 \\
k_2 &= 3, \quad v_2 = 1 \\
k_3 &= 4, \quad v_3 = 3.
\end{aligned}$$

are met, and k_3 and k_2 ensure that the restrictions stipulated by k_2 and v_2 are met, so we remove k_1 and v_1 and k_2 and v_2 and reorder all subsequent knots to obtain

$$k_1 = 4, \quad v_1 = 3.$$

A.3. Sampling a column

After we have recorded the knots $\boldsymbol{k} = (k_1, k_2, ...)$ and corresponding restrictions $\boldsymbol{v} = (v_1, v_2, ...)$ for a column, we can begin sampling for that column. Before providing a general rule for sampling a column, we will first demonstrate how sampling is executed for a simple example.

Example

To sample the first column for the graph in Table 4, we record the knots and corresponding restrictions for the first column. These are given by

$$k_1 = 4, \quad v_1 = 3.$$

With the knots and corresponding restrictions recorded, our first step is to sample 1's for rows 1 to $k_1 = 4$. Because $v_1 = 3$, we know that we must sample at least three of these rows to receive a 1. At the same time, we cannot sample more 1's than what the column sum allows, so we may not sample more than three rows to receive a 1 from the first two rows. Thus, we know that we must sample exactly three of the four rows to receive a 1, and we randomly select the three rows.

General column sampling procedure

In general, then, to sample column j we first determine d_1, the total number of 1's to be sampled for the first k_1 rows. As stated before, v_1 denotes the number of ones that must be sampled by row k_1, so this is our lower bound for d_1. At the same time, we cannot sample more 1's than rows, nor can we sample more 1's than what the column sum allows, so we are bounded above by the minimum of k_1 and c_j. Thus, we sample a value d_1 uniformly from $\{v_1, ..., \min\{k_1, c_j\}\}$. Once we have sampled a value for d_1, we sample the rows that are to receive a 1. The procedure for sampling the d_1 rows to receive a 1 is described later. Next, we uniformly sample a value d_2 from $\{\max\{v_2 - d_1, 0\}, \min\{k_2 - k_1, c_j - d_1\}\}$ to determine the number of 1's to be sampled for rows $k_1 + 1$ to k_2, and the row sampling procedure is repeated. This continues until we have either sampled c_j rows to receive ones or have reached our last knot.

Each time we sample a value d_i for a knot k_i, we compute inclusion probabilities for rows $k_{i-1} + 1$ to k_i, and then we randomly sample one of these rows to receive a 1 according to the inclusion probabilities. The inclusion probabilities are then updated for the unsampled rows, and we again randomly select one of these rows to receive a 1 according to the new inclusion probabilities. The procedure is repeated until we have sampled d_i rows.

A.4. Updating

After we have sampled the first column, we decrement by one the row sums for each row that was sampled and record these new row sums $\boldsymbol{r}^{(1)}$. Then we consider the conjugate sequence $\boldsymbol{C}^{(2)}$ and the new row sums and repeat the procedure of ordering row sums in decreasing order, determining knots, recording restrictions corresponding to each knot, and sampling according to these restrictions. In essence, we are sampling the first column for a new $m \times n - 1$ graph with row sums equal to $\boldsymbol{r}^{(1)}$.

A.5. Graph probability

As sampling is occurring, we update the probability of the new graph. Since sampling is not uniform, it is vital that we know the probability of generating the new graph that we observe. Columns are sampled sequentially and conditional on previous columns sampled, so the graph probability is given by the product of the conditional probabilities of the columns. For each column, this conditional probability is simply the product of the uniform probabilities used to choose d_i for each knot k_i and the probabilities of the rows that are sampled for each of those knots. Let $\mathcal{S}$ represent the rows $k_{i-1} + 1$ to k_i, the rows from which we will be sampling d_i times, and A_l $(l = 0, ..., d_i)$ the rows in $\mathcal{S}$ that have been sampled after d_i draws.

Then $\prod_{l=1}^{d_i} P\left(s_l, A_{l-1}^c\right)$ gives the probability of sampling row s_1, then row s_2, and so on. According to the distribution that we will use in computing inclusion probabilities, however, $\prod_{l=1}^{d_i} P\left(s_l, A_{l-1}^c\right)$ does not depend on the ordering of s_1 to s_{d_i}. Consequently, by computing the probability of the permutation that we observe, we can easily compute the probability of the combination of 1's and 0's that are sampled for a particular column, as this will simply be $d_i! \prod_{l=1}^{d_i} P\left(s_l, A_{l-1}^c\right)$.

A.6. Essential algorithmic considerations

Ideally, we would like to sample graphs uniformly. For SIS, this is rarely possible, but we can often sample graphs from a distribution that is nearly uniform. As mentioned previously, sampling graphs from a relative uniform distribution is vital in enabling us to accurately estimate the number of graphs meeting marginal constraints and to approximate the null distribution of a test statistic. Additionally, the effective sample size increases as the sampling distribution approaches a uniform distribution. To ensure that graphs are sampled according to a relative uniform distribution, columns should almost always be arranged in decreasing order by column sum, and sampling of rows to receive a 1 should be done according to the conditional Poisson distribution.

Effective sample size and squared coefficient of variation

The effective sample size gives a calculation of the equivalent uniform probability sample for the sample under consideration. Thus, if we sample N graphs uniformly, our effective sample size is simply N. Kong, Liu, and Wong (1994) show that, in the case of SIS, if we sample N graphs, the effective sample size is $\frac{N}{1+cv^2}$, where cv^2 is the square of the coefficient of variation of the standardized graph weights. If the graph probabilities are $p_1, p_2, ..., p_N$, then the standardized graph probabilities are given by $\frac{\frac{1}{p_1}N}{\sum_{i=1}^N \frac{1}{p_i}}, \frac{\frac{1}{p_2}N}{\sum_{i=1}^N \frac{1}{p_i}}, ..., \frac{\frac{1}{p_N}N}{\sum_{i=1}^N \frac{1}{p_i}}$, which have mean $\mu = 1$. Recall that the coefficient of variation is given by $cv = \frac{\sigma}{\mu}$, so, in the case of SIS, $cv^2 = \sigma^2$. This is approximated by the sample variance

$$\begin{aligned} s^2\left(\frac{\frac{1}{p_i}N}{\sum_{i=1}^N \frac{1}{p_i}}\right) &= \left(\frac{N}{\sum_{i=1}^N \frac{1}{p_i}}\right)^2 s^2\left(\frac{1}{p_i}\right) \\ &= \frac{\frac{1}{N-2}\sum_{i=1}^N \left[\frac{1}{p_i} - \frac{1}{N}\sum_{j=1}^N \frac{1}{p_j}\right]^2}{\left[\frac{1}{N}\sum_{j=1}^N \frac{1}{p_j}\right]^2}. \end{aligned}$$

In essence, cv^2 provides a measure of the distance between a uniform distribution and the SIS distribution, and $1 + cv^2$ measures the efficiency of the SIS distribution, relative to a uniform sampling distribution. To maximize the effective sample size, it is clear that we must minimize cv^2. Chen *et al.* (2005) argue that, in the case of zero-one tables, cv^2 is almost always minimized by rearranging columns so that the column sums are in decreasing order and by sampling rows to receive a 1 according to a conditional Poisson distribution.

Conditional Poisson distribution

The conditional Poisson distribution arises from the conditional distribution of a Poisson-binomial distribution. Borrowing from the notation of Chen *et al.* (2005), the Poisson-binomial distribution is given by a random variable $S_{\boldsymbol{Z}} = Z_1 + \cdots + Z_l$, where $\boldsymbol{Z} = (Z_1, ..., Z_l)$ denote Bernoulli trials with corresponding probability of success $\boldsymbol{p} = (p_1, ..., p_l)$. If we condition on $S_{\boldsymbol{Z}}$, the resulting distribution is the conditional Poisson distribution. Chen, Dempster, and Liu (1994) argue that sampling rows according to such a distribution is much more efficient than sampling rows uniformly. With the conditional Poisson distribution, rows are sampled without replacement with probabilities that are proportional to $\boldsymbol{r}/n$. Let $\mathcal{S}$ again represent the rows from which we will be sampling d_i times and A_l $(l = 0, ..., d_i)$ the rows in $\mathcal{S}$ that have been sampled after l draws. Then row t will be sampled on draw l with probability

$$P\left(t, A_{l-1}^c\right) = \frac{w_t \Psi\left(d_i - l, A_{l-1}^c \backslash t\right)}{(d_i - l + 1)\,\Psi\left(d_i - l + 1, A_{l-1}^c\right)},$$

where $w_t = \frac{r_t}{n - r_t}$ and Ψ is given by the recursive formula

$$\begin{aligned}\Psi(z, A) &= \sum_{B \subset A, |B| = z} \left(\prod_{i \in B} w_i\right) \\ &= \Psi(z, A\backslash\{z\}) + w_z \Psi(z - 1, A\backslash\{z\})\end{aligned}$$

This distribution has the nice property that $\prod_{l=1}^{d_i} P\left(s_l, A_{l-1}^c\right)$ does not depend on the ordering of s_1 to s_{d_i}, greatly simplifying the computation of the graph probability.

Affiliation:

Ryan Admiraal
Department of Statistics
University of Washington
Box 354322
Seattle WA 98195-4332, United States of America
Telephone: +1/206/543-8628
Fax: +1/206/221-6873
E-mail: ryan@stat.washington.edu
URL: http://www.stat.washington.edu/ryan/

Mark S. Handcock
Department of Statistics
University of Washington
Seattle WA 98195-4332, United States of America
E-mail: handcock@stat.washington.edu
URL: http://www.stat.washington.edu/handcock/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 24, Issue 8 *Submitted:* 2008-01-30
February 2008 *Accepted:* 2008-03-25