backbone: An R Package for Backbone Extraction of
Weighted Graphs

Rachel Domagalski
Zachary Neal
Bruce Sagan
December 29, 2019

Abstract
This paper describes how to use the R package backbone for extracting the backbone of a weighted graph. The analysis and visualization of weighted graphs are inherently challenging, and thus it is often of use to only examine edges which are deemed significant. There are several methods for extracting from a weighted graph an unweighted or signed backbone which retains only the significant edges. The backbone package implements four different backbone methods, three of which consider weighted graphs that arise from bipartite projections. This paper describes how to use each of the package’s functions and the algorithms they employ.

Introduction to Social Network Analysis

Social Networks

Social network analysis records information on social interactions as social networks, then analyzes these networks using tools from graph theory to understand how the interactions influence social life. Social networks consist of vertices, the members of the network, and edges, the relationships between them, however there is significant diversity in the types of vertices and edges represented in social networks. These vertices may represent humans with edges between them indicating friendship. They could be businesses and directional edges indicating how information is diffused. The vertices could be cities and edges could be spread of disease through the community. To understand these complex relationships, social network analysis studies patterns of relations through mathematics and statistics [Scott and Carrington, 2014].

One challenge in analyzing social networks is that social relationships vary in strength, leading many social networks to be both dense and complex. In a particular social context, each individual may interact with all others at least to some extent, however some pairs may interact more intensely or frequently than others. For example, in a classroom context, some pairs of students might work together on a project, but other pairs might have been close acquaintances for years. To capture these different intensities of a relationship, we can assign a weight to each edge, where a larger weight describes a stronger relationship in terms of intensity, frequency, or duration, while a lower weight describes a weaker one.

When edges have a weight, we call the network “weighted”. Although these weights encode important information, they also make it more difficult to both visualize and analyze the network. In such cases, it is often of interest to instead focus only on the most “important” edges of a network. We call this reduced network of important edges the backbone. This extracted backbone can either have unweighted edges (i.e. an edge is either present or absent) or signed edges, in which a +1 edge indicates a positive relationship between two vertices and a −1 edge indicates a negative relationship between them.

Weighted graphs often arise from bipartite projections of participation data. In a bipartite projection, an edge between two vertices represent how many events the two have participated in together. Many different methods of backbone extraction exist for bipartite projections, but until now there has not been a single software package that implements the most common methods. The backbone R package brings these methods into a common framework so that researchers can easily use various backbone extraction methods on their own data and compare the results. Currently the package implements four backbone methods: a universal threshold
model, a hypergeometric model, a stochastic degree sequence model, and a fixed degree sequence model. It can be installed from CRAN via `install.packages("backbone")` and used with `library(backbone)`. This manuscript is based on version `backbone 1.1.0`. For further information regarding the CRAN distribution, please visit [https://CRAN.R-project.org/package=backbone](https://CRAN.R-project.org/package=backbone). For additional materials relating to `backbone`, including papers, presentations, and data sets, please visit [http://www.zacharyneal.com/backbone](http://www.zacharyneal.com/backbone).

## Graph Theory

Before discussing methods of backbone extraction that are included in `backbone`, we first review some elementary graph theory to provide a background for the underlying mathematical structures that are used in social network analysis.

A graph $G$ is a set of objects called vertices, and 2-element subsets of the vertices which are called edges. An edge between vertices $i$ and $j$ can be denoted as $e = ij$. Each graph has both a visual representation which can be drawn in the plane, and can be described via an adjacency matrix. The adjacency matrix of an unweighted graph $G$ with $n$ vertices is an $n \times n$ matrix $G = [G_{ij}]$ where $G_{ij} = 1$ if an edge is present between vertex $i$ and vertex $j$, or $G_{ij} = 0$ if that edge is absent [Chartrand et al., 2010]. In some cases, a graph can also be signed, in which case $G_{ij} = 1$ if a positive edge is present between vertex $i$ and vertex $j$, $G_{ij} = -1$ if a negative edge is present, or $G_{ij} = 0$ if the edge is absent.

We call a graph weighted if each edge has an associated numeric value, and unweighted otherwise. In a weighted graph, the value of an edge may capture such properties as the strength or capacity of the edge. In this case, $G_{ij} = w(ij)$ where $w(ij)$ is the weight of the edge $e = ij$. Although weighted graphs contain a large amount of information, there are some cases (e.g., visualization, application of statistical models not developed for weighted graphs) where it is useful to reduce this information by focusing on an unweighted subgraph that contains only the most important edges. We call this subgraph the backbone of $G$, which we denote as $G'$.

Extracting $G'$ from $G$ requires deciding which edges to preserve. This usually involves selecting a threshold $T_{ij}$ such that edges are preserved if they are strictly above the threshold and omitted if they are below it, i.e., $G_{ij}' = 1$ or $G_{ij}' = 0$ depending on whether $G_{ij} > T_{ij}$ or $G_{ij} \leq T_{ij}$, respectively. It is also possible to extract a signed backbone by selecting distinct upper $T_{ij}^+$ and lower $T_{ij}^-$ thresholds, $T_{ij}^- < T_{ij}^+$, such that

$$G_{ij}' = \begin{cases} 
1 & \text{if } G_{ij} > T_{ij}^+ \\
-1 & \text{if } G_{ij} < T_{ij}^- \\
0 & \text{if } T_{ij}^- \leq G_{ij} \leq T_{ij}^+
\end{cases}$$

The key to all backbone extraction methods lies in the selection of $T$. The backbone package provides several different methods for selecting $T$ and thus extracting $G'$ from $G$.

While weighted graphs can be collected from social situations in a variety of ways, `backbone 1.1.0` is focused on weighted graphs that arise from bipartite projections. Bipartite projections are a way to generate a weighted graph from bipartite graphs. We call a graph bipartite if the set of vertices can be partitioned into two sets $U$ and $W$ such that each edge of the graph is of the form $ij$ where $i \in U$ and $j \in W$. The sets $U$ and $W$ are called independent, meaning there are no edges present within the sets. These graphs can be represented by a bi-adjacency matrix, $B$. This matrix uses rows to represent vertices in the set $U$ and columns to represent vertices in the set $W$. We set $B_{ij} = 1$ if there is an edge between vertex $i$ of $U$ and vertex $j$ of $W$, and set $B_{ij} = 0$ otherwise.

We can use bipartite graphs to study social networks where the vertices of $U$ are people, the vertices of $W$ are events, and edges represent participation of a person in an event. These bipartite networks are often also called affiliation networks or two-mode networks. To transform this data into a weighted graph, we project the bipartite adjacency matrix $B$ by multiplying it by its transpose $B^T$. This produces a weighted graph $G = BB^T$ called the bipartite projection, where $G_{ij}$ is equal to the number of events of $W$ in which both $i$ and $j$ participate.
and \( j \) participated, when \( i \neq j \). The value \( G_{ii} \) is equal to the total number of events in which \( i \) participated. Let \( M = \min(G_{ii}, G_{jj}) - (|W| - \max(G_{ii}, G_{jj})) \). The value of each off-diagonal entry \( G_{ij} \) is bounded by

\[
\max(0, M) \leq G_{ij} \leq \min(G_{ii}, G_{jj}).
\]

Bipartite projections are of interest in social network analysis because they allow us to infer a weighted network structure through event participation which is often easier to obtain than taking a survey of the network members. If the number of members of the network is quite large, getting complete and reliable information regarding relationships between members can involve long and repetitive survey techniques which can lead to survey fatigue and costly field work. Additionally, in some settings, individuals may be reluctant to provide information about their relationships. In these cases, it is often beneficial and easier to collect bipartite network information, such as event attendance, then project to infer a weighted network between the social figures.

**Example Data: The 115th Session of the United States Senate**

We outline the use of the backbone package with a network of the 115th session of the United States Senate, consisting of 105 senators and the 3665 bills that they have sponsored [govinfo,]. This data takes the form of a bipartite graph \( B \), the two independent sets of vertices are the senators and the bills. Each edge connects one senator to one bill. Specifically, \( B_{ij} = 1 \) if senator \( i \) sponsored bill \( j \), and otherwise is 0. Below we examine the data set. Notice that the row names correspond to each senator (including their party affiliation and the state they represent) and the column names refer to the bill number.

```r
library(backbone)
senate <- read.csv("senate115.csv", row.names = 1, header = TRUE, check.names = FALSE)
senate <- as.matrix(senate)
dim(senate) #Notice there are 105 senators and 3665 bills

# [1] 105 3665
senate[1:5, 1:10] #view a subset of the dataset

# Alexander  0  1 100 1000 1001 1002 1003 1004 1005 1006 1007
# Baldwin    0  0  0  0  0  0  0  1  0  1  0
# Barrasso   0  0  0  1  0  0  0  0  0  0  0
# Bennet     0  0  0  1  1  0  0  0  1  0  0
# Blumenthal 0  0  0  1  0  0  0  0  0  1  0

A weighted graph \( G \) can be constructed from \( B \) via bipartite projection, where \( G = BB^T \) and \( G_{ij} \) contains the number of bills that both senator \( i \) and senator \( j \) sponsored.

\( G <- \text{as.matrix}(\text{senate}) \) 
\( \text{dim}(G) \ #\text{Notice the graph is now 105 by 105} \)

# [1] 105 105
G[1:5, 1:5]

|       | Alexander | Baldwin | Barrasso | Bennet | Blumenthal |
|-------|-----------|---------|----------|--------|------------|
| Alexander | 98       | 22      | 21       | 21     | 25         |
| Baldwin   | 22       | 476     | 21       | 115    | 242        |
| Barrasso  | 21       | 21      | 142      | 16     | 19         |
| Bennet    | 21       | 115     | 16       | 280    | 128        |
| Blumenthal| 25       | 242     | 19       | 128    | 564        |
The projected graph $G$ now indicates that Senator Lamar Alexander sponsored a total of 98 bills in the 115th session. Among these 98 bills, 22 were co-sponsored by Senator Tammy Baldwin, and 21 were co-sponsored by Senator John Barrasso.

We can use the values of graph $G$ to observe differences between those with similar or dissimilar ideology. Below, we compare the number of bills co-sponsored by two individuals with similar political ideology, Senators Kamala Harris and Elizabeth Warren, versus those with dissimilar ideology, Senators Ted Cruz and Bernie Sanders. The results are consistent with the expectation that legislators sharing a similar ideology engage in more co-sponsored bills.

\[
G["Sen. Harris, Kamala D. [D-CA]", "Sen. Warren, Elizabeth [D-MA"]
\]

## [1] 189

\[
G["Sen. Cruz, Ted [R-TX]", "Sen. Sanders, Bernard [I-VT"]
\]

## [1] 2

The differences in the number of bills co-sponsored prompts an important underlying question: how many bills do two senators have to co-sponsor before we can consider them to be in a political alliance? How few do they have to co-sponsor before we consider them to be avoiding each other? These questions are what the backbone package seeks to answer.

We will demonstrate how to use the backbone package to extract the backbone of $G$, which involves deciding whether to preserve an edge between each pair of senators in the network. This data set provides insight into how the backbone package works because, based on media accounts of the current US political climate, we have some a priori expectations about what structure a properly extracted backbone should have. Specifically, given conditions of partisanship and polarization, we should expect to see positive relationships form primarily between those in the same political party. When we use different backbone methods, we will visualize this network by depicting the Republican senators by red vertices, Democratic senators by blue vertices, and Independent senators by green vertices. We discuss signed backbones in the text, but for visual clarity only visualize binary backbones that contain positive edges. Positive relations of collaboration between pairs of senators will be shown by colored edges with the following color scheme: red for two Republicans, blue for two Democrats, and purple for a Republican and a Democrat.

General Backbone Methods

In this section, we will describe backbone methods that can be applied to any weighted graph, whether the weights are present in a natively unipartite graph, or are the product of a bipartite projection (as is the case in our example data). General backbone methods reduce weighted graphs to either unweighted or signed graphs which are easier to analyze and depict.

Universal Backbone: `universal()`

The simplest approach to backbone extraction applies a single threshold $T$ to all edges. Generally, a threshold value of 0 is usually used to indicate that as long as a pair of vertices have participated in at least one event, or have a nonzero edge weight, their relationship should be counted. However, this can lead to extremely dense and uninformative networks, as we will show with our example data. In any case, if using a single threshold value $T$ is desired, this can be done in the backbone package by using the `universal()` function. The `universal()` function allows the user to extract an unweighted backbone by selecting a single threshold $T$, or extract a signed backbone by selecting upper and lower thresholds $T$ and $T'$. The function `universal()` returns the backbone matrix, a signed (or unweighted) adjacency matrix of a graph. It has a variety of different uses which are demonstrated in the following examples. Using the `senate` data set, if we input the projected matrix $G \leftarrow \text{senate} \% \% \text{t(senate)}$, we can use the universal threshold
on the weighted matrix $G$. If we set an upper threshold of 0, then if two senators have cosponsored one or more bills, there will be an edge between the two. By only designating an upper threshold, the `universal()` function returns an unweighted graph. A signed graph is returned only if a lower threshold is also supplied. We plot our upper threshold of 0 using the igraph package. Note that the resulting graph is extremely dense as only 146 pairs of senators out of the total 5460 number of pairs have not sponsored a bill together, see fig. [I]. The universal threshold backbone is uninformative about the underlying structure of the network.

```r
G <- senate %*% t(senate)  #projected senate dataset, a weighted graph
universal_bb1 <- universal(G, upper = 0)  #if they sponsored at least one bill together
universal_bb1$backbone[1:5,1:5]
```

| # | Alexander | Baldwin | Barrasso | Bennet | Blumenthal |
|---|-----------|---------|----------|--------|------------|
| # Alexander | 0 | 1 | 1 | 1 | 1 |
| # Baldwin | 1 | 0 | 1 | 1 | 1 |
| # Barrasso | 1 | 1 | 0 | 1 | 1 |
| # Bennet | 1 | 1 | 1 | 0 | 1 |
| # Blumenthal | 1 | 1 | 1 | 1 | 0 |

Figure 1: The positive backbone of the US Senate co-sponsorship network with edges retained between two senators if they sponsored at least 1 bill together.

We can also use the `universal()` function on the original bipartite data. When inputting bipartite data, we set parameter `bipartite = TRUE`. The bipartite matrix will be multiplied by its transpose before the threshold is applied. Below, we input the bipartite matrix with the same threshold values as before, returning the same backbone matrix.

```r
universal_bb2 <- universal(senate, upper = 0, bipartite = TRUE)  
#Note both matrices return the same results.
all.equal(universal_bb1$backbone, universal_bb2$backbone)
```

## [1] TRUE

To create a signed backbone, we can apply both an upper and lower threshold value. For instance, we could choose to retain a positive edge if the senators cosponsored at least 100 bills together, and a negative edge if they cosponsored less than 50 bills together. We can do this with the following code. Note that the returned backbone matrix now has both +1 and −1 values.
We can also define a threshold as a function of the edge weights, such as the mean. Any $G_{ij}$ values above the upper threshold are counted as a positive +1 value in the backbone, and any below the lower threshold are counted as a negative −1 value in the backbone. The following code will return a backbone where the positive edges indicate two senators cosponsored more than 1 standard deviation above the mean number of bills and negative edges indicate two senators cosponsored less than 1 standard deviation below the mean number of bills. The graph of the positive edges of this backbone can be seen in fig. 2.

Using the code just given, the senate matrix would first be projected. Then, the standard deviation of the $G_{ij}$ entries is calculated and added to (or subtracted from) to the mean of the $G_{ij}$ values. This value is then used as a threshold of the threshold the projected matrix for the positive (or negative) entries.

Although the application of a universal threshold to extract a backbone network does reduce the complex nature of the original weighted graph, it does not necessarily help with the understanding of the underlying structure of the graph. To obtain meaningfully sparse graphs, we require a principled method of selecting the threshold, and we must allow the threshold to vary for different edges. To improve our backbone results, we move to methods of bipartite projections.

### Bipartite Projection Backbone Methods

The methods described above can be applied to any weighted graph $G$. In this section we describe methods that are designed for weighted graphs $G$ that are the result of bipartite projections, specifically $G = BB^T$ for bipartite graph $B$ [Neal, 2014]. They differ from other methods because they take into account the
information contained in the original bipartite graph $B$. These methods are conditioned on the bipartite graph’s two degree sequences: the row vertex degrees and the column vertex degrees which are called the row and column marginals, respectively. Let $\mathcal{R}$ be a set of restrictions on the row and column marginals and $\mathcal{B}(\mathcal{R})$ the space of all bipartite graphs which satisfy those conditions. We compare the values of the bipartite projection $G_{ij}$ to the probability distributions that describe $(BB^T)_{ij}$ for all bipartite graphs $B \in \mathcal{B}(\mathcal{R})$.

When the probability distribution that describes the null model $\mathcal{B}(\mathcal{R})$ is unknown for a certain $\mathcal{R}$, we construct a distribution via the following algorithm:

1. Construct a bipartite graph $B^*$ that represents a random draw from $\mathcal{B}(\mathcal{R})$.
2. Project $B^*$ (i.e. $B^*B^T$) to obtain a random weighted bipartite projection $G^*$.
3. Repeat steps 1 and 2 $N$ times to sample the space of possible $G^*_ij$.
4. Compare $G_{ij}$ to the observed $G^*_ij$ as follows: Define a backbone $G'$ such that

$$G'_{ij} = \begin{cases} 1 & \text{if } |\{G^*_ij : G^*_ij \geq G_{ij}\}| \leq \alpha N \\ 0 & \text{otherwise} \end{cases}$$

Or define a signed backbone $G'$ such that

$$G'_{ij} = \begin{cases} 1 & \text{if } |\{G^*_ij : G^*_ij \geq G_{ij}\}| \leq (\alpha/2)N \\ -1 & \text{if } |\{G^*_ij : G^*_ij \leq G_{ij}\}| \leq (\alpha/2)N \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha$ is a significance level.

We will refer to the original bipartite graph as $B$, with projection $G = BB^T$, while a generated bipartite matrix will be referred to as $B^*$ with projection $G^* = B^*B^T$. The backbone package implements three ways to count the proportion of times $G^*_ij$ was larger or smaller than $G_{ij}$: the hypergeometric distribution using `hyperg()`, the fixed degree sequence model using `fdsm()`, and the stochastic degree sequence model using `sdsm()`. The hypergeometric null model imposes fixed row marginals on $\mathcal{B}(\mathcal{R})$ but allows the column marginals to vary. The fixed degree sequence model imposed fixed row and column marginals on $\mathcal{B}(\mathcal{R})$. Finally, the stochastic degree sequence model imposes the restriction on $\mathcal{B}(\mathcal{R})$ that row and column marginals are roughly the same as those of $B$. This is computed via the Poisson binomial distribution.

After one of these three functions are used to compute the proportions, the backbone can then be extracted for a given $\alpha$ level using the `backbone.extract()` function. We first describe `backbone.extract()`, then illustrate its use in the context of `hyperg()`, `fdsm()`, and `sdsm()`.

### Extracting the Backbone: backbone.extract()

The `hyperg()`, `fdsm()`, and `sdsm()` functions return two matrices: a positive matrix containing the proportion of times $G^*_ij$ was larger than $G_{ij}$, and a negative matrix containing the proportion of times $G^*_ij$ was smaller than $G_{ij}$. The `backbone.extract()` function allows the user to take these positive and negative matrices and return an unweighted or signed backbone. The `backbone.extract()` function has three parameters: two matrices, positive and negative, returned by one of the functions described above (i.e. `hyperg`, `fdsm`, or `sdsm`), and a significant test value $\alpha$.

One can adjust the precision of the significance test, $\alpha$, to refine the backbone results. The value of $\alpha$ should be between 0 and 1. If only the positive matrix is supplied to the function (i.e. $\alpha$ = NULL), then the $\alpha$ value is equal to the user’s input, and the statistical test is one-tailed yielding an unweighted backbone. If the negative matrix is also supplied to the function, the $\alpha$ value is equal to the user’s input divided by two, and the statistical test is two-tailed yielding a signed backbone.

If an entry in the positive matrix is greater than the $\alpha$ value, it is considered a +1 edge in the backbone. If an entry in the negative matrix is greater than the $\alpha$ value, it is considered a -1 edge in the backbone.
All other values are 0 in the backbone matrix. We demonstrate this function’s use with each of the bipartite projection methods.

**Hypergeometric Backbone: hyperg()**

The `hyperg()` function compares an edge’s observed weight, $G_{ij}$, to the distribution of weights expected in a projection obtained from a random bipartite network where the row vertex degrees are fixed, but the column vertex degrees are allowed to vary. This method of backbone extraction was developed in [Neal, 2013], which showed that the distribution of $G_{ij}^* = (B^*B^{*T})_{ij}$ when only row marginals are fixed is given by the hypergeometric distribution.

The hypergeometric distribution is a discrete probability distribution that models the probability of having $k$ “successes” in a random sample of size $n$ (without replacement) from a population of size $N$, where a total of $K$ objects are classified as a success. The probability mass function given by

$$Pr(X = k) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}}$$

where $X$ is a random variable, characterizes the distribution.

In the case of a bipartite projection, the population $N$ is the total number of events that individuals $i$ and $j$ could participate in, thus $N$ is equal to the total number of columns of $B$, which we will denote $C$. The “successes” are the number of events that both $i$ and $j$ participate in, which we will denote $E$. From the population, individual $i$ participates in $G_{ii}$ events, and individual $j$ participated in $G_{jj}$ events, providing the number of samples. The probability that $i$ and $j$ participate in exactly $E$ events is now given by

$$P(G_{ij} = E) = \frac{\binom{C}{E} \binom{C-E}{G_{ii}-E} \binom{C-G_{ii}}{G_{jj}-E}}{\binom{G_{ii}}{G_{jj}}}. $$

This method ensures that both $i$ and $j$ are still participating in the same number of events we’ve observed them to, but the number of individuals that participate in each event may vary. From the probability mass function we can find the probability of $i$ and $j$ participating in at least $G_{ij}$ events (for positive backbone edges) or at most $G_{ij}$ events (for negative backbone edges).

Specifically, for our example, the hypergeometric function will fix the number of bills that each senator sponsors, while allowing each bill to be sponsored by a varying number of senators. The function will compute the probability of each senator sponsoring the observed number of bills when the bills in which they sponsor were chosen randomly. These probabilities are returned to the user in the variables `positive` and `negative` where `positive` is the matrix of the proportion of times $G_{ij}$ is equal to or above the corresponding entry in $G^*$, and `negative` is the matrix of the proportion of times $G_{ij}$ is equal to or below the corresponding entry in $G^*$.

Following the `hyperg()` function, the user must use the `backbone.extract()` function to find the backbone at a given significance value `alpha`.

```r
hyperg_probs <- hyperg(senate)
```

```r
# Finding the Backbone using Hypergeometric Distribution
hyperg_bb <- backbone.extract(hyperg_probs$positive, hyperg_probs$negative, alpha = .01)
```

We can now examine how this method has changed the appearance of our network, focusing only on the positive edges of the signed backbone in fig. [3] We can see that the hypergeometric function has reduced the density of our network and that we begin to see some of the two party structure that is inherent in the United States senate.
The Fixed Degree Sequence Model: \textsc{fdsm}( )

The fixed degree sequence model compares an edge's observed weight, \( G_{ij} \), to the distribution of weights expected in a projection obtained from a random bipartite network where both the row vertex degrees and column vertex degrees are fixed. This method of backbone extraction was developed in \cite{Zweig:2011}, however the challenge lies in randomly sampling from the space of \( B^* \) with fixed degree sequences.

One method of sampling from a probability distribution is using a Monte Carlo Markov Chain method. A Markov chain is a way to move between elements of a set \( S \), where when currently at state \( x \in S \), the next move is determined by a probability distribution. At each state of the Markov chain, the probability of moving from one state to another only depends on the current state, and all moves before have no impact. These Markov chains describe a sequence of random variables that a process moves through. In a Monte Carlo Markov Chain, MCMC, a series of Markov chains are developed. Specifically, a starting point is chosen from the sample space \( S \), and then a next spot is realized with some given probability. Now starting at this new spot, we continue to pick a next spot in the sample space. We describe this sequence of samples as a random walk. At each step, we have the choice to make the step, or stay in our position \cite{Levin:2017}.

The \textsc{fdsm()} function uses the curveball algorithm \cite{Strona:2014}, which is proven to randomly sample from the space of \( B^* \) with fixed degree sequences \cite{Carstens:2015} and is an efficient way to do Monte Carlo Markov Chain sampling. The curveball algorithm works as follows:

1. First, the biadjacency matrix \( B \) is written as lists of indices. The index \( B_i \) corresponds to the column index of the 1’s in row \( i \).
2. Then, two of these rows are selected at random. A new list \( B_{i-j} \) is constructed to compare them where \( B_{i-j} \) is the indices that are in \( B_i \) and not in \( B_j \).
3. Let \( L_i \) be a list containing the indices of \( B_i \) minus those in \( B_{i-j} \). To \( L_i \) we add \( |L_i| \) elements randomly chosen from \( S = B_{i-j} \cup B_{j-i} \). The elements chosen are removed from \( S \).
4. Combine \( B_j \setminus B_{j-i} \) with remaining elements of \( S = B_{i-j} \cup B_{j-i} \) to form \( L_j \).
5. Now \( L_i \) and \( L_j \) define the rows \( i \) and \( j \) in the biadjacency matrix. Note that it is possible that no changes occurred.
6. Repeat a set number of times to create a new matrix \( B^* \).

This algorithm allows for multiple (or no) changes at each iteration. Since each change keeps the row and column degrees constant, these indices are fixed throughout the entire algorithm, allowing for sampling of
the space of all networks with fixed row and column degree. Strona et. al. [Strona et al., 2014] wrote the
curveball algorithm for R and it is their algorithm that the \texttt{fdsm()} function uses.

The fixed degree sequence model first constructs a random bipartite graph $B^*$ that preserves (to varying
extents, depending on the method) one or both degree sequences [Strona et al., 2018] using the curveball
algorithm. This bipartite graph $B^*$ is then projected (i.e. $B^*B^*$) to obtain a random weighted bipartite
projection $G^*$. These two steps are repeated a number of times to sample the space of possible $G^*_{ij}$. At each
iteration, we compare $G_{ij}$ to the value of $G^*_{ij}$ and keep record of how often it was above, below, or equal
to the generated value. The \texttt{fdsm()} function returns a list containing a matrix \texttt{positive} of the proportion
times $G_{ij}$ is equal to or above the corresponding entry in $G^*$, and a matrix \texttt{negative} containing the
proportion of times $G_{ij}$ is equal to or below the corresponding entry in $G^*$.

The function can also save each value of $G^*_{ij}$ for a given $i,j$. This is useful for visualizing an example of the
empirical null edge weight distribution generated by the model. The values $i,j$ correspond to the row and
column indices of a cell in the projected matrix and can be inputted as either numeric values or a string
containing the row names. These values are returned in variable \texttt{dyad_values}.

Using the fixed degree sequence model on the senate data set will allow us to compare our observed values to
a distribution where each senator sponsors the exact same number of bills and each bill is sponsored by the
exact same number of people. We can find the backbone using the fixed degree sequence model as follows:

```
# Finding the Backbone using Curveball FDSM

fdsm_props <- fdsm(senate, trials = 1000,
                   dyad=c("Sen. Harris, Kamala D. [D-CA]",
                          "Sen. Warren, Elizabeth [D-MA]"))
```

```
## Finding the Backbone using Curveball FDSM
## Estimated time to complete is 119 secs

hist(fdsm_props$dyad_values, freq = TRUE, breaks = 1000,
     col = 4, xlab = "Number of Co-Sponsorships",
     ylab = "Frequency",
     main = "Expected co-sponsorships between Harris and Warren using FDSM")
```

The \texttt{fdsm_props$dyad_values} output is a list of the $G^*_{ij}$ values for each of the 1000 trials, where $i$ =
“Sen. Harris, Kamala D. [D-CA]” and $j$ = “Sen. Warren, Elizabeth [D-MA]”. These values correspond to
the number of bills Senators Harris and Warren would be expected to co-sponsor if: (a) the number of bills
sponsored by Senator Harris was fixed, (b) the number of bills sponsored by Senator Warren was fixed,
and (c) the number of senators sponsoring each bill was fixed. We can compare their actual number of
coo-sponsorships, 189, to what is generated under our null model. We can view a histogram of the expected
coo-sponsorships generated in each of the 1000 trials in fig. 4.

To extract the backbone, we supply the \texttt{backbone.extract()} function with the proportion matrices \texttt{positive}
and \texttt{negative}.

```
fdsm_bb <- backbone.extract(fdsm_props$positive, fdsm_props$negative, alpha = 0.05)
```

Because we have provided both a \texttt{positive} and \texttt{negative} matrix, \texttt{backbone.extract()} returns a signed
backbone matrix by conducting a two-tailed significance test in which \texttt{alpha} is 0.025 on each end of the
distribution. Using the fixed degree sequence model has allowed us to see more of the partisan structure we
assume to be present in the United States Senate, which is visible in fig. 5.

The Stochastic Degree Sequence Model: \texttt{sdsm()} 

The stochastic degree sequence model (sdsm) compares an edge’s observed weight, $G_{ij}$ to the distribution
of weights expected in a projection obtained from a random bipartite network where both the row vertex
degrees and column vertex degrees are \textit{approximately} fixed. That is, the \textit{expected} row vertex degrees and
column vertex degrees in $B^*$ are fixed, but the actual values in a specific instance of $B^*$ may be higher or
Figure 4: A histogram of the expected co-sponsorships between Senators Kamala Harris and Elizabeth Warren. This plot can be used to compare the actual number of co-sponsorships, 189, to what is expected under the fixed degree sequence model.

This method of backbone extraction was developed in [Neal, 2014]. The distribution of graphs in $\mathcal{B}(\mathcal{R})$ that satisfy these row and column restrictions is given by the Poisson binomial distribution.

The Poisson Binomial distribution is a discrete probability distribution. Specifically, it is the distribution of a sum of independent Bernoulli trials that do not have to be identically distributed. A Bernoulli trial is a random experiment with exactly two outcomes, often referred to as “success” and “failure”, or “yes” and “no”. Thus the Poisson Binomial distribution is modeling the probability of getting some $k$ successes in $n$ success/fail trials, where each trial has a specific probability for success. The Poisson Binomial distribution can be described by its probability mass function which is given by

$$Pr(X = k) = \sum_{A \in F_k} \prod_{i \in A} p_i \prod_{j \in A^c} (1 - p_j)$$

where $F_k$ is the set containing all $k$ element subsets of $\{1, 2, \ldots, n\}$, $A$ is a set of $F_k$, and $A^c$ is the complement of $A$ in $\{1, 2, \ldots, n\}$ [Hong, 2013].

The first step in the sdsm model is to fit a binary regression model which estimates the probability for $B_{ij}$ to equal 1. These probabilities will allow us to find the $p_i$’s, $i = 1, 2, \ldots, n$ used in the Poisson Binomial probability mass function. In $\text{sdsm}()$, the $\beta$ parameters in $P_{ij} = Pr(B_{ij} = 1) = \beta_0 + \beta_1 B_i + \beta_2 B_j + \beta_3 (B_i \times B_j)$ are estimated using a binary regression with user chosen link function (e.g. logit, probit, complementary log-log, etc.), where $B_i$ and $B_j$ are the row vertex and column vertex degrees in $B$, respectively. The user can specify which link function to use when calling $\text{sdsm}()$ by using parameter $\text{model}$.

After the $\beta$ parameters are estimated, we can compute the expected probability that $B_{ij}$ is equal 1, then use these computed probabilities $P_{ij}$ in the Poisson binomial probability mass function. Specifically, we obtain a
$n \times m$ matrix $P$ of probabilities where each entry $0 \leq P_{ij} \leq 1$ is the probability of edge $B_{ij} = 1$. We wish to compare our observed edge weights of $G = BB^T$ to a distribution of weights in a graph $G^* = B^* B^{*T}$ where $B^*$ is selected from the set of all $n \times m$ 0-1 matrices where each entry of $B^*$ is picked independently with $P(B_{ij}^* = 1) = P_{ij}$. The distribution of $G_{ij}^*$ values is the Poisson Binomial distribution with parameters $p_1 = P_{11} P_{12}, \ldots, p_n = P_{n1} P_{nn}$.

The $\text{sdsm()}$ function returns the probability of edge weight $G_{ij}^*$ being equal to or above the observed value $G_{ij}$ in the output $\text{positive}$, and the probability of the edge weight $G_{ij}^*$ being equal to or below the observed value $G_{ij}$ in the output $\text{negative}$. These values can then be used in the $\text{backbone.extract()}$ function to extract a backbone.

In the context of the senate co-sponsorship matrix, the stochastic degree sequence model will compare our observed values to a distribution where each senator sponsors roughly the same number of bills, and each bill is sponsored by roughly the same number of people.

sdsm_rna <- sdsm(senate)
## Finding the Backbone using Poisson Binomial SDSM with alpha = 0.05 and tolerance = 0

Notice that the function provides a message indicating an alpha and tolerance value. The $\text{sdsm()}$ function uses the $\text{poibin}$ package to approximate the cumulative distribution function under the Poisson Binomial distribution [Hong, 2013]. The function uses the Refined Normal Approximation to first approximate the probability. Then, if the computed value lies between $\text{alpha-tolerance}$ and $\text{alpha+tolerance}$, the probability is computed exactly using the Discrete Fourier Transform method. The alpha value, defaulted at 0.05, is a proposed significance test value. The tolerance is set to 0 by default but can be changed by the user. If tolerance is set equal to 1, the Discrete Fourier Transform method will be used for every value of the matrices $\text{positive}$ and $\text{negative}$. Below, we use the Discrete Fourier Transform method and use the probit link function.

sdsm_dft <- sdsm(senate, model = "probit", tolerance = 1)

After computing the probabilities using the stochastic degree sequence model, we can use the
backbone.extract() function to find the backbone. We recommend using the same alpha value in the call to sdsm() that is used in backbone.extract().

sdsm_bb <- backbone.extract(sdsm_rna$positive, alpha = 0.05)

We have provided only a $positive matrix, backbone.extract() returns a unweighted backbone matrix by conducting a one-tailed significance test in which alpha is 0.05. Again, we are able to see more of the partisan structure that is suggested to be present in the US Senate in fig. 6, and this visualization provides more information than the extremely dense graphs found using a universal threshold.

The Power of backbone

Using the backbone extraction methods on weighted graphs that arise from bipartite projections results in backbone graphs that are much more informative than those that come from using a universal threshold. When using a single threshold value for each edge in our weighted graph, the resulting graph was extremely dense and didn’t offer any insight into the current political structure of the United States Senate. However, when applying the Hypergeometric model, Fixed Degree Sequence Model, or Stochastic Degree Sequence Model, we begin to see that most positive edges lie between members of the same political party, while negative edges tend to connect members of differing political parties. These results are not unexpected given the current political climate.

The backbone R package provides an easy way to analyze and interpret weighted graphs by reducing them to an unweighted graph that contains only the most important edges. Each included model respects slightly different scenarios (being conditioned on row marginals, column marginals, etc.) and providing an accessible way to use all models will help researchers understand better the underlying structure of their data and what inferences should be drawn from it.
References

[Carstens, 2015] Carstens, C. J. (2015). Proof of uniform sampling of binary matrices with fixed row sums and column sums for the fast curveball algorithm. Physical Review E, 91(4).

[Chartrand et al., 2010] Chartrand, G., Lesniak, L., and Zhang, P. (2010). GRAPHS & DIGRAPHS: Sixth Edition. Chapman and Hall/CRC.

[govinfo, ] govinfo, U. S. G. P. O. G. Bulk data - bill status.

[Hong, 2013] Hong, Y. (2013). On computing the distribution function for the poisson binomial distribution. Computational Statistics & Data Analysis, 59:41–51.

[Levin and Peres, 2017] Levin, D. A. and Peres, Y. (2017). Markov chains and mixing times, volume 107. American Mathematical Soc.

[Neal, 2013] Neal, Z. (2013). Identifying statistically significant edges in one-mode projections. Social Network Analysis and Mining, 3(4):915–924.

[Neal, 2014] Neal, Z. (2014). The backbone of bipartite projections: Inferring relationships from co-authorship, co-sponsorship, co-attendance and other co-behaviors. Social Networks, 39:84–97.

[Scott and Carrington, 2014] Scott, J. and Carrington, P. (2014). The SAGE Handbook of Social Network Analysis. SAGE Publications Ltd.

[Strona et al., 2014] Strona, G., Nappo, D., Boccacci, F., Fattorini, S., and San-Miguel-Ayanz, J. (2014). A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. Nature Communications, 5:4114.

[Strona et al., 2018] Strona, G., Ulrich, W., and Gotelli, N. J. (2018). Bi-dimensional null model analysis of presence-absence binary matrices. Ecology, 99(1):103–115.

[Zweig and Kaufmann, 2011] Zweig, K. A. and Kaufmann, M. (2011). A systematic approach to the one-mode projection of bipartite graphs. Social Network Analysis and Mining, 1(3):187–218.