Graphical discovery in stochastic actor-oriented models for social network analysis

by

Samantha Carroll Tyner

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Statistics

Iowa State University

Ames, Iowa

2017

DEDICATION

I would like to dedicate this thesis to the all the amazing women in my life, who have guided and encouraged me on my academic journey. Thank you.

Special thanks to Dr. Lara Pudwell, without whom I would have never even considered going to graduate school; Dr. Taddy Kalas, *merci pour votre passion et votre intrépidité*; my sisters, Jordan, Makenna, Cate, and Jenna; my mom, Cathy, and my grandma, Nancy. Thanks for the brains, Grandma. I wish you could be with me to celebrate.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this dissertation.

First and foremost, Dr. Heike Hofmann for her guidance, patience, encouragement, excitement, and support throughout this research and the writing of this thesis. Her insights and words of encouragement and wisdom have often inspired me and gave me confidence in the work when I couldn't muster it for myself. I would also like to thank my committee members for their efforts and contributions to this work: Drs. Alicia Carriquiry, Dianne Cook, Cindy Yu, and extra thanks to Dr. Olga Chyzh for introducing Dr. Hofmann and I to these incredibly memorable stochastic actor-oriented models.

I would additionally like to thank Dr. Bruce Rogers for planting the network bug in my brain, Dr. Eric Hare for his computational assistance, and Dr. Margaret Johnson for her emotional assistance. I couldn't have done this without you.

ABSTRACT

This work presented in this thesis combines statistical models for social networks and network visualization in new and exciting ways. In Chapter 1, a thorough review of the literature in the topics of statistical network models and network visualization is presented. In Chapter 2, we focus in on one type of model for dynamic social networks: the stochastic actor-oriented models (SAOMs), introduced by Snijders (1996). Unlike other network models, SAOMs are not very well understood, so we use model visualization techniques inspired by those introduced in Wickham et al (2015) in order to make the models a little less murky. The SAOMs are a prime example of a set of models that can benefit greatly from application of model visualization, and with the help of static and dynamic visualizations, we bring the hidden model fitting processes into the foreground, eventually leading to a better understanding and higher accessibility of stochastic actor-oriented models for social network analysts. In Chapter 3, we further explore the SAOMs using the visual inference methodology of Buja et al. (2009). We construct significance tests of model parameters, goodness-of-fit tests, and power calculations for the objective function parameters in SAOMs using visual inference. In this way, we can explore complex network data more completely than traditional significance and goodness-of-fit methods that rely on one-dimensional derived features of networks do. In Chapter 4, we present an `R` package for drawing networks using the popular grammar of graphics `R` plotting paradigm, `ggplot2` (Wickham 2016). We close with a discussion of the limitations of the work and directions for the future in Chapter 5.

CHAPTER 1. LITERATURE REVIEW

Social networks have been studied for decades, beginning with a few foundational works, the most well known of which is the 1967 study, “The Small World Problem” by Stanley Milgram (Goldenberg et al., 2010). But in recent years, the study of social networks has grown wildly in popularity due to an increase in the availability of and easy of access to social network data. The digital revolution has led to the creation of social media, linking people from all over the world in a way we never have been before. Now that platforms like Facebook, Twitter, and LinkedIn permeate our world, just about everyone knows what social networks are. In academic circles, collaboration networks are a type of social network that have been extensively studied and can even be a point of pride, like a mathematician’s Erdös number (Grossman, 2016). Social networks are a rich source of knowledge, but the data format does not fit easily within traditional data collection paradigms. Traditionally, data collection involves a set of units of the same, or at least similar, kind, on which observations are made. The storage of traditional data is simple and organized: rows contain variable values collected from units. These units can be people, plants, animals, stocks, objects, fields, and anything else under the sun, but one social network consits of many units, yet on the whole is just one observation. When observing a social network, one observes the possibly very numerous actors (also referred to as vertices or nodes) and the relationships (also referred to as edges or ties) between those actors. One can also collect information on the nodes and the edges separately, such as the age or gender of people and the length of their relationship or how strong it is in a friendship network. Thus, information on the entire network is more difficult to store than traditional data with which statisticians usually work.

This apparent difficulty has not stopped researchers in many different fields from studying social and other types of networks. Sociologists work with human relationship networks of all kinds imaginable, biologists work with protein-protein interaction networks, neurologists use fMRI scans to study biologic neural networks, and the list goes on. These disciplines worked separately for many years, each developing their own measures, softwares, and theories about the fundamental properties of networks. And although statisticians were comparatively late to the party, many statistical models exist for network analysis. Beginning with the classic Erdös-Rényi random graph model and varying in structure, complexity, and application to include longitudinal network data, such as continuous time markov chain models (Goldenberg et al., 2010). The many varying models that exist just for social network analysis are impressive, but I focus my research on one type of continuous time markov chain (CTMC) models, called Stochastic Actor-Oriented Models (SAOMs). A full introduction to the various models that exist for social network analysis is presented in Section 1.1, and a full introduction to the structure and theory of SAOMs is presented in Section 1.1.4.

1.1 Statistical Models for Social Networks

The literature on statistical models for networks is extensive. In their thorough "Survey of Statistical Models", Goldenberg et al. (2010) separate these models into two primary classes: static and dynamic. I discuss the several types of models as they relate to stochastic, actor-oriented models, the models of my primary focus, in each of these two categories after a brief introductory section on general network terminology and notation.

1.1.1 Basic Network Terminology and Notation

Formally, a network is defined by a collection of nodes, also referred to as vertices or actors, and the set of ties, also referred to as edges or relationships, between them. Let x denote a network. The network's collection of nodes, its nodeset, is written $\mathcal{N}$, and its collection of edges, its edgeset, is written. $\mathcal{E}$. Typically, the nodes are numbered so that $\mathcal{N} = \{1, 2, \dots, n\}$, where n is the total number of nodes in the network. The edgeset is usually described as a set of pairs, written as x_{ij} or $i \rightsquigarrow j$ or (i, j), where $i \neq j \in \mathcal{N}$. In an undirected network, the ordering of i and j does not matter: there is no parent-child relationship, to use a term from graph theory, just a connection of some kind. In a directed graph, however, the order does matter: the tie x_{ij} is not equivalent to the tie x_{ji}. In a simple, undirected graph, the number of possible edges is $\binom{n}{2}$, while in simple, directed graphs it is $n(n-1)$, assuming no self-loops (also called self-ties or simply loops) and only allowing for at most one edge between any two nodes.

In statistical network analysis, an observed network is written as x, while X denotes an unobserved network being treated as a random variable. I assume binary network ties throughout: if the edge between nodes i and j is present, $x_{ij} = 1$, whereas $x_{ij} = 0$ if the edge is not present. If x is undirected, then $x_{ij} = x_{ji} \forall i \neq j \in \mathcal{N}$. If x is directed, then x_{ij} may equal x_{ji}, but this is not required and should not be assumed. Note that the definition of binary edge variables makes the assumption that edges are unweighted and that their cannot be more than one edge between two nodes. It is possible for networks to have weighted edges or multiple ties between nodes, but the models I discuss here, including the stochastic actor-oriented models that are my primary focus, are all for unweighted networks.

A network x can also be expressed as an $n \times n$ matrix of 0s and 1s called the adjacency matrix, denoted A. The ij^{th} entry of this matrix, a_{ij} is 1 if there is an edge between nodes i and j and 0 otherwise. The diagonal entries of this matrix, a_{ii} are structurally 0, as self-ties or self-loops are not allowed as mentioned above.

1.1.2 Static Network Models

The Erdös-Rényi random graph model is widely regarded as the first random graph model (Goldenberg et al., 2010). This model, first introduced in Erdös and Rényi (Erdös and Rényi, 1959), describes random, undirected networks. Edges x_{ij} are selected at random from all possible edges. The parameter in this model is p, the probability that an edge exists between any two nodes in the network. The number of edges in the network, $e = \sum_{i<j} x_{ij}$, has likelihood

$$f(e|p,n) = p^e(1-p)^{\binom{n}{2}-e}.$$

The properties and asymptotic behavior of this network model are well-established (Goldenberg et al., 2010). Nodes in networks generated using this model all have about the same degree, or number of incident edges, which, in practice, is a very unrealistic property for a network to have. As such, many other models have been devised over the years as a way to better capture the network creation process underlying real-world networks.

In order to better model real-world networks, the exponential random graph family of models (ERGMs) was developed. These are also referred to as p^* models after the first use of the exponential family form in the p_1 model for directed networks of Holland and Leinhardt (Holland and Leinhardt, 1981). This class of models uses structural properties of the network as sufficient statistics in the likelihood. The properties used are different for directed and undirected graphs. Some statistics used for directed networks are the outdegree of the nodes, $x_{i+} = \sum_{j=\neq i} x_{ij}$, the indegree of the nodes, $x_{+i} = \sum_{j \neq i} x_{ji}$, and the number of reciprocal ties of the nodes, $x_{i,recip} = \sum_{j \neq i} x_{ij} x_{ji}$. For undirected networks, however, structures that are considered are the number of triangles in the network, $T(x) = \sum_{i \neq j \neq h} x_{ij} x_{ih} x_{jh}$, or the number of k-stars, $S_k(x) = \sum_i \binom{x_{i+}}{k}$, where $k = 2$ is most commonly chosen. The likelihood for ERGMs is written in terms of the whole network, x. The general form of the likelihood for x is

$$f(x|\boldsymbol{\beta}) = \frac{1}{\psi(\boldsymbol{\beta})} \exp\left(\sum_k \beta_k s_k(x) \right),$$

where β_k are parameters corresponding to K sufficient statistics chosen by the researcher and $\psi(\beta)$ is the normalizing constant. A problem with this model arises when one considers the nested nature of the sufficient statistics. For example, an edge can be contained in a 2-star, which can be contained in a triangle. So, the sufficent statistics can be dependent. Despite this flaw, this type of ERGM has been studied extensively, and many methods for parameter estimation exist, for example in the `R` packages `statnet` and `sna` (R Core Team, 2016; Handcock et al., 2008; Butts, 2014).

Other models are extensions of the Erdös-Rényi (ER) random graph model, including the preferential attachment model or the small world model, which are among the first network models that consider a network *formation* process over time. Eventually, network models expanded to include dynamic models, which consider changing network states in time.

1.1.3 Dynamic Network Models

Dynamic networks models are extremely important because of how realistic they are. Social networks do not form spontaneously: they evolve over time. Ties can be added and deleted, and new nodes can join the network. Modeling the process of network changes over time is more complex but ultimately more useful if done correctly. The work on dynamic network models began with fairly straightforward random graph models that are quasi-dynamic extensions of the classic Erdös-Rényi model.

A model is quasi-dynamic if it models a *static* network via an *underlying* dynamic process. The first is the preferential attachment model of Barabsi and Albert (Barabási and Albert, 1999). Given n_0 nodes to start, at each time point t a new node is added with $n_t \leq n_0$ ties to the nodels already in the network. The n_t new ties are assigned proportionally based on the degree of each existing node. It is quasi-dynamic because it is usually used to model one scale-free network oberervation. The preferential attachment model is also referred to as the "rich-get-richer" model because it results in a network where there are a few nodes with very high degree.

Another quasi-dynamic model is the small-world model of Watts and Strogatz (Watts and Strogatz, 1998b). Given n nodes to start, each with k edges that form a ring lattice (nodes layed out in a circle and connected to their k closest neigbors), edges are randomly "rewired" with probability p. This results in networks with the small-world property: let L be the average distance between any two nodes in the graph, and if the graph has the small-world property, $L \propto \log(n)$ as n increases (Watts and Strogatz, 1998b).

Truly dynamic models consider the same network observed at multiple points in time. To indicate a dynamic network, we write $x(t)$ instead of x for the network observation at time t. Dynamic network models can be in discrete or continuous time.

One such model in discrete time is an extension of the ERGM family. It models the transition probability, the probability of moving from the current network $x(t-1)$ to a potential future network, $x(t)$, that differs from $x(t-1)$ by one tie. The form of this probability is similar to the likelihood of the static ERGM model:

$$Pr(x(t)|x(t-1)) = \frac{1}{\psi(\boldsymbol{\beta})} \exp\left\{\sum_k \beta_k s_k(x(t), x(t-1))\right\},$$

where the s_k for $k = 1, \dots, K$, are structural network statistics, similar to, but not the same as, the network statistics defined for static ERGMs. Some examples of statistics used are, the density of edges of the network, $s_1(x(t), x(t-1))$, or the stability of the network between time $t-1$ and time t. The density of a network is a ratio of the number of edges to the number of nodes in the network at the next time point, $s_1(x(t), x(t-1)) = \frac{1}{n-1}\sum_{i \neq j} x_{ij}(t)$. The stability is a measure of how many changes were made in the network between two time points, relative to the number of nodes, $s_2(x(t), x(t-1)) = \frac{1}{n-1}\sum_{i \neq j}(x_{ij}(t)x_{ij}(t-1) + (1 - x_{ij}(t))(1 - x_{ij}(t-1)))$. The likelihood of the entire network for all its states in *discrete* time is the joint probability of each transition step:

$$Pr(x(1), x(2), \dots x(T)) = \prod_{t=2}^{T} Pr(x(t)|x(t-1)).$$

The family of dynamic network models in continuous time, of which stochastic, actor-oriented models are a member, are called continuous time Markov Chain (CTMC) models. These models are founded in the theory of continuous time Markov processes. Let $\{X(t), | t \in \mathcal{T}\}$ be a stochastic process in a continuous time interval $\mathcal{T}$ and finite state space $\mathcal{X}$. For any two timepoints $t_a < t_b \in \mathcal{T}$, the future state of the network, $X(t_b)$, depends only on the current state of the network, $X(t_a)$, and not any other previous network state. This is the Markov property, which for CTMCs is written as:

$$Pr(X(t_b) = \tilde{x}|X(t) = x(t), \quad \forall t \leq t_a) = Pr(X(t_b) = \tilde{x}|X(t_a) = x(t_a))$$

where $\tilde{x}$ is a potential future state in $\mathcal{X}$ and $x(t_a)$ is the present, observed state of the network. Assuming this probability relies only on the length of time that passes, $t_b - t_a$, then $X(t)$ has a stationary transition distribution. Then, the transition matrix for the process $X(t)$ is

$$Pr(t_b - t_a) \equiv \Big[Pr(X(t_b) = \tilde{x}|X(t_a) = x(t_a))\Big]_{x, \tilde{x} \in \mathcal{X}}.$$

Write $t_b - t_a = t'$. Then, thanks to the stationarity of $X(t)$, the transition matrix of $X(t)$, $Pr(t')$ is equal to the matrix exponenial $\exp(t'\mathbf{Q})$, where $\mathbf{Q}$ is called the *intensity matrix* in the CTMC

Table 1.1: Some propensity functions to describe the network dynamics in CTMC models.

Model	$\mathbf{q_{ij}(x) =}$	**Brief Description**
Independent arc	$\lambda_{x_{ij}}$	Edges are independent and have equal probability of changing from 0 to 1 and from 1 to 0
Reciprocity	$\lambda_{x_{ij}} + \mu_{x_{ij}} x_{ji}$	Rate of change depends on the presence of reciprocal edge.
Popularity	$\lambda_{x_{ij}} + \pi_{x_{ij}} x_{+j}$	Rate of change is dependent on the indegree of the child node, j
Expansiveness	$\lambda_{x_{ij}} + \pi_{x_{ij}} x_{i+}$	Rate of change is dependent on the outdegree of the parent node, i

literature. The elements of this matrix will be defined in greater detail later, but one should note that the rows of $\mathbf{Q}$ are constructed to always sum to 0, and that each element also determines the probability of changing from one state to the other as a function of time.

For network modelling with CTMCs, the state space $\mathcal{X}$ is the set of all $2^{n(n-1)}$ possible networks with n nodes and directed, binary edges. Let x denote the current state of the network. From this network, there are $n(n-1)$ possible networks that x could become by changing just one edge variable, x_{ij} to its opposite value, $1 - x_{ij}$. Then, let $q_{ij}(x)$ be the propensity for for x_{ij} to become $1 - x_{ij}$ given x. This function $q_{ij}(x)$ "completely specifies the dynamics of the network model" (Goldenberg et al., 2010, p. 48). There are many forms in this family of models, which differ only in their choice of $q_{ij}(x)$. A list of some fairly simple choices for $q_{ij}(x)$ is provided in Table 1.1.

Additional definitions of $q_{ij}(x)$ are more complicated. These next set of models rely on two different underlying mechanisms: one that determines which node is given the opportunity to change and one that determines the propensity of change. First, I consider the subset of models with edge-oriented dynamics. Let $x(i \rightsquigarrow j)$ denote the network that differs from x by just one node, x_{ij}, which takes on the value $1 - x_{ij}$ in $x(i \rightsquigarrow j)$. Then, write the probability that node x_{ij} changes to $1 - x_{ij}$ as

$$p_{ij}(x) = \frac{\exp(f(\boldsymbol{\beta}, x(i \rightsquigarrow j)))}{\exp(f(\boldsymbol{\beta}, x)) + \exp(f(\boldsymbol{\beta}, x(i \rightsquigarrow j)))},$$

where $f(\boldsymbol{\beta}, x) = \sum_k \beta_k s_k(x)$ is called the potential or objective function (Goldenberg et al., 2010). The β_k are parameter values associated with the network statistics that are also used in ERGMs. For more definitions of the possible $s_k(x)$, see Table 1.2. The opportunity for change in this model is controlled by a constant rate parameter, α. The wait time between a change of any edge in the network is exponentially distributed with parameter α. So, the function $q_{ij}(x)$ is defined as $\alpha p_{ij}(x)$.

The next subset of models rely on node-oriented dynamics. These are very similar to the edge-oriented dynamics but the rate parameter and propensity to change are defined with respect to the nodes instead of the edges. Now, each node has its own rate at which it gets an opportunity for change, α_i. Additionally, the objective function is defined for each node, $f_i(\boldsymbol{\beta}, x) = \sum_k \beta_k s_{ik}(x)$. This changes the definition of the statistics used slightly, from global statistics to local statistics with ego node i. Thus, the propensity function becomes $q_{ij}(x) = \alpha_i p_{ij}(x)$.

Finally, stochastic, actor-oriented models belong to the set of CTMC models that combine edge and node dynamics so that the propensity function becomes a hybrid of the prior two: $q_{ij}(x) = \alpha p_{ij}(x)$ where α is a constant rate of edge change, while p_{ij} is the propensity to change edge x_{ij} using the node-oriented objective function $f_i(\boldsymbol{\beta}, x)$. These are described in greater detail in Section 1.1.4.

1.1.4 Stochastic Actor-Oriented Models for Longitudinal Social Networks.

A Stochastic Actor-Oriented Model (SAOM) is a model that is changing in time in order to accomodate for observations from the same network made at different points in time and that allows for changes in network structure due to actor-level covariates. These two properties are crucial to understanding networks as they exist naturally. Most social networks, even holding constant the set of actors over time, are ever-changing as relationships decay or grow, and most actors (or nodes) in social networks have inherent properties that could affect how they change their place within the network.

1.1.4.1 Terminology, Notation, and Mathematical Definition of SAOMs

A longitudinal network is a network consisting of the same set of n nodes that is changing over time, and is observed at M discrete time points, $t_1, \ldots, t_M$. We denote these network observations $x(t_1), \ldots, x(t_M)$. The SAOM assumes that this longitudinal network is embedded within a continuous time markov process (CTMP), call it $X(T)$. This process is almost entirely unobserved. The process $X(T)$ theoretically exists outside of the range of observation, but for simplicity of notation, assume that the beginning of the process, $X(0)$ is equivalent to the first observation $x(t_1)$, while the end of the process $X(\infty)$ is equivalent to the last observation $x(t_M)$. The observations $x(t_1), \ldots, x(t_M)$ are observed states of the process, $x(t_1) \equiv X(0), x(t_2) \equiv X(T_{t_2}), \ldots, x(t_{M-1}) \equiv X(T_{t_{M-1}}), x(t_M) \equiv X(\infty)$, but the time points t_m and T_{t_m} for $m = 2, \ldots M-1$ are not equivalent. The process $X(T)$ is a series of single tie changes, in which one actor at a time is given the opportunity to add or remove one outgoing tie. These opportunities for change can arise at a different rate for each actor, and the overall rate of change, the distribution of the waiting times that *any* actor will be given the opportunity to change is a function of all actors' rates. Additionally, once an actor is given the chance to change a tie, it tries to maximize a sort of utility function based on the current and potential future states of the network. These functions are described in detail in subsections 1.1.4.2 and 1.1.4.3.

1.1.4.2 The Rate Function

For the network x and each actor i in the network, the rate function dictates how often the actor i gets to change its ties, x_{ij}, to other nodes $j \neq i$ in the network. This rate can depend on the time period of observation, some actor-level covariates or some actor-level network statistics. The rate function can be unique to each actor, and is denoted λ_i. The most general form is $\lambda_i(\alpha, \rho, x, m)$, where α is a simple rate of change parameter, ρ is a parameter or a vector of parameters corresponding to one or more covariates, $x \in \mathcal{X}$ is the current state of the network, and m indicates the time point of the current network observation, t_m. The rate function determines how quickly actor i gets an opportunity to change one of its ties, x_{ij} in the time period $t_m \leq T < t_{m+1}$. We assume that the actors i are conditionally independent given their current ties, $x_{i1}, \ldots, x_{in}$. This assumption leads to the rate function for the whole network:

$$\lambda(\alpha, \rho, x, m) = \sum_i \lambda_i(\alpha, \rho, x, m).$$

In order to achieve the memorylessness property of a Markov process, for any time point, T, where $t_m \leq T < t_{m+1}$, the waiting time to the next change opportunity by actor i is exponentially distributed with expected value $(\lambda_i(\alpha, \rho, x, m))^{-1}$. Thus, the waiting time to the next change opportunity by *any* actor in the network is also exponentially distributed with mean $(\lambda(\alpha, \rho, x, m))^{-1}$, where

$$\lambda(\alpha, \rho, x, m) = \sum_i \lambda_i(\alpha, \rho, x, m)$$

.

There are many possibilities for the rate function, λ_i. The simplest is that it is constant over all actors and all unobserved timepoints between observations $x(t_m)$ and $x(t_{m+1})$, $\lambda_i(\alpha, \rho, x, m)) = \alpha_m$. The rate function can also depend on covariate values, call them $\mathbf{z}_i(t_m)$, of the actors, or structural network elements such as outdegree, or both. For instance, assume $\lambda_i(\alpha, \rho, x, m)) = \lambda_{i1}\lambda_{i2}\lambda_{i3}$, where λ_{i1} is constant over all actors within a time period (t_m, t_{m+1}), λ_{i2} depends on the actor covariates, and λ_{i3} depends on a structural network property for node i. λ_{i1} might be written as α_m. λ_{i2} might be written as

$$\lambda_{i2} = \exp\left(\sum_h \rho_h z_{ih}(t_m)\right),$$

where there are $h = 1, \ldots, H$ actor covariates of interest, each with their own parameter ρ_h. λ_{i3} can be written as a function of the outdegree of node i, denoted x_{i+} with its own parameter α_{H+1}, so that, for example,

$$\lambda_{i3} = \frac{x_{i+}}{n-1}\exp(\alpha_{H+1}) + \left(1 - \frac{x_{i+}}{n-1}\right)\exp(-\alpha_{H+1}).$$

When $H = 0$, this form of λ_{i3} is equivalent to the model proposed by Wasserman (1980), which is one of the first models proposed for modeling dynamic networks as continuous-time Markov processes (Snijders, 2001). Once a change occurs, according to the rate of change for the whole network, $\lambda(\cdot)$, the probability that actor i is the node with the power to change a tie is

$$\frac{\lambda_i(\alpha, \rho, x, m))}{\sum_i \lambda_i(\alpha, \rho, x, m))}$$

.

1.1.4.3 The Objective Function

Thanks to the conditional dependence assumptions in the model, we can consider the objective function for each node separately, since only one tie from one node is changing at a time. The objective function is written as

$$f_i(\boldsymbol{\beta}, x) = \sum_k \beta_k s_{ik}(x, \mathbf{Z}),$$

for $x \in \mathcal{X}$ and $\mathbf{Z}$ the matrix of covariates. The vector $\boldsymbol{\beta}$ are the parameters of the model with corresponding network and covariate statistics, $s_{ik}(x, \mathbf{Z})$, for $k = 1, \ldots, K$. Given the focal or ego node, i, there are n possible steps for the actor i to take: either one of all current ties $x_{ij} = 1$ will be destroyed, a new tie will be created, or no change will occur.

The parameters, $\boldsymbol{\beta}$, are attached to various actor-level network statistics, $s_{ik}(x)$. There are always at least two parameters, β_1 for the outdegree of a node, and β_2 for the number of reciprocal ties held by a node (Snijders, 2001, p. 371). There are many possible parameters β to add to the model. They can be split up into two groups: first, the structural effects, which only depend on the structure of the network. The inclusion of these effects has origin in the ERGMs discussed in Section 1.1.2. These effects are written in terms of the edge variables x_{ij}, for $i \neq j$. The second set of effects are the actor-level or covariate effects. These effects also depend on the structure of the network. They are written in terms of x_{ij} but also in terms of the covariates, $\mathbf{Z}$. A table of some possible structural and covariate effects is given in 1.2.

When node i is given the chance to change a node, we assume that they wish to maximize the value of their objective function $f_i(\boldsymbol{\beta}, x)$ plus a random element, $U_i(x)$, where the $U_i(x)$ are from "the type 1 extreme value distribution (or Gumbel distribution) with mean 0 and scale parameter 1" (Snijders, 2001, p. 368). This distribution, which is also known as the log-Weibull distribution,

Table 1.2: Some of the possible effects to be included in the stochastic actor-oriented models in `RSiena`. There are many more possible effects, but we only consider a select few here. For a complete list, see the RSiena manual (Ripley et al 2016).

Structural Effects	
outdegree	$s_{i1}(x) = \sum_j x_{ij}$
reciprocity	$s_{i2}(x) = \sum_j x_{ij}x_{ji}$
transitive triplets	$s_{i3}(x) = \sum_{j,h} x_{ij}x_{jh}x_{ih}$
Covariate Effects	
covariate-alter	$s_{i4}(x) = \sum_j x_{ij}z_j$
covariate-ego	$s_{i5}(x) = z_i \sum_j x_{ij}$
same covariate	$s_{i6}(x) = \sum_j x_{ij}\mathbb{I}(z_i = z_j)$
jumping transitive triplets	$s_{i7}(x) = \sum_{j \neq h} x_{ij}x_{ih}x_{hj}\mathbb{I}(z_i = z_h \neq z_j)$

has probability distribution function, using μ for the mean parameter and σ for the scale parameter, of

$$f(u|\mu, \sigma) = \frac{1}{\sigma} \exp\left\{-\left(\frac{u-\mu}{\sigma} + e^{-\frac{u-\mu}{\sigma}}\right)\right\}.$$

Using this distribution is convenient because it allows the probablity the actor i chooses to change its tie to actor j in terms of the objective function alone. Let $p_{ij}(\boldsymbol{\beta}, x)$ be this probability. Next, write the network x in its potential future state, where the tie x_{ij} has changed to $1 - x_{ij}$, as $x(i \rightsquigarrow j)$. Then, the probility that the tie x_{ij} changes is

$$p_{ij}(\boldsymbol{\beta}, x) = \frac{\exp\{f_i(\boldsymbol{\beta}, x(i \rightsquigarrow j))\}}{\sum_{h \neq i} \exp\{f_i(\boldsymbol{\beta}, x(i \rightsquigarrow h))\}}$$

1.1.4.4 A SAOM as a CTMC

In order to fit this model definition back into the original context of the CTMC described in Section 1.1.3, it must be written in terms of its intensity matrix, $\mathbf{Q}$. This matrix describes the rate of change between states of the process. For networks, there are a very large number of possible states, $2^{n(n-1)}$, so the intensity matrix is a square matrix of that dimension. But, thanks to the property of SAOMs that the states are allowed to change only one tie at a time, there are only n possible states given the current state, $n-1$ of which are uniquely determined by the node i that is given the opportunity to change. Thus, the intensity matrix $\mathbf{Q}$ is very sparse, with only $n(n-1)+1$ non-zero entries in each row. Note that $n(n-1)$ of these represent the possible states that are one edge different from a given state, and the additional non-zero entry is for the state to remain the same. All other entries in a row are zero because those column states cannot be reached from the row state by just one change as dictated by the SAOM. The entries of $\mathbf{Q}$ are defined as follows: let $b \neq c \in \{1, 2, \ldots, 2^{n(n-1)}\}$ be indices of two different possible states of the network, $x^b, x^c \in \mathcal{X}$. Then the bc^{th} entry of Q is:

$$q(x^b, x^c) = \begin{cases} q_{ij}(\alpha, \rho, \boldsymbol{\beta}, x^b) = \lambda_i(\alpha, \rho, x^b, m) p_{ij}(\boldsymbol{\beta}, x^b) & \text{if } x^c \in \{x^b(i \rightsquigarrow j) | \text{ any } i \neq j \in \mathcal{N}\} \\ 0 & \text{if } x^c \text{ differs from } x^b \text{ by more than 1 tie} \\ -\sum_{i \neq j} q_{ij}(\alpha, \rho, \boldsymbol{\beta}, x^b) & \text{if } x^b = x^c \end{cases}$$

Thus, the rate of change between any two states that differ by only one tie, x_{ij}, is the product of the rate at which actor i gets to change a tie and the probability that the tie that will change is the tie to node j.[1] Furthermore, the theory of continuous time Markov chains gives that the matrix of transition probabilities between observation times t_{m-1} and t_m is dependent only on the difference between timepoints, $t_m - t_{m-1}$. Following the same definition for transition probabilities in Section 1.1.3, the matrix of transition probabilities is

$$e^{(t_m - t_{m-1})\mathbf{Q}},$$

where $\mathbf{Q}$ is the matrix defined above and e^X for a real or complex square matrix X is equal to $\sum_{k=0}^{\infty} \frac{1}{k!} X^k$.

1.1.4.5 Model Fitting for SAOMs

Stochastic actor-oriented models are "too complicated for the calculation of likelihoods or estimators in closed form, but they represent stochastic processes which can be easily simulated" (Snijders et al., 2010b, p. 568). Thus, calculation of the method of moments estimates of param-

[1] Just to be clear, the change is from x_{ij}^b to $x_{ij}^c = 1 - x_{ij}^b$.

eters in SAOMs is done via Markov Chain Monte Carlo (MCMC) approximation. The algorithm presented here were first presented in Snijders (2001).

The vector of parameters that need to be estimated is

$$\boldsymbol{\theta} = (\alpha_2, \ldots, \alpha_M, \beta_1, \ldots, \beta_K).$$

The length of $\boldsymbol{\theta}$ is $L = M - 1 + K$, where M is the number of network observations and K is the number of parameters included in the objective function, $f_i(\boldsymbol{\beta}, x)$. The corresponding sufficient statistics for estimating the rate parameters, α_m, are the number of edges that have changed between $x(t_{m-1})$ and x_{t_m}, $C_2, \ldots, C_M$, where $C_m = \sum_{i \neq j} |x_{ij}(t_m) - x_{ij}(t_{m-1})|$. Let The corresponding sufficient statistics for estimating the rate parameters, β_k, are the corresponding values of the node-level statistics, some of which are seen in Table 1.2, summed over all nodes for each network observation, $S_{2k}, \ldots, S_{Mk}$ where $S_{mk} = \sum_i s_{ik}(x(t_m))$ for $m = 2, \ldots, M$. Denote the whole vector of sufficient statistics as $\mathbf{S} = \big(C_2, \ldots, C_M, S_{2k}, \ldots, S_{Mk}\big)$.

The method of moments estimator of $\boldsymbol{\theta}$ is the solution to $E_{\boldsymbol{\theta}}[\mathbf{S}] = \mathbf{s}$ where $\mathbf{s}$ are the observed values of $\mathbf{S}$ in $x(t_2), \ldots, x(t_M)$. Following Snijders (2001), the estimate $\hat{\boldsymbol{\theta}}$ can be separated into the vectors $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$, which are the solutions to the system of equations

$$E_{\boldsymbol{\alpha}}[C_m | x(t_m)] = c_m$$
$$\sum_{m=1}^{M-1} E_{\boldsymbol{\beta}}[S_{mk} | x(t_m)] = \sum_{m=1}^{M-1} s_{mk}.$$

The solutions to these moment equations are, unless the model is extremely simple, not able to be calculated explicitly. Because of this, random simulation of networks with the desired distribution can be used in Markov Chain Monte Carlo simulation of the moment estimates. Given a starting value $\boldsymbol{\theta}^{(0)}$, the updating step of the simulation is, for iterations $b = 0, \ldots, B$:

$$\boldsymbol{\theta}^{(b+1)} = \boldsymbol{\theta}^{(b)} + a_b D_0^{-1}(\mathbf{S}_b - \mathbf{s})$$

where $\mathbf{S}_b$ is drawn from the distribution of the model under $\boldsymbol{\theta} = \boldsymbol{\theta}^{(b)}$, $\mathbf{s}$ are the observed statistics, D_0 is a positive diagonal matrix, usually the identity, and a_b is called the *gain sequence* and is some sequence of positive values that approach 0 as $b \to \infty$ at about the same rate as b^{-r} for some $0.5 < r < 1$. The method of moments estimator, $\tilde{\boldsymbol{\theta}}$ is then an average of the B iterations, $\tilde{\boldsymbol{\theta}} = \frac{1}{B}\sum_{b=1}^{B} \boldsymbol{\theta}^{(b)}$. It must be the average in order to obtain optimal convergence (Snijders, 2001).

This algorithm is implemented in the R software package `RSiena` for computation of parameter estimates for various SAOMs (Ripley et al., 2013). This is the software I use for model fitting in 2 and 3.

Model Selection and Testing for SAOMs A likelihood ratio test was also developed in Snijders et al. (2010b), but it has yet to be implemented in the software RSIENA for parameter estimation of SAOMs. Tests of the elements of $\boldsymbol{\beta}$ are, however, availabile in RSIENA. Both t-tests and Wald-type tests are implemented. A goodness-of-fit test is also implemented, but it only assesses the fit of a model with respect to the "auxiliary statistics of networks [...] that are not explicitly fit by a particular effect" (Ripley et al., 2017, p. 53). It is this lack of goodness-of-fit testing that led my research down the path of applying visual inference principles and protocols to hypothesis testing for SAOMs.

1.2 Network Visualization

Network visualization, also called network mapping, is a very well-established subfield of network analysis. As networks have such a non-traditional data structure, visualization has always been of the utmost importance to understanding the structre of a network.

1.2.1 Layout Algorithms

The key difficulty with network visualization that does not arise with most other types of data visualization is the lack of a well-defined axis. This is not something one has to think hard about for most data visualizations. If the variables are numerical, histograms, scatterplots, or time series plots are straightforward to construct: one variable on the x-axis, another on the y-axis in 2D Euclidean space. If the variables are categorical, bar charts and mosaic plots can be constructed in this same space. If the data are spatial, there is a well-defined space In pretty much any case, the location and labels of the data and axes can be defined with very little struggle. With network data, however, this is a more difficult problem.

Network visualizations are made by representing nodes with points in 2D Euclidean space, just like one would with any other data set, and then by representing edges by connecting the points with lines if there is an edge between the two nodes. But, because there is no natural placement of the points, a random placement is used, then adjusted iteratively via a layout algorithm, of which there are many kinds. I will focus on the 2D layout algorithms only because I work later with the `ggplot2` package to visualize networks, and this package only has 2D drawing capabilities.

Some layout algorithms were designed to mimic physical systems, drawing the graphs based on the "forces" connecting them. The network's edges act as springs pushing and pulling the nodes in 2D space. Some force-directed layout algorithms are:

- Kamada-Kawai: first introduced in Kamada and Kawai (1989). Has "symmetric drawings, a relatively small number of edge crossings, and almost congruent drawings of isomorphic graphs" (Kamada and Kawai, 1989, p. 15).
- Fruchterman-Reingold: first introduced in Fruchterman and Reingold (1991). Primary advantage is speed over Kamada-Kawai (Fruchterman and Reingold, 1991, p. 1161).
- Spring embedding: first introduced in Eades (1984). Other force-directed layouts are refinements of this original algorithm.
- Target diagram: nodes placed in concentric circles with hig-centrality nodes placed nearer to the center of the circle. First introduced in Brandes et al. (2003).

Other layout algorithms depend on the mathematical properties of the network's adjacency matrix or some other function or propterty of the network. Algorithms of this kind are:

- Eigen: node placement is based on the eigenvalues of the adjacency matrix

- Hall: node placement is based on the last two eigenvectors of the Laplacian of the adjacency matrix
- Multidimensional Scaling (MDS): node placement is based on metric multidimensional scaling of a given distance matrix. Distance metric can vary.
- Principal Coordinates: node placement is based on the eigenvalues of a given covariance or correlation matrix.

Some layout algorithms only exists for certain types of networks:

- Reingold-Tilford: for trees
- Sugiyama: for layerd directed acyclic graphs

Finally, some layout methods just place the nodes randomly or in a simple ordering:

- Random: places nodes randomly according to some distribution, usually uniform or some Gaussian distribution.
- Grid: places nodes on a 2D grid
- Circle: places nodes in a circle in numerical order by ID number

These layout algorithms have been provided in several R packages for network visualization. Another important aspect of network visualization is the addition of varirable information into the properties of the points and segments of the network visualization. For example, the size of the point, the width of the line, and the color of these these can all be mapped to the points and segments making up the network visualization. This is discussed further in Section 1.2.3.

The visualization methods outlined above are all for static networks. There has been little work done on how to visualize dynamic networks. The only R package to my knowledge that attempts dynamic network visualization is `ndtv` by Bender-deMoll (2016). I will use this package to help visualize the continuous time Markov chain underlying the SAOM dynamics. The goal is to better understand the network changes, how they change, and better see the differences, but it may turn out to be less effective than looking at side-by-side comparisons of two network observations.

1.2.2 R Packages

There is a multitude of R packages that exist for network analysis, and many, if not most, of them contain some sort of built-in functionality for visualizing networks. The most popular of these is probably the `igraph` package by Csardi and Nepusz (2006). This package is extensive, and contains much more than methods for network visualization. It contains tools for both 2D and 3D visualization of networks. The 2D layouts it contains are `random`, `circle`, `star`, `grid`, `graphopt`, `bipartite`, `fruchterman_reingold`,`kamada_kawai`, `mds`, `grid_fruchterman_reingold`, `lgl`, `reingold_tilford`, `reingold_tilford_circular`, and `sugiyama`.

Another popular package for network analysis is `sna` by Butts (2014). This package was designed specifically for social network analysis (sna), so it also contains much more capabilities for network analysis in addition to visualization. Like `igraph`, `sna` contains both 2D and 3D layout methods. The 2D layout algorithms available in `sna` are `circle`, `circrand`, `eigen`, `fruchtermanreingold`, `geodist`, `hall`, `kamadakawai`, `mds`, `princoord`, `random`, `rmds`, `segeo`, `seham`, `spring`, `springrepulse`, and `target`.

Research into possible layout algorithms is important, but it ignores some of the things that statisticians usually consider when visualizing data. For instance, since the location of points in 2D space contains no information about the data, how else should this information be visualized? As an example, consider a friendship network of students at a university. Representing this network as simple points and lines leaves a lot of information out. Some information that could be incorporated includes the students' majors, year in school, and whether the students have ties through their classes or their extracurricular activities. In the network visualization, this information can be mapped to color of point, shape of point, and linetype, respectively. Adding this aesthetic information helps to make up for the loss of two dimensions of visual perception and to bring the network visualization into the world of statistical graphics.

1.2.3 The Importance of the `ggplot2` Package

The `gg` of `ggplot2` is for the "grammar of graphics". The grammar of graphics is a well-defined theory for creating statistical graphics described in Wilkinson (1999) and Wickham (2009). In the grammar, a plot has layers, each of which has four distinct pieces: the data and aesthetic mapping, a statistical transformation, a geometric object, and a position adjustment. The aesthetic mapping takes the data and *maps* the variables in the data frame to visual features. Some of these features are horizontal and vertical placement in the plane, size of the geometric object and color of the geometric object. The statistical transformation dictates how to transform the data to the values that create the visual feature. Some `stats` are `identity` (no change in data), `bin`, and `smooth`. The geometric object or `geom` is the tool used to draw a plot layer. Some `geoms` are `point`, `line` and `bar`. Finally, the position adjustment is there to slightly change the position of the visual features in order to better view the data. This is typically only a probelm with discrete data, where overplotting can occur. Some position adjustments are `identity`, `jitter`, and `dodge`.

With the theory well defined and constructed, the `ggplot2` package allows for creation of rich, visually dense plots. The user can combine multiple aesthetic mappings to view four variables at once or view many data sets of similar scale at once. The widespread use of `ggplot2` and the many packages that have built upon `ggplot2` to create visualizations above and beyond what it is capable of by itself make the `ggplot2` package an ideal framework on which to build additional methods of network visualization in R.

First, the data structure required in `ggplot2` is fairly simple: data frames. Some other network packages contain network data structures unique to them, like the `igraph` class of data in the `igraph` packages or the `network` class of data in the `network` package. These unique structures come with unqiue syntax that can make customizing visualizations tricky. Additionally, the default visualizations in these packages are not very pleasing to the eye, as is shown with the random graph examples from `igraph` and `network` in Figure 1.1. As I will discuss in 4, network visualization within the `ggplot2` framework results in beautiful, easily customizable plots.

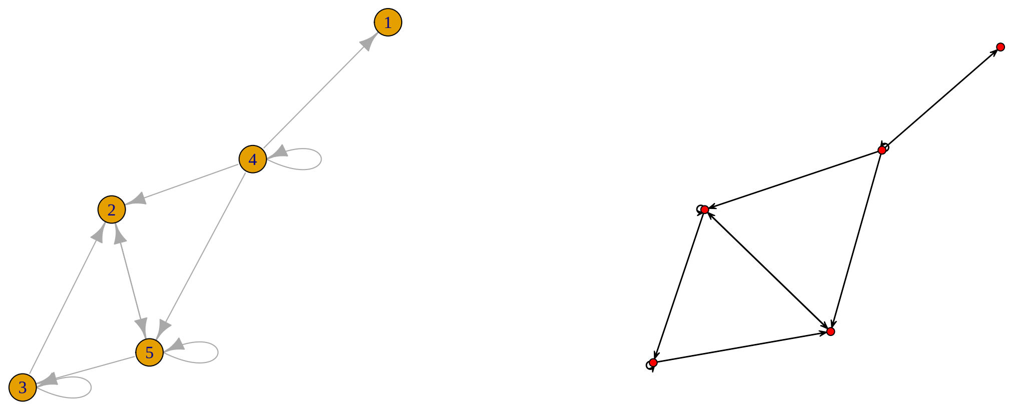


Figure 1.1: The same random network plotted with the default options in `igraph` (at left) and `network` (at right).

By creating a way to visualize networks in `ggplot2`, we open up network visualization to a set of visual tools and approaches that

I will use network visualization in the `ggplot2` framework to graphically explore the SAOMs. By using visual inference, I will learn about the importance of the many possible model parameters in SAOMs and about how they affect the visible structure of the network.

1.3 What is Visual Inference?

Viewing plots of data is an important part of exploratory data analysis (EDA) and of model diagnostics (MD). In EDA, plots guide the analyst to discovering relationships between variables in their data, while in MD, plots help the analyst determine if the model chosen is appropriate. In EDA, the analyst may notice that a covariate is strongly correlated with the dependent variable by drawing a scatterplot, leading the analyst to choose a simple linear model. But in MD, the analyst could later notice a pattern in the residuals plotted against the covariate, indicating that the variance of the dependent variable is not constant across changing values of the covariate.

These steps of EDA and MD have become so engrained in statistical practice that they are taught in introductory statistics courses. But, how can we formalized this visual discovery process?

1.3.1 A Formal Definition and Construction

The idea of visual inference was first introduced in Buja et al. (2009). In this seminal work, the authors outline two protocol for visual tests of hypotheses, the "Rorschach" the "lineup". The former allows one "to measure a data analyst's tendency to overinterpret plots in which there is no or only spurious structure," while the latter has the viewer "identify the plot of the real data from among a set of decoys [. . .] under the veil of ignorance" (Buja et al., 2009, p. 4368-9).

They begin by formalizing the definition of the set of discoverable (i.e. visible) features of a plot as a set of test statistics, denoted $T^{(i)}(\mathbf{y})(i \in I)$. The value $\mathbf{y}$ is the data in the plot, and the set I is the hard-to-define set of all possible visual features one could discover in a plot. Then they consider a general null hypotheses scenario, H_0, from which the data could have arisen. Samples are then taken from this null model and the same plot is made for the samples as was made for the data. These plots are called "null plots" while the other is the "data plot". The idea is that if an "analyst" sees a feature in the data, and also in the null plots, then the data cannot be said to come from a different scenario than H_0.

Generating samples from H_0 is not trivial. The authors provide three types of sampling available for creating the null plots: conditional sampling given a minimally sufficient statistic, parametric bootstrap sampling, and Bayesian posterior predictive sampling. (Buja et al., 2009, p. 4367). Once the null plots are generated, they are presented to an analyst through the Rorschach and lineup protocols.

In the Rorschach protocal, the analyst looks at a series of plots and describes any features or structures that stand out to them. These plots will all be null plots, but the analyst should not know this. The protocol administrator should also not know whether or not the data plot is in the series of plots. Then, these results are examined by the researcher, who determines what tendency the analyst have to "over-interpret" plot structure.

In the lineup protocol, the analyst looks at M plots that are laid out in a grid. $M-1$ of these plots will be null plots, while one is the data plot. For $M=20$, the probability of choosing the data plot from among the null plots is 0.05, providing us with an inferentially valid p-value of $\alpha = 0.05$. The lineup protocal has several special features. First, there is no need for pre-specification of the visual feature the analyst should identify. They can simple be asked to pick the most different or most special plot. Second, the analyst can self-administer the lineup once, thereby becoming a data point in their own experiment. Next, it is possible that 2 or more plots can be selected from among the M plots, as ranked data methods can be used for data analysis. Finally, the procedure can have as many repetitions as possible, as long as the analysts are independently selected and have not previously viewed the plot of the data. This can lead to extremely small p-values for inference, with the smallest possible being 0.05^K for K analysts, assuming all K selected the data plot from the lineup. Formally, the p-value of a lineup of size M evaluated by K analysts is

$$Pr(X \geq x) = 1 - Binom_{K,\frac{1}{M}}(x-1)$$

where X is the number of analysts who correctly identify the data plot, x the observed value X for an experiment, and $Binom_{K,\frac{1}{M}}(x)$ is the probability mass function of the binomial distribution with K trials and probability of success $\frac{1}{M}$ evaluated at the observed x. Type I error, the probability that a test rejects H_0 when it is true, is also formally defined as $Pr(X \geq x_\alpha)$, where x_α is the number of observers picking the data plot needed so that $P(X \geq x_\alpha | H_0)$ is less than or equal to the chosen value of α. The type II error, the probability that H_0 is not rejected when it is not true, is then $P(X < x_\alpha)$, where X and x_α are defined as above. Additionally, the power of the test given the true state, either when H_0 is true or when it is not, is the probability that the test rejects H_0. When H_0 is true, the power is $1 - Binom_{K,\frac{1}{M}}(x_\alpha - 1)$. If H_0 is not true, the power depends on the specific true state (alternative hypothesis) chosen (Majumder et al., 2013a).

The type of plots shown in visual inference will vary based on the context of the research question and null hypothesis of interest. For example, scatterplots can be shown to test for independence of two variables or for clustering; histograms can be shown to test for distribution of a variable; time series plots can be shown to test for trends; residual plots can be shown to test for presence of structure the model misses; and smoothers can be shown to test for differences in trends between groups. All of these examples are discussed in detail in Buja et al. (2009)

Additional detail to consider is the importance of varying skillsets of analysts, and the effectiveness of each analyst at selecting the data lineup. Some analysts, especially when doing experimentation, will be more visually inclined, or more analytically inclined, and these individual differences can affect the success rate of an analyst, and the rate of identification may need to be modified to account for these differences.

1.3.2 Applications of Visual Inference

There have been two distinct areas of application of visual inference since Buja et al. (2009). The first is true application of the methodology, while the second is understanding the methodology via application of the protocols. In both applications usually rely on the Amazon Mechanical Turk service (Amazon, 2010) or other similar services to show lineups to many participants from different backgrounds quickly.

In true applications, researchers have one or more alternative hypotheses and corresponding nulls on which they perform visual inference tests to show many participants of different backgrounds the lineups. One such paper, Loy et al. (2016), considers the visual inference tests for normality via lineups of Q-Q plots and compares these tests to traditional statistical normality tests. The authors found that visual inference used in this way is a more powerful test for normality than classical tests (Loy et al., 2016). In another direct application, Zhao et al. (2013)use visual inference to establish the existence of a structrue in the RNA sequence of soybean plants where different treatments and conditions alter the gene expression. Yet another application is that of Hofmann et al. (2012a), in which the authors use visual inference to determine which view of a dataset to present so that the important data properties are communicated most accurately and efficiently.

The second type of application, understanding the methodology through application is the type that I pursue in 3. One such instance of this type of application is Chowdhury et al. (2014), in which visual inference is used to better understand problems that arise when viewing high dimension, low sample size data. A second application is that of Loy and Hofmann (2015) in which the authors use visual inference to determine hierarchical model misspecification. In both of these applications, visual inference is used to discover more about the models or structures under investigation. This is how I intend to use visual inference for SAOMs. By using the lineup protocol, I hope to learn more about the effects of parameter selection on SAOMs, and order to do this, I also need to have tools to visualize the networks simulated from SAOMs.

1.4 Summary

Stochastic actor-oriented models are a rich and interesting set of models because of the complicated nature of statistical network modeling and the variety in choice of parameters available to the researcher. In the next three chapters, my aim is to fully characterize the structure and function of these models. I will do this using model visualization, the lineup protocol for visual inference, and the R package, `geomnet` that I created as a part of this work.

CHAPTER 2. STOCHASTIC ACTOR-ORIENTED MODELS: REMOVING THE BLINDFOLD

2.1 Introduction

Social networks have been studied for decades, beginning with a few foundational works, including the 1967 study, "The Small World Problem" by Stanley Milgram (Goldenberg et al., 2010). Examples of social networks include collaboration networks between academic researchers, friendship networks in a school or university, and trade networks between nations. In recent years, the study of social networks has grown in popularity due to an increase in the availability and access to social network data. There are many kinds of social networks, but there are not as many statistical models for social network data. Some network models that have been applied to social networks include the exponential random graph model and latent space models. These models, however, are only for single instance networks. If we only have one network observation, or only care about one state of a complex network in time, like a snapshot of the World Wide Web, using the well-established models for single network observating is not a problem. If, on the other hand, we have many observations of social network over time, these models may not be appropriate because they do not explicitly allow for the network to change as time passes. When studying a network over time, referred to as a *dynamic network*, we need a model that can take the time aspect of the network into account. Models for dynamic social networks have a great deal of modelling potential because of how realistic their structure can be. A social network does not form spontaneously: it evolves over time. Ties are formed and dissolved, and new actors join the social structure. Modeling the underlying mechanisms that create network changes over time is very complex but also provides potential to uncover hidden truths.

In this paper, we concentrate on one type of model for dynamic social networks: the stochastic actor-oriented models (SAOMs), introduced by Snijders (1996). These models are fundamentally different from other social network models because they allow us to incorporate network *and* actor statistics, where other models only rely on the network statistics, to model the changes in the network. Allowing the actor-level statistics to directly effect the structure of the network leads to a more practical and relevant approach to model change in a social network. In the "real" world we expect people with common interests to be more likely to form relationships, and SAOMs allow us to incorporate this intuition in the modeling process.

Unlike other network models, SAOMs are not very well understood. They are relatively new, especially compared to the classic exponential random graph models, and they are not very tractable analytically. Likelihood functions quickly very complex objects to analyze due to the dependency structure inherent in the data. Therefore, computationally more tractable solutions are used to fit estimators, and in particular, SAOMs are often fit to data using a series of Markov Chain Monte Carlo (MCMC) phases for finding method of moments estimators. In order to estimate the parameters of SAOMs, we use the software SIENA, and its `R` implementation `RSiena`, which was developed by Ripley et al. (2013). This software marks a huge contribution to the field of social network analysis, but the many moving pieces involved in parameter estimation are largely "behind the scenes" and hidden from the software user. In this paper, in order to better understand the model-fitting process, we attempt to bring SAOM fits and the fitting process out of their black boxes, by combining the principals of network visualization with those of model visualization as discussed in Wickham et al. (2015). By bringing some light to the underlying methods and structures that are behind the scenes when fitting SAOMs to network data, we aim to help researchers working with the models better understand the implications and analyses of these models.

We get into the SAOM black box by using model visualization (see Wickham et al. (2015)) to display the model in the data space, view collections of models instead of single models, and visually explore the process of fitting the SAOMs as opposed to looking at only the final output. Stochastic actor-oriented models are a prime example of a set of models that can benefit greatly from the application of model visualization. For instance, the models themselves include a continuous-time Markov chain (CTMC) that is completely hidden from the analyst in the model fitting process.

Bringing the CTMC out of the black box and into the light through model visualization can provide researchers with insights into the underlying features of the model. Furthermore, SAOMs can include a great deal of parameters to be added to the model structure, each of which is attached to a network statistic. These statistics are often somewhat, if not highly, correlated, which causes high correlation between the associated parameters in a SAOM. By visualizing collections of SAOMs, we gain a better understanding of these correlations and find ways to deal with them and rectify their effects in the model. In addition, the estimation of the parameters in a SAOM relies on a Robbins-Monro algorithm, and the convergence checks for these estimates rely on simulation from the fitted model. Again, each of these steps are largely kept in the background of the estimation process. With the help of static and dynamic visualizations we bring the hidden model fitting processes into the foreground, eventually leading to a better understanding and higher accessibility of stochastic actor-oriented models for social network analysts.

In Section 2.2, we introduce basic concepts of networks and network visualizations. In Section 2.3, we present the family of stochastic actor-oriented models for social network analysis. In Section 2.4, we combine concepts from Sections 2.2 and 2.3 in an application of the model-vis paradigm, and conclude with a discussion in Section 2.5.

2.2 Networks and their Visualizations

2.2.1 Introduction to Network Structures

Network data is of frequent interest to researchers in a wide array of fields. There are technological networks, like power grids or the internet, information networks, such as citation networks or the World Wide Web, biological networks, like neural networks, and social networks, just to name a few (Newman, 2010). Each of these examples have one thing in common: their data *structure*. There are always units of observation: the power stations, websites, neurons, and people, which we refer to throughout this paper as *nodes* or *actors*. There are also always connections of some kind between those units: the power lines, hyperlinks, electrical signals, and relationships, which we will call *edges* or *ties*. Networks might change over time, like when new websites and hyperlinks are added on the World Wide Web, or when there are new people and relationships in a friendship network. The nodes and edges themselves can also have inherent variables of interest, e.g. the institution of authors in a co-authorship network, or the number of times two authors have collaborated.

The multiple layers of network data structures pose unique problems to network analysts. Some questions that network researchers may aim to answer are: How does the strength of a tie between two nodes affect the overall structure of the network? Do node-level differences affect the formation or dissolution of edges? Which views of the data are most informative for communicating significant effects and other results of statistical analyses of the network of interest? These are, of course, just a few broad questions, and we focus here on the latter, which we aim to answer through visual exploration of network data and models.

2.2.2 Visualizing Network Data

Network visualization, also called network mapping, is a prominent subfield of network analysis. Visualizing network data is uniquely difficult because of the structure of the data itself. Most, if not all, data visualizations rely on well-defined axes inherited from the data. If variables are numerical, histograms, scatterplots, or time series plots are straightforward to construct. If the variables are categorical, bar charts and mosaic plots are available to to the researcher. If the data are spatial, there is a well-defined region in which to view information. Network data, however, are much less cut-and-dried.

There are two primary methods used to visualize networks: node-link diagrams and adjacency matrix visualization (Donald E. Knuth and John J. Watkins, 2013; Fekete, 2009). As a toy example, let us assume that we have five nodes, $\{1, 2, 3, 4, 5\}$, connected by five directed edges: $\{2 \to 4, 3 \to 4, 1 \to 5, 3 \to 5, 5 \to 4\}$ We use this toy data set to demonstrate the two visualization methods in Figure 2.1.

The first method, the node-link diagram, represents nodes with points in 2D Euclidean space and then represents edges by connecting the points with lines when there is an edge between the two nodes. These lines can also have arrows on them indicating the direction of the edge for directed networks. But because there is typically no natural placement of the points unless they have important spatial locations, a random placement of the points is used, then adjusted via a layout algorithm, of which there are many (Gibson et al., 2013).

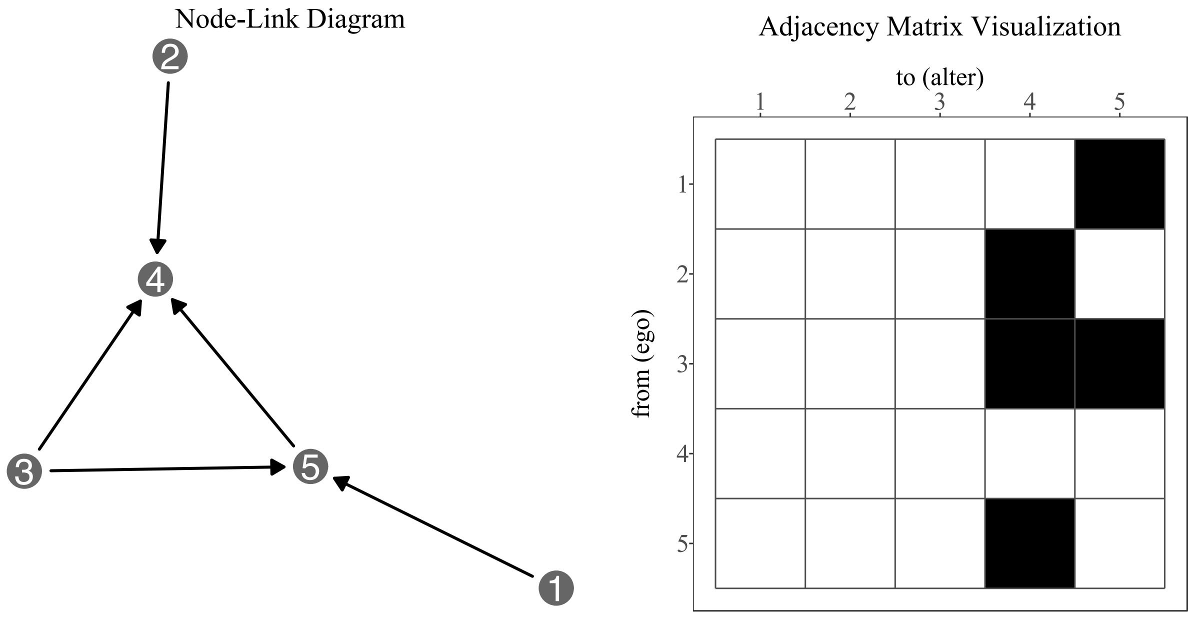


Figure 2.1: On the left, a node-link diagram of our directed toy network, with nodes placed using the Kamada-Kawai algorithm. On the right, the adjacency matrix visualization for that same network.

Some commonly used layout algorithms, such as the Kamada-Kawai layout (Kamada and Kawai, 1989) and the Fruchterman-Reingold layout (Fruchterman and Reingold, 1991), are designed to mimic physical systems, drawing the graphs based on the "forces" connecting them. In these algorithms, the edges of the network act as springs pushing and pulling the nodes in a low dimensional (usually two-dimensional) space. Another algorithm uses multi-dimensional scaling, relying on distance metric and computing a matrix whose entries represent the "distance" between every pair of nodes. There are also layout algorithms that use properties of the adjacency matrix, like its eigenstructure, to place the nodes in 2D space (Gibson et al., 2013). The node-link diagram using the Kamada-Kawai layout algorithm for our toy network is shown in Figure 2.1. Unless otherwise stated, all other node-link diagrams in this paper will use the Kamada-Kawai layout.

The second primary method for network visualization uses the adjacency matrix of the network. The adjaceny matrix of a network, $\mathbf{A}$, describes the edges of a network in matrix form. An entry A_{ij} of $\mathbf{A}$, for two nodes $i \neq j$ in the network is defined as

$$A_{ij} = \begin{cases} 1 & \text{if an edge exists } i \to j \\ 0 & otherwise \end{cases}$$

Note here that our edge variables are binary: we only consider the presence or absence of an edge. If the network has weighted edges, for example an email network where edge weights represent the number of emails sent from one person to another, the entries in the adjacency matrix are the edge weights instead of zeroes and ones. In an undirected network, $A_{ij} = A_{ji}$, but in a directed graph this is only true if there is an edge from i to j and from j to i. Thus, $\mathbf{A}$ is always symmetric for undirected networks, and is symmetric for directed networks only if every edge between two nodes is reciprocated. An adjacency matrix visualization for our toy example is also shown in Figure 2.1.

Each type of visualization comes with its own advantages and disadvantages. For example, paths between two nodes in a network are easier to determine with node-link diagrams than with adjacency matrix visualizations (Ghoniem et al., 2005). In node-link diagrams, node-level information can be incorporated into the visualization by coloring or changing the shape of the points representing the nodes, and edge-level information can be incorporated by coloring the lines, or changing their thickness, linetype, or color. Incorporating a node-level variable into an adjacency matrix visualization is not as straightforward or simple, which is more focused on edges. Adjacency matrix visualization has been found to be particularly useful when the network is very complex, dense, or large, and experimental studies have shown adjacency matrix visualization to be superior to node-link diagrames for large networks. For example, for basic perceptual tasks on networks, including node and edge count, adjacency matrix visualizations outperform node-link diagrams as the size and density of the network increases (Ghoniem et al., 2005). One drawback of the adjacency matrix visualization that Ghoniem et al. found was that edges are overrepresented for undirected graphs, due to the symmetry of $\mathbf{A}$: the edge x_{ij} for $i \neq j$ appears in $\mathbf{A}$ twice: in A_{ij} and A_{ji}, and so it also appears twice in the adjacency matrix visualization. This, however, may actually be an advantage for *directed* graphs, where exactly the correct number of edges is represented in a matrix visualization, due to the fact that the edges x_{ij} and x_{ji} are not interchangeable. A node-link diagram, however, may underrepresent the edge count if the edges x_{ij} and x_{ji} both exist and are drawn on top of one another. Ultimately, however there is not one "correct" way to visualize network information, and we will be using both the node-link and adjacency matrix visualization methods throughout this paper to explore social networks and stochastic actor-oriented models.

2.3 Stochastic Actor-Oriented Models for Longitudinal Social Networks

A Stochastic Actor-Oriented Model (SAOM) is a model that incorporates all three components of dynamic networks: edge, node, and time information. It models the change of a network over time, allowing for changes in network structure due to actor-level covariates. This model was first introduced by Snijders in 1996 (Snijders, 1996). The two titular properties of SAOMs, stochasticity and actor-orientation, are crucial to understanding networks as they exist naturally. Most social networks, even holding constant the set of actors over time, are ever-changing as relationships decay or grow in seemingly random ways, and most actors (or nodes) in social networks have inherent properties that could affect how they change their role within the network, and vice versa.

2.3.1 Definitions, Terminology, and Notation

In this paper, the term *dynamic network* refers to a network, consisting of a fixed set of n nodes, that is changing over time, and is observed at M discrete time points, $t_1, \dots, t_M$ with $t_1 \leq t_2 \leq \cdots \leq t_M$. We denote the network observation at timepoint t_k by $x(t_k)$. In the modelling process, we condition on the first observation, $x(t_1)$. The SAOM assumes that this longitudinal network of discrete observations is embedded within a continuous time Markov chain (CTMC), which we will denote $X(T)$. This process is almost entirely unobserved: we assume that the beginning of the process, $X(0)$, is equivalent to the first network observation $x(t_1)$, while the end of the process,$X(\infty)$, is equivalent to the last observation $x(t_M)$. Nearly all other parts of the process are unseen, with the exception of $x(t_2), \ldots, x(t_{M-1})$. Unlike the first and last observations of the network, these "in-between" observations do not have direct correspondence with steps in the continuous time Markov chain. Thus, the "in-between" observations are considered to be "snapshots" of the network at some point between two steps in the CTMC. The whole process $X(T)$ is a series of single tie changes that happen according to some pre-defined rate function, where one actor at a time is given the opportunity to add or remove one outgoing tie, or to not make any changes. Once an actor is chosen at random according to the rate function, it is "given" the chance to change a tie, and it tries to maximize its utility function based on the current and near future states of the network. We expand on the model description further in the subsequent sections.

2.3.1.1 The Rate Function

For the network x and each actor i in the network, the number of times that an actor i gets to change its ties, x_{ij}, to other nodes $j \neq i$ in the network is dictated by a *rate function* $\rho(x, \mathbf{z}, \boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ are the parameters in the function ρ, and x is the current network state, with covariates of interest $\mathbf{z}$. For this paper, we assume a simple rate function, $\rho(x, \mathbf{z}, \boldsymbol{\alpha}) = \alpha_m$ that is constant across all actors between observations at time t_m to t_{m+1}, thus our rate function is just a rate parameter in the overall model. In general, SAOMs can incorporate covariate values and network statistics into the model, so that each node will have a different rate of change. Other modelling scenarios allow this rate to be more flexible, e.g. a function that depends on the time period of observation, some actor-level covariates or some actor-level network statistics. In our simple model with a simple rate parameter instead of a rate function, the rate parameter dictates how quickly an actor i gets an opportunity to change one of its ties to the other nodes in the network, x_{ij}, for $j \in \{1, \ldots, n\}$ in the time period from t_m to t_{m+1}. If $j = i$, no change in the network is made. The model also assumes that the actors i are conditionally independent given their ties, $x_{i1}, \ldots, x_{in}$ at the current network state. Let $\tau(i|x, m)$ be the wait time until actor i makes its next change from its current state in the network x. Note that m indicates the number of the wave that is conditioned on in the SAOM. For any time point, T, where $t_m \leq T < t_{m+1}$, the waiting time to the next change opportunity by actor i is exponentially distributed with expected value α_m^{-1}. The conditional independence of nodes in the network given their current ties is expressed in Equation 2.1.

$$\tau(i|x,m)|x_{i1}(m),\ldots,x_{in}(m) \overset{\text{iid}}{\sim} Exp(\alpha_m) \tag{2.1}$$

The waiting time to the next change opportunity by *any* actor in the network is also exponentially distributed with expected value $(n\alpha_m)^{-1}$. The distribution of waiting time for the whole network to change, $\tau(x|m) = \sum_i \tau_i(m)|x_{i1}(m),\ldots,x_{in}(m)$ can then be written as

$$\tau(x|m) \sim Exp(n\alpha_m) \tag{2.2}$$

The parameter for the wait time for the whole network $n\alpha_m$ is the rate at which any tie change occurs. The estimation of this parameter is straightforward: a the method of moments is used to estimate the rate with the statistic

$$C = \sum_i \sum_j |x_{ij}(t_{m+1}) = x_{ij}(t_m)|$$

which is the total number of changes from observation at time t_m to the observation at time t_{m+1}.

2.3.1.2 The Objective Function

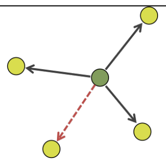
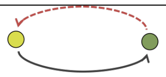
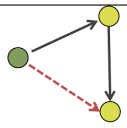
Because of the conditional independence assumptions given in Equation 2.1, we can consider the objective function for each node separately, as only one tie from one node is allowed to change at a time. The node i, which is the node that is chosen to change at the current time point, is called the *ego* node. It has the potential to interact with all other nodes in the network, $j \neq i$. These nodes j, are referred to as *alter* nodes, or simply *alters*. These nodes are acted upon by the ego node, and they only act when they become the ego node at a subsequent time point in the CTMC. For the ego node, i, in the current network state x, its objective function, which it tries to maximize, is written as

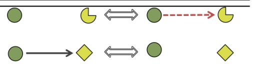
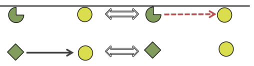
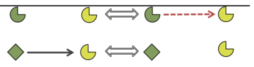
$$f_i(\boldsymbol{\beta}, x) = \sum_k \beta_k s_{ik}(x, \mathbf{Z}), \tag{2.3}$$

for $x \in \mathcal{X}$, the space of all possible directed networks with the n nodes, and $\mathbf{Z}$, the matrix of covariates. The vector $\boldsymbol{\beta}$ contains the parameters of the model with corresponding network and covariate statistics, $s_{ik}(x, \mathbf{Z})$, for $k = 1, \dots, K$. Given the ego node, i, there are n possible steps for the actor i to take: either one of all current ties $x_{ij} = 1$ will be destroyed, a new tie will be created thatis currently $x_{ij} = 0$, or no change will occur.

The parameters, $\boldsymbol{\beta}$, correspond to various actor-level network statistics, $s_{ik}(x)$. According to Snijders (2001) (p. 371), there should be at least two parameters included in the model: β_1 for the outdegree of a node, and β_2 for the number of reciprocal ties held by a node. These effects should seem familiar to readers used to working with the classical exponential random graph model (ERGM) for networks. The outdegree represents the propensity of nodes with a lot of outgoing ties to form more outgoing ties (the "rich get richer" effect), and the reciprocity parameter measures the tendency of outgoing ties to be returned within a network. The statistics corresponding to these effects are written in terms of the edge variables x_{ij}, for $i \neq j$. In the `RSiena` software that we use to fit the SAOMs, there are over 80 possible parameters to add to the model. The formulas for the effects are provided in Ripley et al. (2017). The parameters, β_k, in the model can be split up into two groups: first, the structural effects, whose estimation depends only on the structure of the network, like the outdegree and reciprocity parameters mentioned above. The parameters are included when the researcher hypothesizes that they will model underlying mechanisms of network change. They hope to answer questions such as, “How does the existing network structure influence change in the network?” and “How do the behavior and characteristics of the nodes influence change in the network?” The second set of effects are referred to as the actor-level or covariate effects. These covariate effects also depend on the structure of the network, with the additional inclusion of node-level covariates of interest. The covariate effects are written in terms of the tie variables x_{ij}, but also in terms of the covariates, $\mathbf{Z}$. A table of some possible structural and covariate effects is given

Table 2.1: Some of the possible effects to be included in the stochastic actor-oriented models in RSiena. There are many more possible effects, but we only consider a select few here. For a complete list, see the `RSiena` manual (Ripley et al. 2017).

Structural Effects		
outdegree	$s_{i1}(x) = \sum_j x_{ij}$	
reciprocity	$s_{i2}(x) = \sum_j x_{ij}x_{ji}$	
transitive triplets	$s_{i3}(x) = \sum_{j,h} x_{ij}x_{jh}x_{ih}$	

Covariate Effects		
covariate-alter	$s_{i4}(x) = \sum_j x_{ij}z_j$	
covariate-ego	$s_{i5}(x) = z_i \sum_j x_{ij}$	
same covariate	$s_{i6}(x) = \sum_j x_{ij}\mathbb{I}(z_i = z_j)$	

in Table 2.1. For a complete list of the network and covariate statistics that can currently be included in the objective function, see Ripley et al. (2017).

When node i is given the chance to change a tie, it attempts to maximize the value of its objective function $f_i(\boldsymbol{\beta}, x)$ as well as a random element, $U_i(x)$, to account for unknown attraction between nodes. The additional random element is included to account for any random, unexplainable change in the network ties. Snijders (2005) recommends that the $U_i(x)$ be random draws from a type 1 extreme value distribution. This distribution, which is also known as the log-Weibull distribution, has probability distribution function of:

$$g(u|\mu, \sigma) = \frac{1}{\sigma} \exp\left\{-\left(\frac{u-\mu}{\sigma} + \exp^{-\frac{u-\mu}{\sigma}}\right)\right\}. \tag{2.4}$$

using μ for the mean parameter and σ for the scale parameter.

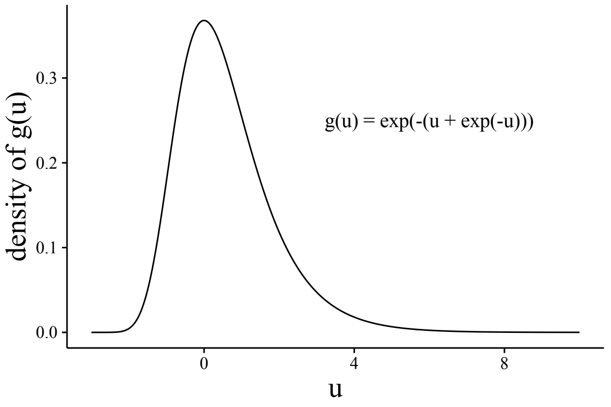


Figure 2.2: The probability distribution function for the type 1 extreme value distribution, also known as the log-Weibull or Gumbel distribution with location parameter $\mu = 0$ and scale parameter $\sigma = 0$.

The probability density function for the type 1 extreme value distribution is shown in Figure 2.2. For mean $\mu = 0$ and scale $\sigma = 1$ as Snijders (2005) suggests is convenient because it leads to a simple probability formula for the probability that actor i chooses to change its tie to actor j that can be written *only* in terms of the objective function. Let $p_{ij}(\boldsymbol{\beta}, x)$ be the probability that actor i chooses to change its tie to actor j. Next, we write the network x in its potential future state, $x(i \rightsquigarrow j)$, where the tie x_{ij} has changed to $1 - x_{ij}$. Then, the probility that the tie x_{ij} changes is

$$p_{ij}(\boldsymbol{\beta}, x) = \frac{\exp\left\{f_i(\boldsymbol{\beta}, x(i \rightsquigarrow j))\right\}}{\sum_{h \neq i} \exp\left\{f_i(\boldsymbol{\beta}, x(i \rightsquigarrow h))\right\}} \tag{2.5}$$

When $i = j$ in p_{ij}, the numerator represents the exponential of the value of the objective function when evaluated at the current network state. When the value of the objective function is high at the current state, the probability of not making a change in a microstep is also high. In the CTMC, when actor i may make a change, it chooses which tie $x_{i1,\dots,x_{in}}$ to change at random according to the probabilities $p_{ij}(\boldsymbol{\beta}, x)$. The objective function and the resulting values of p_{ij} are combined with the rate function to fully describe the CTMC that is used to model network change in a SAOM.

2.3.1.3 Continuous Time Markov Chain (CTMC)

In the continuous-time Markov chain literature, see for instance Yin and Zhang (2010), chains are characterized by their *generator* or *intensity* matrix $\mathbf{Q}$. This matrix describes the rate of change between two states of the CTMC process, and the rows of this matrix always add to zero. For directed networks with binary edge variables like the ones we will be working with, there are a very large number of possible states for a directed network with n nodes. We denote the state space as $\mathcal{X}$, a set which contains $2^{n(n-1)}$ states: there are two possible states for an edge, $\{0, 1\}$, and there are $n(n-1)$ edge relationships because the network is directed and we exclude self-ties. The intensity matrix for a CTMC in a SAOM is then a square matrix of dimension $2^{n(n-1)} \times 2^{n(n-1)}$. Only one tie changes at a time in the CTMC, resulting in $n(n-1)$ reachable states from the current network state. Thus, the intensity matrix $\mathbf{Q}$ is very sparse, with only $n(n-1)+1$ non-zero entries in each row. Note that $n(n-1)$ of these entries represent the possible states that are one edge different from a given state, while the additional non-zero entry is for the state to remain unchanged. All other entries in a row are structural zeroes because those network states cannot be reached from the current state in a single change.

The two pieces of a SAOM, the rate function/parameter and the objective function, each contribute to the entries of the intensity matrix to describe the rate of change between two network states. The entries of $\mathbf{Q}$ are defined as follows: let $b \neq c \in \{1, 2, \dots, 2^{n(n-1)}\}$ be indices of two different possible states of the network, $x^b, x^c \in \mathcal{X}$. Then the bc^{th} entry of Q is:

$$q_{bc} = \begin{cases} q_{ij} = \alpha_m p_{ij}(\boldsymbol{\beta}, x^b) & \text{if } x^c \in \{x^b(i \rightsquigarrow j) |\text{ any } i \neq j \in \{1, \dots, n\}\} \\ 0 & \text{if } \sum_i \sum_j |x^c_{ij} - x^b_{ij}| > 1 \\ -\sum_{i \neq j} q_{ij} & \text{if } x^b = x^c \end{cases}$$

Thus, the rate of change between any two states, x^b and x^c, that differ by only one tie x_{ij}, is the product of the rate at which actor i gets to change a tie and the probability that the tie that will change is the tie to node j. This matrix $\mathbf{Q}$ is the foundation for estimation of a SAOM.

2.3.2 Fitting Models to Data

To fit a SAOM to observations of a dynamic network, we use the package `RSiena` (Ripley et al., 2013). This package uses simulation methods to estimate parameter values using either the method of moments or maximum likelihood estimation. In this paper, use the method of moments estimation because the theory behind it was established in Snijders (1996), while the maximum likelihood estimation methods were not fully established until Snijders et al. (2010c), though `RSiena` contains capabilities to use maximum likelihood estimation. We also use the score function method for estimating the derivatives of the expected values, as opposed to the finite differences method, both of which are outlined in detail in Snijders (2016).

For the score function method, the SIENA software uses a Robbins-Monro algorithm (see Robbins and Monro (1951)) to estimate the solution of the moment equation

$$E_{\theta}S = s_{obs}$$

where θ is the vector of rate and objective function parameters, and s_{obs} is the observed vector of model statistics, S. The entire algorithm is provided in Snijders (2016).

There are three phases in the the SIENA algorithm, as described in Ripley et al. (2017); Snijders (2016). The first phase performs initial estimation of the score functions for use in the Robbins-Monro procedure for method-of-moments estimation. The second phase carries out the Robbins-Monro algorithm and obtains estimates of the parameter values through iterative updates and simulation from the CTMC at current parameter values. The third phase uses the parameter vector estimated in phase two to estimate the score functions and covariance matrix of the parameter estimate, and also carries out convergence checks. In each of the the first two phases, the estimation procedure also uses "microsteps" that simulate from the model as it exists in its current state in order to update either the score functions or the parameter estimates. These simulated microsteps are observed instances of the continuous-time Markov chain that is the backbone of the stochastic actor-oriented model. In Section 2.4, we further explore these phases in the SIENA method-of-moments algorithm through visualization, bringing them out of the "black-box" and into the light.

2.3.3 Model Goodness-of-Fit

The `RSiena` software that fits the models to data also includes a goodnes-of-fit function for examining model fit, `sienaGOF()`. This function "assess[es] the fit of the model with respect to auxiliary statistics of networks" (Ripley et al., 2017, p. 53). Examples of auxiliary statistics include the out- or indegree distribution on the nodes, with the option for users to input their own statistics to examine. The goodness-of-fit is evaluated as follows:

1. The auxiliary statistics are computed on the observed data and on N simulated observations from the model. Typically, $N = 1000$.

2. The mean vector and covariance matrix of the statistics on the simulations from the model are computed.

3. The Mahalanobis distance from the observed statistics to the distribution of the simulated statistics is computed using the mean and covariance found in step 2.

4. The Mahalanobis distance from each of the N simulations to the same distribution is computed, and the Mahalanobis distance of the observed data is compared to this distribution of distances.

5. An empirical p-value is found by computing the proportion of simulated distances found in step 4 that are as large or larger than the Mahalanobis distance from the data. A SAOM is thus considered a good fit if p is large.

The `plot.sienaGOF()` function allows us to visualize this fit. This function draws a box plot and a violin plot at each value of the statistic of interest observed in the simulations: on the x-axis, the out(in)degrees observed in the data $(0, 1, 2, 3, \dots)$, and on the y-axis, the cumulative number of times that out(in)degree value appears in the simulations and the data. In order to compare the distribution of the counts of nodes with the specified degree as calculated on the simulated networks to the counts observed in the true data, red points connected by red lines representing the observed values are superimposed on the boxplots. If the red points lie "well within" the simulated values, the model is a good fit to the data. This plot is not shown because it is not intuitive for understanding network data. Boxplots separated by the many outdegree values observed do not communicate *how* the nodes are connected, just that some have more connections than others. In order to understand the fit of the model, we should try to understand how well the model captures the *overall* structure of the network, not just one or two summary measurements of that structure. In Section 2.4.2, we propose a new way of visualizing goodness-of-fit that uses the traditional node-link diagram to visualize the entire network instead of numerical summaries of the the network.

2.3.4 Example Data

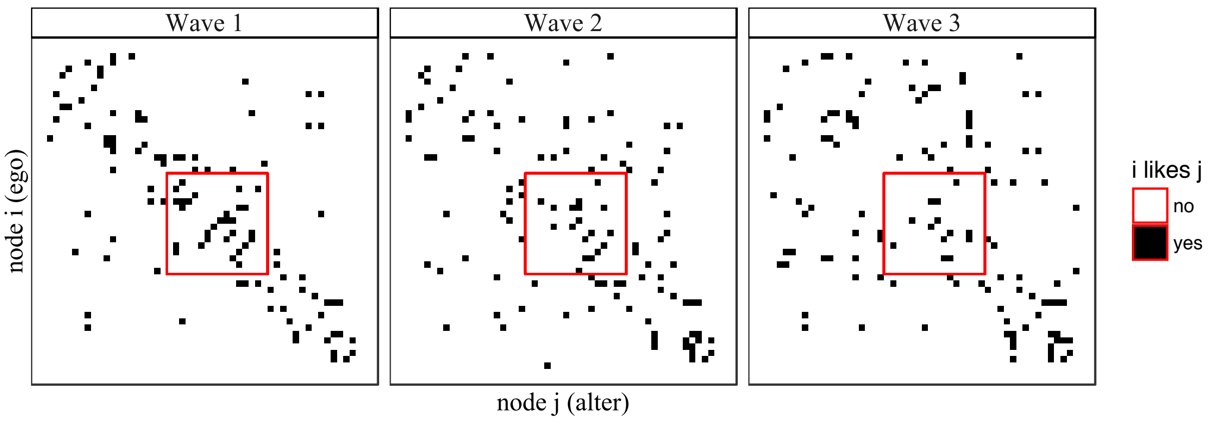


Figure 2.3: A visualization of the adjacency matrices of the three waves of network observations in the "Teenage Friends and Lifestyle Study" data. The subset we will be using is outlined in red.

To guide our visual exploration of stochastic actor-oriented models, we use two data sources. The first is a subset of the 50 actor dataset from the "Teenage Friends and Lifestyle Study" that is provided on the `RSiena` webpage. These data come from Michell and Amos (1997), and we chose to only work with a subset of the data to make network visualizations less busy and to make any changes in the network more noticeable. To determine which subset to select, we visualized all waves of the full network using the adjacency matrix visualization approach, which we show in Figure 2.3. This adjacency matrix visualization is different from the one in Figure 2.1 because it does not show the node IDs on the axes. The network in Figure 2.3 is much larger than our toy data example,

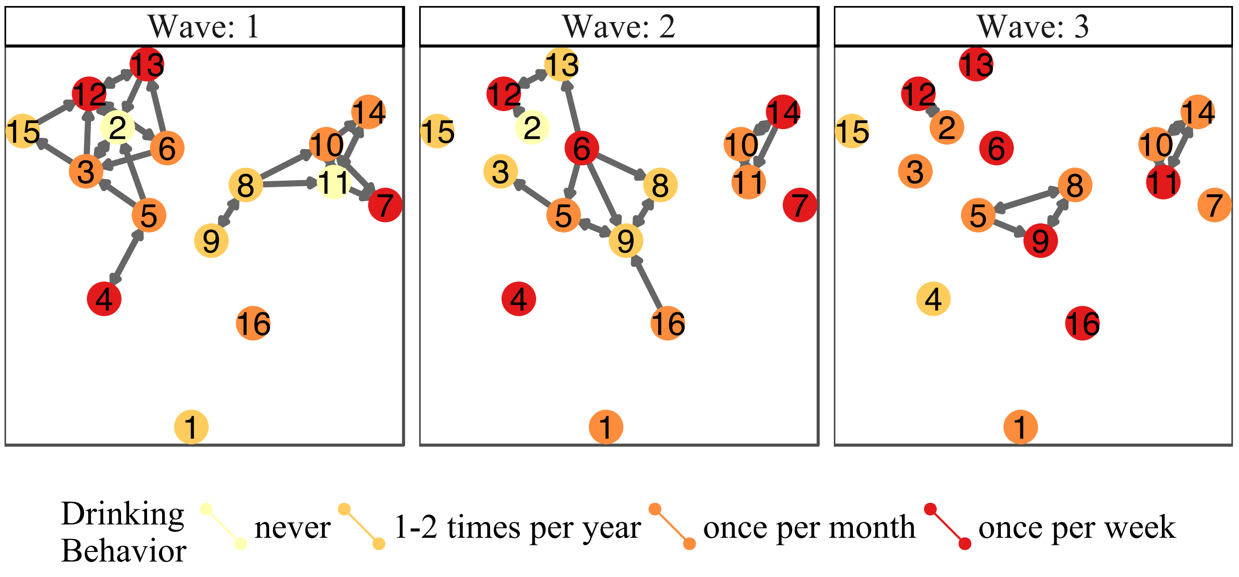


Figure 2.4: The smaller friendship network data we will be modelling throughout the paper. In the first wave, we can see there are two large, separate friend group. By the second wave, three students with heavier drinking behavior have separated from their group, while the others become members of the other group. By the third wave, the large group has almost completely broken up. We want to capture these changes with our SAOMs.

so we remove the node labels to remove clutter. In all three adjacency matrices, the ego and the alter nodes are ordered by node ID, 1-50, which were determined arbitrarily by the relationships in wave 1. The subset we selected is outlined in red in the visualization. This subset contained actors 20 through 35 and the ties between them, as well as the drinking behavior of each actor at each of the three waves. This specific subset was chosen because it showed somewhat higher connectivity than other subsets, as we've emphasized in the visualizations of the three network observations in Figure 2.3. For model fitting, we condition on wave 1 and estimate the parameters of the models from the second and third waves. We will also be working with one actor level categorical covariate, drinking behavior. This variable has five values in the original data: (1) does not drink, (2) drinks once or twice a year, (3) drinks once a month, (4) drinks once a week, and (5) drinks more than once a week. The network and the actor covariate values are visualized using a node-link diagram in Figure 2.4. In the node-link diagram, the nodes are colored according to the drinking behavior of that student. Over time, we can see that the students tend to drink more and become increasingly isolated into smaller groups. An analysis of this type of data with a SAOM should capture these dynamics in a way that allows the researcher to draw conclusions about the nature of the network and behavioral forces at play.

The second data example we use is a collaboration network in the United States Senate during the 111^{th} through 114^{th} Congresses. These sessions of congress correspond to the years of Barack Obama's presidency, from 2009-2016.[1]. In this network, ties are directed from senator i to senator j when senator i signs on as a cosponsor to the bill that senator j authored. There are (somewhat surprisingly) many hundreds of ties between senators when they are connected in this way, so we simplify the network by computing a single value for each senator-senator collaboration called the *weighted propensity to cosponsor* (WPC). This value is defined in Gross et al. (2008) as

$$WPC_{ij} = \frac{\sum_{k=1}^{n_j} \frac{Y_{ij(k)}}{c_{j(k)}}}{\sum_{k=1}^{n_j} \frac{1}{c_{j(k)}}} \tag{2.6}$$

where n_j is the number of bills in a congressional session authored by senator j, $c_{j(k)}$ is the number of cosponsors on senator j's k^{th} bill, where $k \in \{1, \dots, n_j\}$, and $Y_{ij(k)}$ is a binary variable that is 1 if senator i cosponsored senator j's k^{th} bill, and is 0 otherwise. This measure ranges in value from 0 to 1, where $WPC_{ij} = 1$ if senator i is a cosponsor on every one of senator j's bills and $WPC_{ij} = 0$ if senator i is never a cosponsor any of senator j's bills.

Because we require binary edges for our models, we focus only on very strong collaborations. For our senate collaboration networks, x, edges are defined as

$$x_{ij} = \begin{cases} 1 & \text{if } WPC_{ij} > 0.25 \\ 0 & \text{if } WPC_{ij} \leq 0.25. \end{cases}$$

The networks we constructed for the four senates during President Obama's administration are shown in Figure 2.5. In Section 2.4, we fit several stochastic actor-oriented models to these data sets use those models to guide our further exploration of SAOMs.

[1]Details of how this data can be downloaded are provided by Franois Briatte at https://github.com/briatte/congress

2.4 Model Visualizations

Every good data analysis includes both numerical and visual summaries of the data, so why restrict model description and diagnostics to numerical summaries? The concept of model visualization was developed to complement traditional model diagnostic tools. Typically, numerical summaries such as R^2 are used to assess model fit, and the occasional visualization, like a residual plot, are used to determine how well the model fits the data. Wickham et al. outline three separate ideas, each of which are can be referred to simply as the "model": the model family, the model form, and the fitted model. The latter is primarily what one thinks of first when considering a model in a data analysis, where a specified model is fit to data, and parameter estimates and other numerical summaries, such as R^2 are reported. In the context of SAOMs, the fitted model contains the form of the rate and objective functions, the estimated rate parameters, and the estimated objective function parameters. The model form describes the the model *before* the fitting process, defining which parameters are in the model within the context of the larger model family. In SAOMs, the model form includes description of the rate and objective functions and the variables therein that describe how the network evolves over time. Finally, the model family is the broadest description of the model. This is the type of model that you wish to fit to the data, and is chosen based on the problem, data, and knowledge at hand. For example, we chose to use a SAOM to model network data over an exponential random graph model (ERGM) because we believe that actor-level variables effect network structure and formation, and we wanted to model the network changes over time.

The model family, the model form, and the fitted model can each be visualized according to the three principals of model visualization: we can view the model in the data space, visualize collections of models, and explore the process of fitting the model, not just the end result. Since we have already decided on our model family, SAOMs, we now shift our focus to the fitted model and the model form. Specifically, we want to learn more about how the model form we choose affects the fitted model by using our example data sets and our visualization toolbox. We begin by introducing the five models that we fit to our example data. Next we use the five models to guide our visual explorations of SAOMs. We first use novel tools and ideas to view a SAOM in the data space of a dynamic network. We then explore collections of the same models fit many times to the example data to see how the simulation processes in `RSiena` affect the model fits. Finally, we look behind the scenes and into the individual steps of the continuous time Markov chain to learn more about how this "hidden data" mechanism works and how it results in a fitted model.

2.4.1 The Models

We first consider the 16 actor subset of the teenage friends and lifestyle data available on the `RSiena` website (Ripley et al., 2013). To this data, we fit three different SAOMs. Each SAOM used a simple rate function, α_m, and an objective function with two or three parameters. The first model, M1, contains the absolute minimum number of parameters in the objective function $f_i(x)$:

$$f_i(x)^{M1} = \beta_1 s_{i1} + \beta_2 s_{i2},$$

where s_{i1} is the density network statistic and s_{i2} is the reciprocity network statistic for actor i at the current network state x. The second and third models, M2 and M3, contain one additional parameter each in the objective function which were determined by a Wald-type test provided in the `RSiena` software to be significant, with p-values less than 0.05 (Ripley et al., 2016). The M2 model contains an actor-level covariate parameter, and the M3 model contains an additional strutural effect in the objective function.

$$\begin{aligned} f_i(x)^{M2} &= \beta_1 s_{i1} + \beta_2 s_{i2} + \beta_3 s_{i3} \\ f_i(x)^{M3} &= \beta_1 s_{i1} + \beta_2 s_{i2} + \beta_4 s_{i4}, \end{aligned}$$

where $s_{i3} = \sum_{j \neq h} x_{ij} x_{ih} x_{hj} \mathbb{I}(z_i = z_h \neq z_j)$, and $s_{i4} = |\{j : x_{ij} = 0, \sum_h x_{ih} x_{hj} \geq 2\}|$. These statistics are known as the number of jumping transitive triplets and the number of doubly achieved distances two effect, respectively. The first statistic emphasizes triad relationships that are formed between actors from different covariate groups, while the other emphasizes indirect ties between actors. The covariate groups are determined by the student's drinking behavior, the values of which are numeric and mean-centered. The four values in our data are {never, 1-2 times per year, once per month, once per week}, which after being converted to numeric and mean-centered become $\{-0.8125, -1.8125, 0.1875, 1.1875\}$. The two additional effects are visually represented and further described in Figure 2.7 and Figure 2.8, respectively.

In Section 2.4.3.3, we also fit models M1, M2, and M3 to the senate collaboration data for comparison. For fitting M2 to the senate data, we use the number of bills authored by each senator as the node covariate for the jumping transitive triplet variable. In terms of the senate data, then, the value of β_3 should dictate how willing a senator is to coauthor a bill on which the author of the bill has a different level of authorship, assuming there is an intermediary between the two who has the same authorship level as the first.

Due to the intractability of SAOMs, it is difficult to know for certain how to interpret a fitted value of a parameter. We can make educated guesses based on the definition of the effect and the sign of the fitted value, but a direct interpretation is not always possible. But, by exploring these models more visually, we aim to understand these effects, their interpretations, and the model fitting process better.

2.4.2 View the model in the data space

The first way we hope to better understand stochastic actor-oriented models is by viewing the model(s) in the data space. In Wickham et al., they happen to define the *data space* as "the region over which we can reliably make inferences, usually a hypercube containing the data" (Wickham et al., 2015, p. 206). But what does this definition mean for network data? For dynamic social networks, there are a few different data "spaces":

1. the actors and their corresponding covariates,
2. the edges and their correpsonding variables that describe the relationships between the nodes, and
3. the time, both the continuous unobserved time and the discrete observed time points, over which the network evolves

These three data pieces can be visualized together in various ways. The traditional node-link visualization uses one of many algorithms to layout the actors as points in 2D space, then draws segments connecting the points in 2D if there is an edge between two nodes, and draws nothing otherwise. The time aspect can be visualized by drawing each network observation in time and placing the observed timepoints side-by-side.

Because longitudinal network data consist of three different "spaces" of data, viewing the model in the data space can depend on which aspect of the *model* and the *data* we are interested in viewing. Incorporating data covariates into the network structure allows us to assess whether the ties between nodes are affected by how nodes behave over time. In this instance, we would want to view predictions over time. A SAOM can also model behavior change over time, taking both the node and edge information into account. In this case, a plot of predicted covariate values over time would put the model into the time and node data space. Most likely, however, is that we would want to view all of the data spaces simultaneously.

One tool that can bring the node, edge, and time data spaces together in this way is the `R` package `geomnet` (Tyner and Hofmann, 2016a). Different visual features in the node-link diagram can be tied to the underlying node or edge data. The color, size, and shape of the points can be used to represent variables in the node data, while the color, linewidth, and linetype of the lines between points can be used to represent the edge variables. In a social network, node data might be age, gender, and occupation of the person in the network, and edge data might be length of connection between two people, how the people first met (school, work, church, etc.), and how often they interact, and we can view the network at different timepoints side-by-side to see its evolution. Pulling all of this information together with `geomnet` allows the entire data space to be viewed at once.

To demonstrate, we use `geomnet` to visualize the connections in the 111^{th} United States Senate at two different points: when Hillary Clinton was in the senate, and after she left to become Secretary of State. Clinton was only in the 111^{th} senate for 17 days, from January 3, 2009, to January 20, 2009, when she was in the middle of her second term as senator from New York. In that time, she authored two bills and was a cosponsor on 17 other bills. With Clinton included in the node-link diagram, the senate looks much more highly collaborative than it does without her in the diagram. We can compare the number of bills authored throughout the senate by mapping the size of the node to the numer of bills authored by that senator. We also map shape to sex, and keep color mapped to party of the senator. In addition, we can see the strength of the tie by mapping the linewidth of the edge to the WPC value between the two senators. In this single visualization, we have viewed node information (number of bills authored by a senator and the sex and party of that senator), edge information (the direction and strength of ties between two senators), and time (before and after Clinton left the senate).

Another way to view the model in the data space is through simulation from the model. No single network alone simulated from a SAOM is going to look like the data or represent the model, just as no single value simulated from the standard normal distribution will look like a bell-shaped curve. Statisticians would prefer to look at a sample, or at least a summary statistic or two, so it would therefore better to visualize many simulations together. From a statistician's point of view, a stumbling block with statistical network models generally is the lack of an "average network" or "expected network" value. Statisticians frequently rely on averages and expected values in data analyses, but statistical network models, especially those as complex as SAOMs, lack a single, intuitive expected value measure. We could talk about expected values of parameters, but the parameters can be hard to interpret. Expected values of parameters are important, but if they cannot directly tell us anything about our dependent variable (networks in this case), they lose some value to us. Furthermore, there is no way to talk about the expected value of an observation simulated from a statistical network model. How then, can we arrive at an "average" network? We answer this question through visualization. For network data, one way we view an "average" network is through a summary network drawn using the traditional node-link diagram. In Figure 2.11, we show an *average network* created with 1,000 simulations of the second wave of the network from Model 1. To make this average network, we first simulated 1,000 wave 2 and wave 3 observations of our small friendship example data from model M1, for which parameters had previously been estimated. We then combine the 1,000 instances of wave 2, and count up the number of times each edge appears in a simulation. Then, we combine these 1,000 networks into a single network with edgeweight equal to the proportion of time that edge appears in our 1,000 simulations. This weighted edgelist is the network we draw using the node-link diagram. An edge is only drawn in the average network if it appears in more than 5% of the simulations (in at least 51 of the 1000 simulations), with edges that appear more frequently emphasized by thicker linewidths and a darker color, representing the proportion of times appeared in the model simulations. On either side of the average network in Figure 2.11, we show the actual data, wave 1 on the left, and wave 2 on the right. We can see that the structure of the average network is much more similar to the first wave than to the second wave. However, the simulations are supposed to represent the second wave of data, which is shown on the right in Figure 2.11. This is an indication that the simple model, M1, is doing a very poor job of capturing the change mechanism from the first to the second wave of observation. The average network can thus be used to help determine model goodness-of-fit. Because the the average network looks more like the first wave than the second wave, we can use the visualization in Figure 2.11 as evidence of poor model fit.

Another potential goodness-of-fit visualization that places the model in the node and edge data space is a lineup like those proposed in Buja et al. (2009). A *lineup* “asks the witness to identify the plot of the real data from among a set of decoys, the null plots, under the veil of ignorance” (Buja et al., 2009, p. 4369). It can be thought of like a police lineup, where the “suspect” is in a lineup among several innocent lookalike fillers, and a witness picking the suspect out of the lineup is considered evidence against the suspect. In data and model visualization, the “suspect” is a plot of the true data, while the “filler” is composed of several plots of mock data, simulated from a hypothesized model. If the true data stands out among the simulated data, that is taken as evidence of poor model fit, whereas if the true data is difficult to identify among the simulated data, that is taked as evidence of good model fit. An application of the lineup protocal can be found in Hofmann et al. (2012b), where the authors examine, for instance, the differences between polar and cartesian coordinates for plotting categorical data, and density plots and box plots for determining distributional differences. They pose questions to experiment participants such as, "Which plot is most different from the others?" for the first example and, "In which plot is the blue group furthest to the right?" for the distributional differences. The data visualizations examined in Hofmann et al. (2012b) are less complex than a node-link network diagram. What questions should we ask for network data visualizations? Asking participants to identify the most different plot may be difficult. In the network lineup shown in Figure 2.12, the second wave of the small friendship network is shown among five simulated networks from model M1 using parameter values estimated from the data. What makes these plots "different"? It seems possible to argue for any one of the six plots in Figure 2.12 as most different. So, we guide participants to look at the overall structure of the graphs to determine which has the most and least complex structure. The least complex plot, number six, has no triplets, while the most complex plot, number three, has three triplets, and in fact, plot three is the data. We have found in an experiment where all the node-link diagrams shown are based on simulated data, one observation from an "alternative" model and the rest from a "null" model, the triangular shape of the triplets stands out the most to participants. We can use the lineup protocol to help better understand SAOMs because it allows us to view the model in the data space by placing observations the model side-by-side with the data and examining the differences. This can also help us determine the significant structural effects, if any, of the parameters in the model on observations simulated from the model. The triangular shape mentioned above becomes more prevalent when a transitive triplet parameter in included in a model, but it does not always cause triplets to form in the simulated data. This requires more investigation with larger data sets and more complex models, but is promising for the development of additional goodness-of-fit measures for network models.

2.4.3 Visualizing collections of models

There are many possible ways to collect models together. We could look at the same models fit to different data, different models fit to the same data, or because of the nature of SAOMs, we could fit the same models to the same data many times to see how the simulations change. For the SAOMs, we decided that there were four collections that were most important:

1. the collections resulting from exploring the space of all possible models;
2. the collections we get when varying model settings;
3. the results from fitting the same model form to different data;
4. the results from fitting the same model to the same data many times.

We chose these four collections because they each explore something different about SAOMs. The first takes the many dozens of parameters available to include in a SAOM into account, which easily translates into the second by showing how those many parameters affect the fitted models. Then, we look at how the same model looks when fit to different sets of dynamic network data. Finally, we look at the results from fitting the same model to the same data because the MCMCs and CTMCs that make up a SAOM lead to different parameter estimates every time, so it important to see how the results can vary.

2.4.3.1 Exploring the space of all possible models

The `RSiena` manual contains over eighty possible effects to include in the model. In order to select parameters to include in the models for our example data, we searched through the possible effects available to model given the data structure to find *significant* effects. We tested for significance using the Wald-type tests built into `RSiena` for one-at-a-time effects testing. We start with the outdegree and reciprocity measures as the foundation of the models we fit, then add one evaluation effect, fit the model, test the additional effect for significance, and repeat for all possible parameters to add to the model. We performed this procedure for both the small friendship and the senate collaboration data. The results for significant effects are shown in Figure 2.13. We see in both the friendship data and the senate data results that most of the significant effects have absolute value less than ten. In addition, the p-values for the effects from the friendship data are more spread out than the p-values for the senate data, which are concentrated at about 0.02 or less. This may suggest that larger data sets tend to result generally in smaller p-values, just like a larger sample size results in smaller p-values in a t-test.

2.4.3.2 Varying model settings

We have varied model settings already by choosing models M1, M2, M3 to fit to our small friendship network data set. In Section 2.4.3.4, we fit these models to the data 1,000 times, and in this section, we explore simulations from these three models given the mean values of from the 1,000 fitted parameter values as the parameters in our models. From each of these three models using the means of parameter estimates as our fixed parameter values, we simulated 1,000 observations from each of the three models. In this process, we condition on the first friendship network observation, and the second and third observations are simulated from the SAOM models with the given parameter values. From these simulations, we first create a visualization that represents an average network.

To create the average network visualization shown in Figure 2.14, we follow the same procedure as in Section 2.4.2, counting occurrences of each possible edge in the simulations, resulting in a summary network with weighted edges representing the number of times an edge appeared in the simulated wave 2 when simulating from the SAOM 1,000 times. As in Figure 2.11, edges only appear in the average networks if they appear more than 5% of the time in the simulations. In Figure 2.14, we show the "average" network from the three models we fit and the first and second waves of data. Comparing the three averages to waves 1 and 2, we see that they have very similar structure to wave 1. Model 2, which included the transitive triplet parameter, seems to have created a larger connected component overall than models 1 and 3. In particular, if we look at the group of nodes $\{10, 11, 14\}$, we see they are very strongly connected within the three average networks, and they are completely separate from the other nodes in the true wave 2. None of the three average networks show node 16 gaining ties as it does in wave two, nor do they show nodes 4 and 7 becoming isolated. In Model 2, however, the ties to node 7 appear much weaker than in Model 1 or Model 2, suggesting that of the three, Model 2 may be the best fit for our data.

Table 2.2: The means (standard deviations) of parameter values estimated from repeated fittings of $M1, M2, M3$ to the small friendship network and the senate collaboration network. Each model was fit 1,000 times to the friend data, while each model was fit 100 times to the senate data.

	Friendship Data			Senate Data		
	M1	M2	M3	M1	M2	M3
α_1	4.660 (0.059)	5.176 (0.068)	4.712 (0.060)	3.344 (0.016)	3.349 (0.016)	3.340 (0.016)
α_2	1.930 (0.026)	2.017 (0.028)	1.979 (0.027)	2.480 (0.017)	2.487 (0.015)	2.483 (0.014)
α_3	–	–	–	2.221 (0.017)	2.227 (0.017)	2.224 (0.016)
β_1	-3.597 (0.033)	-4.104 (0.038)	-3.589 (0.035)	-4.979 (0.027)	-4.993 (0.025)	-4.987 (0.021)
β_2	4.149 (0.050)	4.277 (0.052)	4.230 (0.050)	4.954 (0.046)	4.974 (0.040)	4.970 (0.035)
β_3	–	3.209 (0.053)	–	–	-1.175 (0.789)	–
β_4	–	–	-7.582 (1.746)	–	–	-1.048 (0.486)

2.4.3.3 Fitting the same model to different data

As mentioned in Section 3.3.4, we fit the models M1, M2, and M3 to both the small friendship network and the senate collaboration network. We fit each model to the friendship data 1,000 times, and to the senate data 100 times. The means and standard deviations of the parameter estimates for each combination of model and data are given in Table 2.2.

Looking at Figure 2.15 and Table 2.2, we see a few patterns in the estimates from both models. First, we can see in both the table and the density of the estimates that the same relationship between the outdegree parameter, β_1, and the reciprocity parameter, β_2. In both data sets and across all three models, the estimates of β_1 are all negative and hover between -5 and -3, while the estimates of β_2 are all positive and hover between four and five. This suggests that in both data sets, nodes are *discouraged* from forming ties that are outgoing without being reciprocated, while also being *encouraged* to form outgoing ties to nodes that tie to them. In both data sets, people seem to want to have reciprocated relationships: teenage girls want be friends with other girls that reciprocate their friendship, and senators want to coauthor bills with senators who have also been coauthors on their bills. We explore the relationship between β_1 and β_2 further in Section 2.4.3.4.

The inclusion of β_3, the jumping transitive triplet parameter, for the friendship data had a noticeable effect on the other parameters in the model. The same cannot be said for the inclusion of β_3 for the senate data. The covariate used in the senate data for the jumping transitive triplet calculation was the number of bills authored by the ego node in the given year. The number of bills authored by a senator varies wildly, from no bills to 114 bills authored in two years, so this effect could simply be nonsensical for the senate data, since senators are less likely to have the same number of bills authors than teenage girls are to have the same drinking behavior. Looking at the estimates of β_4, we see that the estimates for the senate data are near zero, suggesting this effect, which considers indirect ties, is not important for the senate data: indirect relationships do not describe the senate collaboration structure as much as they do the teenage friendship structure.

2.4.3.4 Fitting the same model to the same data

To our small friendship network, we fit three models, M1, M2, and M3, using `RSiena` 1,000 times each. We then looked at the distribution of the fitted values, which are shown in Figure 2.16. We can see from these distributions that the inclusion of the jumping transitive triplet parameter, β_3 is obviously affecting the distributions of the other four parameters included in all models, α_1, α_2, β_1, and β_2. When β_3 is included, its estimate is postitive, meaning that friendships between two girls with different drinking behaviors tend to form when there is an intermediary who is already friends with the two girls. The inclusion of this parameter leads to increases in the rate parameters' estimates, suggesting that encouraging the transitive triplet behavior means that the girls would also change friends more frequently. The outegree parameter, β_1 decreases when β_3 is included, while the reciprocity parameter, β_2 increases. This implies the girls in the data prefer to form closer friend groups, as indicated by reciprocated ties and jumping transitive triplet formation, as opposed to being popular and having many friends. Having many friends who do not reciprocate is discouraged by M2. In comparison with models M1 and M3, model M2 typically has higher estimates of the rate parameter, meaning that the inclusion of the covariate statistic in the model leads to higher estimates of the number of times, on average, a node gets to change its ties. It is not clear, however, that the addition of a parameter to the objective function *should* effect the estimation of the rate parameters, so we continue to explore the collection of parameter estimates.

To further investigate the odd relationship between the parameter values, we look at correlations between each of the parameter estimates in each model. In Figure 2.17, we examine correlations between each of pair of parameters within each model and overall. The strongest correlation within each model is between β_1 and β_2, with absolute value of correlation between those two parameter values greater than 0.90 in all three models. The β_1 parameter is also highly correlated with the β_3 parameter within model M2, but it is not as highly correlated with the β_4 parameter in model M3. It might therefore be advisable to consider only models that either allow β_1, or β_2. Looking at the high correlation with α, we might switch to a model without β_1.

2.4.4 Explore algorithms, not just end result

The last principle of model visualization is to explore the process of fitting the model, instead of just focusing on the end result. This principle is perhaps the most important for SAOMs because the model fitting process in `RSiena` involves several simulation steps that are hidden from the user. Hiding the MCMC steps is practical and efficient if a researcher is primarily interested in fitting one model to a set of longitudinal network data, obtaining parameter estimates, and drawing conclusions or making predictions. We are more interested in *how* the models are fit, so we extracted and explored the different steps of that process instead of allowing them stay hidden.

A key component of each step of the SIENA method of moments algorithm is the "microstep" process. A series of microsteps is obtained by simulating from the model in its current state, $x(t_m)$ with current parameter values $\theta_0 = \{\alpha_{1_0}, \dots \alpha_{m-1_0}, \beta_{1_0}, \dots, \beta_{K_0}\}$, to the next state, $x(t_{m+1})$. This microstep process stops when the simulated network has achieved the same number of differences, C, from $x(t_m)$ as $x(t_{m+1})$, where

$$C = \sum_{i \neq j} |x_{ij}(t_{m+1}) - x_{ij}(t_m)|.$$

This simulation process follows the steps of the continuous-time Markov chain. Each tie change in the CTMC is referred to as one "microstep". At each microstep, an "ego node" is selected to make a change, and the chosen ego node randomly makes one change in its ties according to the probabilities, $\{p_{ij} : i \neq j \in \{1, \dots, n\}\}$ defined in Equation 2.5, determined by its objective function. The options for change are (1) removing a current tie, (2) adding a new tie, or (3) making no change at all. Saving and exploring all of these steps is not computationally efficient if one is only interested in estimating parameter values, but they can be saved and extracted using options in `RSiena`, which is what we did to create our visualizations. Between two network observations $x(t_m)$ and $x(t_{m+1})$, there can be dozens, hundreds, or even thousands of microsteps, depending on the size of the network and the number of changes between two network observations. We wanted to view these in-between steps in order to better understand the behavior of the underlying continuous-time Markov chain.

The first vizualization we present here is an animation of the simulated microsteps that form the transition steps of the CTMC from wave 1 to wave 2 of the small friendship network example shown in Figure 2.4 when fitting model M1. Movies similar to this animation were used to visualize the changes of dynamic networks in Moody et al. (2005). When each ego node is selected in a microstep, it is emphasized in the animation, then the associated edge either appears or disappears. If there are no changes at a particular microstep, no changes are seen. Some frames of the animations are shown in Figure 2.18, and the full movie can be viewed at https://vimeo.com/240089108.

The top row of Figure 2.18 shows an edge being removed, and the bottom row shows one being added. In both cases, the ego node chosen to act change color from black to red, and they also increase greatly in size. In the case of an edge being removed, in the top row, the edge that currently exists is emphasized with the same color and size change that the node gets, and as the animation proceeds the edge shrinks to nothing, as the ego node shrinks and changes color back to its original black. If an edge is being added, as in the bottom row of the figure, the ego node's appearance changes in the same way as when the edge is being removed, while the edge appears colored red from nothing, and grows to a large size, then changes color and size to match the rest of the edges, while the node shrinks and changes color to match the other nodes.

In the network animation, we see the possible steps of the unobserved CTMC process that is underlying the SAOM fit to the data. We see each part of the model come into play. First, we see the rate at which the nodes are selected to change. Then, we see the result of the actor maximizing its objective function by either deleting or adding a node. In addition, the layout of the nodes changes as edges are removed or added, which gives us a better sense of how the overall network structure changes with these individual tie changes.

We next use animation to view the changing structure of the adjacency matrix the microsteps. The adjacency matrices for the three waves of friendship data as shown in Figure 2.4 are ordered by node id. There are 16 nodes in the data, numbered 1-16, and that order is used on the x and y axes for the matrix visualization. Viewing the adjacency matrices with this arbitrary ordering does not provide much information to the viewer about the underlying structure of the network. This lack of perceived structure would be exacerbated in an animation, so we adjust the ordering so that the viewer can better perceive the structure of the network. This process is known as matrix seriation (Liiv, 2010).

To reorder the vertices for the matrix visualization, we first constructed a cumulative adjaceny matrix, $\mathbf{A}^{cum}$, for the series of microsteps simulating the network from $x(t_m)$ to $x(t_{m+1})$. A single entry in the cumulative adjacency matrix, $\mathbf{A}_{ij}^{cum}$, is the total number of times the edge from node i to node j appears in the network from the initial observation, $x(t_m) \equiv X(0)$ to the final result of the last microstep, $X(R)$, where R is the total number of microsteps taken:

$$\mathbf{A}_{ij}^{cum} = \sum_{r=0}^{R} X_{ij}(r).$$

We then performed a principal component analysis (PCA) on $\mathbf{A}^{cum}$, and used the values of the first principal component to order the vertices on the x and y axes for the adjacency matrix animation. For one such series of microsteps simulated by `RSiena`, we present the adjacency matrix ordered by the (arbitrary) vertex id alongside the seriated adjacency matrix using the first principal component loading on the cumulative adjacency matrix, $\mathbf{A}^{cum}$, in Figure 2.19.

Using PCA on $\mathbf{A}^{cum}$ to order the rows and columns of the adjacency matrix visualization clearly shows the two distinct connected components in the first wave of the network, which are difficult to find in the arbitrarily ordered visualization. We also use the PCA seriated layout to fix the layout in the animation of one of the microstep process simulations. This animation, some frames of which are shown in Figure 2.20 is very simple: a square appears or disappears in the animation as that edge appears or disappears in the microstep process. Through this animation, we can see edges appearing, and then later on disappearing. These in-between steps are not shown when we look at the network at our discrete observation points, so by viewing this animation we can gain a better understanding of the underlying dynamics of this model. The full movie can be viewed at https://vimeo.com/240092677.

We also attempt to better understand the microstep process by visualizing the observed transition probabilites for the first microstep in the process. We only do the first step of many because the `RSiena` transition probabilities for any two edges i, j after the first step are only directly comparable for identical ego node states due to the conditioning on the current network state in the model. By ego node state, we mean the current set of incoming and outgoing ties to the ego node, i. Put more precisely, let the vector $X_i = (\{x_{ik}\}, \{x_{ki}\})_{k \neq i} \in \{0, 1\}^{2(n-1)}$ represent the set of all tie variables involving node i in the current network. For p_{ij} to be comparable for any two network states x and x', the value of the vector X_i must be the same in both states. It is possible to incorporate this information into our visualizations, but for now we look at only the first step, because we know without needing any complicated conditioning steps that the previous node state X_i is identical in all cases. Thus we have 1,000 transition probabilities to examine: one transition probability for the first microstep taken for each of the simulations. In Figure 2.21, we present each ego node and the resulting probabilities of tie changes. The probability shown by the bars is the theoretical probability, according to the objective function, of the ego node changing its tie to the alter node, while the probability shown by the points is the empirical probability of that change being made. The empirical probability is calculated by counting all instances of the ego node first, then computing the proportion of each different alter node change. In most cases, they are almost identical, which demonstrates that the algorithm is performing about as expected. There are, however, many steps that are never taken. Some ego nodes, like 1, 4, and 16, really explore the space of all possible outgoing ties. On the other hand, nodes like 13 and 15 explore very few possible outgoing ties. It is unclear why this would be happening; it may have something to do with the ties in wave 1, where 13 and 15 have multiple connections, while 1 and 16 have none, but there are several other counterexamples of well connected nodes that explore the space more. In addition, we can see that the ties changed most often are removing ties, not adding ties. This tracks with the overall pattern of the data: we see the most ties in the first wave of data, and the least ties in the third wave.

Another way to view these transition probabilities is through the adjacency matrix visualization. In Figure 2.22, we build on the concept of the ordered adjacency matrix of Figure 2.19. This heatmap shows the transition probabilities of all ties that are changed in the first microstep of the 1,000 simulations. The heatmap is noticeably sparse: of the $16^2 = 256$ possible steps for the CTMC to take, only 103, or about 40%, are taken in the 1,000 simulated chains. This reinforces what we saw in Figure 2.21, where there are many paths not taken. This effect could only be exacerbated as more steps are taken in the CTMC, leading to a very large area of our network space, $\mathcal{X}$, completely unexplored by the SAOM model fitting process.

We also wanted to better understand the entire microstep process from the first wave, all the way through to the second wave. The number of steps taken from wave 1 to wave 2 varies. In one set of 1,000 simulations from Model 1, the smallest number of steps taken was 58, and the longest was 248, with a mean of 106 and a median of 103. In the 1,000 simulations, the standard deviation of the number of microsteps was 22.8. In Figure 2.23, we see two simulations of the process from wave 1 to wave 2, with wave 1 shown on the left, and wave 2 shown on the right. In each of the three plots, the y-axis contains the edges sorted by how often they appear in the networks along the way. We can see that some edges are there in the beginning, but disappear and never come back, while others appear a few steps in, only to dispappear again. There are also some edges that were observed in wave 2 that don't appear at all in the microstep process in a given simulation. So, even though the CTMC makes about the right *number* of changes as it was designed to do, the changes it is making are not necessarily in the right direction.

We also combine 1,000 simulations from model M1 into a visualization like the one shown in Figure 2.23. We first assign each possible edge an edge ID number so that we can keep track of it throughout all the microsteps and all the simlations. Then, we count up the total number of times each edge appears in the microstep process in each of the 1,000 simulations for use as an ordering variable later. We also count up the number of times an edge occurs in each microstep number in the 1,000 simulations. Since the number of microsteps in the process varies, the number of times an edge occurs decreases as the microstep number increases. Next, we compute a proportion, which we call the occurrence percentage, which is the number of times the edge occurred in a microstep divided by 1,000. Finally, we visualize all this information together in Figure 2.24. In this plot, all possible edges are shown, and we see that every one of the $16 \times 15 = 240$ possible edges in the network occurs at some point in the simulation process. We also see, however, that the process struggles to focus in on the edges in the second wave of the data. Ideally, we would like to see more occurrences of the edges which appear in the second wave of data. But, about half of the edges in wave two are in the bottom half when ordered by number of occurrences, meaning they do not appear as much as they would if the model was truly excellent at capturing the mechanisms of tie change in the network. This solidifies what we found in Figures 2.21 and 2.22: the model M1 and the SAOM fitting process do not explore the data space enough to adequately capture the network change mechanism.

2.5 Discussion

We have used novel visualization methods in order to better understand the family of models known as stochastic actor-oriented models for social network data. By looking at the underlying algorithms, visualizing collections of these models, and viewing the model in the data space, we have been able to gain knowledge and appreciation for these complicated models and everything that goes into them.

We have only just begun to scratch the surface of these complicated and multi-layered models for social networks. The `RSiena` software is incredibly powerful, and can fit a whole slew of much more flexible stochastic actor-oriented models than we have examined here. If a researcher thinks the network structure or an actor covariate effects the rate of change of the network, there is a way to incorporate that belief into the rate function of the SAOM. More than one actor-level covariate can be included in the model, and way more than three parameters can be included in the objective function itself. In addition, `RSiena` allows the user to tell it which parameters lead to tie creation, and which parameters lead to tie endowment, or dissolution. We have used "evaluation" parameters, which assume that creation and endowment are equal (Ripley et al., 2017). Finally, SAOMs and `RSiena` are able to also model behavior change of the actors in the network, which again is a capability we did not explore here.

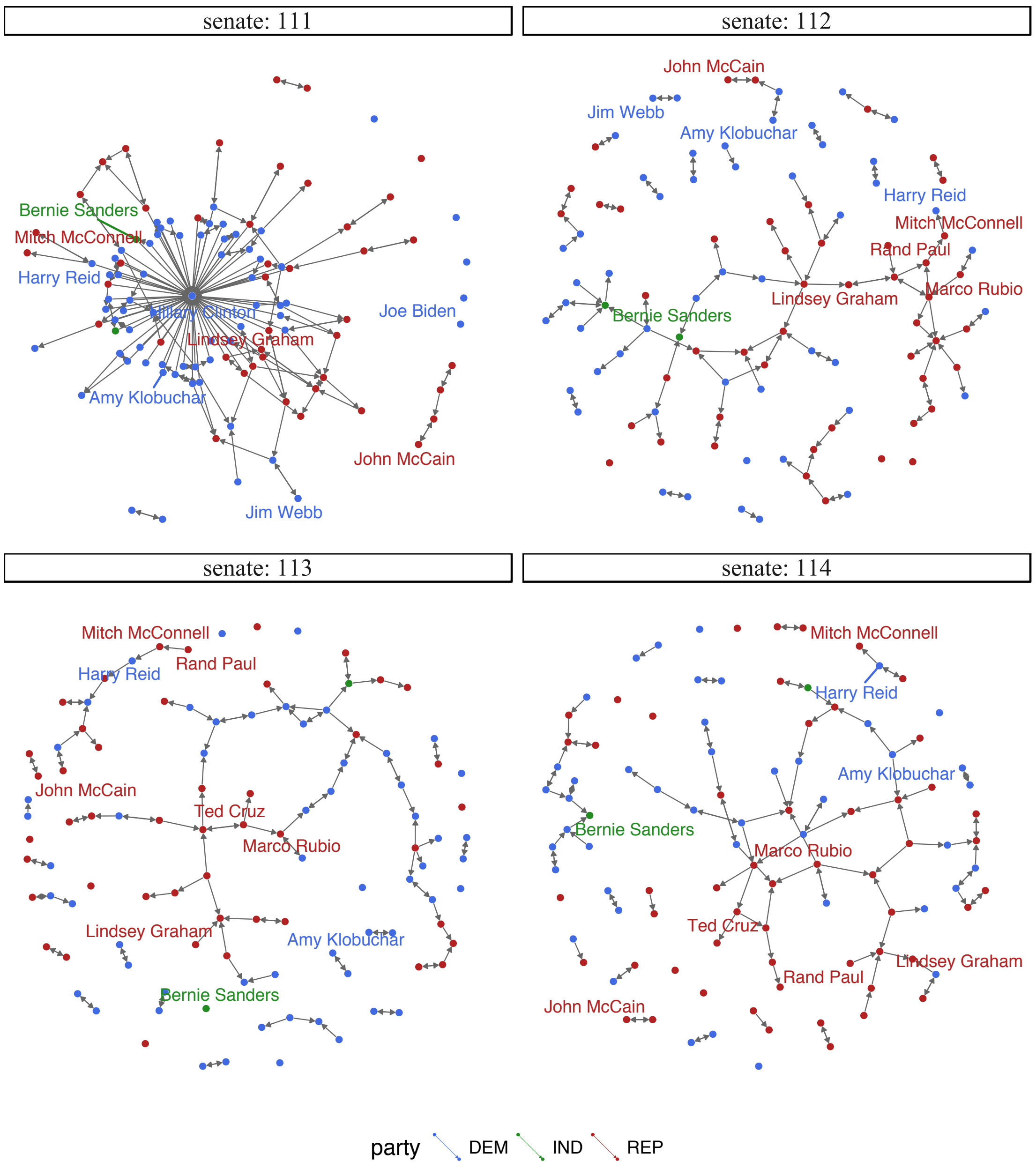


Figure 2.5: The collaboration network in the four senates during the Obama years, 2009-2016. Edges are shown only if the weighted propensity to cosponsor from one senator to another is greater than 0.25. We use the Fruchterman-Reingold algorithm to layout the node-link diagram.

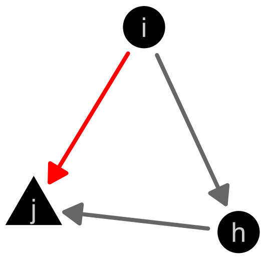


Figure 2.7 Realization of a jumping transitive triplet, where i is the focal actor, j is the target actor, and h is the intermediary. The group of the actors is represented by the shape of the node.

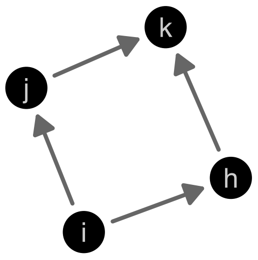


Figure 2.8 Doubly achieved distance between actors i and k.

Figure 2.9: The additional network effects included in our models fit to the friends data. On the left, a jumping transitive triplet (JTT). On the right, a doubly achieved distance between i and k.

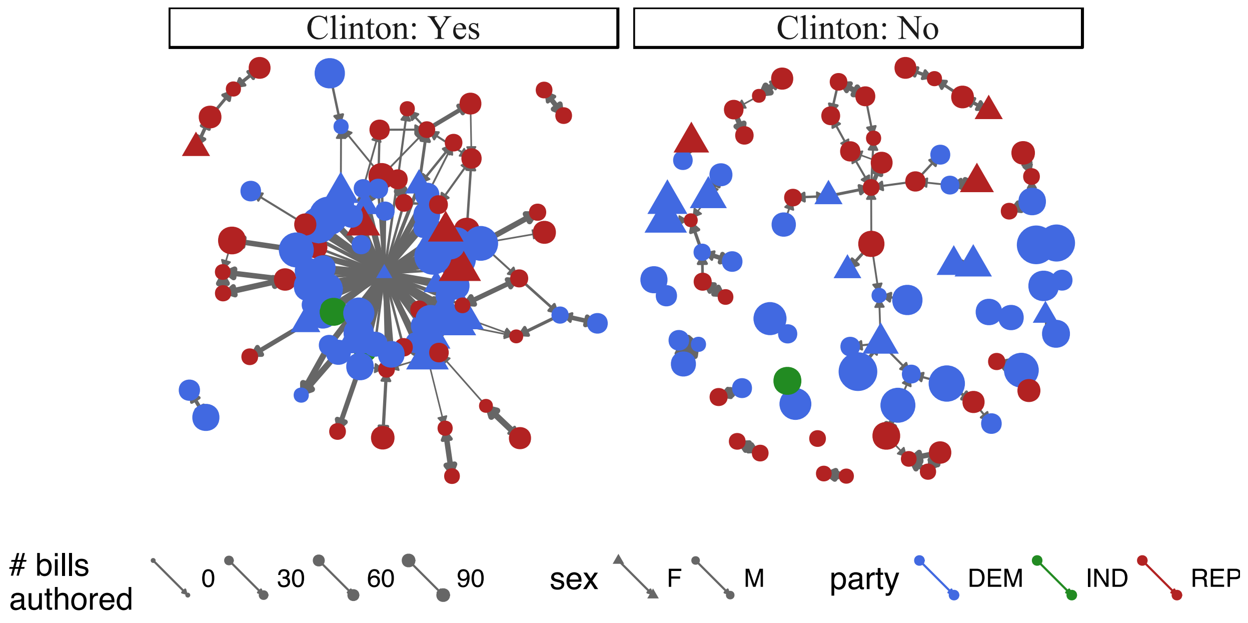


Figure 2.10: The 111th Senate at two discrete time points: while Clinton was in the senate in the first few weeks of 2009 (on the left) and after she left the senate to become Secretary of State (on the right). We put some potential model covariates, sex and party of the senator and the number of bills they authored in the data space through the shape, color, and size of the nodes, respectively. We also map the strength of the tie, the WPC, to the width of the edge between two nodes. Thicker lines implies higher propensity to collaborate. In addition, senators with no ties higher than $WPC = 0.25$ are not shown.

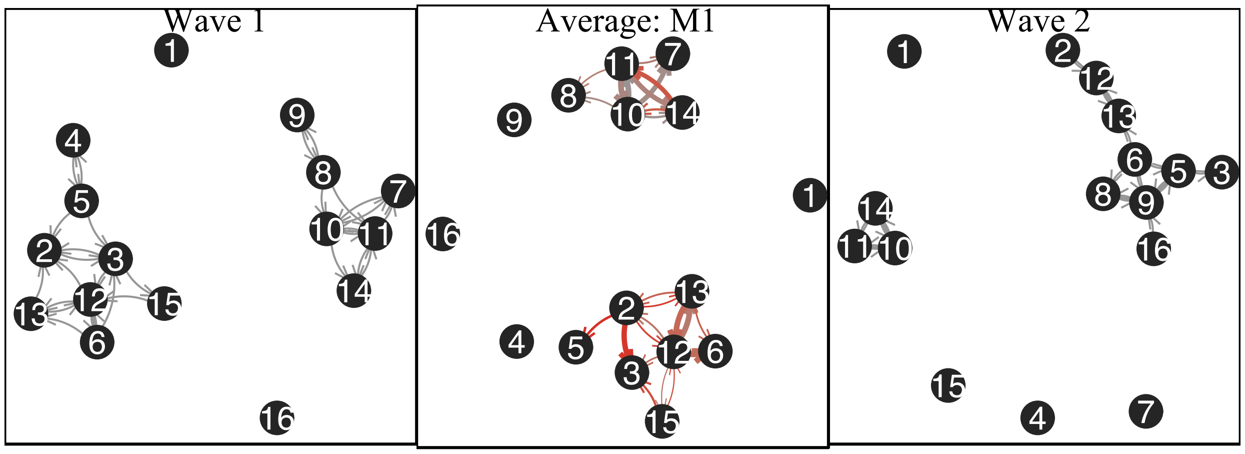


Figure 2.11: On the left, the first wave of observed data that is conditioned on in the model. On the right, the second wave of observed data. In the middle, a summary network from the first model fit to the data. This summary network represents 1,000 simulations of wave 2 using the values from the simple fitted model M1.

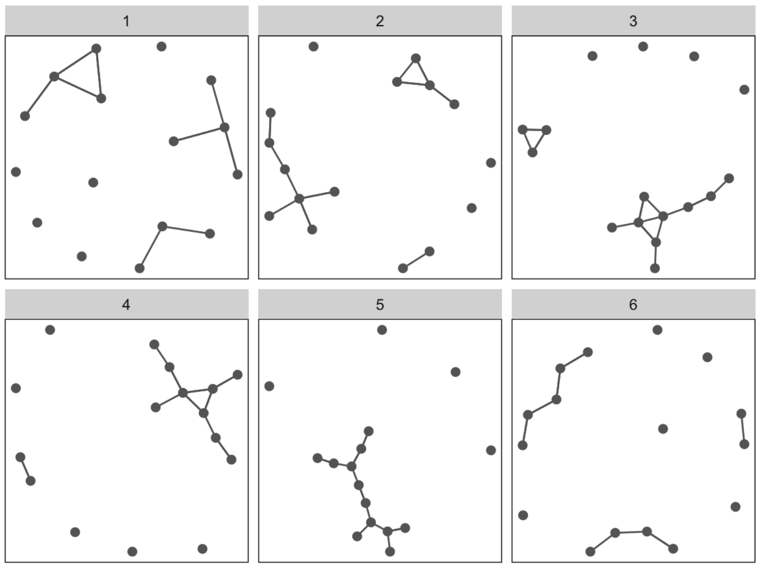
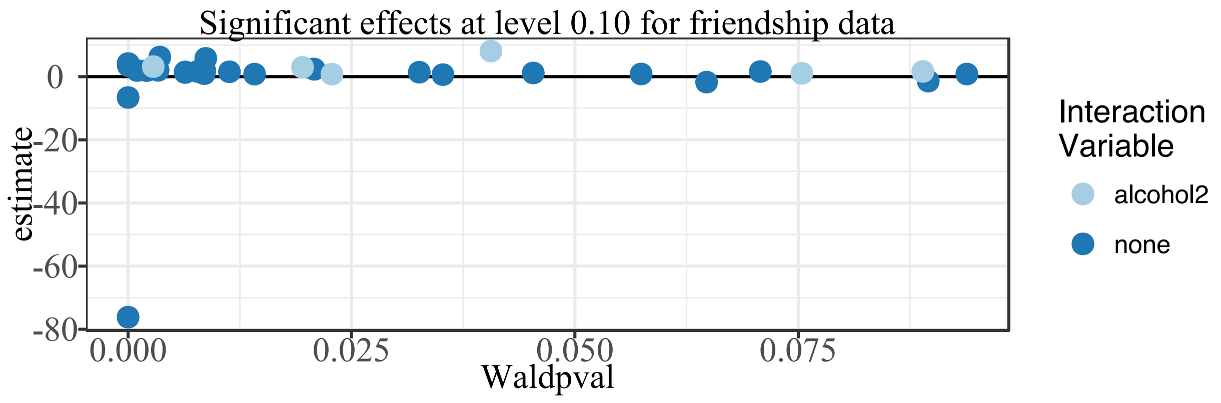


Figure 2.12: A small lineup of node-link diagrams showing the second wave of our small friendship network among five networks simulated from model M1.



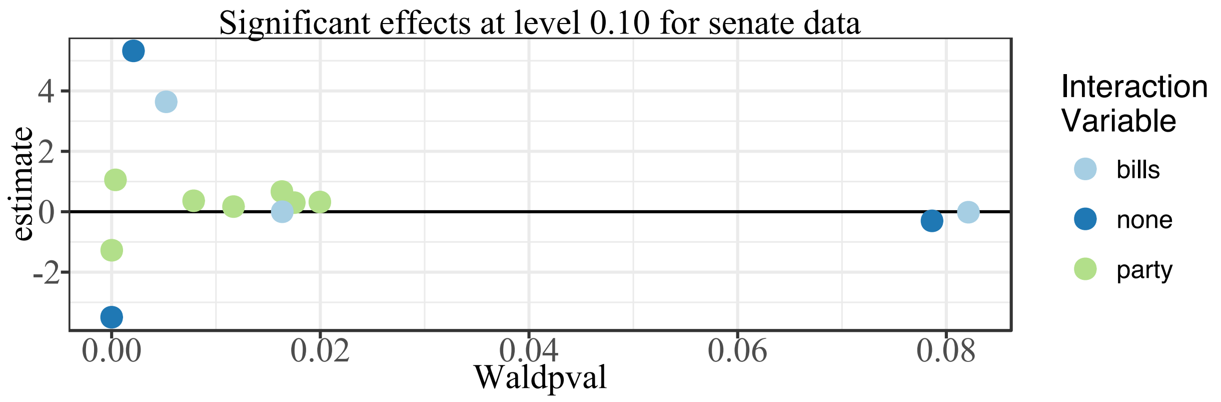


Figure 2.13: Significant effects for the two data sets, at a significance level of 0.10 or lower as calculated by the Wald-type test available in the SIENA software.

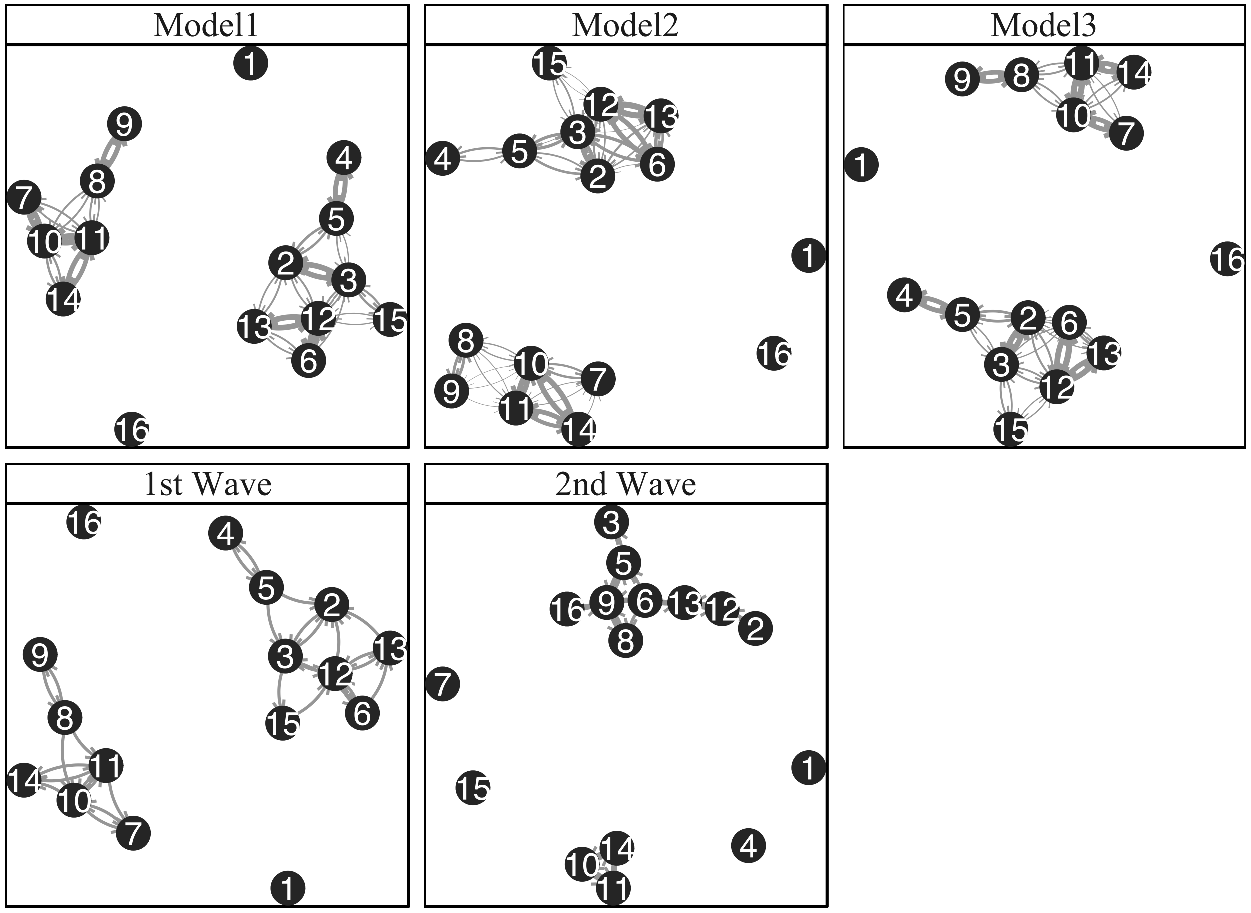


Figure 2.14: The node-link diagrams from the three "average" networks that we calculated are in the top row, and the true wave 1 and wave 2 data are shown in the bottom row above. There is some difference between the three models, but overall, these three models cannot capture the structure in the true second wave of data.

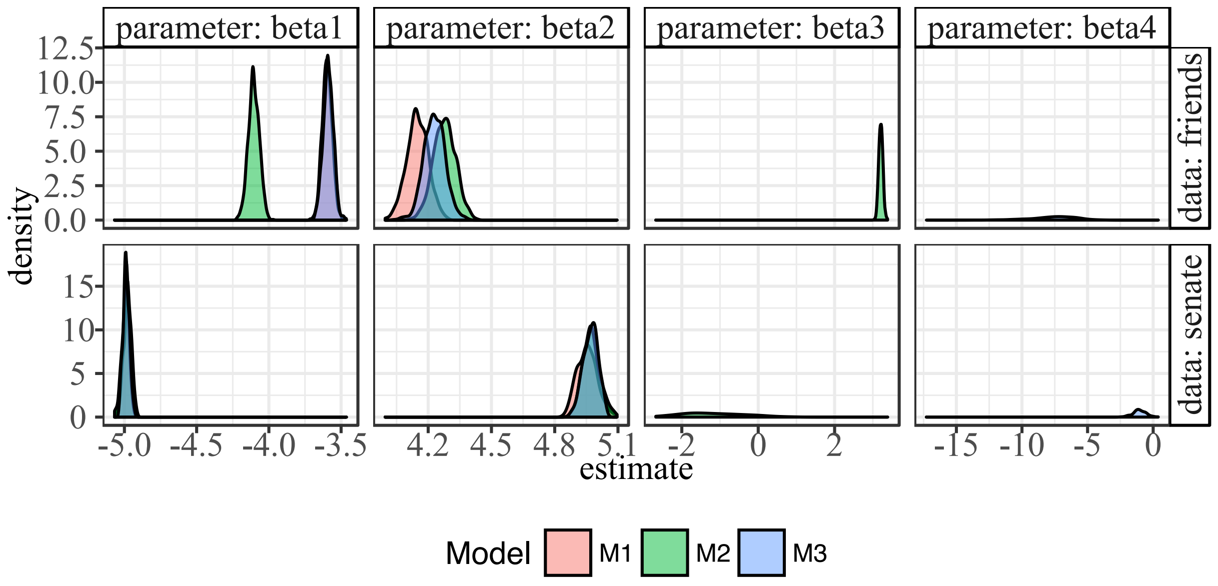


Figure 2.15: Density plots of the repeated estimates from fitting models M1, M2, and M3 to the friendship and senate example data. Note that the rate parameters are excluded since the two data sets have different numbers of waves. The first two parameters, outdegree and reciprocity, have the same relationship for both data sets: the outdegree parameter estimate is strongly negative, while the reciprocity estimate is strongly postitive. For the friendship data, the inclusion of the transitive triplet parameter has strong effect on the estimates of the other two parameters, while it does not affect the estimates of the other two parameters for the senate data.

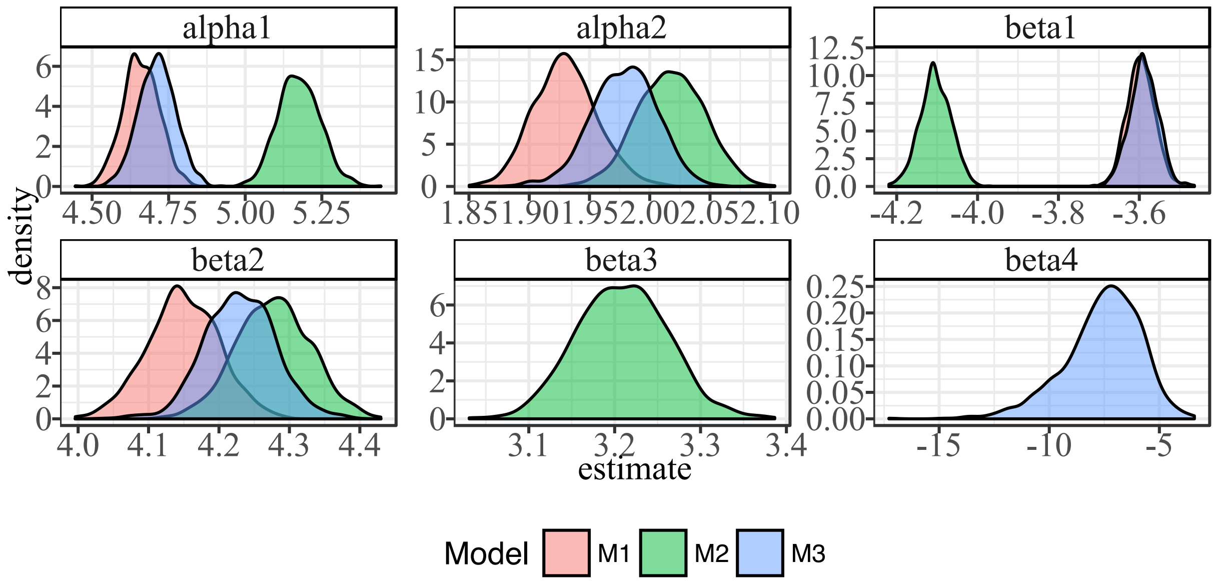


Figure 2.16: Distribution of fitted parameter values for our three SAOMs. The inclusion of β_3 or β_4 clearly has an effect on the distribution of the rate parameters, α_1 and α_2.

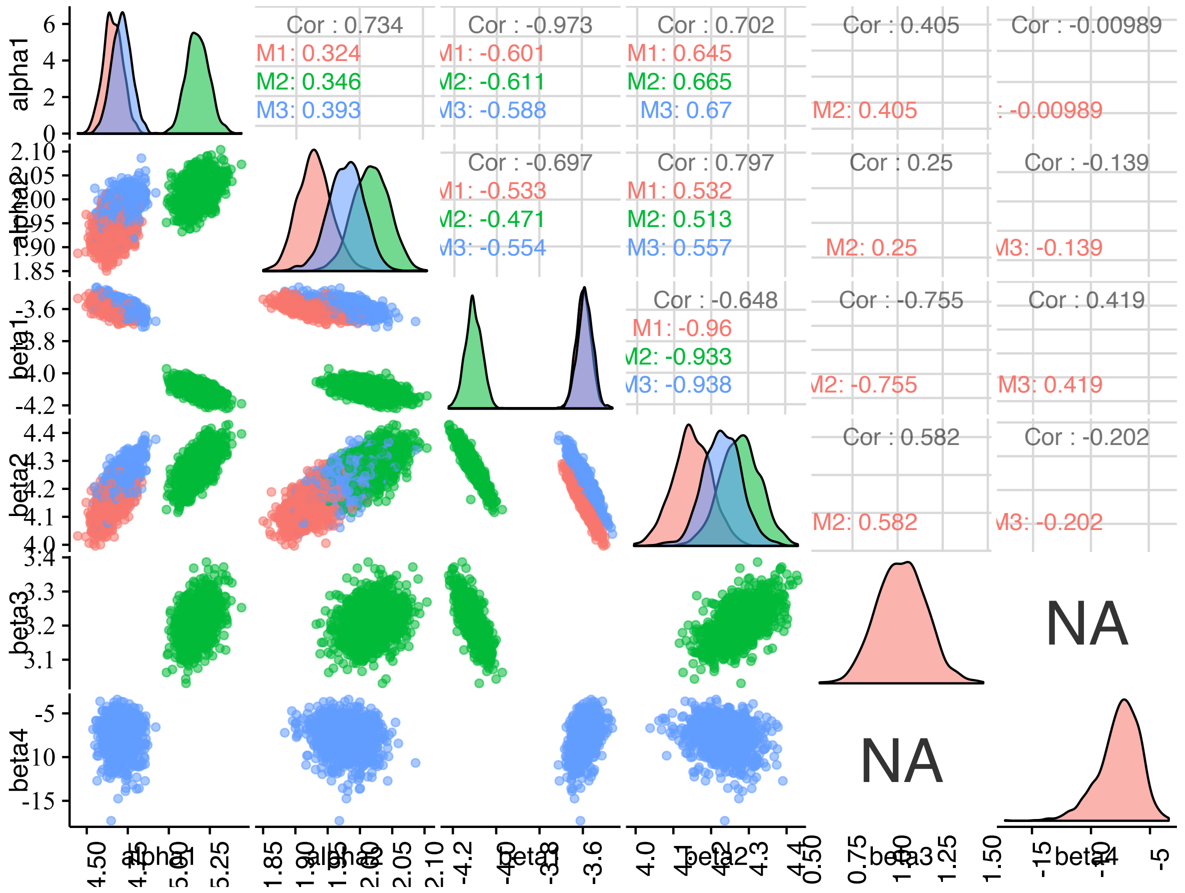


Figure 2.17: A matrix of plots demonstrating the strong correlations between parameter estimate in our SAOMs. The strongest correlation within each model is between β_1 and β_2.

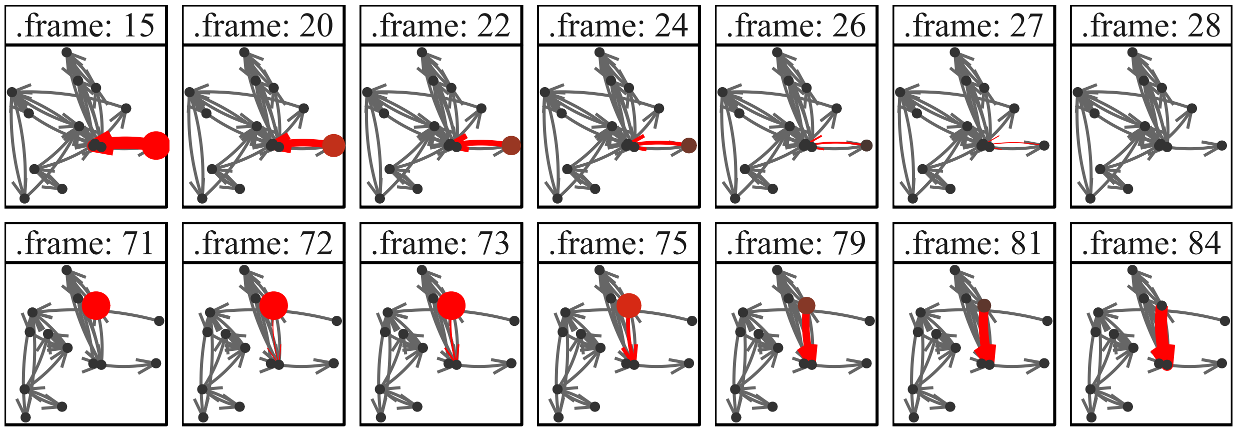
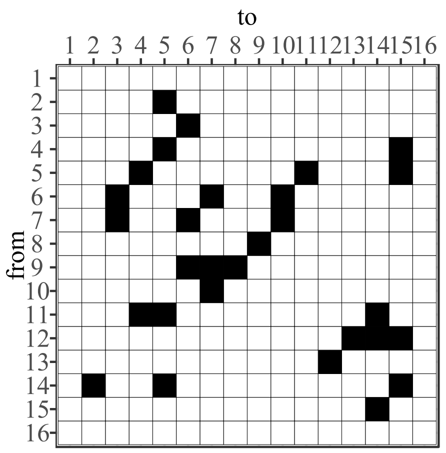


Figure 2.18: A selection of images in the microstep animation. The selected edges and nodes are emphasized by changing the size and the color, then when edges are removed, they fade out, shrinking in size, while the nodes change color and shrink to blend in with the other nodes.



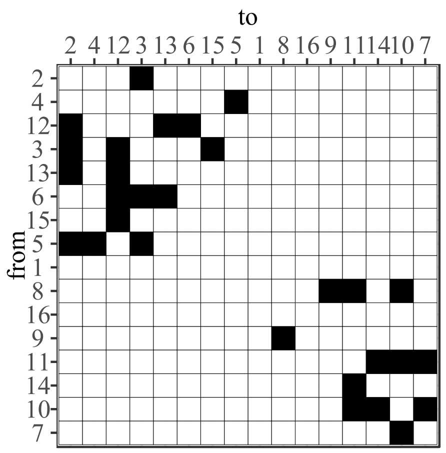


Figure 2.19: On the left, the starting friendship network represented in adjacency matrix form, ordered by vertex id. On the right, the same adjacency matrix is presented after ordering the vertices by one repetition of the microstep simulation process from wave one to wave two.

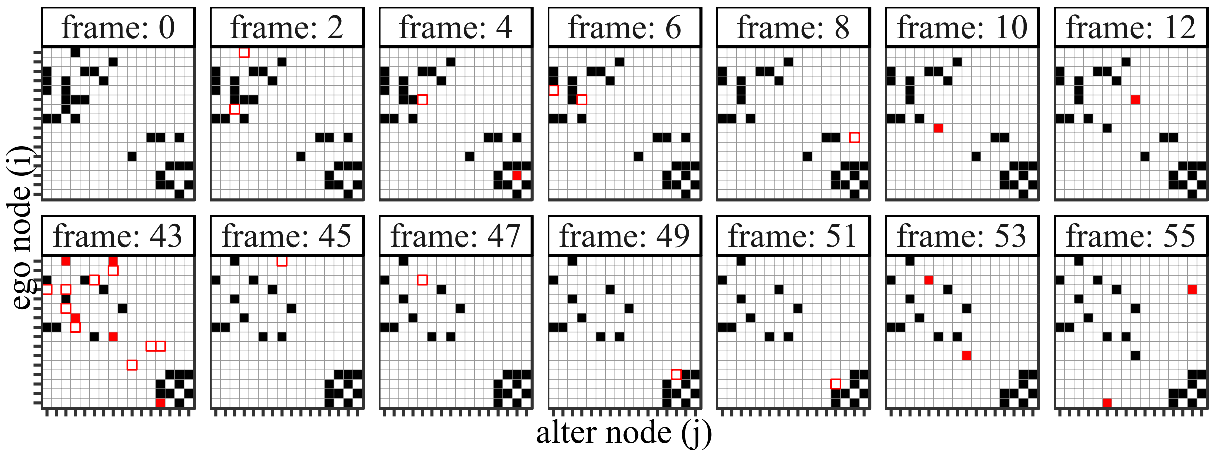


Figure 2.20: A selection of frames from the adjacency matrix visualization animation for one series of microsteps. (Ego and alter node labels are removed to declutter the graph.) At the beginning of the animation, shown in the top row, there are two clearly connected components: one in the top left corner, and one in the bottom right corner. By the end, the component in the top left has spread out, while the bottom right component has shrunk, but remains closely connected.

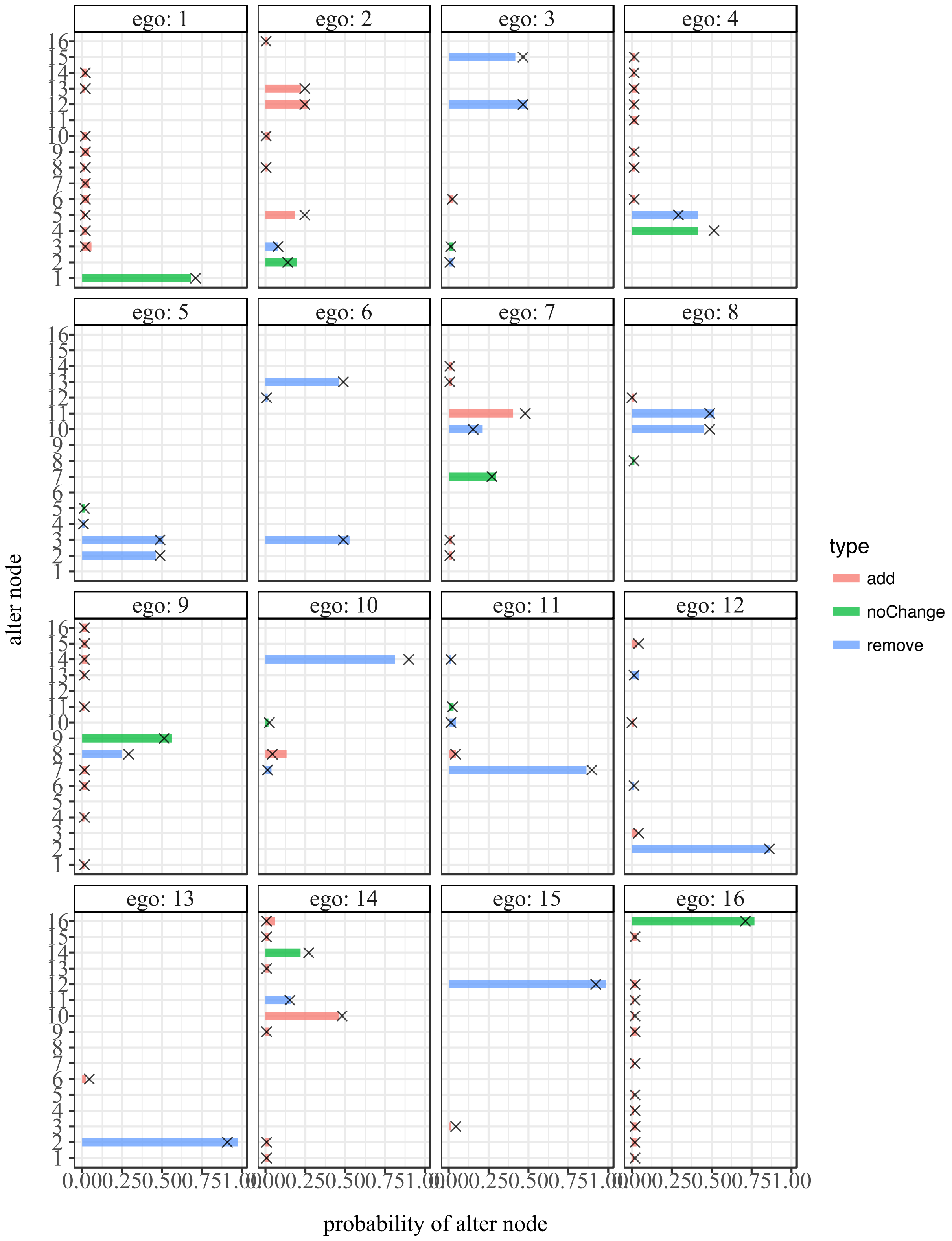


Figure 2.21: Each panel shows the theoretical (as lines) and empirical (as points) probabilities of the chosen ego node changing its tie to each of the other nodes. The color of the line indicates whether the tie from the ego to the alter node is being added, removed, or if there is no change to the network in this step.

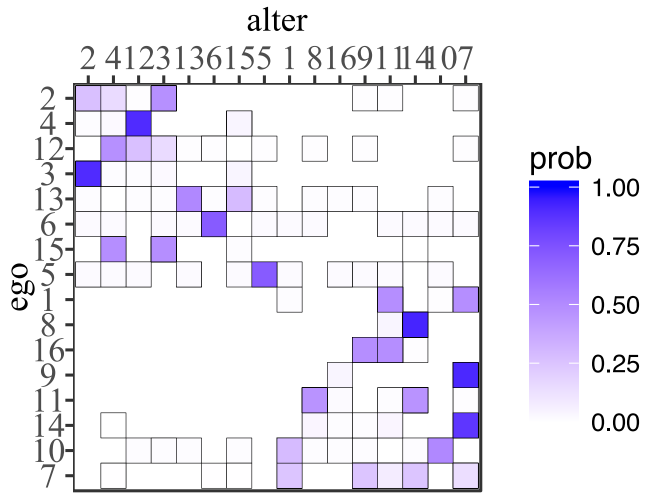


Figure 2.22: A heatmap showing the empirical transition probabilities for the first microstep in 1,000 simulations. The acting ego node is on the y-axis, and the alter node is on the x-axis. There were many ties with empirical probability zero.

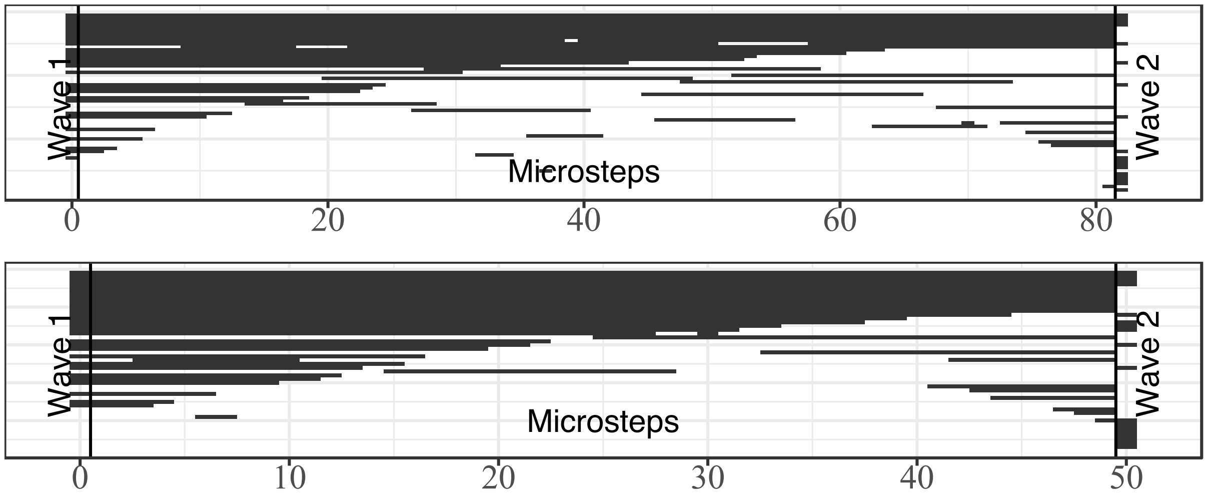


Figure 2.23: Two simulations (out of 1,000) of the microstep process from wave 1 to wave 2. The x axis is the microstep number, with step 0 representing the first wave of data and the final step representing the second wave of data. We can see that many edges are underrepresented in this process: they are in the second wave, but never appear in the microsteps.

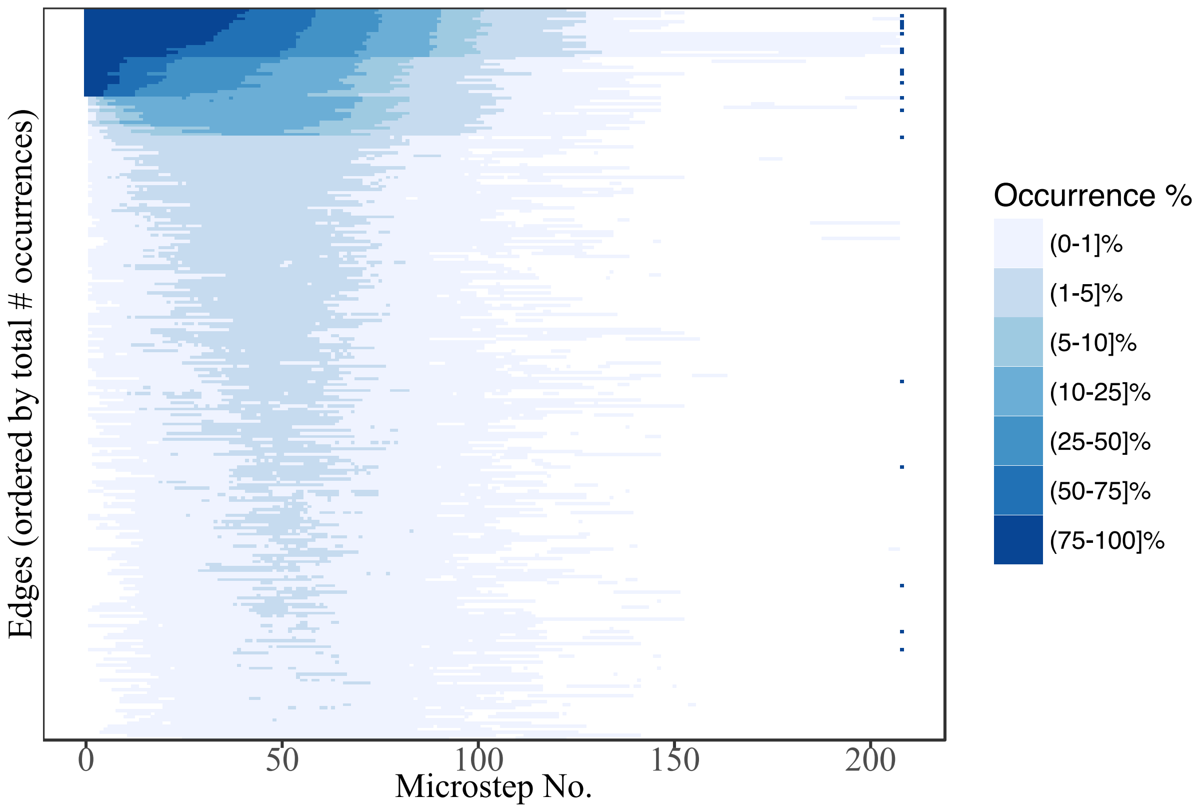


Figure 2.24: Visualizing all microsteps taken in 1,000 simulations from the model M1. The occurrence percent is split up into groups to correspond with its distribution: only about 10% of the edges appear more than 10% of the time in the 1,000 simulations, while about 60% appear less than 1% of the time. The first wave network is shown at microstep 0, and the second wave of the network is shown as the last microstep for comparison. We see that it is rare for a microstep process to last longer than 150 steps, and also that the edges that appear past the 150th step tend to be in either the first wave or the second wave.

CHAPTER 3. VISUAL INFERENCE FOR SIGNIFICANCE AND GOODNESS-OF-FIT TESTING OF STOCHASTIC ACTOR-ORIENTED MODELS

3.1 Introduction

Three of the most important pieces of statistical modeling are significance testing of model parameters, model goodness-of-fit tests, and determining power of a hypothesis test. In the first, the data are usually assumed to come from a simple model under the null hypothesis, and additional parameters are tested to determine whether they significantly contribute to explaining the variability in the data. In the second, the model of interest is examined to determine how well it fits the data. In the final, the ability of the hypothesis test in question to detect the difference between the null and alternative hypothesis is determined. All three of these aspects of statistical modeling increase greatly in difficulty as the complexity of the model of interest increases.

Some particularly complicated sets of models are those designed to model network change. A *network* is any set of things, such as people, computers, or neurons, that are connected in some way, through social relations, internet connection, or electrical impulses in the brain. We refer to the "things" in the network as *nodes*, or *actors* in a social network, and the connections as *edges*, or *ties* in a social network. Dependencies inherent to the data make network objects particularly difficult to model. The difficulty of modeling increases when we go beyond single instances of a network and consider the dynamics of network change between observed instances. This type of modelling for dynamic networks is often performed on social network data, such as friendship networks among students or the spread of HIV in drug users sharing needles. It is known that most network models lack the asymptotics required to perform many well-known goodness-of-fit tests, and the maximum likelihood estimation of parameters is so difficult that it can make significance testing difficult as well (Goldenberg et al., 2010).

In order to circumvent these difficulties, we propose new methods for significance testing of parameters, goodness-of-fit testing, and power calculations for a set of social network models: stochastic actor-oriented models for dynamic network data (Snijders, 1996). Specifically, we are using *visual inference* in place of traditional statistical methods for social network models, such as Wald or t-tests for significance of parameters and in- and outdegree distribution metrics for determining goodness-of-fit. Visual inference, introduced by Buja et al. (2009), allows us to look at the *entire* dataset simulated from a network model as opposed to a (set of) usually one-dimensional metric(s) derived from the network such as outdegree, or a p-value for a single parameter in the model. By using visual inference to supplement traditional statistical tests, we gain a new understanding of parameters in SAOMs and fit of SAOMs to data.

The paper is outlined as follows: Section 3.2 gives a basic overview of visual inference and the lineup protocol. Section 3.3 provides an introduction to the our models of interest, stochastic actor-oriented models. Section 3.4 details how we define significance testing and goodness of fit procedures for SAOMs through visual inference, and Section 3.5 details the results of a visual inference survey of Amazon Mechanical Turk workers. We close with a discussion in Section 3.6.

3.2 Visual Inference

Data visualizations are an important component of data analysis, providing a mechanism for discovering patterns in data. Pioneering research by Gelman (2004), Buja et al. (2009) and Majumder et al. (2013b) provide methods to quantify the significance of discoveries made from visualizations. Buja et al. (2009) introduced two protocols, the Rorschach and the lineup protocol, which bridge the gulf between traditional statistical inference and exploratory data analysis. Here, we use the lineup protocol. Under this protocol, we begin with a data set of interest to us, such as a network, and a visualization of this data, such as a node-link diagram. We would like to know the model that generated this data. There are two hypothesis considered: a null hypothesis which states the model that is assumed to have generated the data, and an alternative hypothesis that the data were not generated under this model. Then, the plot of the data of interest is placed randomly among a set of $m - 1$ null plots, which are visualizations of data simulated from the null model(where $m = 20$, usually). Human observers are then asked to examine the lineup and to identify the plot that looks most different from the others. If an observer identifies the data plot, this is quantifiable evidence against the null hypothesis. Since an observer has a chance of 1 in m to pick the data plot from the lineup by simply guessing, i.e. in a situation where the data plot is virtually indistinguishable from the null plots, the evidence grows in strength with the number of independent observers identifying the data plot.

The lineup protocol places a plot firmly in the framework of hypothesis tests: a plot of the data is considered to be the test statistic, which is compared against the sampling distribution under the null hypothesis represented by the null plots. Obviously, the null generating mechanism, i.e. the method of obtaining the data for null plots, is crucial for the lineup protocol, as the null hypothesis directly affects the choice of null generating method. Null generating methods are typically based on (a) simulation, if the null hypothesis allows us to directly specify a parametric model, (b) sampling, as for example in the case of large data sets, or (c) permutation of the original data, see for instance Good (2005), which allows for non-parametric testing that preserves marginal distributions while ensuring independence in higher dimensions. The model of interest here allows us to simulate directly from a parametric model for dynamic social network data.

The lineup protocol was formally tested in a head-to-head comparison with the equivalent conventional test in Majumder et al. (2013b). The experiment utilized human subjects from Amazon's Mechanical Turk (Amazon, 2010) and used simulation to control conditions. The results suggest that visual inference is comparable to conventional tests in a controlled conventional setting. This provides support for its appropriateness for testing in real exploratory situations where no conventional test exists or is difficult.

3.3 Stochastic Actor-Oriented Models

Stochastic Actor-Oriented Models (SAOMs) are a family of models for dynamic network data that incorporate both network structure and node-level information to describe how a network observed on two or more occasions changes over time (Snijders, 1996). The two titular properties of SAOMs, stochasticity and actor-orientation, are crucial to understanding networks as they exist naturally: social networks are ever-changing as relationships decay or grow in seemingly random ways, and the actors in them have characteristics that could affect how they change their ties to other nodes in the network. These unique properties allow for the fitting of some very complicated models to inherently complex data, so it can be exceedingly difficult to interpret parameters and their corresponding estimates. The sheer amount of available parameters to include in the model combined with the difficulty of interpretation make parameter selection and goodness-of-fit testing burdensome as well.

Broadly, a SAOM takes network structure and node covariate information into account in two ways and models the network changes one-at-a-time as a continuous time Markov chain (CTMC). First, the rate of change between states is dictated by a rate function that describes *how often* changes in the network occur, and secondly, the objective function describes *what* those state changes are. As in most other network models, the variables of interest are the edges of the network, x_{ij}, where x_{ij} denotes the edge between nodes i and j, where $i, j \in \{1, 2, \ldots, n =$ the number of nodes$\}$. x_{ij} is modelled as a binary variable, i.e.

$$x_{ij} = \begin{cases} 1 & \text{if an edge from } i \text{ to } j \text{ exists} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Edges are treated as *directed*, i.e. in general $x_{ij} \neq x_{ji}$, and self-referencing edges or loops are not allowed, i.e. $x_{ii} = 0$ for all i. The network is observed $M \geq 2$ times at time points $t_1 < t_2 < \ldots < t_M$, and the entire network at time point t_m is denoted as $x(t_m)$. In sections 3.3.1 and 3.3.2 we discuss the rate and objective functions of a SOAM in more depth. Additional details on SOAMs can be found in Snijders (1996, 2001); Snijders et al. (2010a,b, 2007); Snijders (2017),

3.3.1 Rate Function

All changes in SAOMs are treated as changes made by the nodes, or *actors*, in the network, i.e. each actor, i, gets a chance to make a change according to the rate function, typically denoted

λ_i, which dictates when relationships between nodes in the network can change. In general, the rate function can take the network structure e.g. outdegree of node i, and the node covariates into account, but we use the simple rate function, which is constant over all nodes in a given time period, so that $\lambda_i \equiv \alpha_m \; \forall i \in \{1, 2, \ldots, n\}$. We denote the rate from t_m to t_{m+1} as α_m for $m = 1, \ldots, M-1$. Using this notation, the *waiting time* to the next chance for actor i to make a change is exponentially distributed with expected value α_m^{-1}. Since the rate is the same for all actors, the waiting time for *any* actor to get the oppurtunity to change its set of ties is exponentially distributed with expected value $(n\alpha_m)^{-1}$.

3.3.2 Objective Function

After actor i has been given the opportunity to change, it probabilistically chooses one of its current ties, x_{ij}, to change. The probability that actor i changes its current tie to actor j is determined by the *objective function* of the model and a random component, U, which can be thought of as encompassing any other factors that may be influencing the changes the node makes not accounted for by the parameters in the model. Actor i is aiming to maximize its corresponding objective function f_i given the current state of the network, x and the node-level covariates, $\mathbf{Z}$, given as:

$$f_i(x, \boldsymbol{\beta}, \mathbf{Z}) = \sum_{k=1}^{K} \beta_k s_{ik}(x, \mathbf{Z}), \tag{3.2}$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_K)$ are additional model parameters, each associated with some network statistics, $s_{ik}(x, \mathbf{Z})$, $s_{ik}(x, \mathbf{Z})$, calculated with respect to actor i at the current network state x. Network statistics range from the simple outdegree, $s_i(x) = \sum_{i \neq j} x_{ij}$, to the more complicated *transitive triplets jumping to different covariate*, $s_i(x, \mathbf{Z}) = \sum_{i \neq j \neq h} x_{ij} x_{ih} x_{hj} \cdot \mathbb{I}(z_i = z_h \neq z_j)$. Version 1.2-3 of `RSiena` (Ripley et al., 2013), the software used to fit the models here, provides over 80 possible effects that can be included in the objective function. We discuss these statistics in more detail in Section 3.3.4.

Objective function $f_i(x, \boldsymbol{\beta}, \mathbf{Z})$ and random component U combine to form the *transition probability*, p_{ij}, of the network changing from its current state x to the state with changed tie x_{ij}, denoted as $x(i \rightsquigarrow j)$:

$$p_{ij} = \frac{\exp\{f_i(x(i \rightsquigarrow j), \boldsymbol{\beta}, \mathbf{Z})\}}{\sum_h \exp\{f_i(x(i \rightsquigarrow h), \boldsymbol{\beta}, \mathbf{Z})\}} \tag{3.3}$$

This probability dictates which edge change is made by the acting node. The acting node can also choose to *not* change at all. This is most likely to occur when the numerator, as calculated for the current state of the network, is larger than for any changes $x(i \rightsquigarrow j)$ that could be made.

According to Ripley et al. (2017), at least two parameters must be included in the objective function: the density and the reciprocity. We denote the density, or out-degree, parameter by β_1 and the associated statistic as $s_{i1}(x) = \sum_j x_{ij}$. Similarly, we denote the reciprocity parameter by β_2 and the associated statistic as $s_{i2}(x) = \sum_j x_{ij}x_{ji}$. We will refer to the very simple model with only these two parameters in the objective function as M1.

3.3.3 Example Data

The data we use are collaboration networks in the United States Senate during the 111^{th} through 114^{th} Congresses, overlapping with Barack Obama's presidency. These senates began on January 6, 2009, the start date of the 111^{th}, and ended on January 3, 2017, the last date of the 114^{th}[1]. In the US Senate, there are three ways that senators can show support for a piece of legislation: they can author the bill, cosponsor the bill, and vote for the bill. We use cosponsorship as a metric because it results in a network that is unimodal (all nodes are senators) and directed. In this network, ties are directed from senator i to senator j when senator i signs on as a cosponsor to the bill that senator j authored. There are many hundreds of ties between senators when they are connected in this way, so we simplify the network by computing a single value for each senator-senator collaboration called the *weighted propensity to cosponsor* (WPC). This value is defined in Gross et al. (2008) as

$$WPC_{ij} = \frac{\sum\limits_{k=1}^{n_j} \frac{Y_{ij(k)}}{c_{j(k)}}}{\sum\limits_{k=1}^{n_j} \frac{1}{c_{j(k)}}} \tag{3.4}$$

[1]Details of how this data can be downloaded are provided by Franois Briatte at https://github.com/briatte/congress.

where n_j is the number of bills in a congressional session authored by senator j, $c_{j(k)}$ is the number of cosponsors on senator j's k^{th} bill, where $k \in \{1, \ldots, n_j\}$, and $Y_{ij(k)}$ is a binary variable that is 1 if senator i cosponsored senator j's k^{th} bill, and is 0 otherwise. This measure ranges in value from 0 to 1, where $WPC_{ij} = 1$ if senator i is a cosponsor on every one of senator j's bills and $WPC_{ij} = 0$ if senator i is never a cosponsor any of senator j's bills. Because SAOMs require binary edges, we construct the edges as follows:

$$x_{ij} = \begin{cases} 1 & WPC_{ij} > 0.25 \\ 0 & WPC_{ij} \leq 0.25 \end{cases} \tag{3.5}$$

For each of the four senate sessions, in addition to the WPC value between any two senators in the session, we have three node covariates: the party affiliation of each senator, the number of bills they authored in each session, and their gender. We explored each of these covariates in the model to determine if they affect the overall network structure and how ties are formed between senators. The node-link diagram representations of the data we use for modelling are shown in Figure 3.1. We have labelled some of the nodes in these networks whose names will be familiar to US readers, because they are leaders in their party or they have run for president. The size of the nodes represent how many bills the senator authored in a session, the color represents party affiliation, and the shape represent gender. In each of the four sessions, there is one very large connected component tying many of the prominent senators together, with many smaller groups of two to ten senators surrounding the larger component. In each senate, the structure changes slightly as new senators arrive or come to prominence.

For Senate 111, for instance, we see Hillary Clinton, serving out her second term in the senate until she became Secretary of State. She is isolated in Figure 3.1, but in actuality, she had many cosponsors on two pieces of legislation she authored in that short time, as is shown in Figure 3.2. We chose to remove Clinton and her edges from the network because they make the overall structure look so different from the other three senates, where the pattern is not typical of a senate in any other year. We suspect that because Hillary Clinton had recently run for President had just been selected as President-elect Obama's Secretary of State, the cosponsorships were largely symbolic, so the 111^{th} Senate without Hillary Clinton is more typical than the 111^{th} Senate with her.

In legislative cosponsorship networks, it is well known that party affiliation, reciprocity of relationships, and whether senators are "workhorses" who author many bills or "showhorses" who author few bills, are major influences on structure (Ringe et al., 2017). We focus on these covariates, plus the additional sex covariate when choosing which SAO models to fit to the data.

3.3.4 Models of Interest

In addition to considering already well-known effects in legislative networks for application of our significance and goodness-of-fit methods, we first fit many other possible models and selected a few significant effects. To determine the effects that we would move forward with, we followed this procedure:

1. Define the simple effects structure of the data: the rate parameters and the outdegree and reciprocity parameters.

2. Add each additional possible effect, as determined by the effects documentation function, in `RSiena` one-at-a-time to the model's objective function (Ripley et al., 2013).

3. Fit each model to the data and check for convergence.

    (a) If the model converged, move to 4.

    (b) If the model did not converge, use the previous fitted values as starting values and repeat 5 times or until convergence, whichever comes first.

4. Test the added parameter for significance using a Wald-type test.

5. Report out the estimate of the additional parameter, its standard error, Wald p value, and convergence criterion.

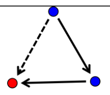
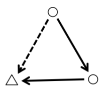
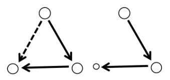
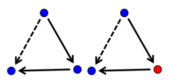
After completing the procedure for all model effects, we selected effects whose estimates converged, had a Wald p-value of less than 0.10, and seemed to have a reasonable interpretation for our data according to well-known properties of legislative networks (Ringe et al., 2017).

The parameters we use for the remainder of the paper are detailed in Table 3.1. The most significant effect was the jumping transitive triplet (JTT) parameter for the party covariate, which was estimated to be about -6 with a standard error of 0.11, resulting in a p-value of less than 0.0001. This estimate of the parameter associated with this statistic considers the number of transitive closures formed between two senators from different parties. The negative estimate is an indication that forming transitive ties between two people from different parties is discouraged, which tracks with the divisive nature of American politics, where party affilitation is the dominant effects. Another significant effect was the same JTT parameter for the sex covariate, with an estimate of about 3 with a standard error of 0.89. This parameter also consideres transitive closures, but for senators of different genders. The positive value indicates that transitive ties between senators of different genders are more likley to form. The covariate-related similarity score-weighted transitive triplets parameter estimate for the number of bills authored by a senator was also significant. We chose to look at similarity because the number of bills authored is more continuous than gender or party. The similarity measure is computed as:

$$sim_{ij}^{b} = \frac{\max_{hk} |b_h - b_k| - |b_i - b_j|}{\max_{hk} |b_h - b_k|} \tag{3.6}$$

where $\max_{hk} |b_h - b_k|$ is the range of number of bills authored by senators, and b_i, b_j are the number of bills authored by senators i, j repectively in the senate period. This effect was estimated at about 10 with standard error of 3.9. The high positive estimate suggests senators are encouraged to collaborate with other senators who author about the same number of bills they do. This tendency of senators to cosponsor bills written by senators who are similarly "prolific" corresponds to another well-known property of the U.S. Senate structure: the tendency of senators to be either "workhorses" or "showhorses". Senators known as workhorses author many pieces of legislation in a session, and largely stay out of the public arena. The showhorse senators, on the other hand, author relatively few pieces of legislation, and tend to appear on television, radio, and other media

Table 3.1: The additional effects we used in the SAOMs fit to the senate data. * - $sim^b_{ij} = \frac{\max_{hk}|b_h - b_k| - |b_i - b_j|}{\max_{hk}|b_h - b_k|}$ is the similarity score between two senators based on the number of bills authored, and $\overline{sim}^b = \frac{1}{n(n-1)}\sum_{i \neq j} sim^b_{ij}$ is the average bill similarity score between any two senators.

β_k	**Effect name**	**Interaction Variable**	**Formula**	**Picture**	**Initial estimate**	**Wald p-value**
β_3	jumping transitive triplet	party	$s_{i3}(x, \mathbf{p}) = \sum_{j \neq h} x_{ij}x_{ih}x_{hj} \cdot \mathbb{I}(p_i = p_h \neq p_j)$		-5.884	< 0.0001
β_4	jumping transitive triplet	sex	$s_{i4}(x, \mathbf{s}) = \sum_{j \neq h} x_{ij}x_{ih}x_{hj} \cdot \mathbb{I}(s_i = s_h \neq s_j)$		3.335	0.0002
β_5	similarity transitive triplet	bills	$s_{i5}(x, \mathbf{b}) = \sum_j x_{ij}x_{ih}x_{hj} \cdot (sim^b_{ij} - \overline{sim}^b)^*$		9.821	0.0128
β_6	same transtive triplet	party	$s_{i6}(x, \mathbf{p}) = \sum_j x_{ij}x_{ih}x_{hj} \cdot \mathbb{I}(p_i = p_j)$		1.306	0.0642

a great deal. Finally, we found the same party transitive triplet effect was also significant, with a fitted value of 1.3 and standard error of 0.7, meaning that transitive relationships between senators tend to form when they are from the same party, exactly as we would expect in a legislative body in a country with extremely entrenced partisan divides as in the US.

We examine a total of six models, each identified by its objective function:

1. Model M1: $f_i(x, \boldsymbol{\beta}) = \beta_1 s_{i1}(x) + \beta_2 s_{i2}(x)$
2. Model M3: $f_i(x, \boldsymbol{\beta}, \mathbf{p}) = \beta_1 s_{i1}(x) + \beta_2 s_{i2}(x) + \beta_3 s_{i3}(x, \mathbf{p})$
3. Model M4: $f_i(x, \boldsymbol{\beta}, \mathbf{s}) = \beta_1 s_{i1}(x) + \beta_2 s_{i2}(x) + \beta_4 s_{i4}(x, \mathbf{s})$
4. Model M5: $f_i(x, \boldsymbol{\beta}, \mathbf{b}) = \beta_1 s_{i1}(x) + \beta_2 s_{i2}(x) + \beta_5 s_{i5}(x, \mathbf{b})$
5. Model M6: $f_i(x, \boldsymbol{\beta}, \mathbf{p}) = \beta_1 s_{i1}(x) + \beta_2 s_{i2}(x) + \beta_6 s_{i6}(x, \mathbf{p})$
6. Model M7: $f_i(x, \boldsymbol{\beta}, \mathbf{p}, \mathbf{b}, \mathbf{s}) = \beta_1 s_{i1}(x) + \beta_2 s_{i2}(x) + \beta_4 s_{i4}(x, \mathbf{s}) + \beta_5 s_{i5}(x, \mathbf{b}) + \beta_6 s_{i6}(x, \mathbf{p})$

Table 3.2: The final estimates from repeated estimation of our models of interest. When simulating from these models, these are the estimates that we will use unless otherwise stated.

Model	$\hat{\alpha}_1$	$\hat{\alpha}_2$	$\hat{\alpha}_3$	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$	$\hat{\beta}_4$	$\hat{\beta}_5$	$\hat{\beta}_6$
M1	2.441	2.46	2.204	-4.903	4.893	–	–	–	–
M3	2.44	2.46	2.204	-4.902	4.893	-3.45	–	–	–
M4	2.438	2.461	2.211	-4.918	4.898	–	3.34	–	–
M5	2.442	2.459	2.206	-4.917	4.89	–	–	10.091	–
M6	2.443	2.461	2.205	-4.911	4.881	–	–	–	1.329
M7	2.441	2.459	2.21	-4.923	4.892	–	2.374	6.966	0.205

We fit models M1 through M6 in `RSiena` using Markov Chain Monte Carlo (MCMC) methods to approximate the method of moments estimates of the parameters. Because the estimation is done through MCMC simulation, we fit each model to the data 1,000 times to get a better estimate of the true value of $\boldsymbol{\beta}$. From the simulations that converged, which made up over 90% of the fits for each model, we computed the mean of the 1,000 estimates of each parameter to get final estimates of $\hat{\boldsymbol{\beta}}$ for each model, which are given in Table 3.2.

We want to explore the role of each of these parameters in the objective functions for each model. So, we use the estimates given in Table 3.2 to simulate from models M1 through M6. We discuss the simulation procedure and how we use the simulations in Section 3.4.

3.4 Experiment Set-Up

We want to explore three different aspects of the SAOM models using the lineup protocol: (1) significance of parameters, (2) goodness-of-fit of a model, and (3) visual power of the effects. Each one of these situations requires a different setup, which we describe in detail, but we make ue of the lineup protocol for all of these aspects.

In each lineup, we include plots from two models: the null model and an alternative model. The definition of the null and alternative model varies with the aspect of the SAOMs we are exploring.

Typically, a lineup shows sets of 20 plots at a time c.f. Loy and Hofmann (2015); Vander Plas and Hofmann (2015), but we determined that not enough structure could be shown in each plot for 20 node-link diagrams. We chose to expose our participants to only six plots at a time in order to show the node-link diagrams in more detail and to lower cognitive load for participants. To construct a lineup, we simulate five networks from the null model and one network from the alternative model. An example of a lineup like those shown to our participants is given on the right side of the image in Figure 3.3. In this lineup, model M4 is the alternative model, and model M1 is the null model.

To simulate lineups from the models we used the `siena07` function in `RSiena` (Ripley et al., 2013). Sections 3.4.1 through 3.4.3 describe in detail how we set up the lineups, which parameter values we simulate from, and why. Sets of the lineups we create are shown to independent observers recruited through Amazon Mechanical Turk for feedback (more details on the Turk setup in Section 3.5).

3.4.1 Significance Testing

In the significance testing protocol, a parameter of interest is selected to test, say β_k. The hypotheses we use to generate lineups are:

$$H_0 : \beta_k = 0 \qquad \text{versus} \qquad H_A : \beta_k \neq 0 \tag{3.7}$$

Under the null hypothesis, we assume that the model that generated the network data is M_1, the simplest model presented in Section 3.3.4. Thus, the five null plots in the lineup are simulations from M1 with β_1, β_2 set to the estimates given in Table 3.2 for M1. The alternate model is the model with β_1, β_2, and β_k in the objective function, with the remaining plot simulated from the appropriate model with values set to the estimates from Table 3.2.

The lineup generated under this scenario is shown to a number of independent viewers. If an observer picks out the alternative data plot, that is evidence against the null hypothesis, while picking one of the null plots is evidence in favor of the null hypothesis. The significance tests that we perform in our experiment are for β_3 and β_4, making the alternative models M3 and M4, respectively, for three repitions each.

3.4.2 Goodness-of-Fit

For the goodness-of-fit tests, we compare one model of interest, say Mi to the data. The hypothesis we use to generate lineups are

H_0: The data come from model Mi

H_A: The data come from some other, unknown model

To generate the null plots, we simulate five networks from model Mi using the corresponding parameters in Table 3.2. We pick a wave to focus on, wave two, which is the first simulated network, and among these five plots, we place a node-link diagram of the true second wave data. We cannot show the data more than once to each participant, so we examine several different models with three repitions each in our Amazon Mechanical Turk experiment, with each participant never seeing the true data wave twice. The models we chose for goodness-of-fit testing are M3, M4, M5, and M7.

3.4.3 Visual Power

Through visual inference, we want to determine at which point an effect becomes noticeable in a SAOM. By *noticeable*, we mean that the inclusion of the effect alters the appearance of networks simulated from a model to a degree that viewers are able to reliably pick out a node-link diagram rendered from data simulated from a model *with* the effect in a lineup among plots *without* the effect. This is a way to determine the power of the visual significance test. We explore all parameters in the objective function, $\beta_1, \ldots, \beta_6$ in this way.

In model M1, with only two parameters in the objective function, we varied both the density and reciprocity parameter values one at a time, keeping all other parameters at their fitted values given in Table 3.2. In models M3 through M6, we vary the additional parameter, β_3 through β_6, respectively. Thus, we have six different parameters of interest to us in total: $\beta_1, \ldots, \beta_6$. We want to determine how the size of these parameters affects the overall structure of the network data simulated from the models M1 through M6, so we also vary the value of the parameters in order to determine at what level the effects become noticeable.

To determine the threshold at which an effect becomes noticeable, we examine six different levels of the effect, three negative and three positive ones. Figure 3.4 shows a sketch of what the detection probability by participants' looks like hypothetically with varying effect size: the higher in absolute value the parameter is, the more likely participants are to choose the alternative model out of the lineup. To determine the exact values of the six levels we want to test for each effect, we started with the estimates of the parameter at hand (see Section 3.3.4), and used small negative and positive factors to determine at what point *we* noticed the effect of the parameter in simulations from the changed models. In Figure 3.4, we demonstrate these values with the vertical lines labeled "easy," "medium," and "hard." We expected most viewers to see the effects at the "easy" values, and we expected very few, if any, viewers to see the effects at the "hard" values.

To decide on the values of the parameters to use for each difficulty level, we constructed an online application that created the lineup protocol for us to be the guinea pigs of our own experiment (Swan, 2013). A screen shot of the app we created with the `shiny` package by Chang et al. (2017) is shown in Figure 3.3. On the left side of the screen, the user[2] can input the information necessary for creating a lineup of the models M1 through M7 for the data in Section 3.3.3: first, choose to simulate only one plot from the specified model (analogous to changing the alternative model in the lineup protocol) or to simulate $M - 1$ plots from the specified model (analogous to changing the null model in the lineup protocol); the model of interest; the wave of the data to examine; if model M1 is selected, whether to alter the density or the reciprocity parameter; the size of the lineup; the amount by which to multiply the effect selected; a random seed for replicability; and a layout algorithm to use for the node-link diagrams. There is also a checkbox if the user wishes the nodes to be colored by the size of the connected component to which they belong. The plot(s) that appear(s) that are *not* from the model specified using the other options are simulated from model M1 with the estimates of the rate parameters, and β_1 and β_2 as given in Table 3.2.

[2]Please visit `https://sctyner.shinyapps.io/saom_lineup_creation/` to create lineups constructed from the models we present for this data for yourself.

Using the “Picking Lineups” web application, we settled on six parameter values to test for each of our six effects, $\beta_1, \dots \beta_6$. The complete details of the parameters tested using the lineup protocol is given in Table 3.3. In the case of both β_4 and β_6, we could not determine any values for negative effects that made the data simulated from M4 and M6 look different than null model simulations from model M1. We hypothesized that this was due to the negative effects *removing* visually interesting structural elements as opposed to *adding* noticeable structural elements. Therefore we decided that the lesser experienced participants in our experiment would also not be able to. Instead of testing the negative values of these effects, we are examining a different scenario: we placed 5 simulations from model M4 or M6 with positive values of the parameter with one simulation from model M1 in a lineup. We will refer to this as the “reverse” lineup scenario. We used the reverse scenario to determine if the perception of the effect size is symmetric: if an effect is noticed $x\%$ of the time at value $\beta_k = \beta_{k_0}$ when *one* simulation from the corresponding model is placed among *five* null plots from model M1, then when *five* simulations from the model with $\beta_k = \beta_{k_0}$ are put in a lineup with *one* simulation from model M1, the plot from the simpler model should be noticed about $x\%$ of the time as well.

Table 3.3: All conditions used for the MTurk experiment. For parameters $\beta_1, \beta_2, \beta_3$, and β_5 M1 served as null model. For β_4 and β_6, null model M1 and the alternative model switch roles in the reversed lineups, i.e. five plots show data simluated from the laternative model and only one plot shows data from M1.

Parameter	Condition	Easy Value	Medium Value	Hard Value
beta1	neg	-7.354	-6.6187	-5.883
	pos	-3.922	-4.1674	-4.412
beta2	neg	0.000	0.0005	0.049
	pos	7.340	6.8504	6.361
beta3	neg	-17.249	-10.3497	-3.450
	pos	10.350	6.8998	5.175
beta5	neg	-30.272	-20.1817	-10.091
	pos	20.182	17.6590	16.145
beta4	regular	8.351	6.6806	5.010
	reverse	6.681	5.0105	3.340
beta6	regular	5.316	3.9872	3.323
	reverse	5.316	3.9872	3.323

3.5 Experiment Results

We recruited 250 participants for our experiment through Amazon Mechanical Turk. Each participant was presented with some brief training material before beginning the experiment. After agreeing to participate, the participants were shown two trial plots, one where the data plot was the most different from the others due to its relatively complex structure, while the other trial included a data plot that was most different from the others due to its comparatively simple structure. Only when participants were able to correctly identify the data plot from the trial lineups were they allowed to begin the experiment. Each participant was randomly assigned 13 lineups to look at. The were asked to select one or more plots that they perceived as "most different" from the others, and provide a reasoning for their choice. They could select from "Most simple overall structure" or "Most complex overall structure", corresponding to what they saw in the examples and trials, or they could choose "Other" and provide their own text description of their reasoning. These language in these reasons is purposefully vague: we want participants to tell us what they see. We do not want to tell them what they "should" be seeing. In this way, we can truly determine what effects are *noticeable*.

Twelve of the 13 lineups that the participants saw were used for the significance testing and the visual power methods discussed in Sections 3.4.1 and 3.4.3. The final of the 13 lineups shown to participants contained the true data from the 112th senate shown in Section 3.3.3 placed among five other plots from models M3, M4, M5, and M7 as discussed in Section 3.4.2. Each participant only saw the data one time in order to avoid bias. Upon completion of the 13 lineups, each participant was paid \$1.75.

3.5.1 Significance Testing

For a SAOM, there are two ways a conventional significance test of the parameters can be performed. In `RSiena`, there are t-type and Wald-type tests for a single parameter and for multiple parameters. The t-type test statistic is simply the parameter estimate divided by its standard error, and compared to a standard normal distribution (Ripley et al., 2017). The Wald-type test statistic for a single parameter, β_k is

$$\frac{(\hat{\beta_k})^2}{var(\hat{\beta_k})} \sim \chi^2_1, \tag{3.8}$$

which is compared to a Chi-square distribution with one degree of freedom(Ripley et al., 2017). Testing the significance of multiple parameters depends on the hypothesis we wish to test, and a $P \times K$ matrix, A, must be appropriately designed to test the P hypotheses of interest. The null hypothesis is that $A\boldsymbol{\beta} = \mathbf{0}$, and the test statistic is

$$(A\hat{\boldsymbol{\beta}})'\hat{\Sigma}^{-1}A\hat{\boldsymbol{\beta}} \sim \chi^2_P, \tag{3.9}$$

where $\hat{\Sigma}$ is the estimated covariance matrix of $\boldsymbol{\beta}$. This statistic is then compared to a Chi-square distribution with P degrees of freedom.

Both the parameters we test for significance using the lineup protocol, β_3 and β_4, were determined to be statistically significant using Equation 3.8. The correspoding results from the significance tests we performed using the lineup protocol are given in Table 3.4. Corresponding to traditional methods, if enough participants pick out the alternative plot to result in a p-value less than 0.05, we reject the null hypothesis that the true value of the additional parameter, either β_3 or β_4, is equal to zero.

Table 3.4: Experiment results for the two parameters for which we performed significance tests. There were three lineups for each parameter, so there are three results for each plot.

Lineup ID	parameter	# Alt. Model Picks	Total Views	p-value
3131	beta3	4	29	0.60654
3132	beta3	26	31	0.00001
3133	beta3	2	27	0.80053
3141	beta4	10	23	0.03420
3142	beta4	3	37	0.77965
3143	beta4	10	29	0.09619

The p-values were calculated using the `vinference` package by Hofmann and Röttger (2016). This package contains methods to calculate *Visual distributions* for lineup experiment data. The distribution depends on the number of evaluations of a plot, K, the size of the lineup, m, and the lineup scenario, which here is that each lineup containing the same data and the same set of null plots is shown to K independent observers. The visual inference family of distributions is similar to the binomial distribution, but takes the dependency among the m plots in a single lineup shown to multiple viewers into account. Using these p-values, all but one lineup results in a rejection of the null hypothesis at Type-I error rate of $\alpha = 0.05$. We see that the p-values for visual inference here are highly variable.

The lineup for significance testing of β_3 which resulted in a very small p-value and rejection of the null hypothesis is shown in Figure 3.5. Another significance lineup for model M3, which resulted in failure to reject the null hypothesis, is shown in Figure 3.6. When viewing Figure 3.5, 26 of 31 viewers chose the plot from M3, while only 2 of 27 chose the plot from M3 when viewing Figure 3.6. The most common choice in the latter was panel two, which 16 of 27 viewers chose as the most different due to its large connected component, making it seem more complex than the others. In viewing these two lineups, it is evident that there is a large amount of variability. It is difficult to see that five of the six come from the same model when they can all look different in their own way. Thus, the variability in results is introduced through the null plots generated from M1, as not all simulations look alike. In addition, the necessarily small number of null plots do not give the viewer as complete of a view of the null model as the usual 19 null plots would. The results of the significance tests given in Table 3.4 for β_3 and β_4 are not definitive. For the test of β_3, two of the three tests are not significant, while the third is highly significant. For the test of β_4, one test is significant, one is decidedly not significant, and the third is significant at the level of 0.10. Thus, unlike the Wald-type tests described at the beginning of this section, there is no way to decisively reject or to fail to reject the null hypothesis that the parameter value is 0. We include all of the lineups shown to our participants in the appendix.

3.5.2 Goodness-of-Fit Testing

Goodness-of-fit testing for network models is notoriously difficult. Most network models, other than the most simple, lack the necessary asymptotics for developing goodness-of-fit methods (Goldenberg et al., 2010). Some methods have been developed based on what Ripley et al. call “auxiliary statistics" such as the indegree or outdegree distribution on the nodes. In `RSiena`, the `sienaGOF` function performs goodness-of-fit testing as follows:

1. Auxiliary statistics, such as the cumulative outdegree counts on the nodes, are computed on the observed data ($\mathbf{u}_d$) and on N simulated observations from the model ($\mathbf{u}_1 \dots \mathbf{u}_N$). (Usually, $N = 1000$)

2. The mean vector, $\overline{\mathbf{u}}$ and covariance matrix, $\mathbf{S}$ of the statistics on the simulations from the model are computed, and the Mahalanobis distance, $d_M(\mathbf{u})$ from the observed statistics to the distribution of the simulated statistics is computed:

$$d_M(\mathbf{u}) = \sqrt{(\mathbf{u} - \overline{\mathbf{u}})' S^{-1} (\mathbf{u} - \overline{\mathbf{u}})} \tag{3.10}$$

3. The Mahalanobis distance for each of the N simulations is calculated and $d_M(\mathbf{u}_d)$ is compared to this distribution of distances.

4. An empirical p-value is found by computing the proportion of simulated distances found in step 4 that are as large or larger than $d_M(\mathbf{u}_d)$. A SAOM is thus considered a good fit to the data if p is large. A plot comparing the data to the simulations is also considered, and a similar plot is shown in Figure 3.7 for the outdegree distribution of small data set, shown in the points and connected lines, with the simulated values of $\mathbf{u}_d$ shown in boxplots and overlaid violin plots.

The `RSiena` software also provides a Rao score-type test for goodness-of-fit for assessing one or more parameters, the test statistic of which is compared to a Chi-square distribution with P degrees of freedom, where P has the same definition as in Section 3.5.1. For full detail on the score-type test, see Schweinberger (2012).

These methods are both restriced: the `sienaGOF` method only considers *one* measure on the data and simulations from the model, while the score-type tests only consider *subsets* of parameters, "nuisance parameters" in Schweinberger (2012), not the entire set of parameters. By using visual inference instead of more traditional statistical methods, we hope to perform a more *holistic* goodness-of-fit test.

Using the lineup protocol, we show each Amazon Mechanical Turk worker the data once, in a lineup with five other plots of simulated data from one of the models we chose. We examined four different models, M3, M4, M5, and M7, and examined three repetitions of each, for a total of 12 goodness-of-fit lineups. In each lineup, the "null model" is one of the four models and the "alternative" model is the true, unknown model that generated the senate network data. The hypotheses for our goodness-of-fit tests are:

H_0: The senate network data come from (or could have come from) the null model, Mi.

H_A: The senate network data do not come from the null model.

If a lineup viewer picks out the data among the five simulations from the null model, it is evidence against the null hypothesis. On the contrary, if the lineup viewer picks one of the null plots, that is evidence in favor of the null hypothesis. Because the size of the lineups is small, the probability of picking the data by chance is high, $\frac{1}{6}$, but if *many* independent viewers pick out the data from the nulls, the evidence in favor of the alternative hypothesis becomes stronger. Results from our MTurk goodness-of-fit plots are provided in Table 3.5.

Table 3.5: An overview of the results from the 12 goodness-of-fit lineup tests.

Model	Replicate	Data Picks	Total Viewers	p-value
M3	1	29	36	< 0.0001
jtt party	2	13	18	0.0004
	3	16	20	< 0.0001
M4	1	13	16	< 0.0001
jtt sex	2	7	20	0.1150
	3	29	34	< 0.0001
M5	1	9	21	0.0414
stt party	2	21	24	< 0.0001
	3	14	16	< 0.0001
M7	1	17	20	< 0.0001
	2	14	28	0.0093
	3	28	37	< 0.0001

The p-values were again computed using the `vinference` package by Hofmann and Röttger (2016). The lineup that resulted in a failure to reject the null hypothesis is shown in Figure 3.8.

The null model in this lineup is M5, and the senate data is shown in panel number $3^2 - 7$. However, the panel most participants chose was number four, and the most common reasoning for that choice was that it had the most simple structure. Some of the other panels, such as three and six, in Figure 3.8 have large connected components that are similar in size to the connected component of the data plot shown in panel two. Thus, model M5 is sometimes capable of capturing the network structure of the senate collaboration data.

The smallest p-value for one of the goodness-of-fit lineups was for the third replicate of the null model M5. This result contrasts with our previous finding that the only lineup to fail to reject the null was also when the null model was M5. This lineup is shown in Figure 3.10. In the remaining replicate of M5 as the null model, 13 of 16 viewers identified the data plot, corresponding to a p-values of less than 0.0001, just like the third replicate. This variability in results is similar to the variability we found in Section 3.5.1. This variability is again introduced through the plots simulated from null model, and does not provide us with a clear cut decision resulting the hypothesis test. For model M5, we can neither reject nor fail to reject the null hypothesis that the data come from model M5. This is evidence that the goodness-of-fit of network models cannot always be determined by one dimensional derived features, such as p-value shown on the x-axis in Figure 3.7.

For the other models for which we tested goodness-of-fit, however, we *do* have significant evidence from all three replicates to reject the null hypothesis that the null model generated the data. For models M3, M5, and M7, these goodness-of-fit tests have rejected the null hypotheses that the senate data come from these models. We hypothesized that the model with the most effects, M7, would be the best fit. However, as shown in Figure 3.9, the model does not capture the overall structure very well at all. The rest of the goodness-of-fit lineups as shown to participants are provided in the appendix.

We believe this goodness-of-fit testing method holds promise for the future of social network analysis. The participants in our experiments are very good overall at picking out the data when it is noticeably different from the null plots in the lineups. In addition, as in replicate three for null model M4, when the null plots contain similarly sized structures as the data plot, our participants have a hard time distinguishing the data. We believe that running these tests multiple times using several different sets of null models to adequately explore the possible structures generated by the models is a step in the right direction for a more comprehensive goodness-of-fit test for network models.

3.5.3 Visual Power

A summary of the results from our experiment is shown as points in Figure 3.11. On the x axis, we plot the value of the parameter of interest, and on the y axis, the proportion of times the alternative data plot was picked out for each lineup. The results are split into groups based on the value of the parameter and the lineup type. We can see clear patterns in the added parameters $\beta_3, \ldots, \beta_6$: as the parameter value approaches 0, fewer participants identified the alternative plot. Similarly, as β_1, β_2 approach their estimated values $\hat{\beta}_1, \hat{\beta}_2$, fewer people are able to identify the alternative plot.

We further explore this relationship between identification of the alternative data in the lineup and the parameter of interest, effect size, and lineup type with a generalized linear mixed model that provides us with an estimate of the power of the visual significance test. The response variable, Y_{ijkm}, is binary, indicating whether participant m picked the alternative data plot in lineup type j, rep k, for effect i. There is one continuous covariate x, which is the centered and scaled size of the effect of interest from which the alternative data were simulated, the values of which are labeled "easy", "medium", and "hard" in Table 3.3 according to how difficult we thought the Turk participants would find each lineup. In Equation 3.11, $i \in \{1, 2, 3, 4, 6\}$ corresponds to the effects $\beta_1, \ldots, \beta_6$, respectively, $j \in \{-1, 1\}$, and $k \in \{1, 2, 3\}$. We also include random effects in the model: one for each lineup, δ_{ijk}, and one for each participant, ϵ_m, and fit a hierarchical model given in Equation 3.11.

$$
\begin{aligned}
Y_{ijkm} &\sim \text{Bernoulli}(\pi_{ijkm}) \\
\text{logit}(\pi_{ijkm}) &= \alpha_{ij} + \gamma_{ij}x + \delta_{ijk} + \epsilon_m \\
\delta_{ijk} &\overset{iid}{\sim} N(0, \sigma^2_\delta) \\
\epsilon_m &\overset{iid}{\sim} N(0, \sigma^2_\epsilon)
\end{aligned}
\tag{3.11}
$$

The results of fitting this model, including estimates of parameters, standard errors, p-values, and the odds ratio multipliers, using `glmer` from the `lme4` package are summarized in Table 3.6 (Bates et al., 2015). For each combination of parameter and lineup type, the expected value of the link function for a new lineup and a new observer with parameter value x is

$$
E[\text{logit}(\pi_{ij})] = \alpha_{ij} + \gamma_{ij}x \tag{3.12}
$$

and the corresponding probability of picking out the alternative data plot is

$$
\pi_{ij} = \frac{\exp\{\alpha_{ij} + \gamma_{ij}x\}}{1 + \exp\{\alpha_{ij} + \gamma_{ij}x\}} \tag{3.13}
$$

In Figure 3.11, we see a clear trend in all parameters except β_1 and β_2 that as the parameter value approaches zero from either side, the probability of picking the data plot in a lineup of size six descreases. For β_3 and β_5, the slope of the fitted line is *much* steeper for positive values of the parameter than for negative values, meaning that our participants perceived differences more often for postitive parameter values than for negative parameter values. This finding is similar to that of Harrison et al. (2014), who found that people detect positive correlations sooner and better than negative correlations.

We expand portions of Figure 3.11 in Figures 3.12-3.14. These figures show the same prediction regions as in Figure 3.11, plus some additional predictions outside of the data range shown in gray. Again, the points represent the results from the experiment. In all three of these figures, the lack of symmetry is apparent. In the reverse lineup scenario shown in Figure 3.14, the probability of prediction is consistently far less than the probability of prediction in the regular lineup scenario. This demonstrates that the visual signal of one plot from M4 among five plots from M1 is much

Table 3.6: Summary of the results from fitting the model given in Equation 3.11. Significance levels: * - < 0.10; ** - < 0.05; † - < 0.01; ‡ - < 0.001

Parameter	Estimate	Std Error	p-value	Odds Multiplier
α_{1+}	37.297	6.573	<0.0001‡	>1e+06
α_{1-}	−9.933	3.473	0.0042†	<0.0001
γ_{1+}	75.356	13.532	<0.0001‡	>1e+06
γ_{1-}	−14.504	4.804	0.0025†	<0.0001
α_{2+}	−6.833	4.466	0.1260	0.0011
α_{2-}	−17.001	2.236	<0.0001‡	<0.0001
γ_{2+}	13.771	7.446	0.0644*	956752.4844
γ_{2-}	−229.16	31.306	<0.0001‡	<0.0001
α_{3+}	−2.801	0.949	0.0032†	0.0608
α_{3-}	−2.644	0.811	0.0011†	0.0711
γ_{3+}	4.474	1.389	0.0013†	87.7507
γ_{3-}	−1.108	0.609	0.0690*	0.3304
α_{4+}	−2.078	0.954	0.0293**	0.1252
α_{4-}	−2.692	1.322	0.0417**	0.0678
γ_{4+}	4.247	2.147	0.0479**	69.8675
γ_{4-}	2.403	2.187	0.2719	11.0585
α_{5+}	−5.84	2.989	0.0507*	0.0029
α_{5-}	−4.686	0.86	<0.0001‡	0.0092
γ_{5+}	3.264	1.756	0.0630*	26.1487
γ_{5-}	−2.176	0.387	<0.0001‡	0.1136
α_{6+}	−1.164	1.226	0.3425	0.3123
α_{6-}	−5.929	1.28	<0.0001‡	0.0027
γ_{6+}	5.76	3.524	0.1021	317.4753
γ_{6-}	15.092	3.605	<0.0001‡	>1e+06
σ^2_δ	0.564	–	–	–
σ^2_ϵ	0.342	–	–	–

stronger than that of one plot from M1 among five plots from M4. We posit that the latter is a more difficult task because it involves noticing a *lack of structure* as opposed to the presence of more structure. We can see a similar effect in Figure 3.13. At a value of $\beta_5 = 20$, the model predicts a probability of about 0.60 that a new viewer of a new lineup will identify the alternative data plot. At a value of $\beta_5 = -20$, however, the model predicts this same probability to be about 0.40. This again demonstrates that the presence of structure is detected sooner and more frequently than the absence of structure.

For β_4 and β_6, where one plot simulated from M1 was placed among five plots from the corresponding model, we see that the predictions for the reverse lineup type (-1), are less than the standard lineup type (1) for all values of the parameter that we have. This contradicts our hypothesis for this scenario, which was that these two scenarios would perform similarly. One of the lineups for the $\beta_4 = 6.681$, lineup type 1 scenario is given in Figure 3.15, and a corresponding lineup for the lineup type -1 scenario is given in Figure 3.16. For identical values of the parameter, viewers had a harder time identifying the different plot when they were selected the most "simple" structure, detecting M1 in five plots from the more complicated model, than they did identifying the most "complex" structure, the plot from the more complicated model, from the five plots from M1. This result is also similar to that of Harrison et al. (2014) because it emphasizes the difficulty of picking out the absence of an effect relative to picking out the presence of an effect.

3.6 Discussion

By using visual inference methods, we have developed new ways to perform significance and goodness-of-fit testing for a complicated and intractable set of statistical models for social network data. We have also developed a way to determine the power of these new visual tests. Our methods can be used to supplement traditional methods and check our assumptions about network models. The traditional methods only look at one piece or derived measure of a network model, whereas our methods look at the models holistically for a broader sense of what it means for a parameter to be significant or a model to be a good fit. By looking at an entire network simulated from a SAOM side-by-side with other instances of networks simulated from another model, instead of singular features, we develop an idea of the model in terms of the *data* itself, instead of in terms of statistical summaries of the data. These methods place the model in the data space, instead of summarizing or compressing the data to place it in the model space.

Furthermore, we have found the visual power of some effects in the object function of a SAOM for this particular senate data example, and we have shown that, for the same effects, there is a lot of variability in results from significance and goodness of fit tests. Because the visual tests we performed show a great deal of variability, we can see that the decisions with respect to the significance of a parameter or the goodness of fit of a model to data are not as cut-and-dried as the more traditional methods would have us believe.

These results do not come without limitations. In visual inference, the null plots are supposed to play the role of good representatives of the null model. Here, the number of null plots is reduced to five, which increases the variability seen in a single lineup dramatically, and can unfortunately lead to very different conclusions for the same lineup scenario. Furthermore, these results do not generalize to all SAOMs or to some subset of SAOMs. The lineups shown are made for only one set of data, and it is not clear whether the power results transfer, nor is it clear to what degree if they do transfer, to other situations with different number of actors, different edge densities, or different layout algorithm of the node-link diagram. We can make some generalizations about what participants are picking up on in the lineups based on their feedback and previous research, but we cannot apply our hierarchical model directly to lineups constructed for new data or new models or parameters.

We hope to apply these methods further for different types of network data and different types of network models. We accept the limitations of this type of network data visualization, in that even in small instances, the cognitive load of looking at a lineup is very high for the average observer. We would therefore like to explore larger datasets, different layout algorithms, and different ways of visualizing network data, such adjacency matrix visualizations, using visual inference to see if similar patterns emerge.

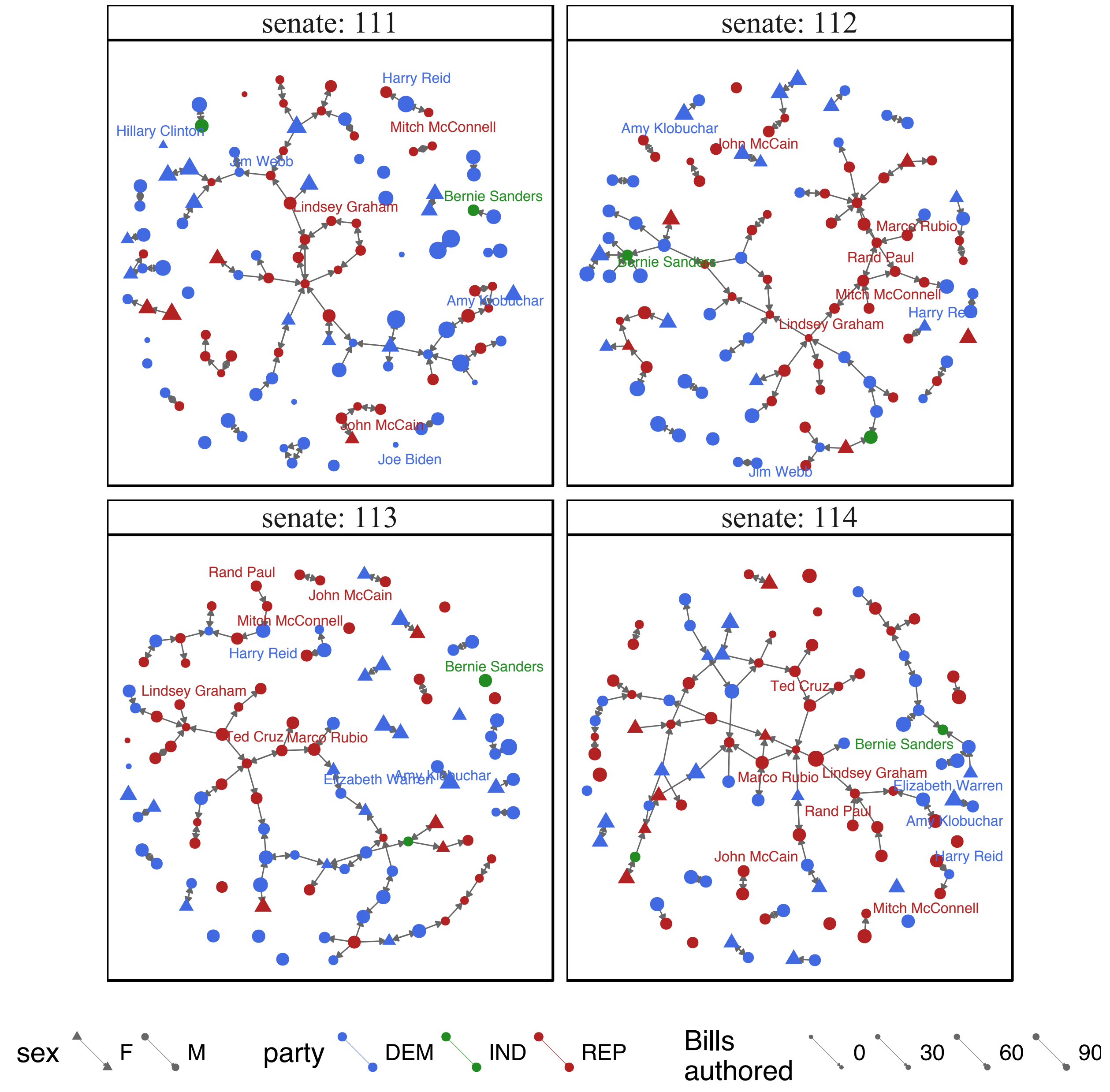


Figure 3.1: The four senate collaboration networks that we use as our example data to visually assess the SAOM effects. Color represents party, shape represents gender, and size represents number of bills authored in a session. The Frucherman-Reingold layout is shown.

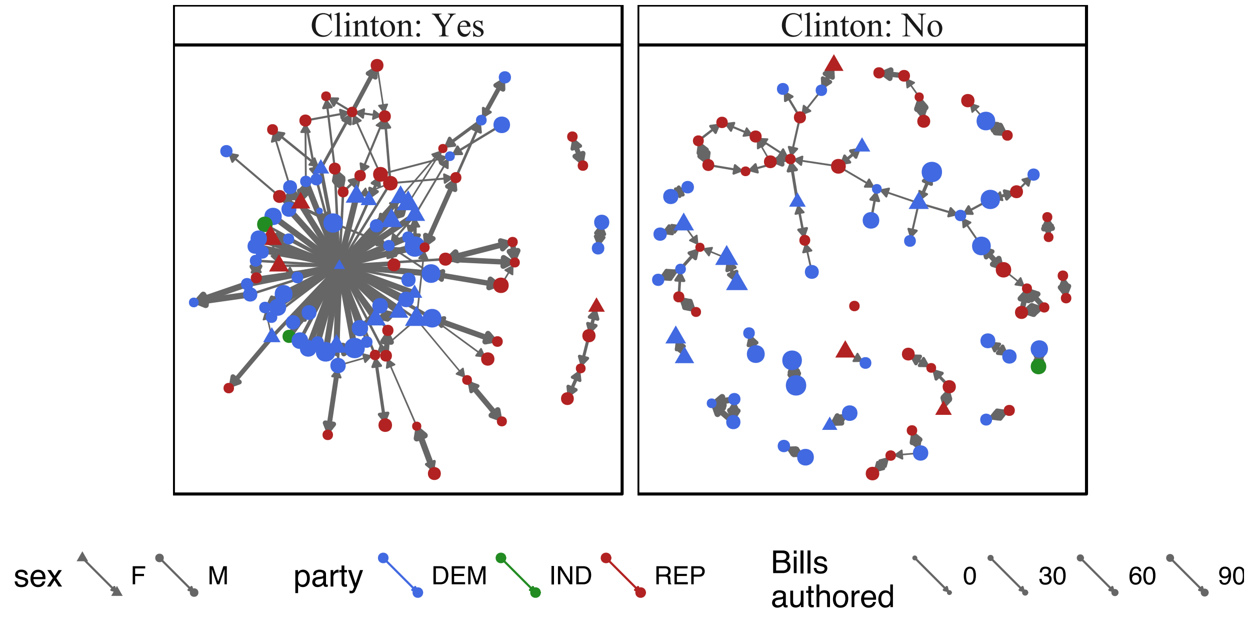
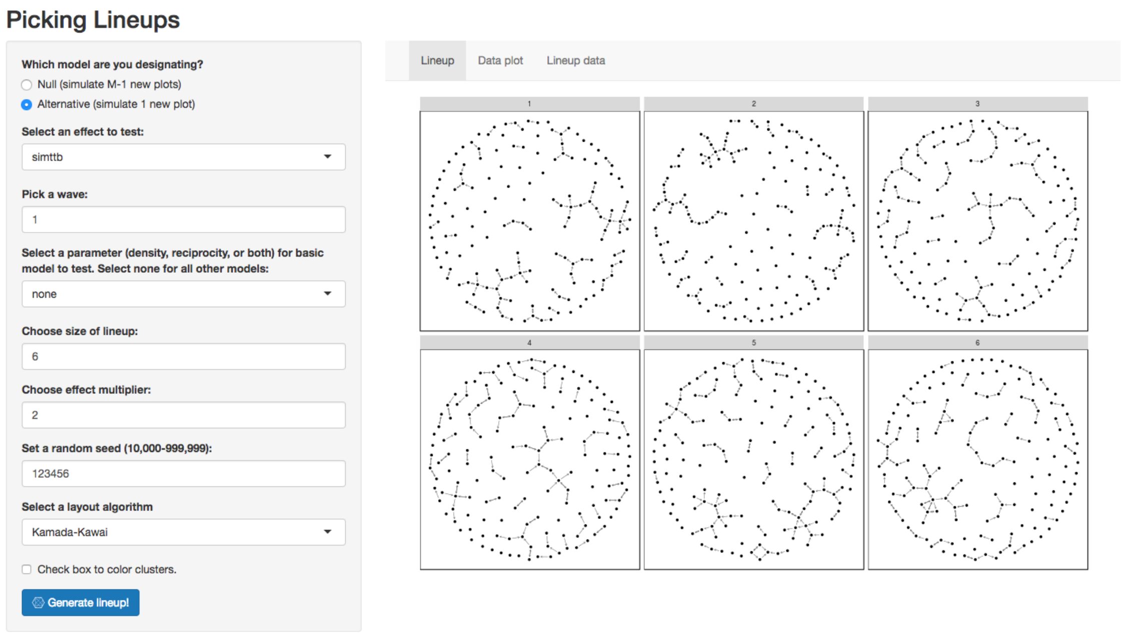


Figure 3.2: We removed Hillary Clinton's ties from the network because she had abnormally high collaboration with senators during the time she was in the 111th senate and before she left office to become Secretary of State.


Picking Lineups

Which model are you designating?
Null (simulate M-1 new plots)
[✓ Alternative (simulate 1 new plot)]

Select an effect to test: simttb

Pick a wave: 1

Select a parameter (density, reciprocity, or both) for basic model to test. Select none for all other models: none

Choose size of lineup: 6

Choose effect multiplier: 2

Set a random seed (10,000-999,999): 123456

Select a layout algorithm: Kamada-Kawai

Check box to color clusters.

Figure 3.3: A screen shot of the web application we created to design our lineup experiment. More details about this application are given in Section 3.4.3. In the lineup, M5 is the alternative model with β_5 set to twice its estimated value given in Table 3.2. One plot simulated from this model is placed at random among five observations simulated from the null model, M1. Participants of the study are asked to identify the most different plot.

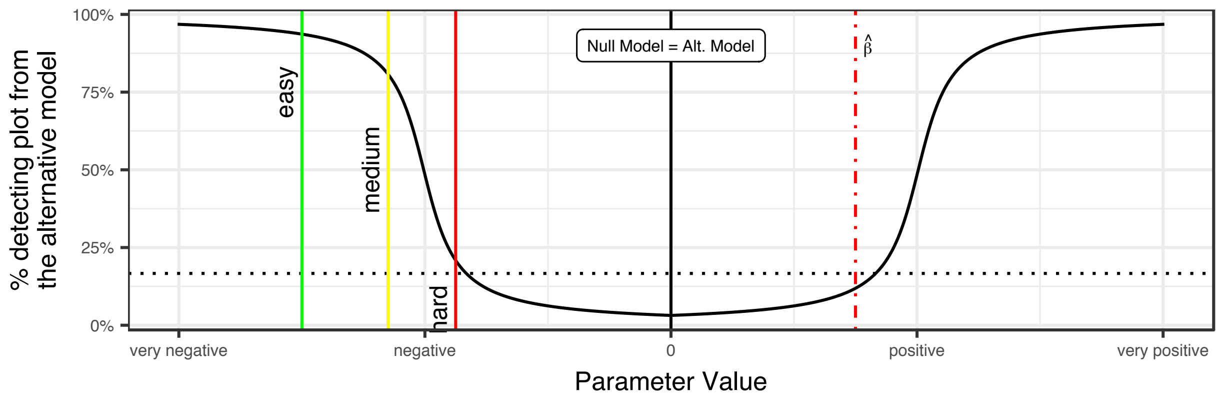


Figure 3.4: We hypothesize that as the parameter value of interest increases in absolute value, more viewers of the lineup will pick the alternative data out of a lineup. Note that the significance test we construct in Section 3.4.1 is just one point on the line below, represented by the vertical dotted line labeled $\hat{\beta}$. The easy, medium, and hard lines represent how we determined which values of the parameters to show to our participants, and the horizontal dotted line shows the type-I error for one viewer of a lineup of size 6.

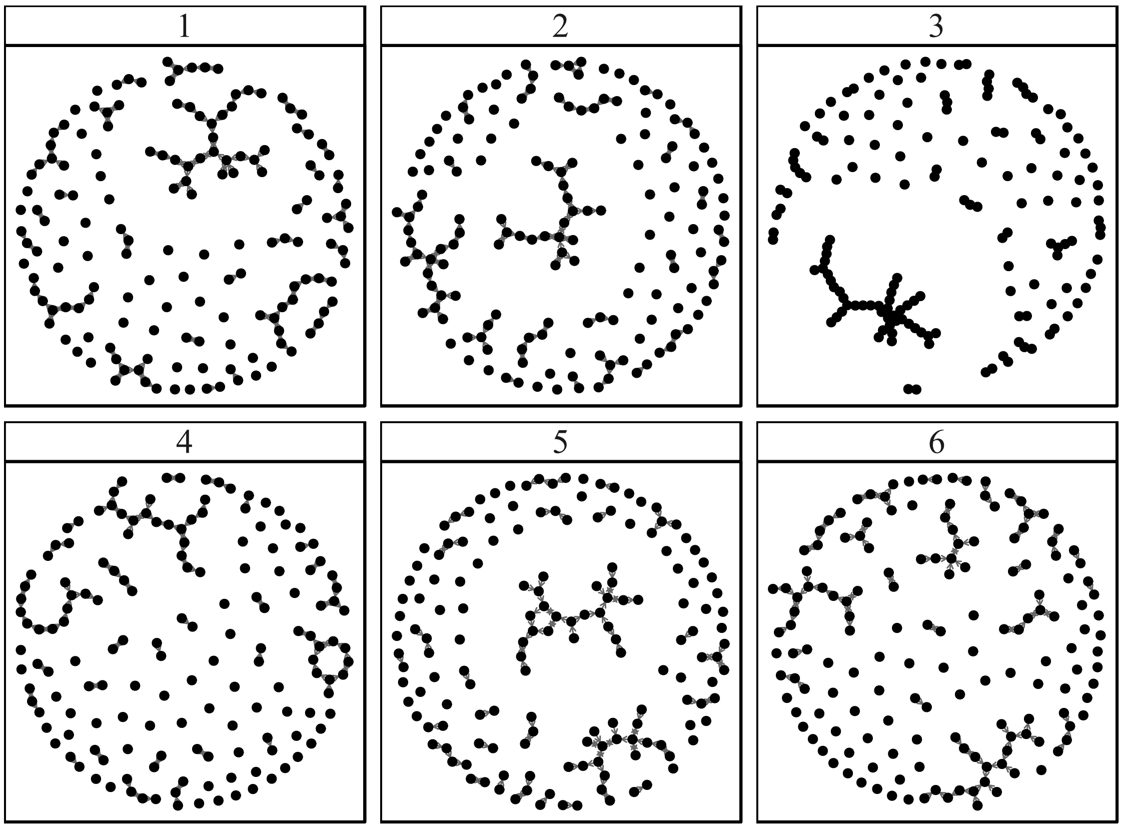


Figure 3.5: The lineup which caused a rejection of the null hypothesis that $\beta_3 = 0$. The network simulated from model M3 is found in panel $\sqrt{16} - 1$, and the remaining panels show networks simulated from model M1.

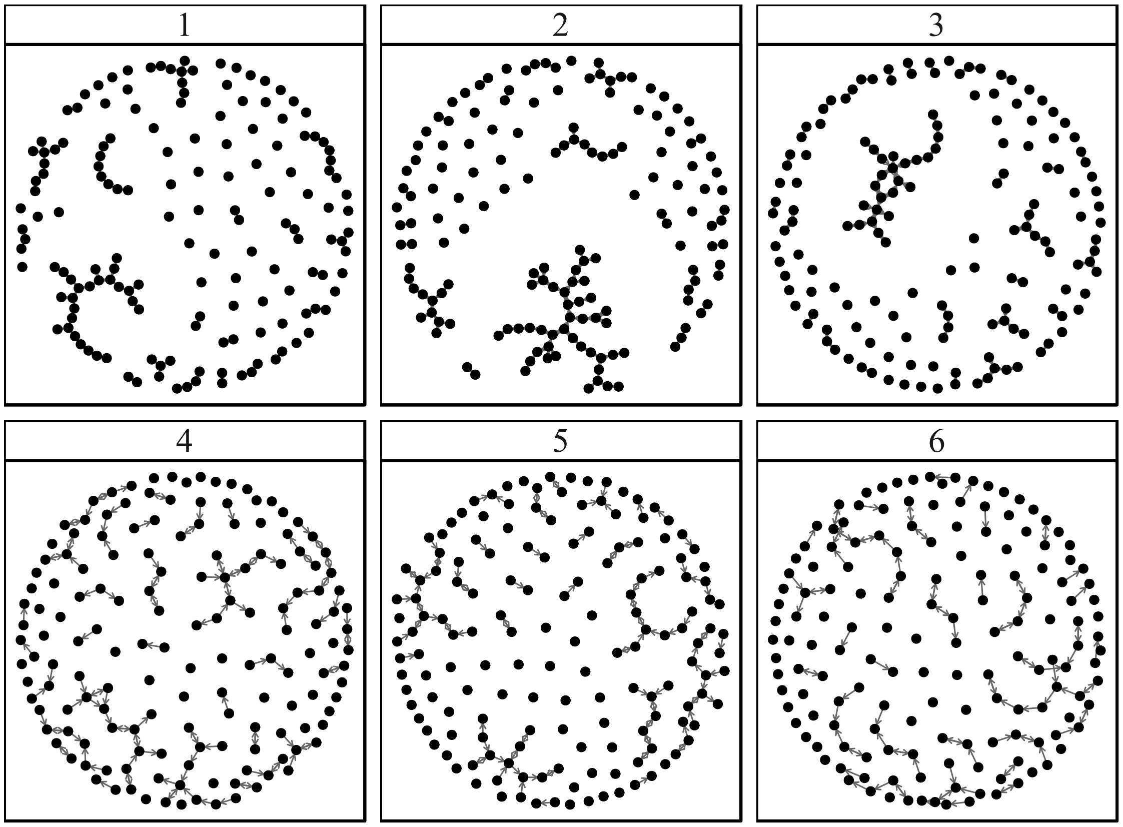


Figure 3.6: One of the lineups which failed to reject the null hypothesis that $\beta_3 = 0$. The network simulated from model M3 is found in panel $\sqrt{25} - 4$, and the remaining panels show networks simulated from model M1.

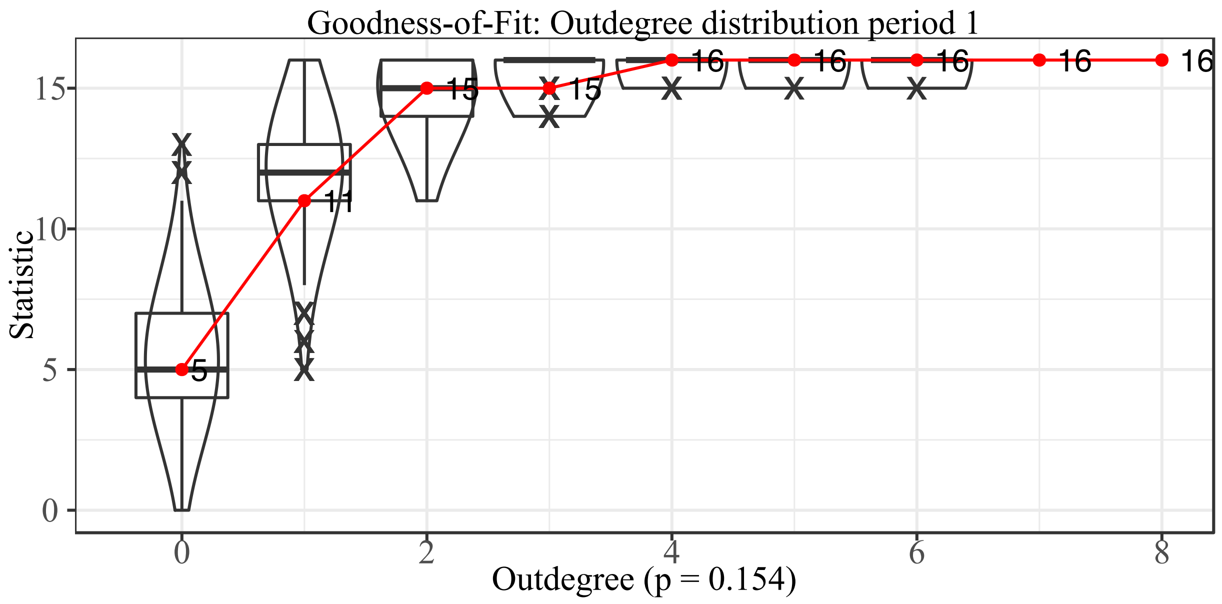


Figure 3.7: An example of what a goodness-of-fit plot from `RSiena` looks like. The overlaid boxplots and violin plots show the distribution of each of the outdegree count values on the simulated networks, and the red points and lines are the observed data values.

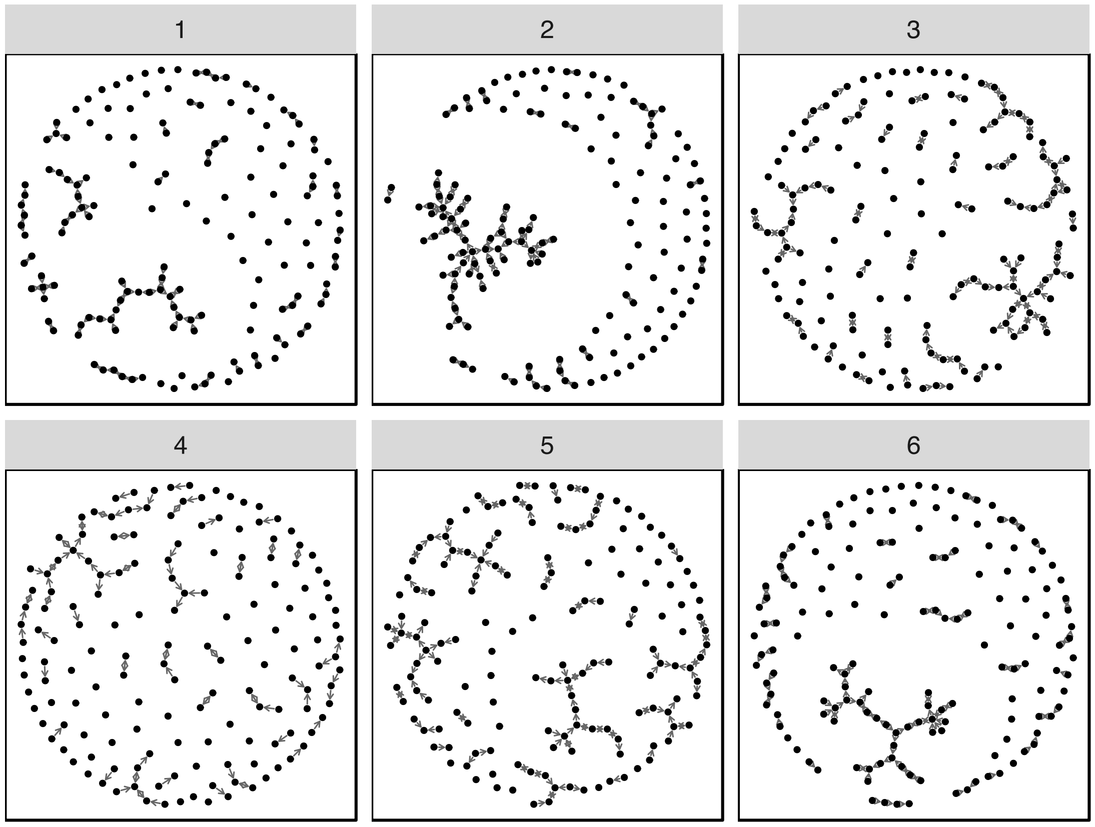


Figure 3.8: The goodness-of-fit lineup that failed to reject the null hypothesis. The null model for this lineup is M5. Only 7 of 20 viewers of this lineup selected the data plot as the most different from the others. The most commonly chosen panel was number four, which has a relatively simple structure compared to panels 2, 3, and 6 especially.

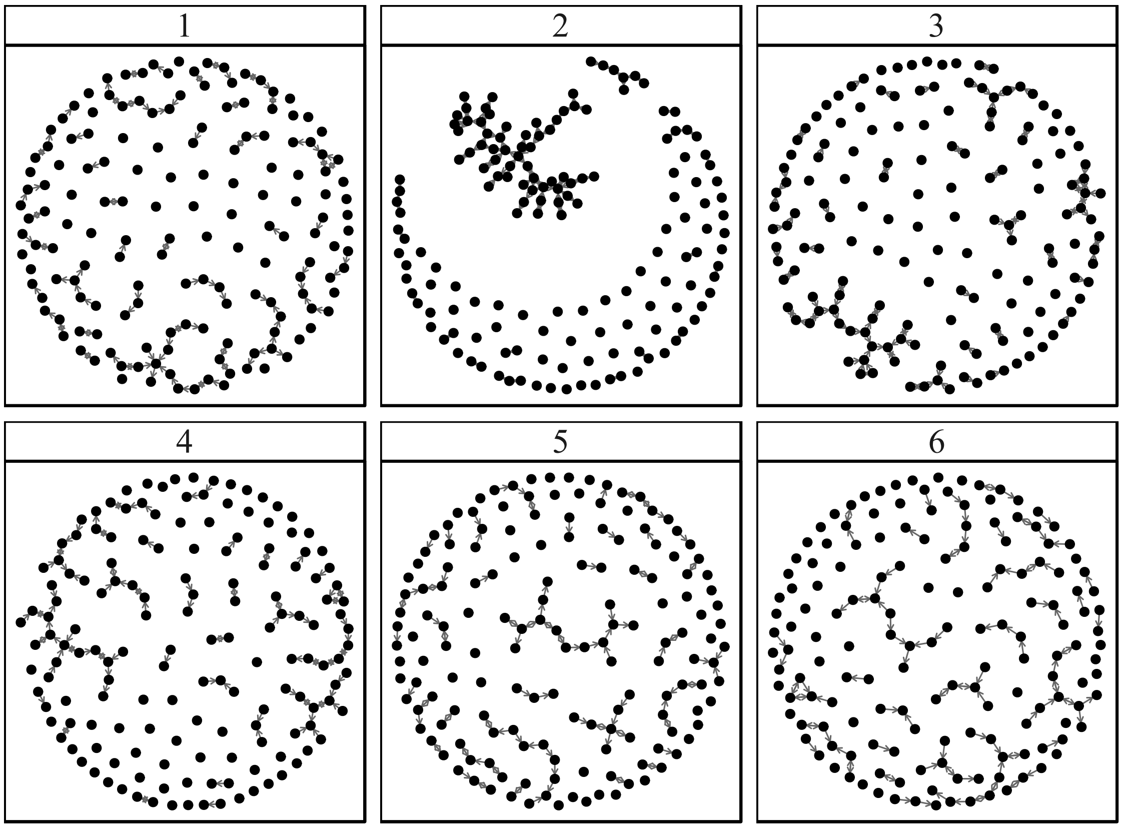


Figure 3.9: One repitition of a goodness-of-fit lineup testing modle M7. The senate data are shown in panel two, and it is evident that none of the other five panels, which show data simulated from model M7, come close to creating the large connected component that is central to the structure of the senate data.

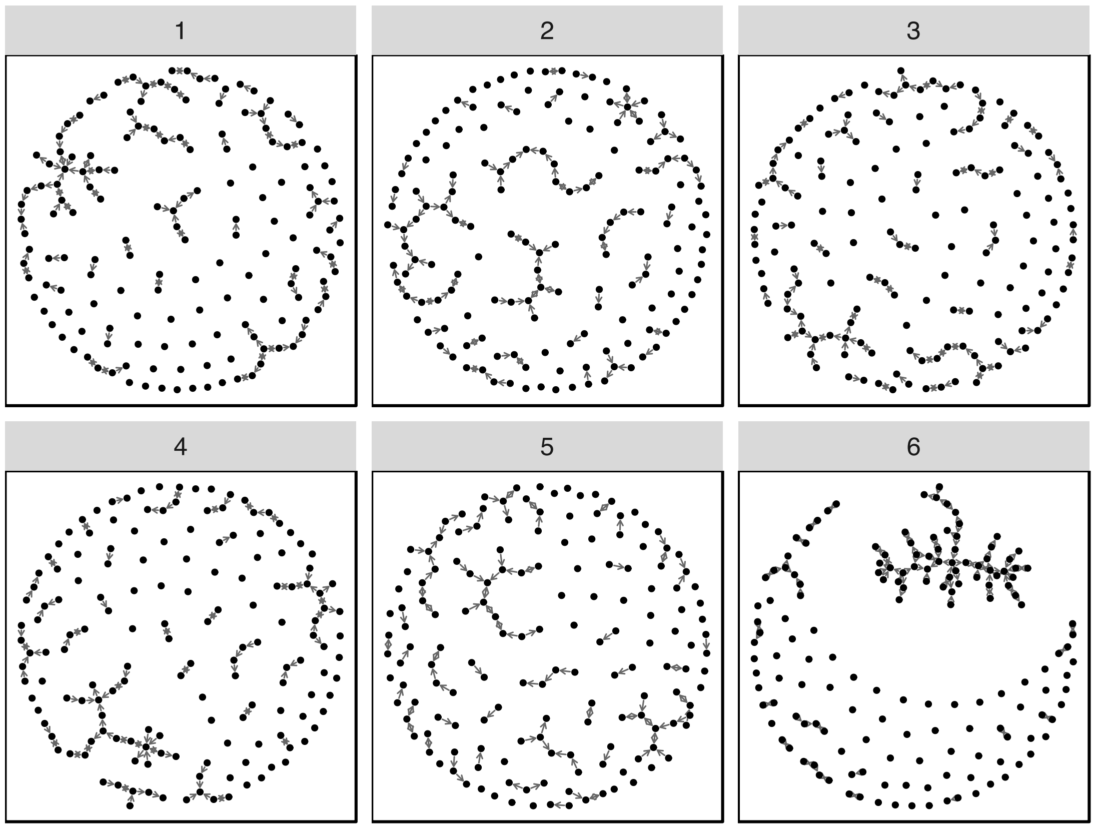


Figure 3.10: The lineup resulting in the smallest p-value rejecting the null hypothesis. Surprisingly, this another repetition for M5 as the null model.

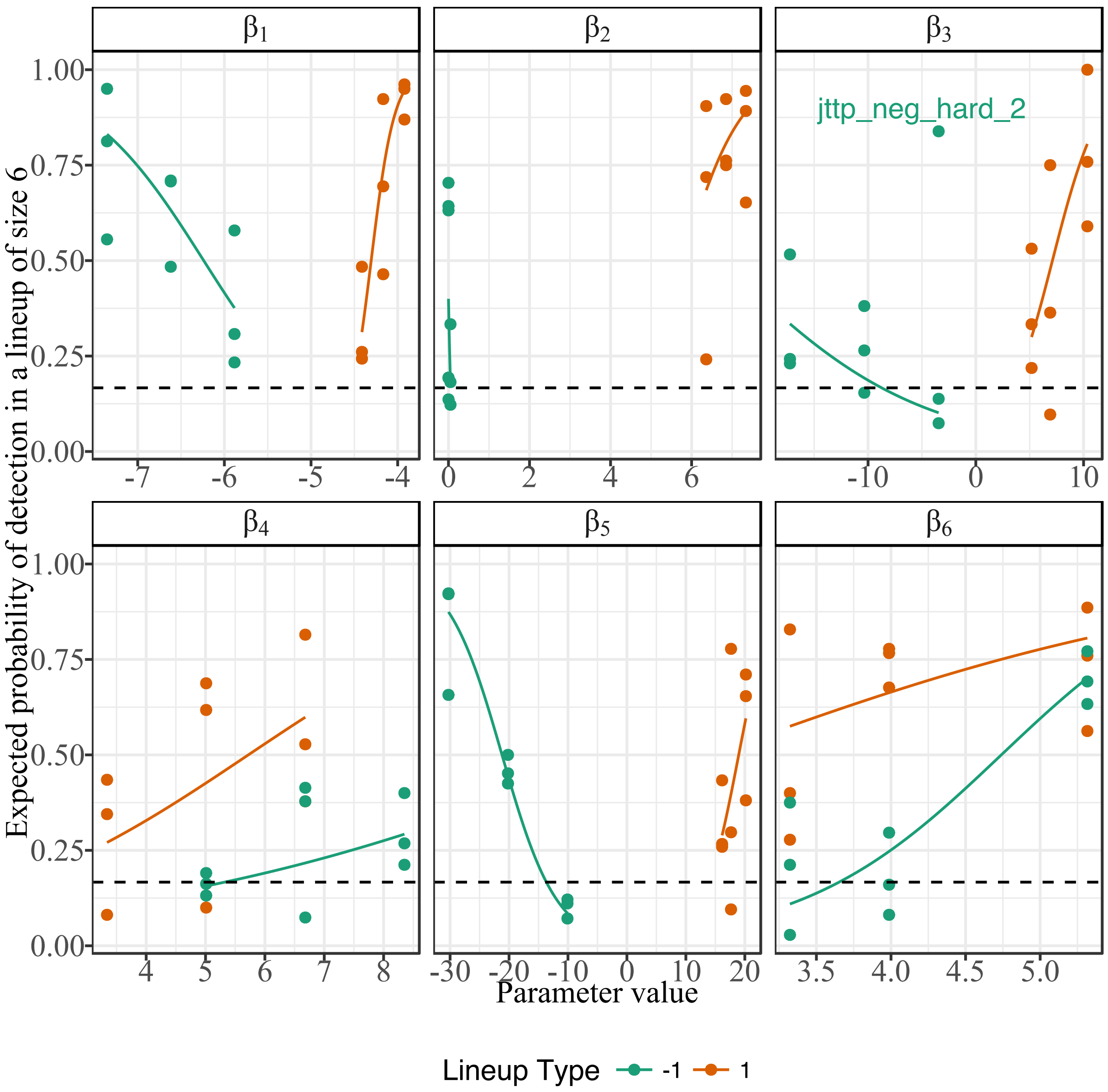


Figure 3.11: Predictions from our generalized linear mixed effects model given in Equation efeq:glmm. The lines show the expected probability of detecting the alternative data in a lineup of size 6 for new observers of new lineups is plotted on the y-axis, and the size of the parameter of interest is on the x-axis. The proportions detected by our Turk participants for each lineup group are shown by the points, with the probability of picking out the data plot at random shown by a horizontal line at 1/6. The lineup marked as "outlier" was removed from modeling. The panel for the reciprocity parameter, β_2 is also presented in Figure 3.12 in more detail.

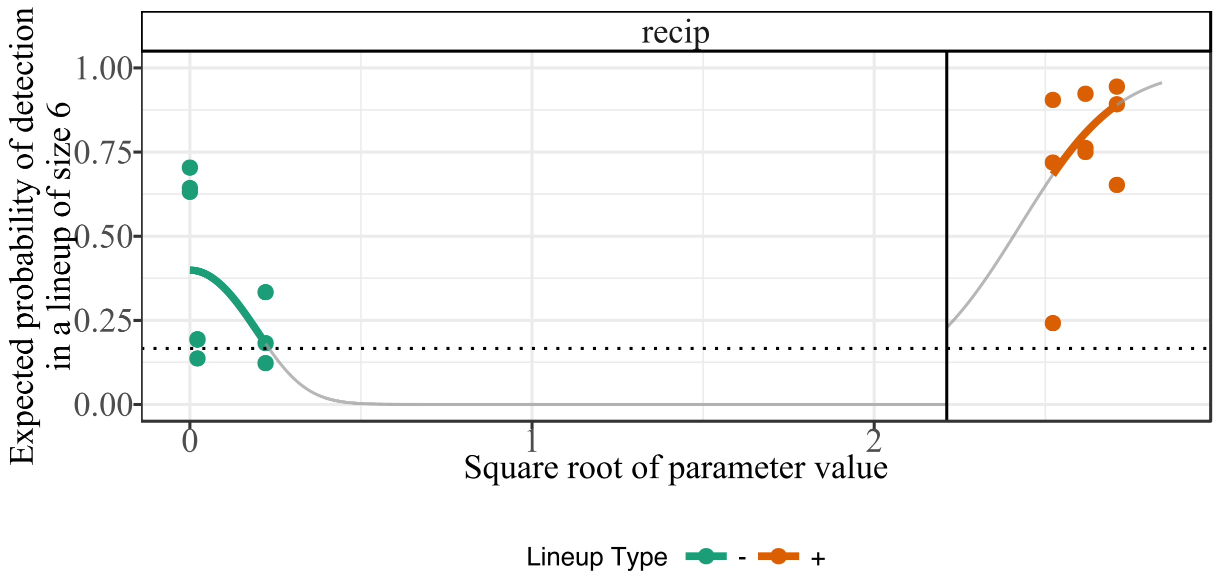


Figure 3.12: The top middle panel of Figure 3.11 expanded to show greater detail. The square root of the parameter value is shown on the x-axis. For this parameter, as its value approaches zero, the probability of identifying the alternate data model decreases, then increases, which is noticeably different from the pattern exhibited by the others. Again, a horizontal line is drawn at 1/6, the chance of selecting the data plot at random.

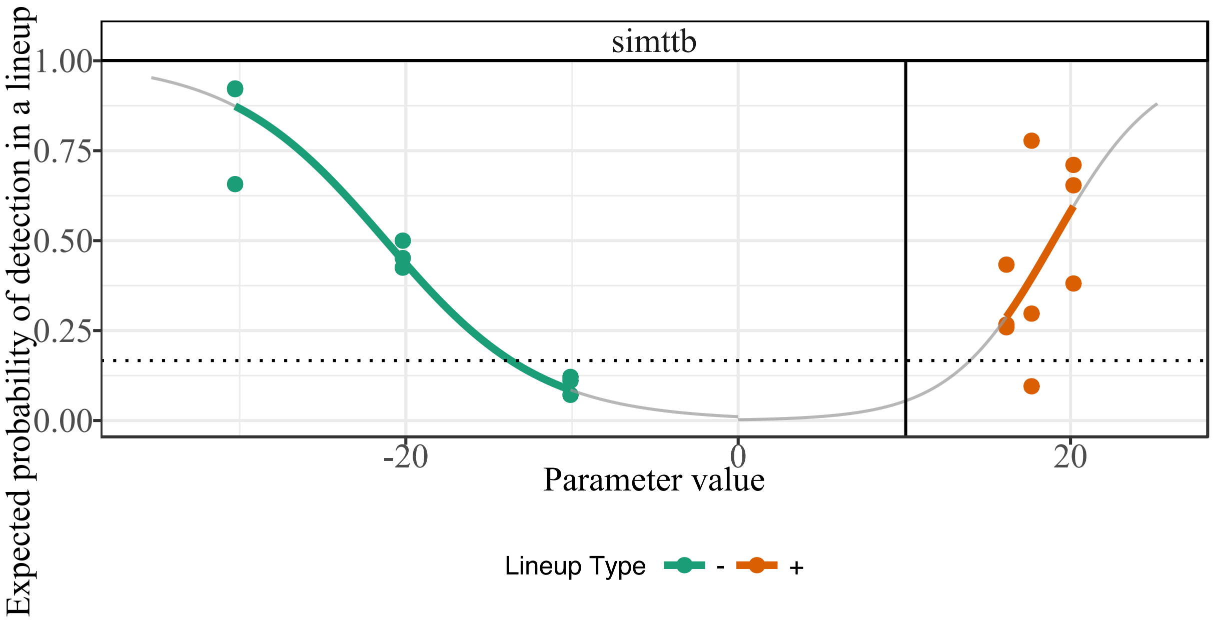


Figure 3.13: The bottom middle panel of Figure 3.11 expanded to show greater detail. The parameter value is shown on the x-axis. This parameter most closely follows our hypothesis shown in Figure 3.4. However, the result is not symmetric. According to the model, people will detect the effect at lower values and with greater frequency as the value increases when it is positive instead of negative.

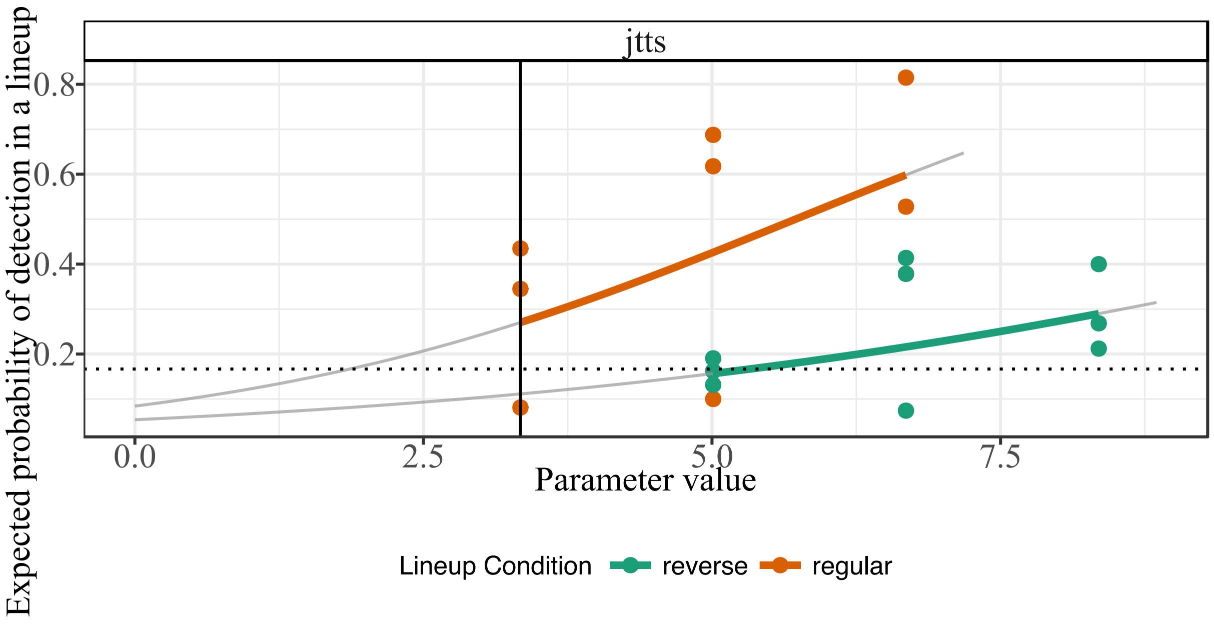


Figure 3.14: The bottom left panel of Figure 3.11 expanded to show greater detail. The parameter value is shown on the x-axis. The "reverse" lineup has a much flatter slope than the "regular" lineup, which means the participants had a harder time detecting a more simple M1 structure among many more complex M4 structures. Reversing the lineup scenario was not symmetric as we hypothesized.

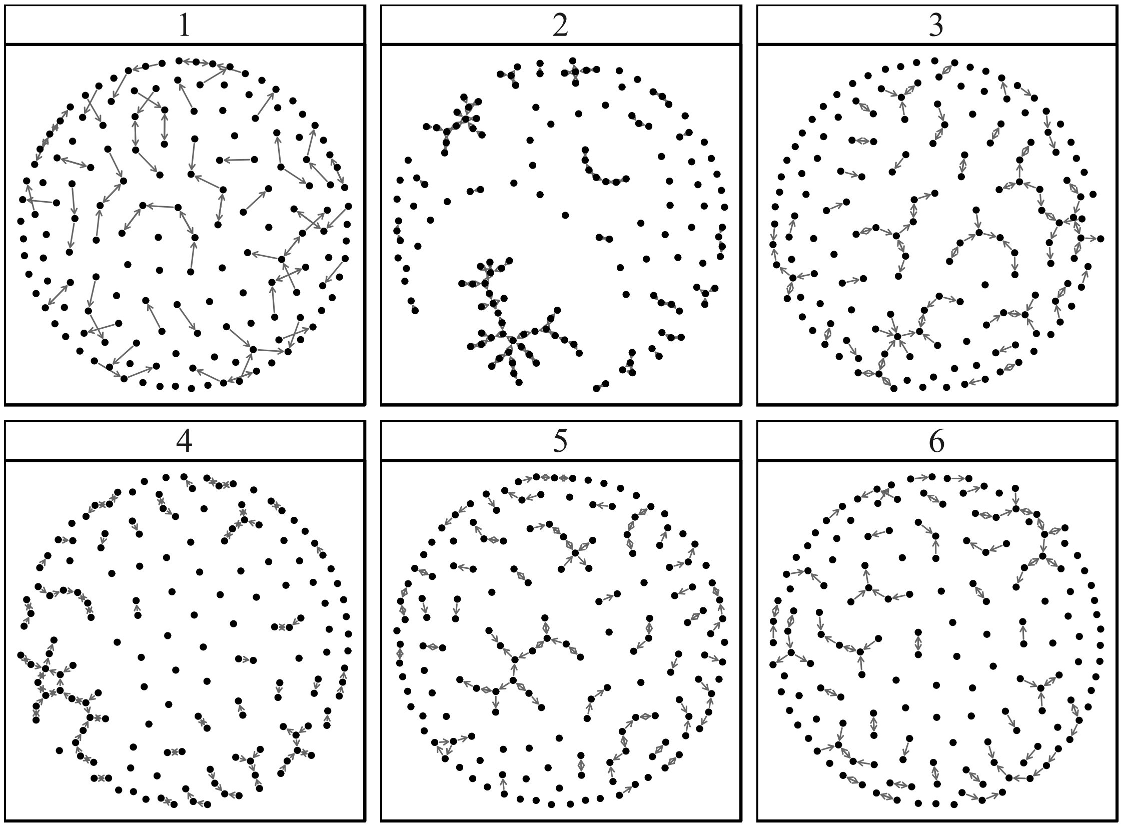


Figure 3.15: In our experiment, 52.8% of viewers of this plot selected the plot from the alternative model, M4. The "reverse" of this lineup is given in Figure 3.16, where 41.4% of viewers selected the plot from the alternative model, M1. Here, the alternative plot is $\sqrt{25}-3$.

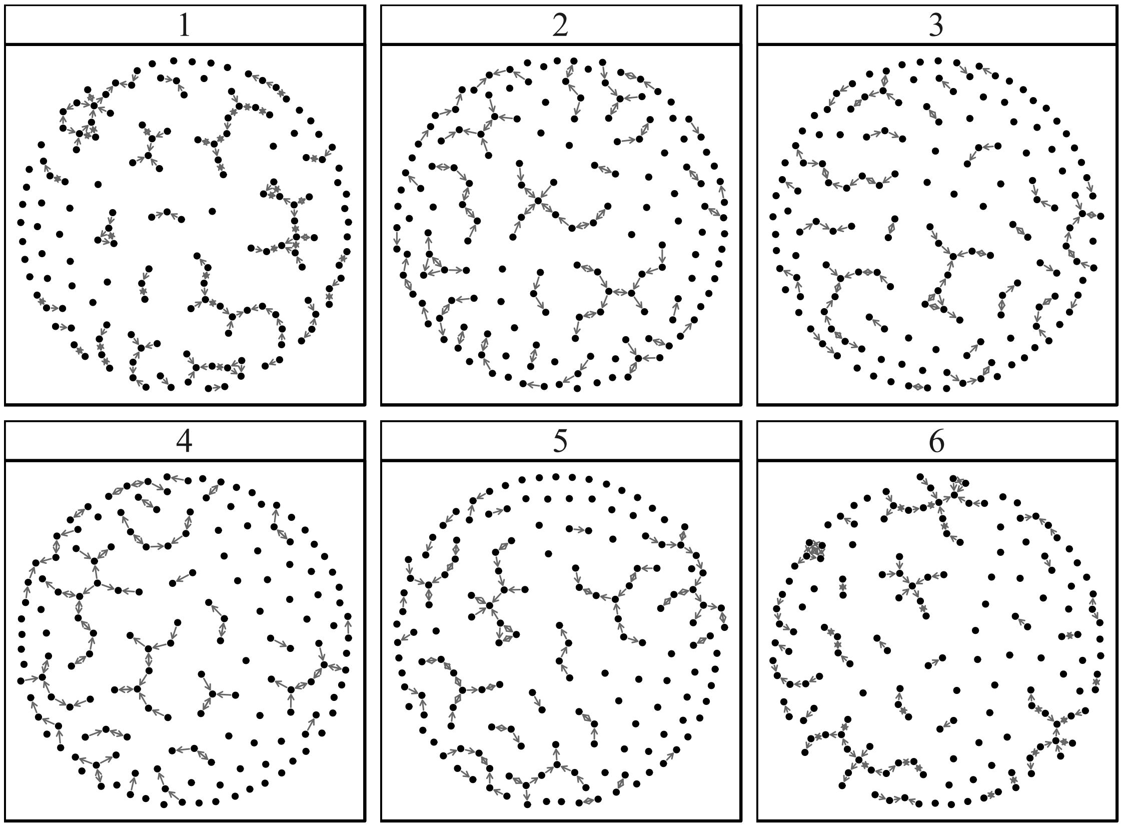


Figure 3.16: In our experiment, 41.4% of viewers of this plot selected the plot from the alternative model, M1. The "reverse" of this lineup is given in Figure 3.15, where 52.8% of viewers selected the plot from the alternative model, M1. Here, the alternative plot is $\sqrt{25} - 1$.

CHAPTER 4. DRAWING NETWORK DATA WITH THE R PACKAGE `ggplot2`

Author's Note: This Chapter is a version of paper I authored with Heike Hofmann (Iowa State University) and François Briatte (European School of Political and Social Sciences) that has been published in *The R Journal*. I developed the package 'geomnet' ("our package") with Dr. Hofmann in 2015, and I have maintained it since. Separately from the implementation of the `geom_net()` function in our package, Briatte implemented graph visualization in two different approaches: the `ggnet2()` function in the `GGally` package and the `ggnetwork` package. This paper gives an overview of the three different approaches, highlights why the package `ggplot2` is generally well suited for graph/network visualization, and discusses the pros and cons of the three methods, with several reproducible examples for each implementation.

There are many kinds of networks, and networks are extensively studied across many disciplines (Watts, 2004). For instance, social network analysis is a longstanding and prominent sub-field of sociology, and the study of biological networks, such as protein-protein interaction networks or metabolic networks, is a notable sub-field of biology (Prell, 2011; Junker and Schreiber, 2008). In addition, the ubiquity of social media platforms, like Facebook, Twitter, and LinkedIn, has brought the concepts of networks out of academia and into the mainstream. Though these disciplines and the many others that study networks are themselves very different and specialized, they can all benefit from good network visualization tools.

Many R packages already exist to manipulate network objects, such as `igraph` by Csardi and Nepusz (2006), `sna` by Butts (2014), and `network` by Butts et al. (2014); Butts (2008). Each one of these packages were developed with a focus of analyzing network data and not necessarily for rendering visualizations of networks. Though these packages do have network visualization capabilities, visualization was not intended as their primary purpose. This is by no means a critique or an inherently negative aspect of these packages: they are all hugely important tools for network analysis that we have relied on heavily in our own work. We have found, however, that visualizing network data in these packages requires a lot of extra work if one is accustomed to working with more common data structures such as vectors, data frames, or arrays. The visualization tools in these packages require detailed knowledge of each one of them and their syntax in order to build meaningful network visualizations with them. This is obviously not a problem if the user is very familiar with network structures and has already spent time working with network data. If, however, the user is new to network data or is more comfortable working with the aforementioned common data structures, they could find the learning curve for these packages burdensome.

The packages described in this paper have, by contrast, have one primary purpose: to create beautiful network visualizations by providing a wrapper of existing network layout capabilities (see for example the `statnet` suite of packages by Handcock et al. (2008)) to the popular `ggplot2` package (Wickham, 2009). And so, our focus here is not on adding to the analysis of network data or to the field of graph drawing (Tamassia, 2013) but rather it is on implementing existing graph drawing capabilities in the `ggplot2` framework, using the common data frame structure. The `ggplot2` package is hugely popular, and many other packages and tools interface with it in order to better visualize a wide variety of data types. By creating a `ggplot2` implementation, we hope to place network visualization within a large, active community of data visualization enthusiasts, bringing new eyes and potentially new innovations to the field of network visualization. With our approaches, we have two primary audiences in mind. The first audience is made up of frequent users of network structures and those who are fluent in the language of packages such as `network` or `igraph`. This audience will find that two of our three approaches (`ggnet2` and `ggnetwork`) directly incorporate the network structures and functions with which they are familiar with into the less familiar visualization paradigm of `ggplot2` (Briatte, 2016). The second audience, targeted by `geomnet`, consists of those users who are not familiar with network structures, but are familiar with data manipulation and tidying, and who happen to find themselves examining some data that can be expressed as a network (Tyner and Hofmann, 2016a). For this audience, we do the heavy network lifting internally, while also relying on their familiarity with `ggplot2` externally.

The `ggplot2` package was designed as an implementation of the 'grammar of graphics' proposed by Wilkinson (1999), and it has become extremely popular among R users.[1]

Because the syntax implemented in the `ggplot2` package is extendable to different kinds of visualizations, many packages have built additional functionality on top of the `ggplot2` framework. Examples include the `ggmap` package by Kahle and Wickham (2013) for spatial visualization, the `ggfortify` package for visualizing statistical models (see Horikoshi and Tang (2015), Tang et al. (2016)), the package `ggally` by Schloerke et al. (2016), which encompasses various complementary visualization techniques to `ggplot2`, and the `ggbio` and `ggtree` Bioconductor packages by Yin et al. (2012) and Yu et al. (tted), which both provide visualizations for biological data. These packages have expanded the utility of `ggplot2`, likely resulting in an increase of its user base. We hope to appeal to this user base and potentially add to it by applying the benefits of the grammar of graphics implemented in `ggplot2` to network visualization.

Our efforts rely upon recent changes to `ggplot2`, which allow users to more easily extend the package through additional geometries or `geoms`.[2]

In the remainder of this paper, we present three different approaches to network visualization through `ggplot2` wrappers. The first is a function, `ggnet2` from the `ggally` package, that acts as a wrapper around a network object to create a `ggplot2` graph. The second is a package, `geomnet`, that combines all network pieces (nodes, edges, and labels) into a single `geom` and is intended to look the most like other `ggplot2` `geoms` in use. The final is another package, `ggnetwork`, that performs some data manipulation and aliases other `geoms` in order to layer the different network aspects one on top of the other. Section 4.1 introduces the basic terminology of networks and illustrates their

[1]In order to give an indication of how large the user base of `ggplot2` is, we looked at its usage statistics from January 1, 2016 to December 31, 2016 (see `http://cran-logs.rstudio.com/`). Over this period, the `ggplot2` package was downloaded over 3.2 million times from CRAN, which amounts to almost 9,000 downloads per day. Almost 800 R packages import or depend on `ggplot2`.

[2]Version 2.1.0, released 1 March 2016. See `https://cran.r-project.org/web/packages/ggplot2/news.html` for the full list of changes in `ggplot2` 2.1.0, as well as the new package vignette, "Extending ggplot2", which explains how the internal `ggproto` system of object-oriented programming can be used to create new geoms.

ubiquity in natural and social life. Section 4.2 then discusses the structure and capabilities of each of the three approaches that we offer. Section 4.3 extends that discussion through several examples ranging from simple to complex networks, for which we provide the code corresponding to each approach alongside its graphical result. We follow with some considerations of runtime behavior in plotting networks in Section 4.4 before closing with a discussion.

4.1 Brief introduction to networks

In its essence, a *network* is simply a set of vertices connected in pairs by a set of edges (Newman, 2010). Throughout this paper, we also use the term *node* to refer to vertices, as well as the terms *ties* or *relationships* to refer to edges, depending on context. The two sets of graphical objects that make up a network visualization, points and segments between them, have been used to examine a huge variety and quantity of information across many different fields of study. For instance, networks of scientific collaboration, a food web of marine animals, and American college football games are all covered in a paper on community detection in networks by Girvan and Newman (2002). Additionally, Buldyrev et al. (2010) study node failure in interdependent networks like power grids. Social networks such as links between television and film actors found on `http://www.imdb.com/` and neural networks, like the completely mapped neural network of the *C. elegans* worm are also extensively studied (Watts and Strogatz, 1998a).

These examples show that networks can vary widely in scope and complexity: the smallest connected network is simply one edge between two vertices, while one of the most commonly used and most complex networks, the world wide web, has billions of vertices (Web pages) and billions of edges (hyperlinks) connecting them. Additionally, the edges in a network can be directed or undirected: *directed* edges represent an ordering of vertices, like a relationship extending from one vertex to another, where switching the direction would change the structure of the network. The World Wide Web is an example of a directed network because hyperlinks connect one Web page to another, but not necessarily the other way around. *Undirected* edges are simply connections between vertices where order does not matter. Co-authorship networks are examples of undirected networks, where nodes are authors and they are connected by an edge if they have written an academic publication together.

As a reference example, we turn to a specific instance of a social network. A *social network* is a network that everyone is a part of in one way or another, whether through friends, family, or other human interactions. We do not necessarily refer here to social media like Facebook or LinkedIn, but rather to the connections we form with other people. To demonstrate the functionality of our tools for plotting networks, we have chosen an example of a social network from the popular television show *Mad Men*. This network, which was compiled by Chang (2013) and made available in `gcookbook` (Chang, 2012), consists of 52 vertices and 87 edges. Each vertex represents a character on the show, and there is an edge between every two characters who have had a romantic relationship.

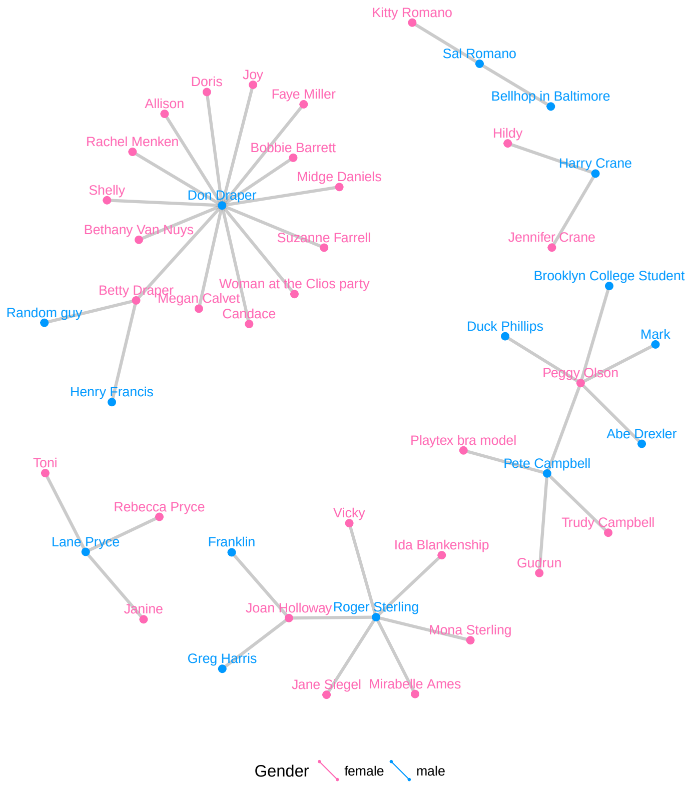


Figure 4.1: Graph of the characters in the show Mad Men who are linked by a romantic relationship.

Figure 4.1 is a visualization of this network. In the plot, we can see one central character who has many more relationships than any other character. This vertex represents the main character of the show, Don Draper, who is quite the "ladies' man." Networks like this one, no matter how simple or complex, are everywhere, and we hope to provide the curious reader with a straightforward way to visualize any network they choose.

Coloring the vertices or edges in a graph is a quick way to visualize grouping and helps with pattern or cluster detection. The vertices in a network and the edges between them compose the structure of a network, and being able to visually discover patterns among them is a key part of network analysis. Viewing multiple layouts of the same network can also help reveal patterns or clusters that would not be discovered when only viewing one layout or analyzing only its underlying adjacency matrix.

4.2 Three implementations of network visualizations

We present two basic approaches to using the `ggplot2` framework for network visualization. First, we implement network visualizations by providing a wrapper function, `ggnet2` for the user to visualize a network using `ggplot2` elements (Schloerke et al., 2016). Second, we implement network visualizations using layering in `ggplot2`. For the second approach, we have two ways of creating a network visualization. The first, `geomnet`, wraps all network structures, including vertices, edges, and vertex labels into a single `geom`. The second, `ggnetwork`, implements each of these structural components in an independent `geom` and layers them to create the visualization (Briatte, 2016). In each package, our goal is to provide users with a way to map network properties to aesthetic properties of graphs that is familiar to them and straightforward to implement. Each package has a slightly different approach to accomplish this goal, and we will discuss all of these approaches in this section. For each implementation, we also provide the code necessary to create Figure 4.1, and describe the arguments used. We conclude the section with a side-by-side comparison of the features available in all three implementations in Table 4.1.

4.2.1 ggnet2

The `ggnet2` function is a part of the `ggally` package, a suite of functions developed to extend the plotting capabilities of `ggplot2` (Schloerke et al., 2016). A detailed description of the `ggnet2` function is available from within the package as a vignette. Some example code to recreate Figure 4.1 using `ggnet2` is presented in Figure 4.2.

```
library(GGally)
library(network)
# make the data available
data(madmen, package = 'geomnet')
# data step for both ggnet2 and ggnetwork
# create undirected network
mm.net <- network(madmen$edges[, 1:2], directed = FALSE)
mm.net # glance at network object
# create node attribute (gender)
rownames(madmen$vertices) <- madmen$vertices$label
mm.net %v% "gender" <- as.character(
  madmen$vertices[ network.vertex.names(mm.net), "Gender"]
)
# gender color palette
mm.col <- c("female" = "#ff69b4", "male" = "#0099ff")
# create plot for ggnet2
set.seed(10052016)
ggnet2(mm.net, color = mm.col[ mm.net %v% "gender" ],
       labelon = TRUE, label.color = mm.col[ mm.net %v% "gender" ],
       size = 2, vjust = -0.6, mode = "kamadakawai", label.size = 3)
```

Figure 4.2: The code required to generate Figure 4.1 using the `ggnet2` function in the `ggally` package.

The `ggnet2` function offers a large range of network visualization functionality in a single function call. Although its result is a `ggplot2` object that can be further styled with `ggplot2` scales and themes, the syntax of the `ggnet2` function is designed to be easily understood by the users, who may not be familiar with `ggplot2` objects. The aesthetics relating to the nodes are controlled by arguments such as `node.alpha` or `node.color`, while those relating to the edges are controlled by arguments starting with `edge`. Additionally, as seen in Figure 4.2, the usual `ggplot2` arguments like `color` can be used without the prefix to map node attributes to aesthetic values. The arguments with the `node.` prefix are aliased versions for readability of the code. Thus, while `ggnet2` applies the grammar of graphics to network objects, the function itself still works very much like the plotting functions of the `igraph` and `network` packages: a long series of arguments is used to control every possible aspect of how the network should be visualized.

The `ggnet2` function takes a single network object as input. This initial object might be an object of class `"network"` from the `network` package (with the exception of hypergraphs or multiplex graphs), or any data structure that can be coerced to an object of that class via functions in the `network` package, such as an incidence matrix, an adjacency matrix, or an edge list. Additionally, if the `intergraph` package (Bojanowski, 2015) is installed, the function also accepts a network object of class `"igraph"`. Internally, the function converts the network object to two data frames: one for edges and another one for nodes. It then passes them to `ggplot2`. Each of the two data frames contain the information required by `ggplot2` to plot segments and points respectively, such as a shape for the points (nodes) and a line type for the segments (edges). The final result returned to the user is a plot with a minimum of two layers, or more if there are edge and/or node labels.

The `mode` argument of `ggnet2` controls how the nodes of the network are to be positioned in the plot returned by the function. This argument can take any of the layout values supported by the `gplot.layout` function of the `sna` package, and defaults to `fruchtermanreingold`, which places the nodes through the force-directed layout algorithm by Fruchterman and Reingold (1991). In the example presented in Figure 4.2, the Kamada-Kawai layout is used by adding `mode = "kamadakawai"` to the function call. Many other possible layouts and their parameters can also be passed to `ggnet2` through the `layout.par` argument. For a list of possible layouts and their arguments, see `?sna::gplot.layout`.

Other arguments passed to the `ggnet2` function offer extensive control over the aesthetics of the plot that it returns, including the addition of edge and/or node labels and their respective aesthetics. Arguments such as `node.shape` or `edge.lty`, which control the shape of the nodes and the line type of the edges, respectively, can take a single global value, a vector of global values, or the name of an edge or vertex attribute to be used as an aesthetic mapping. This feature is used to change the size of the nodes and the node labels in Figure 4.2 by including `size = 2` and `label.size = 3` in the function call.

This last functionality builds on one of the strengths of the `"network"` class, which can store information on network edges and nodes as attributes that are then accessible to the user through the `%e%` and `%v%` operators respectively.[3] Usage examples of these operators can be seen in Figure 4.2. The attribute of gender is assigned to nodes, which in turn is accessed to color the nodes and node labels by gender. If the `ggnet2` function is given the `node.alpha = "importance"` argument, it will interpret it as an attempt to map the vertex attribute called `importance` to the transparency level of the nodes. This works exactly like the command `net %v% "importance"`, which returns the vertex attribute `importance` of the `"network"` object `net`. This functionality allows the `ggnet2` function to work in a similar fashion to `ggplot2` mappings of aesthetics within the `aes` operator.

The `ggnet2` function also provides a few network-specific options, such as sizing the nodes as a function of their unweighted degree, or using the primary and secondary modes of a bipartite network as an aesthetic mapping for the nodes.

All in all, the `ggnet2` function combines two different kinds of processes: it translates a network object into a data frame suitable for plotting with `ggplot2`, and it applies network-related aesthetic operations to that data frame, such as coloring the edges in function of the color of the nodes that they connect.

4.2.2 geomnet

4.2.2.1 Data structure

The package `geomnet` implements network visualization in a single `ggplot2` layer. A stable version is available on CRAN, with a development version available at https://github.com/sctyner/geomnet. The package has two main functions: `stat_net`, which performs all of the calculations, and `geom_net`, which renders the plot. It also contains the secondary functions `geom_circle` and

[3] See p. 22-24 of Butts et al. (2014). The equivalent operators in the `igraph` package are called `E` and `V`.

```
# also loads ggplot2
library(geomnet)

# data step: join the edge and node data with a fortify call
MMnet <- fortify(as.edgedf(madmen$edges), madmen$vertices)
# create plot
set.seed(10052016)
ggplot(data = MMnet, aes(from_id = from_id, to_id = to_id)) +
  geom_net(aes(colour = Gender), layout.alg = "kamadakawai",
           size = 2, labelon = TRUE, vjust = -0.6, ecolour = "grey60",
           directed =FALSE, fontsize = 3, ealpha = 0.5) +
  scale_colour_manual(values = c("#FF69B4", "#0099ff")) +
  xlim(c(-0.05, 1.05)) +
  theme_net() +
  theme(legend.position = "bottom")
```

Figure 4.3: The code required to generate Figure 4.1 using the `geom_net` function in the `geomnet` package.

`theme_net`, which assist, respectively, in drawing self-referencing edges and removing axes and other background elements from the plots. The approach in `geomnet` is similar to the implementation of other, native `ggplot2` geoms, such as `geom_smooth`. When using `geom_smooth`, the user does not need to know about any of the internals of the loess function, and similarly, when using `geomnet`, the user is not expected to know about the internals of the layout algorithm, just the name of the algorithm they'd like to use. On the other hand, if users are comfortable with network analysis, the entire body of layout methods provided by the `sna` package is available to them through the parameters `layout.alg` and `layout.par`.

In network analysis there are usually two sources of information: one data set consisting of a description of the nodes, represented as the vertices in the network and vertex attributes, and another data set detailing the relationship between these nodes, i.e. it consists of the edge list and any additional edge attributes. The minimum amount of information needed is a vector of all vertex labels and a two column data frame that encodes the edge list of the network. In order for this geometry to work, these two data sets need to be combined into a single data frame. For this, we implemented several new `fortify` methods for producing the correct data structure from different S3 objects that encode network information. Supported classes are `"network"` from the `sna` and `network` packages, `"igraph"` from the `igraph` package, `"adjmat"`, and `"edgedf"`. The last two are new classes introduced in `geomnet` that are identical to the `"matrix"` and `"data.frame"` classes, respectively. We created these new classes and the functions `as.adjmat()` and `as.edgedf()` so that network data in adjacency matrix and edgelist (data frame) formats can have their own fortify functions, separate from the very generic `"matrix"` and `"data.frame"` classes. These fortify functions combine the edge and the node information using a full join. A full join is used because generally, there will be some vertices that are sinks in the network because they only show up in the 'to' column, and so we accommodate for these by adding artificial edges in the data set that have missing information for the 'to' column. The user may also pass two data frames to the function, e.g. `data = edge_data` and `vertices = vertex_data`, but we recommend using the `fortify` methods whenever possible.

A usage example of the `fortify.edgedf` method is presented in Figure 4.3 with the creation of the `MMnet` data set. Two dataframes, `madmen$edges` and `madmen$vertices` are joined to create the required data. The first few rows of these data sets and their merged result are below.

```
head(as.edgedf(madmen$edges), 3)
##        from_id         to_id
## 1 Betty Draper Henry Francis
## 2 Betty Draper    Random guy
## 3   Don Draper       Allison
head(madmen$vertices, 3)
##          label Gender
## 1 Betty Draper female
## 2   Don Draper   male
## 3  Harry Crane   male
head(fortify(as.edgedf(madmen$edges), madmen$vertices), 3)
```

```
##        from_id         to_id Gender
## 1 Betty Draper Henry Francis female
## 2 Betty Draper    Random guy female
## 3   Don Draper       Allison   male
```

The formal requirements of `stat_net` are two columns, called `from_id` and `to_id`. During this routine, columns `x, y` and `xend, yend` are calculated and used as a required input for `geom_net`. Other variables may also be included for each edge, such as the edge weight, in-degree, out-degree or grouping variable.

4.2.2.2 Parameters and aesthetics

Parameters that are currently implemented in `geom_net` are:

- **layout:** the `layout.alg` parameter takes a character value corresponding to the possible network layouts in the `sna` package that are available within the `gplot.layout.*()` family of functions. The default layout algorithm used is the Kamada-Kawai layout, a force-directed layout for undirected networks (Kamada and Kawai, 1989).

  In `sna`, for each layout there is a corresponding set of possible layout parameters, `layout.par`, which can be passed as a list to `geom_net`. If the user wishes to create small multiples using `ggplot2 facets`, they can use `fiteach`, a logical value specifying whether the same layout should be used for all panels (default) or each panel's data should be fit separately. Finally, the `singletons` parameter is a logical value that dictates whether or not to include nodes with zero indegree and zero outdegree in the visualization. The default is set to `TRUE`, and if set to `FALSE` nodes will only appear in panels where they have indegree or outdegree of at least one.

- **vertices:** any of `ggplot2`'s aesthetics relating to points: `colour`, `size`, `shape`, `alpha`, `x`, and `y` are available and used for specifying the appearance of nodes in the network. For example `aes(colour = Gender)` is used in Figure 4.3 to color the nodes and node labels according to the gender of each character.

- **edges:** for edges we distinguish between two different sets of aesthetics: aesthetics that only relate to line attributes, such as `linewidth` and `linetype`, and aesthetics that are also used by the point `geom`. The former can be used in the same way as they are used in `geom_segment`, while the latter, like `alpha` or `colour`, for instance, are used for vertices unless separately specified. Instead, use the parameters `ecolour` or `ealpha`, which are only applied to the edges. If the `group` variable is specified, a new variable, called `samegroup` is added during the layout process. This variable is `TRUE`, if an edge is between two vertices of the same group, and `FALSE` otherwise. If `samegroup` is `TRUE`, the corresponding edge will be colored using the same color as the vertices it connects. If the edge is between vertices of a different group, the default grey shade is used for the edge.

  The parameter `curvature` is set to zero by default, but if specified, leads to curved edges using the newly implemented `ggplot2` geom `geom_curve` instead of the regular `geom_segment`. Note that the edge specific aesthetics that overwrite node aesthetics are currently considered as 'as.is' values: they do not get a legend and are not scaled within the ggplot2 framework. This is done to avoid any clashes between node and edge scales.

  self-referencing vertices: some networks contain self references, i.e. an edge has the same vertex id in its from and to columns. If the parameter `selfloops` is set to `TRUE`, a circle is drawn using the new `geom_circle` next to the vertex to represent this self reference.

- **arrow:** whenever the parameter `directed` is set from its default state to `TRUE`, arrows are drawn from the 'from' to the 'to' node, with tips pointing towards the 'to' node. By default, arrows have an absolute size of 10 points. The entire structure of the arrow can be changed by passing an `arrow` object from the `grid` package to the `arrow` argument. If the user doesn't wish to change the whole arrow object, the parameters `arrowsize` and `arrowgap` are also available. The `arrowsize` argument is of a positive numeric value that is used as a multiple of the original arrow size, i.e. `arrowsize = 2` shows arrow tips at twice their original size. The parameter `arrowgap` can be used to avoid overplotting of the arrow tips by the nodes, `arrowgap` specifies a proportion by which the edge should be shrunk with default of 0.05. A value of 0.5 will result in edges drawn only half way from the 'from' node to the 'to' node.

- **labels:** the `labelon` argument is a logical parameter, which when set to `TRUE` will label the nodes with their IDs, as is in Figure 4.1. The `aes` option `label` can also be used to label nodes, in which case the nodes are labeled with the value corresponding to their respective values of the provided variable. If `colour` is specified for the nodes, the same values are used for the labels, unless `labelcolour` is specified. If `fontsize` is specified, it changes the label size to that value in points. Other parameter values, such as `vjust` and `hjust` help in adjusting labels relative to the nodes. The parameters work in the same fashion as in native `ggplot2` `geoms`. Additionally, the label can be drawn by using `geom_text` (the default) or using the new `geom_label` in `ggplot2` by adding `labelgeom = "label"` to the arguments in `geom_net`. Finally, with the help of the package `ggrepel` by Slowikowski (2017) we have implemented the logical `repel` argument, which when true, uses `geom_text_repel` or `geom_label_repel` to plot the labels instead of `geom_text` or `geom_label`, respectively. Using `repel` can be extremely useful when the networks are dense or the labels are long, as in Figure 4.1, helping to solve a common problem with many network visualizations.

4.2.3 ggnetwork

`ggnetwork` is a small R package that mimics the behavior of `geomnet` by defining several geoms to achieve similar results.

The approach taken by the `ggnetwork` package is to alias some of the native geoms of the `ggplot2` package. An aliased geom is simply a variant of an already existing one. The `ggplot2` package contains several examples of aliased geoms, such as `geom_histogram`, which is a variant of `geom_bar`. (See Table 4.6 of Wickham (2009) for an example).

Following that logic, the `ggnetwork` package adds four aliased geometries to `ggplot2`:

```
# create plot for ggnetwork. uses same data created for ggnet2 function
library(ggnetwork)
set.seed(10052016)
ggplot(data = ggnetwork(mm.net, layout = "kamadakawai"),
       aes(x, y, xend = xend, yend = yend)) +
  geom_edges(color = "grey50") + # draw edge layer
  geom_nodes(aes(colour = gender), size = 2) + # draw node layer
  geom_nodetext(aes(colour = gender, label = vertex.names),
                size = 3, vjust = -0.6) + # draw node label layer
  scale_colour_manual(values = mm.col) +
  xlim(c(-0.05, 1.05)) +
  theme_blank() +
  theme(legend.position = "bottom")
```

Figure 4.4: The code required to generate Figure 4.1 using the `ggnetwork` package.

- `geom_nodes`, an alias to `geom_point`;
- `geom_edges`, an alias to either `geom_segment` or `geom_curve`;
- `geom_nodetext`, an alias to `geom_text`; and
- `geom_edgetext`, an alias to `geom_label`.

The four geoms are used to plot nodes, edges, node labels and edge labels, respectively. Two of the geoms that they alias, `geom_curve` and `geom_label`, are part of the new geometries introduced in `ggplot2` version 2.1.0. All four geoms behave exactly like those that they alias, and take exactly the same arguments. The only exception to that rule is the special case of `geom_edges`, which accepts both the arguments of `geom_segment` and those of `geom_curve`; if its `curvature` argument is set to anything but `0` (the default), then `geom_edges` behaves exactly like `geom_curve`; otherwise, it behaves exactly like `geom_segment`. Three of the four availble aliased `geoms` are used in Figure 4.4 to create the visualization of the Mad Men relationship network.

Just like the `ggnet2` function, the `ggnetwork` package takes a single network object as input. This can be an object of class `"network"`, some data structure coercible to that class, or an object of class `"igraph"` when the `intergraph` package is installed. This object is passed to the 'workhorse' function of the package, which is also called `ggnetwork` to create a data frame, and then to the `data` argument of `ggplot()`.

Internally, the `ggnetwork` function starts by computing the `x` and `y` coordinates of all nodes in the network with respect to its `layout` argument, which defaults to the Fruchterman-Reingold layout algorithm (Fruchterman and Reingold, 1991). It then extracts the edge list of the network, to which it adds the coordinates of the sender and receiver nodes as well as all edge-level attributes. The result is a data frame with as many rows as there are edges in the network, and where the `x`, `y`, `xend` and `yend` hold the coordinates of the network edges.

At that stage, the `ggnetwork` function, like the `geomnet` package, performs a left-join of that augmented edge list with the vertex-level attributes of the 'from' nodes. It also adds one self-loop per node, in order to ensure that every node is plotted even when their degree is zero—that is, even if the node is not connected to any other node of the network, and is therefore absent from the edge list. The data frame created by this process contains one row per edge as well as one additional row per node, and features all edge-level and vertex-level attributes initially present in the network.[4]

The `ggnetwork` function also accepts the arguments `arrow.gap` and `by`. Like in `geomnet`, `arrow.gap` slightly shortens the edges of directed networks in order to avoid overplotting edge arrows and nodes. The argument `by` is intended for use with plot facets. Passing an edge attribute as a grouping variable to the `by` argument will cause `ggnetwork` to return a data frame in which each node appears as many times as there are unique values of that edge attribute, using the same coordinates for all occurrences. When that same edge attribute is also passed to either `facet_wrap` or `facet_grid`, each edge of the network will show in only one panel of the plot, and all nodes will appear in each of the panels at the same position. This makes the panels of the plot comparable to

[4]One limitation of this process is that it requires some reserved variable names (`x`, `y`, `xend` and `yend`), which should not also be present as edge-level or vertex-level attributes (otherwise the function will simply break). Similarly, if an edge attribute and a vertex attribute have the same name, like `na`, which the `network` package defines as an attribute for both edges and vertices in order to flag missing data, `ggnetwork` will rename them to `na.x` (for the edge-level attribute) and `na.y` (for the vertex-level attribute).

each other, and allows the user to visualize the network structure as a function of a specific edge attribute, like a temporal attribute.

4.3 Examples

In this section, we demonstrate some of the current capabilities of `ggnet2`, `geomnet`, and `ggnetwork` in a series of side by side examples. While the output is nearly identical for each method of network visualization, the code and implementations differ across the three methods. For each of these examples, we present the code necessary to produce the network visualization in each of the three packages, and discuss each application in detail.

For the following examples we will be loading all three packages under comparison. In practice, only one of these packages would be needed to visualize a network in the `ggplot2` framework:

```
library(ggplot2)
library(GGally)
library(geomnet)
library(ggnetwork)
```

4.3.1 Blood donation

We begin with a very simple example that most should be familiar with: blood donation. In this directed network, there are eight vertices and 27 edges. The vertices represent the eight different blood types in humans that are most important for donation: the ABO blood types A, B, AB, and O, combined with the RhD positive (+) and negative (-) types. The edges are directed: a person whose blood type is that of a *from* vertex can to donate blood to a person whose blood type is that of a corresponding *to* vertex. This network is shown in Figure 4.9. The code to produce each one of the networks is shown above Figure 4.9. We take advantage of each approach's ability to assign identity values to the aesthetic values. The color is changed to a dark red, the size of the nodes is changed to be large enough to accomodate the blood type label, which we also change the color of, and we use the directed and arrow arguments of each implementation to show the precise blood

Table 4.1: Comparing the three different package side-by-side.

Functionality	`ggnet2` (`ggally`)	`geom_net` (`geomnet`)	`geom_nodes`, `geom_edges`, etc (`ggnetwork`)
Data	object of class `"network"` or object easily converted to that class (i.e. incidence or adjacency matrices, edge list) or object of class `"igraph"`	a fortified `"network"`, `"igraph"`, `"edgedf"`, or `"adjmat"` object OR one edge data frame and one node data frame to be merged internally	same as `ggnet2`
Naming conventions	`node._`, `edge._`, `label._`, `edge.label._` for `alpha`, `color`, etc.	arguments identical to `ggplot2` with exception of `ecolor`, `ealpha`	same as `ggplot2`
Layout package & default	`sna`, Fruchterman-Reingold	`sna`, Kamada-Kawai	`sna`, Fruchterman-Reingold
Aesthetic mappings to variables	all `alpha`, `color`, `shape`, `size` for nodes, edges, labels	`colour`, `size`, `shape`, `x`, `y`, `linetype`, `linewidth`, `label`, `group`, `fontsize`	same as `ggplot2`
Arrows	`directed = TRUE`, `arrow.size`, `gap`	`arrowsize`, `gap`, `arrow = arrow()` like `ggplot2`	specify arrows in `geom_edge` like in codegeom_segment, `arrow.gap`
Theme or palette changes	done in the function with arguments like `_.legend`, `_.palette`, etc. and adding `ggplot2` elements	adding `ggplot2` elements	adding `ggplot2` elements
Creating small multiples	created separately, use `grid.arrange` from `gridExtra`	add `group` argument to `fortify()` and use `facet_*()` from `ggplot2`	use `by` argument in `ggnetwork()` and `facet_*()` from `ggplot2`
Edge labelling?	Yes	No	Yes
Draw self-loops?	No	Yes	No

donation relationships. Additionally, we change the node layout to circle, and the placement of the labels with the `hjust` and `vjust` options.

```
# make data accessible
data(blood, package = "geomnet")


# plot with ggnet2 (Figure 5a)
set.seed(12252016)
ggnet2(network(blood$edges[, 1:2], directed=TRUE),
       mode = "circle", size = 15, label = TRUE,
       arrow.size = 10, arrow.gap = 0.05, vjust = 0.5,
       node.color = "darkred", label.color = "grey80")

head(blood$edges,3) # glance at the data
##   from  to group_to
## 1  AB- AB+     same
## 2  AB- AB-     same
## 3  AB+ AB+     same
# plot with geomnet (Figure 5b)
set.seed(12252016)
ggplot(data = blood$edges, aes(from_id = from, to_id = to)) +
  geom_net(colour = "darkred", layout.alg = "circle", labelon = TRUE, size = 15,
           directed = TRUE, vjust = 0.5, labelcolour = "grey80",
           arrowsize = 1.5, linewidth = 0.5, arrowgap = 0.05,
           selfloops = TRUE, ecolour = "grey40") +
  theme_net()

# plot with ggnetwork (Figure 5c)
set.seed(12252016)
```

```
ggplot(ggnetwork(network(blood$edges[, 1:2]),
                 layout = "circle", arrow.gap = 0.05),
       aes(x, y, xend = xend, yend = yend)) +
  geom_edges(color = "grey50",
             arrow = arrow(length = unit(10, "pt"), type = "closed")) +
  geom_nodes(size = 15, color = "darkred") +
  geom_nodetext(aes(label = vertex.names), color = "grey80") +
  theme_blank()
```

In this example every vertex has a self-reference, as blood between two people of matching ABO and RhD type can always be exchanged. The **geomnet** approach shows these self-references as circles looping back to the vertex, which is controlled by using the parameter setting `selfloops = TRUE`.

Figure 4.6 ggnet2

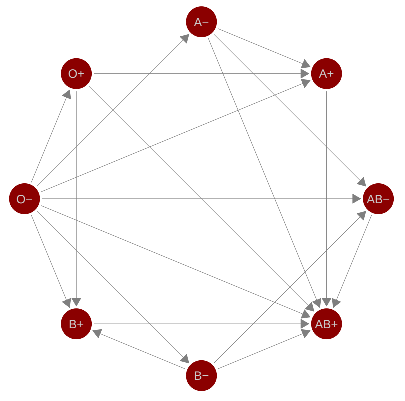


Figure 4.7 geomnet

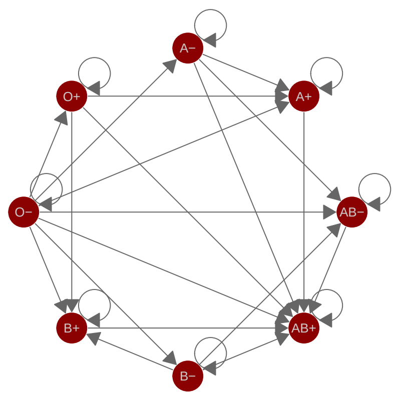


Figure 4.8 ggnetwork

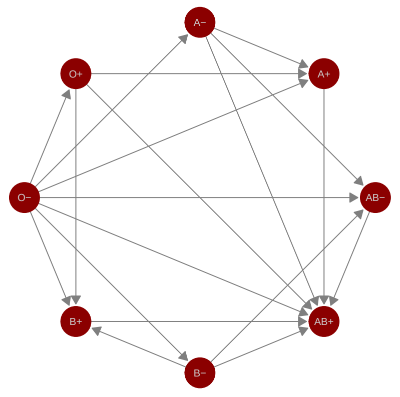


Figure 4.9: Network of blood donation possibilities in humans by ABO and RhD blood types.

`colour` and `size` aesthetics in Figure 4.9 are set to identity values to change the size and color of all vertices. We have also used the `layout` and `label` arguments to change the default Kamada-Kawai layout to a circle layout and to print labels for each of the blood types. The circle layout places blood types of the same ABO type next to each other and spreads the vertices out far enough to distinguish between the various “in” and “out” types. We can tell clearly from this plot that the O-type is the universal donor: it has an out-degree of seven and an in-degree of zero. Additionally, we can see that the AB+ type is the universal recipient, with an in-degree of seven and an out-degree of zero. Anyone looking at this plot can quickly determine which type(s) of blood they can receive and which type(s) can receive their blood.

4.3.2 Email network

The email network comes from the 2014 VAST Challenge (Cook et al., 2014). It is a directed network of emails between company employees with 55 vertices and 9,063 edges. Each vertex represents an employee of the company, and each edge represents an email sent from one employee to another. The arrow of the directed edge points to the recipient of the email. If an email has multiple recipients, multiple edges, one for each recipient, are included in the network. The network contains two business weeks of emails across the entire company. In order to better visualize the structure of the communication network between employees, emails that were sent out to all employees are removed. A glimpse of the data objects used is below.

```
em.net # ggnet2 and ggnetwork
##  Network attributes:
##   vertices = 55
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 4743
##     missing edges= 0
##     non-missing edges= 4743
##
##  Vertex attribute names:
```

```
##     curr_empl_type vertex.names
##
##  Edge attribute names not shown
emailnet[1,c(1:2,7,21)] # geomnet
##                                 from_id
## 1 Ada.Campo-Corrente@gastech.com.kronos
##                                 to_id day
## 1 Ingrid.Barranco@gastech.com.kronos  10
##   CurrentEmploymentType
## 1             Executive
```

Emails taken by themselves form an event network, i.e. edges do not have any temporal duration. Here, however, we can think of emails as observable expressions of the underlying, unobservable, relationship between employees. We can think of this network as a dynamic temporal network, i.e. this network has the potential to change over time. The `ndtv` package by Bender-deMoll (2016) allows the analysis of such networks and provides impressive animations of the underlying dynamics. Here, we are using two static approaches to visualize the network: first, we aggregate emails across the whole time frame (shown in Figure 4.13), then we aggregate emails by day and use small multiples to allow a comparison of day-to-day behavior (shown in Figure 4.19).

For all of the email examples, we have colored the vertices by the variable `CurrentEmploymentType`, which contains the department in the company of which each employee is a part of. There are six distinct clusters in this network which almost perfectly correspond to the six different types of employees in this company: administration, engineering, executive, facilities, information technology, and security. Other features in the code include using `alpha` arguments to change the transparency of the edges, `curvature` argumnets to show mutual communication as two edges instead of one edge with two arrowheads, and the addition of `ggplot2` functions like `scale_colour_brewer` and `theme` to customize the colors of the nodes and their corresponding legend.

In Figure 4.13 we can clearly see the varying densities of communications within departments and the more sparse communication between employees in different departments. We also see that one of the executives only communicates with employees in Facilities, while one of the IT employees frequently communicates with security employees.

A comparison of the results of `ggnet`, `geomnet` and `ggnetwork` reveals some of the more subtle differences between the implementations:

- In the `ggnet2` implementation, the opacity of the edges between employees in the same cluster is higher than it is for the edges between employees in different clusters. This is due to the fact that the email network does not make use of edge weights: instead, every email between two employees is represented by an edge, resulting in edge overplotting. The `edge.alpha` argument has been set to a value smaller than one, therefore multiple emails between two employees create more opaque edges between them. Multiple emails are also taken into account in the `geomnet` package. When there is more than one edge connecting two vertices, the `stat_net` function adds a weight variable to the edge list, which is passed automatically to the layout algorithms and taken into account during layout. This is thanks to the `sna` package, which supports the use of weights in its edge list. In addition to taking weights into account in the layout, we can also make use of them in the visualization. `geomnet` allows to access all of the internal variables created in the visualization process, such as coordinates `..x..`, `..y..` and edge weights `..weight..`. Note the use of the `ggplot2` notation `..` for internal variables.

- In the first two layouts of Figure 4.13, edges between employees who share the same employment type are given the color of that employment type, while edges between employees belonging to different types are plotted in grey. This feature is particularly useful to visualize the amount of within-group connectedness in a network. By contrast, in the last layout, edges are colored according to the sender's employment type, because the `ggnetwork` package does not support coloring edges as a function of node-level attributes.

- Finally, in the last two layouts of Figure 4.13, the `curvature` argument has been set to 0.05, resulting in slightly curved edges in both plots. This feature, which takes advantage of the `geom_curve` geometry released in `ggplot2` 2.1.0, makes it possible to visualize which edges correspond to reciprocal connections; in an email communication network, as one might expect, most edges fall into that category.

To give some insight into how the relations between employees change over time, we facet the network by day: each panel in Figure 4.19 shows email networks associated with each day of the work week. The code for these visualizations is below. The different approaches create small multiples in different ways. The `ggnet2` approach requires that the network be separated, each plot created individually, then placed together using the `grid.arrange` function from the `gridExtra` package (Auguie, 2016). The `geomnet` approach uses the `facet_*` family of functions just as they are used in `ggplot2`, and the `ggnetwork` approach uses the `by` argument in the `ggnetwork` function in combination with the `facet_*` functions. We present the full code for each of these approaches below.

First, the code for the `ggnet2` approach, which results in Figure 4.19(a):

```
# data preparation. first, remove emails sent to all employees
em.day <- subset(email$edges, nrecipients < 54)[, c("From", "to", "day") ]
# for small multiples by day, create one element in a list per day
# (10 days, 10 elements in the list em.day)
em.day <- lapply(unique(em.day$day),
                 function(x) subset(em.day, day == x)[, 1:2 ])
# make the list of edgelists a list of network objects for plotting with ggnet2
em.day <- lapply(em.day, network, directed = TRUE)
# create vertex (employee type) and network (day) attributes for each element in list
for (i in 1:length(em.day)) {
  em.day[[ i ]] %v% "curr_empl_type" <-
    em.cet[ network.vertex.names(em.day[[ i ]]) ]
  em.day[[ i ]] %n% "day" <- unique(email$edges$day)[ i ]
}


# plot ggnet2
# first, make an empty list containing slots for the 10 days (one plot per day)
g <- list(length(em.day))
set.seed(7042016)
# create a ggnet2 plot for each element in the list of networks
for (i in 1:length(em.day)) {
  g[[ i ]] <- ggnet2(em.day[[ i ]], size = 2,
                     color = "curr_empl_type",
                     palette = "Set1", arrow.size = 0,
                     arrow.gap = 0.01, edge.alpha = 0.1,
                     legend.position = "none",
                     mode = "kamadakawai") +
    ggtitle(paste("Day", em.day[[ i ]] %n% "day")) +
    theme(panel.border = element_rect(color = "grey50", fill = NA),
          aspect.ratio = 1)
}
# arrange all of the network plots into one plot window
gridExtra::grid.arrange(grobs = g, nrow = 2)
```

Second, the code for the geomnet approach, which results in Figure 4.19(b):

```
# data step: use the fortify.edgedf group argument to
# combine the edge and node data and allow all nodes to
# show up on all days. Also, remove emails sent to all
# employees
emailnet <- fortify(as.edgedf(subset(email$edges, nrecipients < 54)),
                    email$nodes, group = "day")


# creating the plot

set.seed(7042016)

ggplot(data = emailnet, aes(from_id = from, to_id = to_id)) +

  geom_net(layout.alg = "kamadakawai", singletons = FALSE,

    aes(colour = CurrentEmploymentType,

        group = CurrentEmploymentType,

        linewidth = 2 * (...samegroup.. / 8 + .125)),

        arrowsize = .5,

        directed = TRUE, fiteach = TRUE, ealpha = 0.5, size = 1.5, na.rm = FALSE) +

  scale_colour_brewer("Employment Type", palette = "Set1") +

  theme_net() +

  facet_wrap(~day, nrow = 2, labeller = "label_both") +

  theme(legend.position = "bottom",

        panel.border = element_rect(fill = NA, colour = "grey60"),

        plot.margin = unit(c(0, 0, 0, 0), "mm"))
```

Finally, the code for the `ggnetwork` approach, which results in Figure 4.19(c):

```
# create the network and aesthetics

# first, remove emails sent to all employees

edges <- subset(email$edges, nrecipients < 54)

edges <- edges[, c("From", "to", "day") ]

# Create network class object for plotting with ggnetwork

em.net <- network(edges[, 1:2])

# assign edge attributes (day)

set.edge.attribute(em.net, "day", edges[, 3])

# assign vertex attributes (employee type)
```

```
em.net %v% "curr_empl_type" <- em.cet[ network.vertex.names(em.net) ]


# create the plot
set.seed(7042016)
ggplot(ggnetwork(em.net, arrow.gap = 0.02, by = "day",
                 layout = "kamadakawai"),
       aes(x, y, xend = xend, yend = yend)) +
  geom_edges(
    aes(color = curr_empl_type),
    alpha = 0.25,
    arrow = arrow(length = unit(5, "pt"), type = "closed")) +
  geom_nodes(aes(color = curr_empl_type), size = 1.5) +
  scale_color_brewer("Employment Type", palette = "Set1") +
  facet_wrap(~day, nrow = 2, labeller = "label_both") +
  theme_facet(legend.position = "bottom")
```

Note the two key differences in the visualizations of Figure 4.19: whether singletons (isolated nodes) are plotted (as in the `ggnetwork` method), and whether *one* layout is used across all panels (as for the `ggnetwork` example) or whether individual layouts are fit to each of the subsets (as for the `ggnet2` and the `geomnet` examples). Plotting isolated nodes in `geomnet` is possible by setting `singletons = TRUE`, and it would be possible in `ggnet2` by including all nodes in the creation of the list of networks. Using the same layout for plotting small multiples in `geomnet` is controlled by the argument `fiteach`. By default, `fiteach = TRUE`, but `fiteach = FALSE` results in all panels sharing the same layout. Having the same layout in each panel makes seeing specific differences in ties between nodes easier, while having a different layout in each panel emphasizes the overall structural differences between the sub-networks. It would be interesting to be able to have a hybrid of these two approaches, but at the moment this is beyond the capability of any of the methods. Through the faceting it becomes obvious that there are several days where one or more of the departments does not communicate with any of the other departments. There are only two days, day 13 and day 15, without any isolated department communications. Faceting is one of the major benefits of implementing tools for network visualization in `ggplot2`. Faceting allows the user to quickly separate dense networks into smaller sub-networks for easy visual comparison and analyses, a feature that the other network visualization tools do not have.

4.3.3 `ggplot2` theme elements

This example comes from the `theme()` help page in the `ggplot2` documentation (Wickham, 2009). It is a directed network which shows the structure of the inheritance of theme options in the construction of a `ggplot2` plot. There are 53 vertices and 36 edges in this network. Each vertex represents one possible theme option. There is an arrow from one theme option to another if the element represented by the 'to' vertex inherits its values from the 'from' vertex. For example, the `axis.ticks.x` option inherits its value from the `axis.ticks` value, which in turn inherits its value from the `line` option. Thus, setting the `line` option to a value such as `element_blank()` sets the entire inheritance tree to `element_blank()`, and no lines appear anywhere on the plot background. Code and plots of the inheritance structure are shown in Figure 4.25. A glimpse of the data is below.

```
te.net
##  Network attributes:
##   vertices = 53
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 48
##     missing edges= 0
##     non-missing edges= 48
##
##  Vertex attribute names:
##     size vertex.names
##
## No edge attributes
head(TEnet)
## # A tibble: 6 x 3
## # Groups:   from_id [2]
##   from_id       to_id degree
##    <fctr>      <fctr>  <dbl>
## 1    text       title   6.40
## 2    text legend.text   6.40
## 3    text   axis.text   6.40
## 4    text  strip.text   6.40
## 5    line   axis.line   5.57
## 6    line  axis.ticks   5.57
```

Note the various ways the packages adjust the side of the labels to correspond to the outdegree of the nodes, including the use of the `scale_size_continuous` function in Figure 4.25(c). In each of these plots, it is easy to quickly determine parent-child relationships, and to assess which theme elements are unrelated to all others. Nodes with the most children are the `rect`, `text`, and `line` elements, so we made their labels larger in order to emphasize their importance. In each case, the label size is a function of the out degree of the vertices.

4.3.4 College football

This next example comes from M.E.J. Newman's network data web page (Girvan and Newman, 2002). It is an undirected network consisting of all regular season college football games played between Division I schools in Fall of 2000. There are 115 vertices and 613 edges: each vertex represents a school, and an edge represents a game played between two schools. There is an additional variable in the vertex data frame corresponding to the conference each team belongs to, and there is an additional variable in the edge data frame that is equal to one if the game occurred between teams in the same conference or zero if the game occurred between teams in different conferences. We take a look at the data used in the plots below.

```
fb.net
##  Network attributes:
##   vertices = 115
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 613
##     missing edges= 0
##     non-missing edges= 613
##
##  Vertex attribute names:
##     conf vertex.names
##
##  Edge attribute names:
##     same.conf
head(ftnet)
##    from_id              to_id same.conf         value
## 1 AirForce    NevadaLasVegas         1 Mountain West
## 2    Akron         MiamiOhio         1  Mid-American
## 3    Akron      VirginiaTech         0  Mid-American
## 4    Akron           Buffalo         1  Mid-American
## 5    Akron BowlingGreenState         1  Mid-American
## 6    Akron              Kent         1  Mid-American
##   schools
## 1
## 2
## 3
## 4
## 5
## 6
```

The network of football games is given in Figure 4.32. Here, the `linetype` aesthetic corresponds to games that occur between teams in the same conference or different conferences.
These lines are dotted and solid, respectively. We have also assigned a different color to each conference, so that the vertices and their labels are colored according to their conference. Additionally, in the first two implementations, the edges between two teams in the same conference share that conference color, while edges between teams in different conferences are a default gray color. This coloring and changing of the line types make the structure of the game network easier to view. Additionally, we use the `label` aesthetic in Figure 4.32(b) to label only a few schools that are of interest to us. This is the conference consisting of Navy, Notre Dame, Utah State, Central Florida, and Connecticut, which is spread out, whereas most other conferences' teams are all very close to each other because they play within conference much more than they play out of conference. At the time, these five schools were all independents and did not have a home conference. Without the coloring capability, we would not have been able to pick out that difference as easily.

4.3.5 Southern women

Bipartite (or 'two-mode') networks are networks with two different kinds of nodes and where all ties are formed between these two kinds. Affiliation networks, which represent the ties between individuals and the groups to which they belong, are examples of such networks (see p. 53-54, 123-127 of Newman (2010) for more examples).

One of the classic examples for a two-mode network is the network of 18 Southern women attending 14 social events as collected by Davis et al. (1941) and published e.g. as part of the `tnet` package (Opsahl, 2009). In this data, a woman is linked by an edge to an event if she attended it. One of the questions for these type of networks is gain insight in the interplay between the two different sets of nodes.

The data for the example of the Southern women is reported as edge list in form of 'lady X attending event Y'. With a bit of data preparation as detailed below, we can visualize the graph as shown in Figure 4.38. In creating the plots, we use the `shape` and `colour` aesthetics to map the two different modes to two different shapes and colours.

```
# access the data and rename it for convenience
library(tnet)


data(tnet)
elist <- data.frame(Davis.Southern.women.2mode)
names(elist) <- c("Lady", "Event")
```

The edge list for the Southern women's data consists of women attending events:

```
head(elist,4)
##   Lady Event
## 1    1     1
## 2    1     2
## 3    1     3
## 4    1     4
```

In order to distinguish between nodes from different types, we have to add an additional identifier element, so that we can tell the 'first' woman $L1$ apart from the first event, $E1$.

```
elist$Lady <- paste("L", elist$Lady, sep="")
elist$Event <- paste("E", elist$Event, sep="")


davis <- elist
names(davis) <- c("from", "to")
davis <- rbind(davis, data.frame(from=davis$to, to=davis$from))
davis$type <- factor(c(rep("Lady", nrow(elist)), rep("Event", nrow(elist))))
```

The two different types of nodes are shown by different shapes and colors. We see the familiar relationship between events and groups of women attending these events. Women attending the same events then form a tighter knit subset, while events are also thought of as more similar, if they are attended by the same women. This defines the cluster of events E1 through E5, which are only attended by women 1 through 9, while events E6 through E9 are attended by (almost) everybody making them the core group of events.

4.3.6 Bike sharing in Washington D.C.

The data shows trips taken with bikes from the bike share company Capital Bikeshare[5] during the second quarter of 2015. While this bike sharing company is located in the heart of Washington D.C. the company offers a set of bike stations just outside of Washington in Rockville, MD and north of it. Each station is shown as a vertex, and edges between stations indicate that at least five trips were taken between these two stations; the wider the line, the more trips have been taken between stations. In order to reflect distance between stations, we use as an additional restriction that the fastest trip was at most ten minutes long. Figure 4.44 shows four renderings of this data.

[5] https://secure.capitalbikeshare.com/

The first is a geographically true representation of the area overlaid by lines between bike stations, the other three are networks drawn with `geomnet`, `ggnet2`, and `ggnetwork`, respectively. The code for these renderings is shown below:

```
# make data accessible
data(bikes, package = 'geomnet')
# data step for geomnet
tripnet <- fortify(as.edgedf(bikes$trips), bikes$stations[,c(2,1,3:5)])
# create variable to identify Metro Stations
tripnet$Metro = FALSE
idx <- grep("Metro", tripnet$from_id)
tripnet$Metro[idx] <- TRUE

# plot the bike sharing network shown in Figure 7b
set.seed(1232016)
ggplot(aes(from_id = from_id, to_id = to_id), data = tripnet) +
  geom_net(aes(linewidth = n / 15, colour = Metro),
           labelon = TRUE, repel = TRUE) +
  theme_net() +
  xlim(c(-0.1, 1.1)) +
  scale_colour_manual("Metro Station", values = c("grey40", "darkorange")) +
  theme(legend.position = "bottom")

# data preparation for ggnet2 and ggnetwork
bikes.net <- network(bikes$trips[, 1:2 ], directed = FALSE)
# create edge attribute (number of trips)
network::set.edge.attribute(bikes.net, "n", bikes$trips[, 3 ] / 15)
# create vertex attribute for Metro Station
bikes.net %v% "station" <-  grepl("Metro", network.vertex.names(bikes.net))
bikes.net %v% "station" <-  1 + as.integer(bikes.net %v% "station")
rownames(bikes$stations) <- bikes$stations$name
# create node attributes (coordinates)
bikes.net %v% "lon" <-
  bikes$stations[ network.vertex.names(bikes.net), "long" ]
bikes.net %v% "lat" <-
  bikes$stations[ network.vertex.names(bikes.net), "lat" ]
bikes.col <- c("grey40", "darkorange")

# Non-geographic placement
set.seed(1232016)
ggnet2(bikes.net, mode = "fruchtermanreingold", size = 4, label = TRUE,
       vjust = -0.5, edge.size = "n", layout.exp = 1.1,
       color = bikes.col[ bikes.net %v% "station" ],
       label.color = bikes.col[ bikes.net %v% "station" ])

# Non-geographic placement. Use data from ggnet2 step.
set.seed(1232016)
ggplot(data = ggnetwork(bikes.net, layout = "fruchtermanreingold"),
         aes(x, y, xend = xend, yend = yend)) +
  geom_edges(aes(size = n), color = "grey40") +
  geom_nodes(aes(color = factor(station)), size = 4) +
  geom_nodetext(aes(label = vertex.names, color = factor(station)),
                vjust = -0.5) +
  scale_size_continuous("Trips", breaks = c(2, 4, 6), labels = c(30, 60, 90)) +
  scale_colour_manual("Metro station", labels = c("FALSE", "TRUE"),
                      values = c("grey40", "darkorange")) +
  theme_blank() +
  theme(legend.position = "bottom", legend.box = "horizontal")
```

To plot the geographically correct bike network layout in `geomnet`, we use the `layout.alg = NULL` option and provide the latitude and longitude coordinates of the bike stations from the company's data. A glance of the data that we used in the examples is shown below.

```
bikes.net

##  Network attributes:

##   vertices = 20

##   directed = FALSE

##   hyper = FALSE

##   loops = FALSE

##   multiple = FALSE

##   bipartite = FALSE

##   total edges= 53

##     missing edges= 0

##     non-missing edges= 53

##

##  Vertex attribute names:

##     lat lon station vertex.names

##

##  Edge attribute names:

##     n

head(tripnet[,-c(4:5,8)])

##                          from_id

## 1       Broschart & Blackwell Rd

## 2 Crabbs Branch Way & Calhoun Pl

## 3 Crabbs Branch Way & Calhoun Pl

## 4 Crabbs Branch Way & Calhoun Pl

## 5 Crabbs Branch Way & Calhoun Pl
## 6 Crabbs Branch Way & Calhoun Pl
##                             to_id  n  lat  long Metro
## 1                            <NA> NA 39.1 -77.2 FALSE
## 2 Crabbs Branch Way & Redland Rd 11 39.1 -77.2 FALSE
## 3   Needwood Rd & Eagles Head Ct 14 39.1 -77.2 FALSE
## 4           Rockville Metro East 51 39.1 -77.2 FALSE
## 5           Rockville Metro West  8 39.1 -77.2 FALSE
## 6         Shady Grove Metro West 36 39.1 -77.2 FALSE
```

Because all three approaches result in the same picture, we only show one of these in Figure 4.44a. The code for creating the map is given here:

```
  library(ggmap)
metro_map <- get_map(location = c(left = -77.22257, bottom = 39.05721,
                                  right = -77.11271, top = 39.14247))

# geomnet: overlay bike sharing network on geographic map
  ggmap(metro_map) +
  geom_net(data = tripnet, layout.alg = NULL, labelon = TRUE,
           vjust = -0.5, ealpha = 0.5,
           aes(from_id = from_id,
               to_id = to_id,
               x = long, y = lat,
               linewidth = n / 15,
               colour = Metro)) +
  scale_colour_manual("Metro Station", values = c("grey40", "darkorange")) +
  theme_net() %+replace% theme(aspect.ratio=NULL, legend.position = "bottom") +
  coord_map()
```

We can also make use of the option `layout.alg = NULL` whenever we do not want to use an in-built layout algorithm but make use of a user-defined custom layout. In this case, the coordinates of the layout have to be created outside of the visualization and x and y coordinates have to be made available instead.

4.4 Some considerations of speed

In our examples thus far, we have focused on rather small social or relationship networks and one larger communication network. Now we present an example of a biological network, which comes from Jeong et al. (2001). It is the complete protein-protein interaction network in the yeast species *S. cerevisiae*. There are 2,113 proteins that make up the vertices of this network, with a total of 4480 edges between them. These edges represent "direct physical interactions" between any two proteins (Jeong et al., 2001, p. 42), resulting in a relatively large network. When these interactions and their associated proteins are plotted using the Fruchterman-Reingold layout algorithm, the runtime is extremely long, about 9.5 minutes for 50,000 iterations through the algorithm. The resulting layout is shown in Figure 4.45. When testing the three approaches with the larger network, we decided to use a random layout to save time. Despite its size, each one of the approaches in the `ggplot2` framework can be drawn in a few hundred milliseconds.

Another benefit that emerges from using `ggplot2` for network visualization is the speed at which it can plot fairly large networks. In order to assess the speed gain procured by our three approaches, we ran two separate tests, both of which designate `ggplot2`-based approaches as faster than the plotting functionality offered in the `network` package. They also show the `ggplot2` approaches to be largely on par with the speed provided by the `igraph` package. We first investigate average random layout plotting time of the protein network

shown in Figure 4.45, and then consider average plotting times of increasingly larger random networks. Note that in all tests, default package settings were used. The code to create benchmark results for both of these situations is provided in the vignette of the package `ggCompNet` (Tyner and Hofmann, 2016b). See the Supplementary Material section at the end of this paper for more information.

We plotted the protein interaction network of Figure 4.45 100 times using the `network` and `igraph` packages, and compared their run times to 100 runs each of the three visualization approaches introduced in this paper. The results are shown in Figure 4.46. We can see that on average, the `ggplot2` framework provides a two to three-fold increase in speed over the network package, and that `geomnet` and `ggnetwork` are faster than package `igraph`. The three `ggplot2` approaches also have considerably less variability in time than the `network` package. Despite the large number of vertices, the protein interaction network has a relatively small number of edges (4480 out of over 2.2 million theoretically possible connections resulting in an edge probability of just over 0.0020). Next, we examine networks with a higher edge probability.

The second test relies on random undirected networks in which the probability of an edge between two nodes was set to $p = 0.2$. We generated 100 of these networks at network sizes from 25 to 250 nodes, using increments of 25.

Figure 4.47 summarizes the results of these benchmarks using a convenience sample of machines accessible to the authors, including authors' hardware and additional results from friends' and colleagues' machines. Network sizes are plotted horizontally, execution times of 100 runs under each visualization approach are plotted on the y-axis. Each panel shows a different machine as indicated by the facet label. Note that each panel is scaled separately to account for differences in the overall speed of these machines. What these plots indicate is that we have surprisingly large variability in relative run times across different machines. However, the results support some general findings. The `network` plotting routine is by far the slowest across all machines, while the `igraph` plotting is generally among the fastest. Our three approaches generally feature in between `igraph` and `network` with `ggnet2` being as fast or faster than `igraph` plotting, followed by `ggnetwork` and `geomnet`, which is generally the slowest among the three. These differences become more pronounced as the size of the network increases.

Although speed was not the main rationale for our inquiry into ggplot2-based approaches to network visualization, a speed-based comparison shows a clear advantage of these approaches over the plotting function included in the network package, which very quickly becomes much slower as network size increases.

4.5 Summary and discussion

At first glance, the three visualization approaches may seem nearly identical. However, each one brings unique strengths to the visualization of networks. Out of our three approaches, ggnetwork is most flexible and allows for a re-ordering of layers to emphasize one over the other. The flexibility is useful but does require the user to specify every single part of the network visualization. The geomnet implementation most closely aligns with the existing ggplot2 paradigm because it provides a single layer that can be added to other ggplot2 layers. ggnet2 requires the user to know the least about the ggplot2 framework, while resulting in a valid and extensible ggplot2 object. Many features of the packages would not have been possible, or would have at least been difficult to implement, in prior versions of ggplot2. The increased flexibility of the current development version as well as the added geoms geom_curve and geom_label provided us with a strong, yet flexible, foundation for network visualization. Our approaches also benefit from the speed of ggplot2, making network visualization more efficient than the existing framework of network for a lot of the benchmark examples.

All three approaches rely on the package sna for layouts. This allows the user to access the many layout algorithms available for networks, and in the event that new layouts are implemented in sna, our packages will accommodate them seamlessly. A larger range of layouts is available through igraph, and can be implemented into our packages by setting the respective layout arguments to NULL and passing x,y coordinates calculated from igraph. There are some notable differences between the packages, such as in the parameters used for specific layout algorithms, e.g. igraph allows the use of weights for Fruchterman-Reingold placement, even though it is unclear from the original article how these are supposed to affect the layout. In all three approaches, it is feasible to tap into `igraph`'s functionality in a future version so that the user does not need to calculate the layout separately. Additional future work will explore the implementation of other network data structures, such as the `networkDynamic` class from `statnet`, which would benefit from the faceting capabilities of our implementations. This work will likely incorporate the `fortify` approach of `ggnetwork` and `geomnet::fortify.network()` for converting network data structures to a `ggplot2`-friendly format.

We have found that none of our approaches is unequivocally the best. We can, however, provide some guidance as to which approach is best for which type of user. The main differences between the three methods are in the way that network information is passed into the functions. For `ggnet2` and `ggnetwork`, data management and attribute handling is done through network operators on nodes and edges, while the `geomnet` approach does not require any knowledge of networks or existing network analysis packages from the user. This likely affects the user base of each package. We think that users who are well-versed with networks will find `ggnet2` and `ggnetwork` more intuitive to use than `geomnet`. These users might be looking to `ggplot2` as another avenue to create high-quality visualizations that tap into `ggplot2` advantages such as facetting and, for `ggnetwork`, layering. Users who are already familiar with `ggplot2` and some of the other `tidyverse` packages (see Wickham (2017)), and who find themselves dealing with network data will likely be more attracted to the `geomnet` implementation of network plotting. The data management skills needed for using `geomnet` are basic: some familiarity with the split-apply-combine paradigm, in the form of familiarity with `plyr` or `dplyr`, would be sufficient in order to make full use of the features of `geom_net` (Wickham, 2011). All in all, the three approaches we have presented here provide a wealth of resources to users of all skill sets who are looking to create beautiful network visualizations.

On a personal level we discovered that the collaboration on this paper has helped us to improve upon our initial versions of each of these packages. For instance, the edge coloring in the `ggnet2` function was designed so that edges between two vertices in the same group were colored with that group's vertex color. This inspired an implementation of it in `geomnet` through the traditional `ggplot2` `group` operator. During the process of writing the paper the authors collaborated on a solution for the problem of nodes being plotted on top of arrow tips. This solution was implemented in the geomnet arrow.gap parameter, which allows to re-track the tip of an arrow on a directed edge, and was also added to ggnetwork. In addition, the implementation of a ggplot2 geom for networks within geomnet inspired the creation of the aliased geoms of the ggnetwork package.

Finally, curious users may be interested in how these three packages can fit together and replicate each other, since they are in fact so similar. Thanks to the flexibility inherent to ggnetwork, it is possible to write wrapper functions around ggnetwork functions in order to recreate the behavior and functionality of ggnet2 and geomnet. Simple examples of such wrapper functions, called ggnetwork2 and geom_network, respecively are shown below.

```
library(ggnetwork)
# mimics geom_net behavior
geom_network <- function(edge.param, node.param) {
     edge_ly <- do.call(geom_edges, edge.param)
     node_ly <- do.call(geom_nodes, node.param)
     list(edge_ly, node_ly)
}
# mimics ggnet2 behavoir
ggnetwork2 <- function() { ggplot() + geom_network() }
```

Similarly, geomnet can mimic the the behavior of ggnet2, as shown below.

```
library(geomnet)
geomnet2 <- function(net) {
  ggplot(data = fortify(net),
         aes(from_id = from_id, to_id = to_id)) +
    geom_net()
}
```

Mimicking `ggnetwork` with `geomnet` requires a little bit more work because the native data input for `geomnet` is a `"data.frame"` object fortified with `geomnet` methods, not a `"network"` object. Instead, the internal `ggplot2` function `ggplot_build` allows a plot created with `geomnet` function calls to be recreated with `ggnetwork`-like syntax. An example of using a `geomnet` plot to create a similar plot in the style of `ggnetwork` follows to reproduce Figure 4.9(c).

```
library(geomnet)
library(ggnetwork)
library(dplyr)
# a ggnetwork-like creation using a geomnet plot
data("blood")
# first, create the geomnet plot to access the data later
geomnetplot <- ggplot(data = blood$edges, aes(from_id = from, to_id =
                                                 to)) +
                  geom_net(layout.alg = "circle", selfloops = TRUE) +
                theme_net()
# get the data
dat <- ggplot_build(geomnetplot)$data[[1]]
# ggnetwork-like construction for re-creating network shown in Figure 5
ggplot(data = dat, aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_segment(arrow = arrow(type = 'closed'), colour = 'grey40') +
  geom_point(size = 10, colour = 'darkred') +
  geom_text(aes(label = from), colour = 'grey80', size = 4) +
  geom_circle() +
  theme_blank() + theme(aspect.ratio = 1)
```

Supplementary Material

Software: ggnetwork 0.5.1 and geomnet 0.2.0 were used to create the visualizations. ggnet2 is part of ggally 1.3.0.

Reproducibility: All the code used in the examples is available as a vignette in the CRAN package ggCompNet. There are two vignettes: one for the speed comparisons and one for the visualizations provided in the Examples section. The package also provide our speed test data for creating Figure 4.47. We created this package to accompany this paper with the hope that interested users will compare these methods on their own systems and against their own code. Finally, all of the data we use in the examples, with the exception of the bipartite network example, is included as a part of the geomnet package.

Figure 4.11 ggnet2

```
# make data accessible
data(email, package = 'geomnet')

# create node attribute data
em.cet <- as.character(
  email$nodes$CurrentEmploymentType)
names(em.cet) = email$nodes$label

# remove the emails sent to all employees
edges <- subset(email$edges, nrecipients < 54)
# create network
em.net <- edges[, c("From", "to") ]
em.net <- network(em.net, directed = TRUE)
# create employee type node attribute
em.net %v% "curr_empl_type" <-
  em.cet[ network.vertex.names(em.net) ]
set.seed(10312016)
ggnet2(em.net, color = "curr_empl_type",
       size = 4, palette = "Set1", arrow.gap = 0.(
       arrow.size = 5, edge.alpha = 0.25,
       mode = "fruchtermanreingold",
       edge.color = c("color", "grey50"),
       color.legend = "Employment Type") +
  theme(legend.position = "bottom")
```

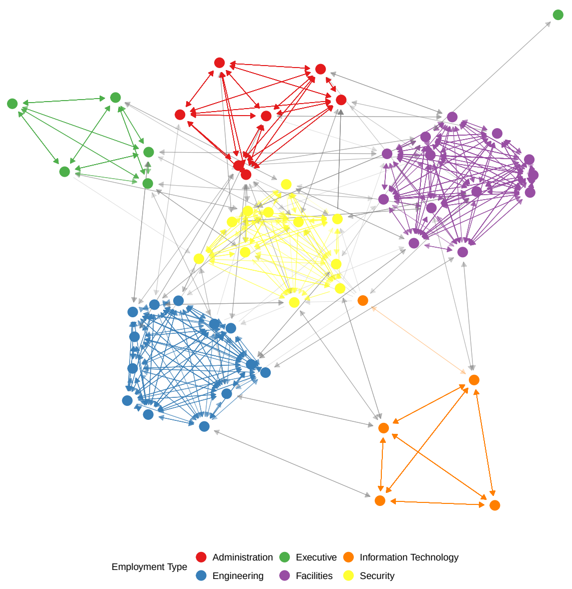


Figure 4.12 geomnet

```
# data step for the geomnet plot
email$edges <- email$edges[, c(1,5,2:4,6:9)]
emailnet <- fortify(
  as.edgedf(subset(email$edges, nrecipients < 54)),
  email$nodes)
set.seed(10312016)
ggplot(data = emailnet,
       aes(from_id = from_id, to_id = to_id)) +
  geom_net(layout.alg = "fruchtermanreingold",
    aes(colour = CurrentEmploymentType,
        group = CurrentEmploymentType,
        linewidth = 3 * (...samegroup.. / 8 + .125)),
    ealpha = 0.25, size = 4, curvature = 0.05,
    directed = TRUE, arrowsize = 0.5) +
  scale_colour_brewer("Employment Type", palette = "Set1") +
  theme_net() +
  theme(legend.position = "bottom")
```

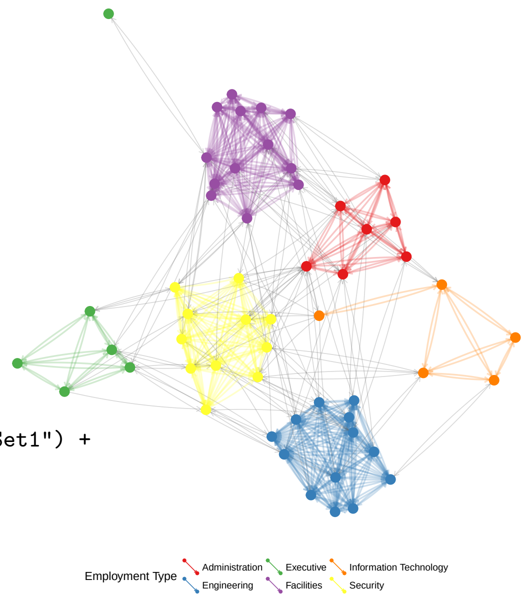


Figure 4.13: Email network within a company over a two week period.

Figure 4.14 ggnetwork

```
# use em.net created in ggnet2step
set.seed(10312016)
ggplot(ggnetwork(em.net, arrow.gap = 0.02,
              layout = "fruchtermanreingold"),
       aes(x, y, xend = xend, yend = yend)) +
  geom_edges(
    aes(color = curr_empl_type),
    alpha = 0.25,
    arrow = arrow(length = unit(5, "pt"),
                  type = "closed"),
    curvature = 0.05) +
  geom_nodes(aes(color = curr_empl_type),
             size = 4) +
  scale_color_brewer("Employment Type",
                     palette = "Set1") +
  theme_blank() +
  theme(legend.position = "bottom")
```

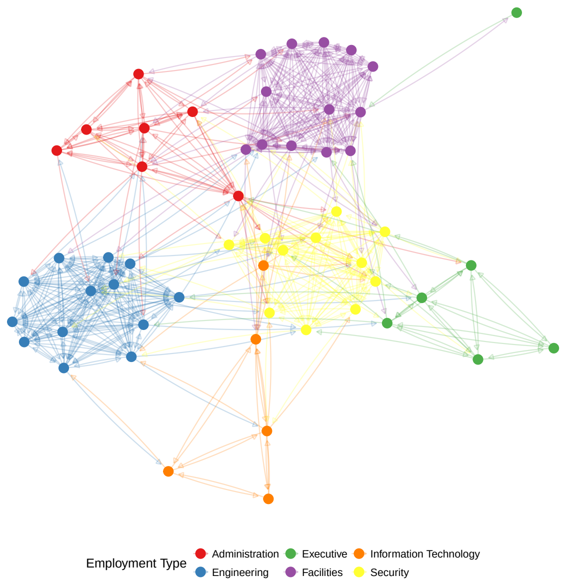


Figure 4.15: (continued) Email network within a company over a two week period.

Figure 4.17 ggnet2

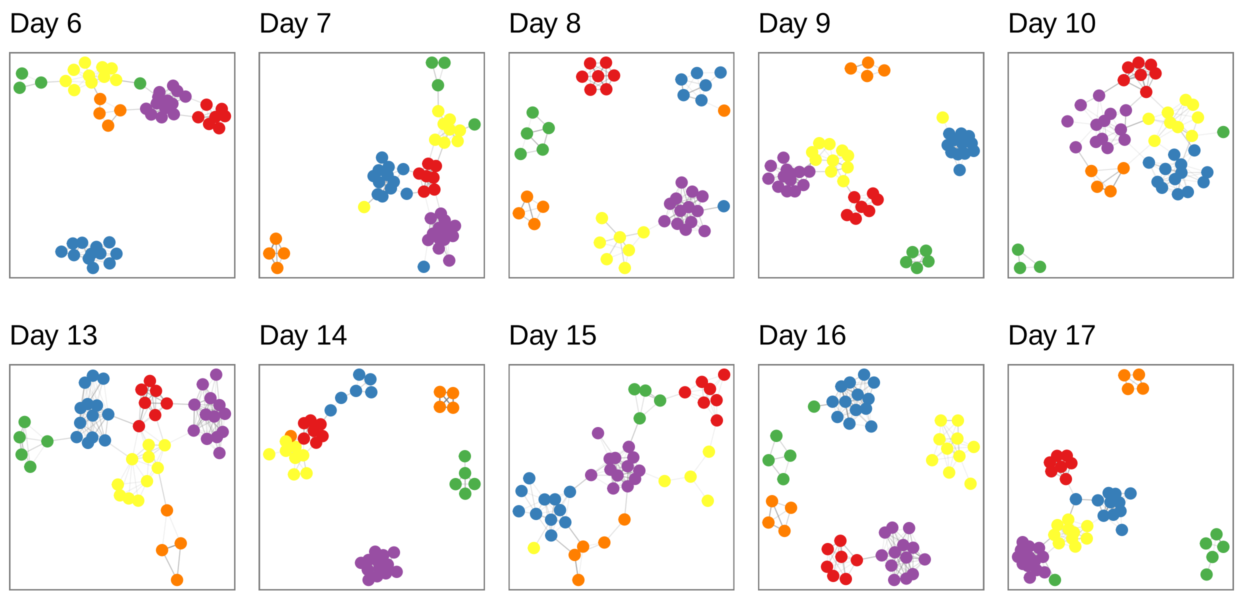


Figure 4.18 geomnet

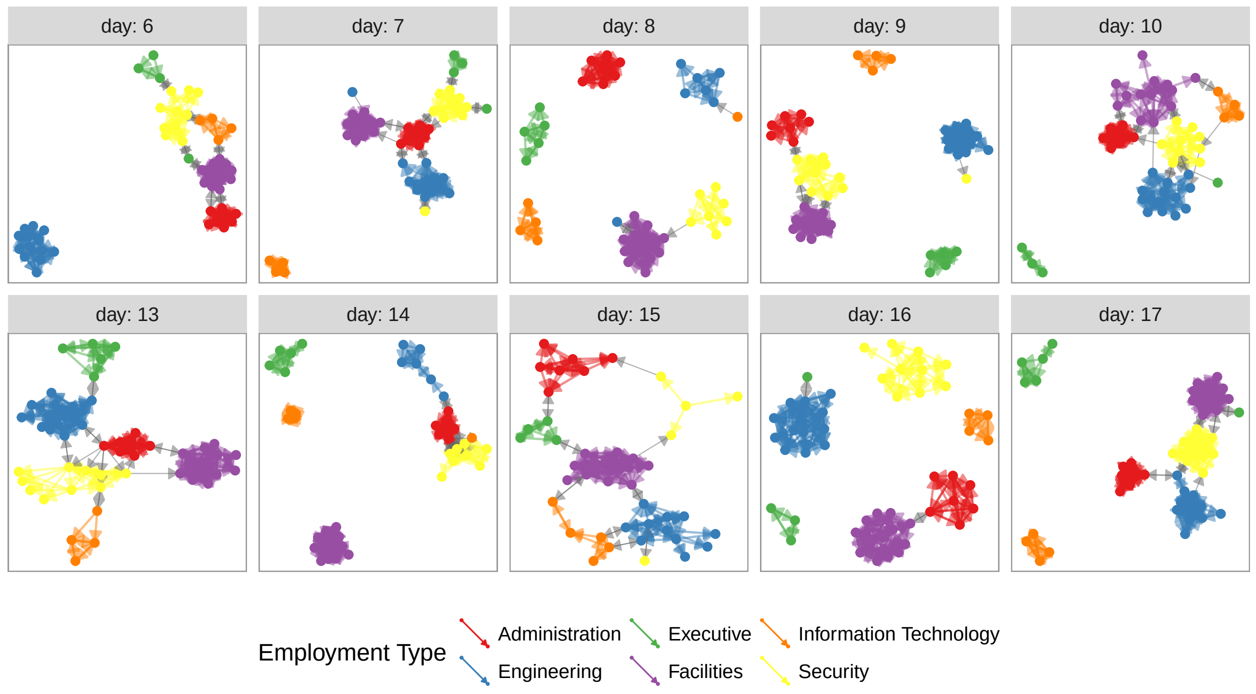


Figure 4.19: The same email network as in Figure 4.13 faceted by day of the week.

Figure 4.20 ggnetwork

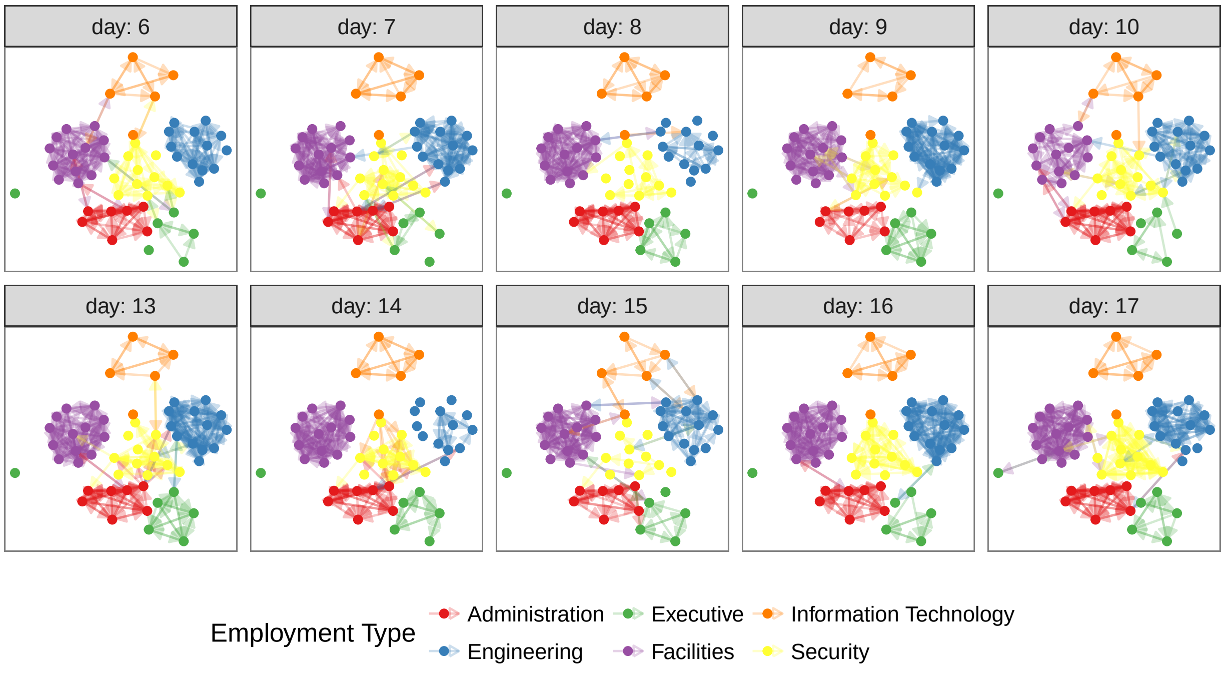


Figure 4.21: (continued) The same email network as in Figure 4.13 faceted by day of the week.

Figure 4.23 ggnet2

```
# make data accessible
data(theme_elements, package = "geomnet")

# create network object
te.net <- network(theme_elements$edges)
# assign node attribut (size based on node degree)
te.net %v% "size" <-
  sqrt(10 * (sna::degree(te.net) + 1))
set.seed(3272016)
ggnet2(te.net, label = TRUE, color = "white",
       label.size = "size", layout.exp = 0.15,
       mode = "fruchtermanreingold")
```

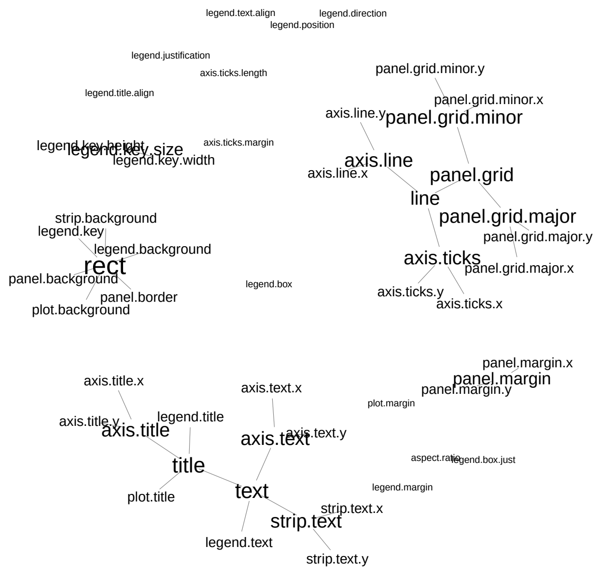


Figure 4.24 geomnet

```
# data step: merge nodes and edges and
# introduce a degree-out variable
# data step: merge nodes and edges and
# introduce a degree-out variable
TEnet <- fortify(
  as.edgedf(theme_elements$edges[,c(2,1)]),
            theme_elements$vertices)
TEnet <- TEnet %>%
  group_by(from_id) %>%
  mutate(degree = sqrt(10 * n() + 1))

# create plot:
set.seed(3272016)
ggplot(data = TEnet,
       aes(from_id = from_id, to_id = to_id)) +
  geom_net(layout.alg = "fruchtermanreingold",
    aes(fontsize = degree), directed = TRUE,
    labelon = TRUE, size = 1, labelcolour = 'black',
    ecolour = "grey70", arrowsize = 0.5,
    linewidth = 0.5, repel = TRUE) +
  theme_net() +
  xlim(c(-0.05, 1.05))
```

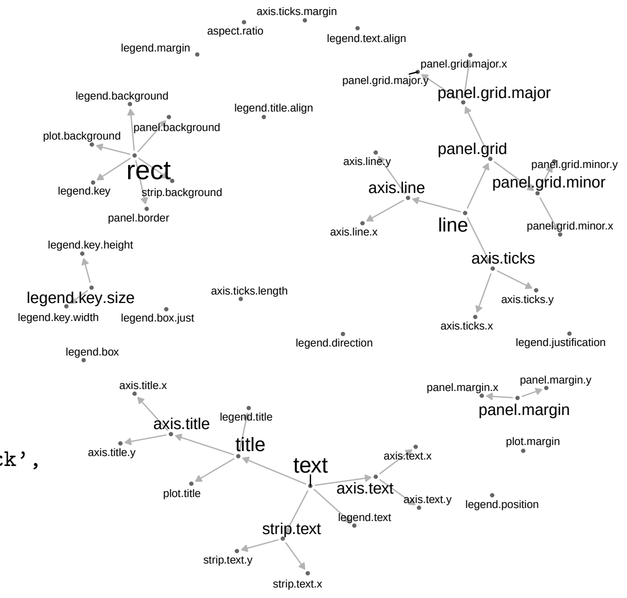


Figure 4.25: Inheritance structure of `ggplot2` theme elements. This is a recreation of the graph found at http://docs.ggplot2.org/current/theme.html.

Figure 4.26 ggnet2

```
set.seed(3272016)
# use network created in ggnet2 data step
ggplot(ggnetwork(te.net,
                 layout = "fruchtermanreingold"),
       aes(x, y, xend = xend, yend = yend)) +
  geom_edges() +
  geom_nodes(size = 12, color = "white") +
  geom_nodetext(
    aes(size = size, label = vertex.names)) +
  scale_size_continuous(range = c(4, 8)) +
  guides(size = FALSE) +
  theme_blank()
```

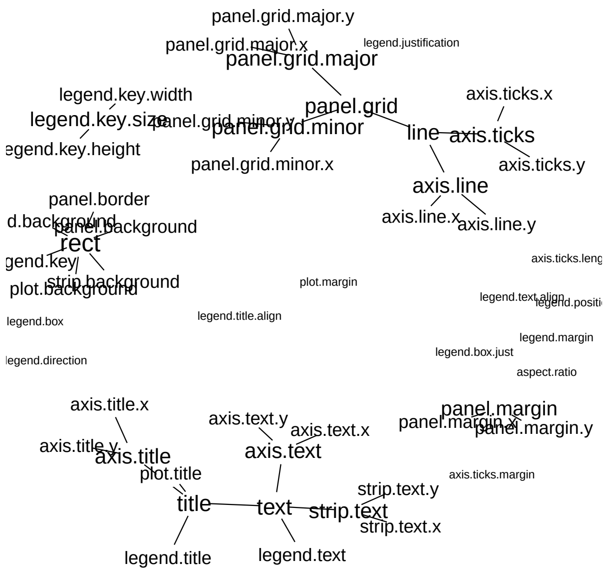


Figure 4.27: (continued) Inheritance structure of `ggplot2` theme elements. This is a recreation of the graph found at http://docs.ggplot2.org/current/theme.html.

Figure 4.29 ggnet2

```
#make data accessible
data(football, package = 'geomnet')
rownames(football$vertices) <-
  football$vertices$label
# create network
fb.net <- network(football$edges[, 1:2],
                  directed = TRUE)
# create node attribute
# (what conference is team in?)
fb.net %v% "conf" <-
  football$vertices[
    network.vertex.names(fb.net), "value"
    ]
# create edge attribute
# (between teams in same conference?)
set.edge.attribute(
  fb.net, "same.conf",
  football$edges$same.conf)
set.seed(5232011)
ggnet2(fb.net, mode = "fruchtermanreingold",
       color = "conf",  palette = "Paired",
       color.legend = "Conference",
       edge.color = c("color", "grey75"))
```

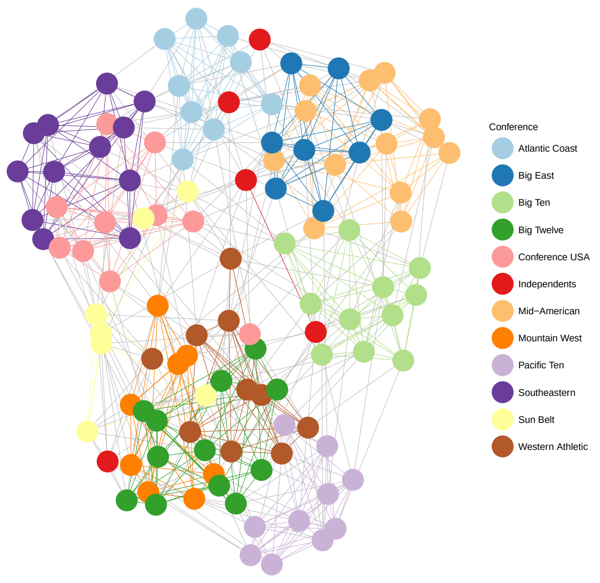


Figure 4.30 geomnet

```
# data step: merge vertices and edges
# data step: merge vertices and edges
ftnet <- fortify(as.edgedf(football$edges),
                 football$vertices)

# create new label variable for independent schools
ftnet$schools <- ifelse(
  ftnet$value == "Independents", ftnet$from_id, "")

# create data plot
set.seed(5232011)
ggplot(data = ftnet,
       aes(from_id = from_id, to_id = to_id)) +
  geom_net(layout.alg = 'fruchtermanreingold',
    aes(colour = value, group = value,
        linetype = factor(same.conf != 1),
        label = schools),
    linewidth = 0.5,
    size = 5, vjust = -0.75, alpha = 0.3) +
  theme_net() +
  theme(legend.position = "bottom") +
  scale_colour_brewer("Conference", palette = "Paired")  +
  guides(linetype = FALSE)
```

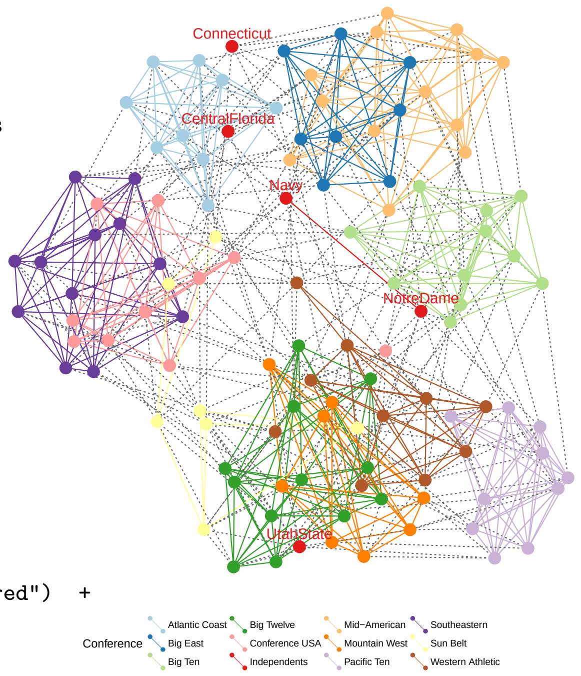


Figure 4.31 ggnetwork

```
# use network from ggnet2 step
set.seed(5232011)
ggplot(
  ggnetwork(
    fb.net,
    layout = "fruchtermanreingold"),
  aes(x, y, xend = xend, yend = yend)) +
  geom_edges(
    aes(linetype = as.factor(same.conf)),
    color = "grey50") +
  geom_nodes(aes(color = conf), size = 4) +
  scale_color_brewer("Conference",
                     palette = "Paired") +
  scale_linetype_manual(values = c(2,1)) +
  guides(linetype = FALSE) +
  theme_blank()
```

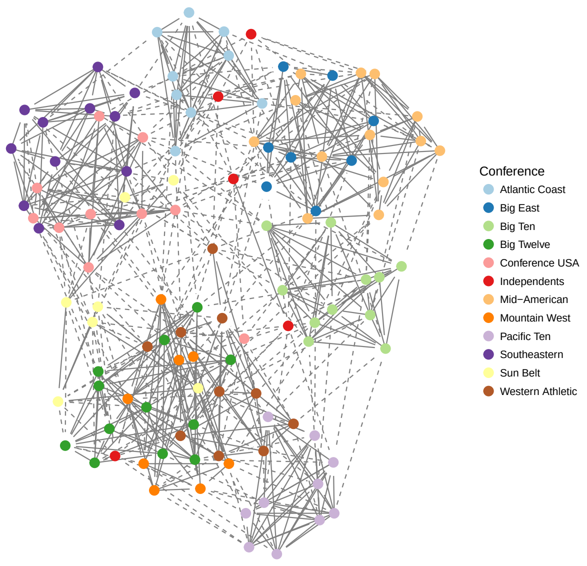


Figure 4.32: (continued) The network of regular season Division I college football games in the season of fall 2000. The vertices and their labels are colored by conference.

Figure 4.34 ggnet2

```
# Southern women network in ggnet2
# create affiliation matrix
bip = xtabs(~Event+Lady, data=elist)

# weighted bipartite network
bip = network(bip,
              matrix.type = "bipartite",
              ignore.eval = FALSE,
              names.eval = "weights")

# detect and color the mode
set.seed(8262013)
ggnet2(bip, color = "mode", palette = "Set2",
       shape = "mode", mode = "kamadakawai",
       size = 15, label = TRUE) +
  theme(legend.position="bottom")
```

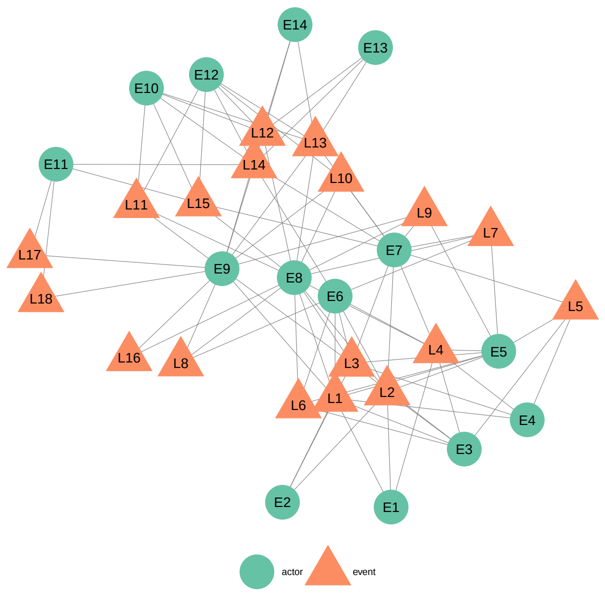


Figure 4.35: Graph of the Southern women data. Women are represented as orange triangles, events as green circles.

Figure 4.36 geomnet

```
# Southern women network in geomnet
# change labelcolour
davis$lcolour <-
  c("white", "black")[as.numeric(davis$type)]

set.seed(8262013)
ggplot(data = davis) +
  geom_net(layout.alg = "kamadakawai",
    aes(from_id = from, to_id = to,
        colour = type, shape = type),
    size = 15, labelon = TRUE, ealpha = 0.25,
    vjust = 0.5, hjust = 0.5,
    labelcolour = davis$lcolour) +
  theme_net() +
  scale_colour_brewer("Type of node", palette = "Set2") +
  scale_shape("Type of node") +
  theme(legend.position = "bottom")
```

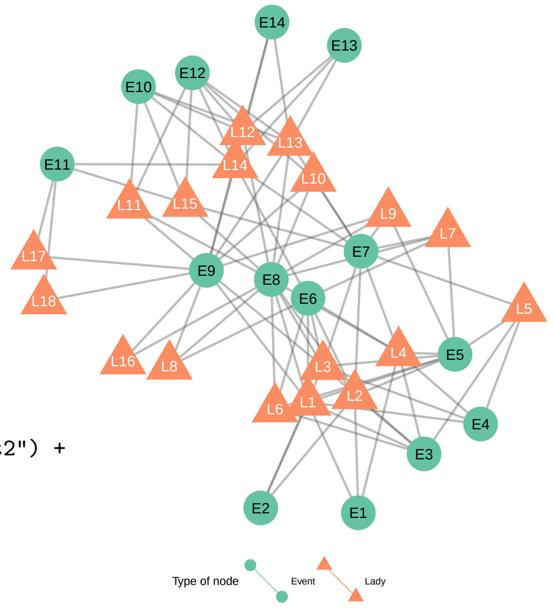


Figure 4.37 ggnetwork

```
# Southern women network in ggnetwork. Use data fı
# assign vertex attributes (Node type and label)
set.vertex.attribute(bip, "mode",
  c(rep("event", 14), rep("woman", 18)))
set.seed(8262013)
ggplot(data = ggnetwork(bip,
        layout = "kamadakawai"),
    aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_edges(colour = "grey80") +
  geom_nodes(aes(colour = mode, shape = mode),
             size = 15) +
  geom_nodetext(aes(label = vertex.names)) +
  scale_colour_brewer(palette = "Set2") +
  theme_blank() +
  theme(legend.position = "bottom")
```

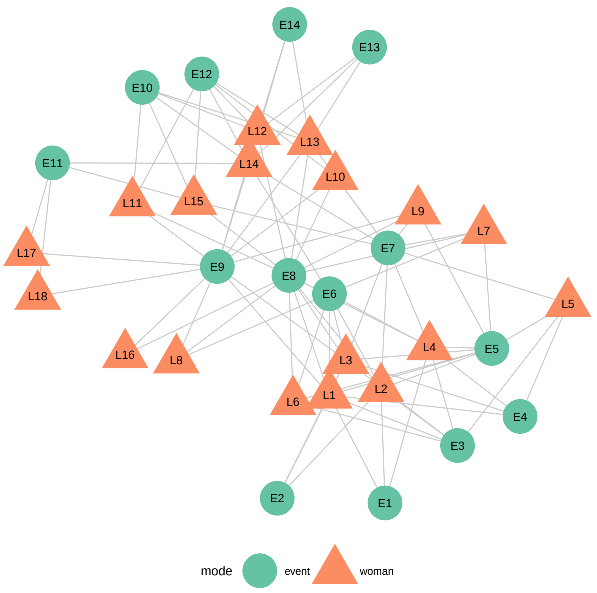


Figure 4.38: Graph of the Southern women data. Women are represented as orange triangles, events as green circles.

Figure 4.40 geographic map

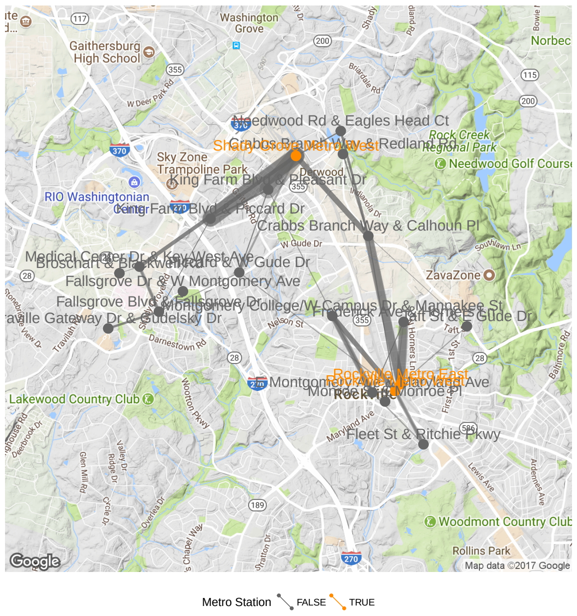

Needwood Rd & Eagles Head Ct
Crabbs Branch Way & Redland Rd
Shady Grove Metro West
King Farm Blvd & Pleasant Dr
King Farm Blvd & Piccard Dr
Crabbs Branch Way & Calhoun Pl
Medical Center Dr & Key West Ave
Broschart & Blackwell Rd
Piccard & W Gude Dr
Fallsgrove Dr & W Montgomery Ave
Fallsgrove Blvd & Fallsgrove Dr
Montgomery College/W Campus Dr & Mannakee St
Traville Gateway Dr & Gudelsky Dr
Frederick Ave & Horners Ln
Taft St & E Gude Dr
Rockville Metro East
Rockville Metro West
E Montgomery Ave & Maryland Ave
Monroe St & Monroe Pl
Fleet St & Ritchie Pkwy
Google
Map data ©2017 Google
Metro Station
FALSE
TRUE

Figure 4.41 geomnet

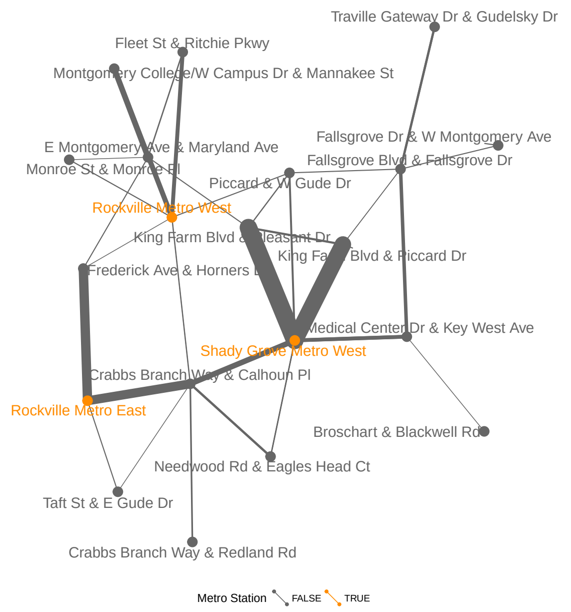

Traville Gateway Dr & Gudelsky Dr
Fleet St & Ritchie Pkwy
Montgomery College/W Campus Dr & Mannakee St
E Montgomery Ave & Maryland Ave
Monroe St & Monroe Pl
Fallsgrove Dr & W Montgomery Ave
Fallsgrove Blvd & Fallsgrove Dr
Piccard & W Gude Dr
Rockville Metro West
King Farm Blvd & Pleasant Dr
King Farm Blvd & Piccard Dr
Frederick Ave & Horners Ln
Medical Center Dr & Key West Ave
Shady Grove Metro West
Crabbs Branch Way & Calhoun Pl
Rockville Metro East
Broschart & Blackwell Rd
Needwood Rd & Eagles Head Ct
Taft St & E Gude Dr
Crabbs Branch Way & Redland Rd
Metro Station
FALSE
TRUE

Figure 4.42 ggnet2

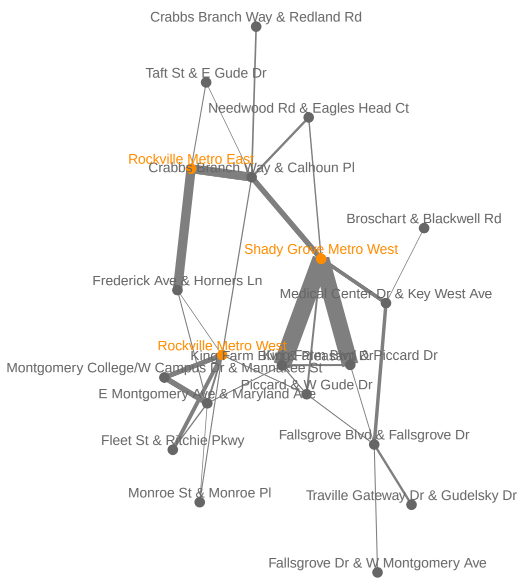

Crabbs Branch Way & Redland Rd
Taft St & E Gude Dr
Needwood Rd & Eagles Head Ct
Rockville Metro East
Crabbs Branch Way & Calhoun Pl
Broschart & Blackwell Rd
Shady Grove Metro West
Frederick Ave & Horners Ln
Medical Center Dr & Key West Ave
Rockville Metro West
King Farm Blvd & Pleasant Dr
King Farm Blvd & Piccard Dr
Montgomery College/W Campus Dr & Mannakee St
Piccard & W Gude Dr
E Montgomery Ave & Maryland Ave
Fallsgrove Blvd & Fallsgrove Dr
Fleet St & Ritchie Pkwy
Monroe St & Monroe Pl
Traville Gateway Dr & Gudelsky Dr
Fallsgrove Dr & W Montgomery Ave

Figure 4.43 ggnetwork

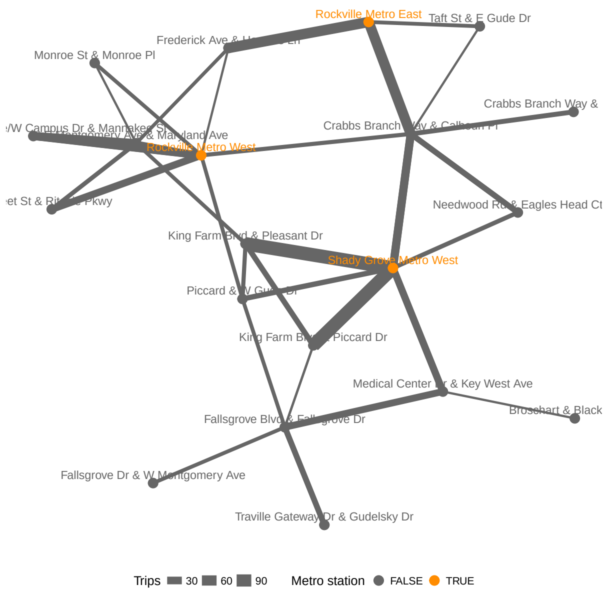

Rockville Metro East
Taft St & E Gude Dr
Frederick Ave & Horners Ln
Monroe St & Monroe Pl
Crabbs Branch Way &
/W Campus Dr & Mannakee St
E Montgomery Ave & Maryland Ave
Rockville Metro West
Crabbs Branch Way & Calhoun Pl
et St & Ritchie Pkwy
Needwood Rd & Eagles Head Ct
King Farm Blvd & Pleasant Dr
Shady Grove Metro West
Piccard & W Gude Dr
King Farm Blvd & Piccard Dr
Medical Center Dr & Key West Ave
Broschart & Black
Fallsgrove Blvd & Fallsgrove Dr
Fallsgrove Dr & W Montgomery Ave
Traville Gateway Dr & Gudelsky Dr
Trips
30
60
90
Metro station
FALSE
TRUE

Figure 4.44: Network of bike trips using a geographically true representation(top left) overlaid on a satellite map, a Kamada-Kawai layout in `geomnet` (top right), a Fruchterman-Reingold layout in `ggnet2` (bottom left) and `ggnetwork` (bottom right). Metro stations are shown in orange. In both the Kamada-Kawai and the Fruchterman-Reingold layouts, metro stations take a much more central position than in the geographically true representation.

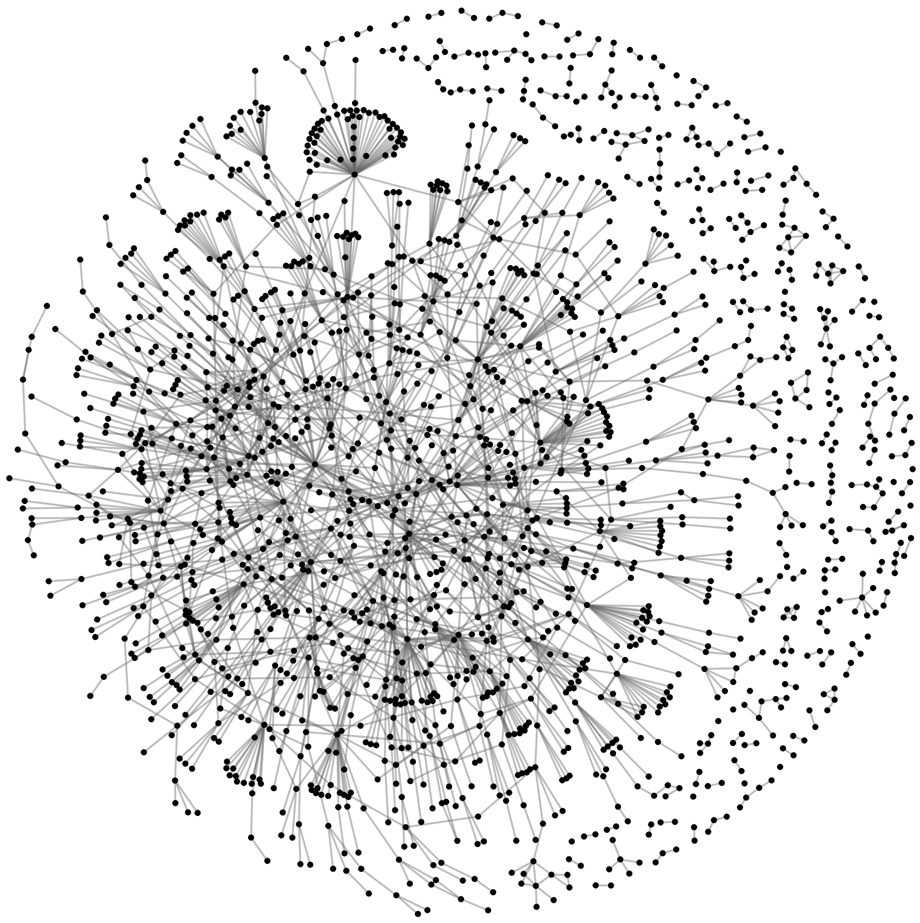


Figure 4.45: Protein-protein interaction network in *S. cerevisiae.* A Fruchterman-Reingold algorithm allowed to run for 50,000 iterations produced the coordinates for the nodes.

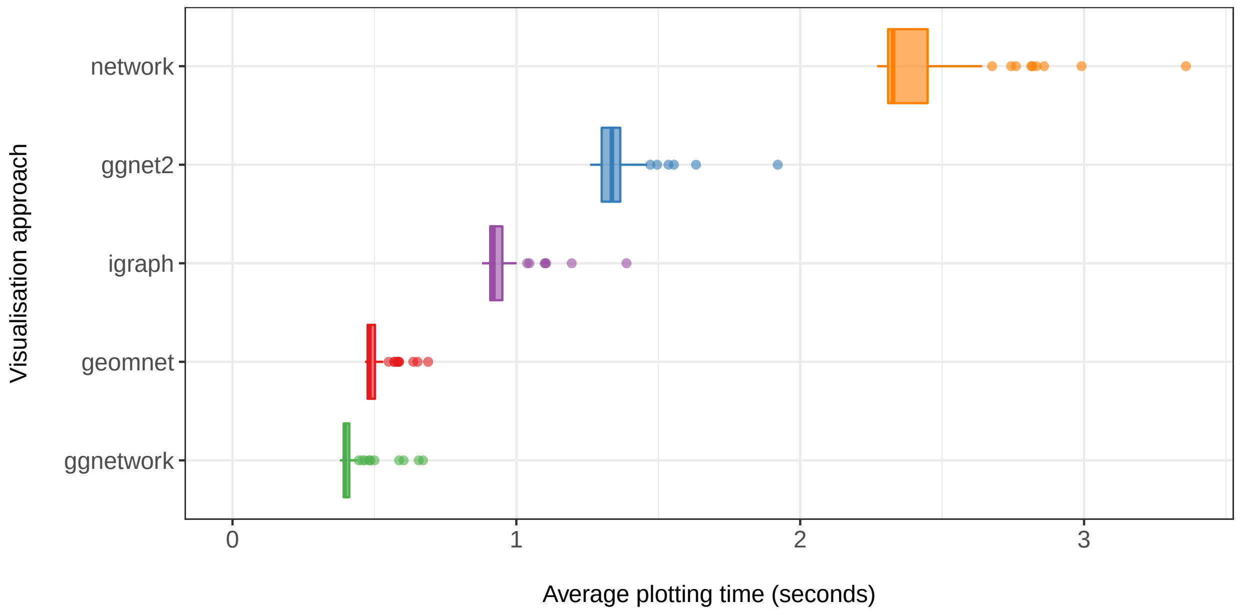


Figure 4.46: Comparison of the times needed for calculating and rendering the previously discussed protein interaction network in the three `ggplot2` approaches and the standard plotting routines of the `network` and `igraph` packages based on 100 evaluations each.

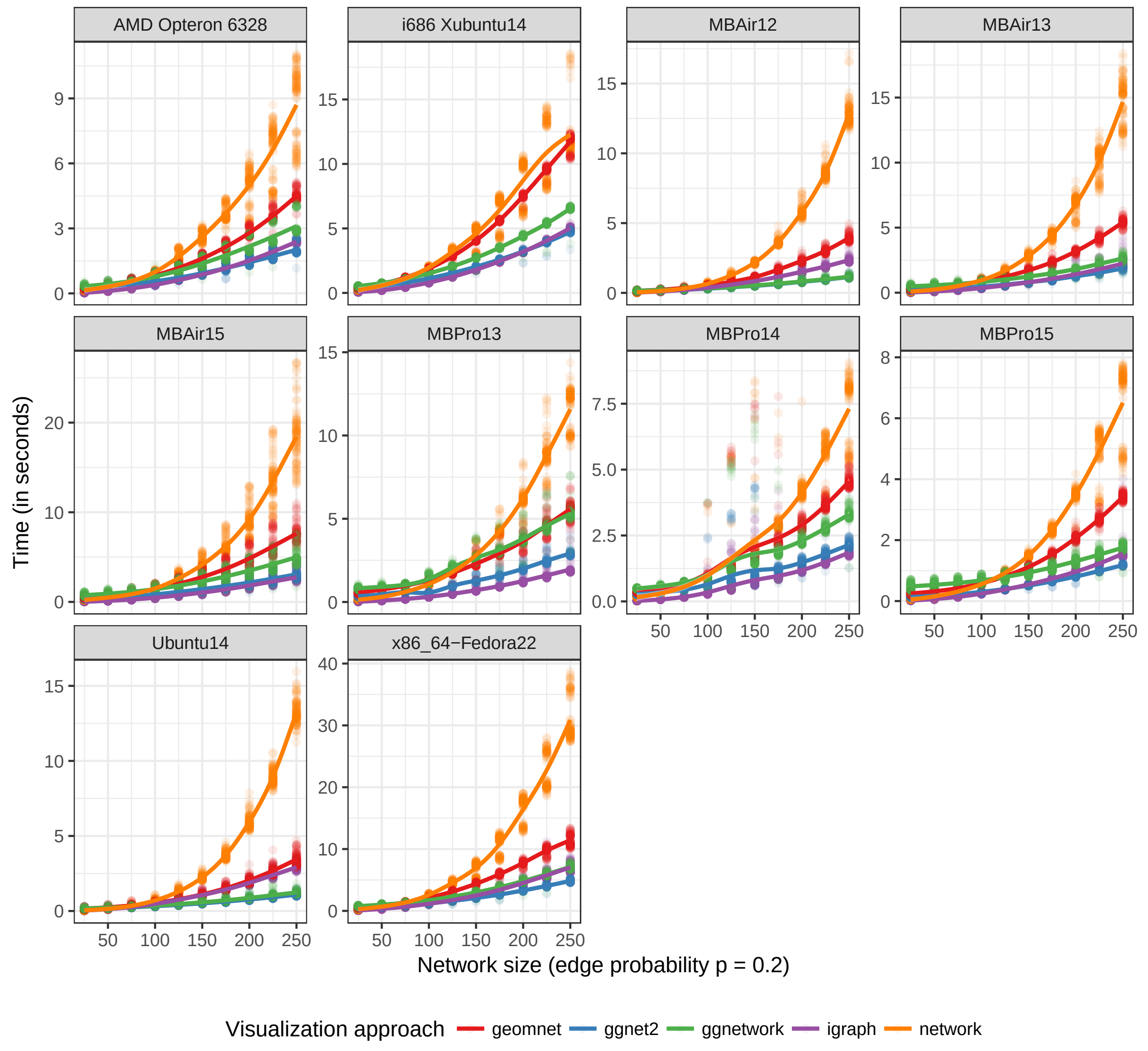


Figure 4.47: Plotting times of random undirected networks of different sizes under each of the available visualization approaches using their default settings. Note that each panel is scaled independently to highlight relative differences in the visualization approaches rather than speed of different hardware.

CHAPTER 5. SUMMARY AND DISCUSSION

There are a vast array of methods for modelling network data. We have chosen to explore one type of model for one type of network data: stochastic actor-oriented models for social network data. These models are complex and relatively new, compared to other statistical network models, so they are difficult to approach. The aim of our work is to bring attention to these models, both for their potential and their potential flaws. Their intractability has led to complex simulation methods for fitting, which in turn has led to these computations being hidden and hard to comprehend. In Chapter 2, we brought these models out of the cave and into the light. By glimpsing the underlying processes behind computation, we have exposed these models for other researchers to fully explore. In Chapter 3, we performed "statistics on street corners" to determine what these models look like, how the parameter values shape the overall structures of the simulated networks, and how well the models actually capture the structure of the data they are modeling. In Chapter 4, we provided a series of tools to the statisticians and sociologists who are interested in modelling dynamic network data, with SAOMs and other network models, that give them beautiful, intutively created graphics of their often complex data.

The suite of stochastic actor-oriented models for dynamic social network data take a lot of time and effort to understand and apply for even the smallest of datasets, as was shown in Chapter 2. We have developed some tools, in Chapter 4, and methods, in Chapters 2 and 3, to lower the "barrier of entry" to working with these models throughout this body of work, but there is still a lot work to be done to further develop these models in a robust statistical framework. Even more work computationally needs to be done to speed up the process of fitting these models. For the relatively small senate collaboration network example of Chapter 2, the `RSiena` software takes minutes on a personal computer to fit a model. With the exponentially increasing amount of network data and the increasing interest in network data in the 21st century, new, faster computing methods need to be developed for these models to remain relevant.

In the future, we would like to explore the space of statistical models for social networks, learning more about them in new and exciting ways, especially through the visualization methods we have developed in this work. We would also like to collect social network data from various sources, whether that be sociological data or social media data, and really push the limits of stochastic actor-oriented models.

BIBLIOGRAPHY

Amazon (2010). *Amazon Mechanical Turk: Artificial Artificial Intelligence*. https://www.mturk.com/mturk/welcome.

Auguie, B. (2016). *gridExtra: Miscellaneous Functions for "Grid" Graphics*. R package version 2.2.1.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

Bates, D., Mächler, M., Bolker, B., and Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48.

Bender-deMoll, S. (2016). *ndtv: Network Dynamic Temporal Visualizations*. R package version 0.10.0.

Bojanowski, M. (2015). *intergraph: Coercion Routines for Network Data Objects*. R package version 2.0-2.

Brandes, U., Kenis, P., and Wagner, D. (2003). Communicating centrality in policy network drawings. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):241–253.

Briatte, F. (2016). *ggnetwork: Geometries to Plot Networks with 'ggplot2'*. R package version 0.5.1.

Buja, A., Cook, D., Hofmann, H., Lawrence, M., Lee, E.-K., Swayne, D. F., and Wickham, H. (2009). Statistical inference for exploratory data analysis and model diagnostics. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 367(1906):4361–4383.

Buldyrev, S. V., Parshani, R., Paul, G., Stanley, H. E., and Havlin, S. (2010). Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025–1028.

Butts, C. T. (2008). network: a Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2).

Butts, C. T. (2014). *sna: Tools for Social Network Analysis*. R package version 2.3-2.

Butts, C. T., Handcock, M. S., and Hunter, D. R. (2014). *network: Classes for Relational Data*. Irvine, CA. R package version 1.10.2.

Chang, W. (2012). *gcookbook: Data for "R Graphics Cookbook"*. R package version 1.0.

Chang, W. (2013). *R graphics cookbook*. O'Reilly, Sebastopol, CA.

Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. (2017). *shiny: Web Application Framework for R*. R package version 1.0.3.

Chowdhury, N. R., Cook, D., Hofmann, H., Majumder, M., Lee, E.-K., and Toth, A. L. (2014). Using visual statistical inference to better understand random class separations in high dimension, low sample size data. *Computational Statistics*, 30(2):293–316.

Cook, K., Grinstein, G., and Whiting, M. (2014). Vast challenge 2014. http://hcil2.cs.umd.edu/newvarepository/benchmarks.php.

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, Complex Systems:1695.

Davis, A., Gardner, B. B., and Gardner, M. R. (1941). *Deep South: A social anthropological study of caste and class*. The University of Chicago Press, Chicago, IL.

Donald E. Knuth, R. W. and John J. Watkins, e. (2013). *Combinatorics: Ancient and Modern*, chapter Two thousand years of combinatorics. Oxford University Press.

Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, 42(11):149–160.

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

Fekete, J.-D. (2009). Visualizing networks using adjacency matrices: Progresses and challenges. In *11th IEEE International Conference on Computer-Aided Design and Computer Graphics*, pages 636– 638.

Fruchterman, T. M. and Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164.

Gelman, A. (2004). Exploratory Data Analysis for Complex Models. *Journal of Computational and Graphical Statistics*, 13(4):755–779.

Ghoniem, M., Fekete, J.-D., and Castagliola, P. (2005). On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4(2):114. Copyright - Palgrave Macmillan Ltd 2005; Document feature - charts; graphs; tables; references; equations; Last updated - 2012-01-28.

Gibson, H., Faith, J., and Vickers, P. (2013). A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357. Copyright - SAGE Publications Jul 2013; Document feature - Tables; ; Last updated - 2013-08-05.

Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(12):7821–7826.

Goldenberg, A., Zheng, A. X., Fienberg, S. E., and Airoldi, E. M. (2010). A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233.

Good, P. (2005). *Permutation, Parametric, and Bootstrap Tests of Hypotheses*. Springer, New York.

Gross, J. H., Kirkland, J. H., and Shalizi, C. R. (2008). Cosponsorship in the u.s. senate: A multi-level two-mode approach to detecting subtle social predictors of legislative support. *Unpublished Manuscript*.

Grossman, J. (2016). *The Erdös Number Project*. http://wwwp.oakland.edu/enp/.

Handcock, M. S., Hunter, D. R., Butts, C. T., Goodreau, S. M., and Morris, M. (2008). statnet: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11.

Harrison, L., Yan, F., Franconeri, S., and Chang, R. (2014). Ranking Visualizations of Correlation Using Weber's Law. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1943–1952.

Hofmann, H., Follett, L., Majumder, M., and Cook, D. (2012a). Graphical Tests for Power Comparison of Competing Designs. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2441–2448.

Hofmann, H., Follett, L., Majumder, M., and Cook, D. (2012b). Graphical tests for power comparison of competing designs. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2441–2448.

Hofmann, H. and Röttger, C. (2016). *vinference: Inference under the lineup protocol.* R package version 0.1.1.

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

Horikoshi, M. and Tang, Y. (2015). *Data Visualization Tools for Statistical Analysis Results.* R package version 0.0.4.

Jeong, H., Mason, S. P., Barabási, A.-L., and Oltvai, Z. N. (2001). Lethality and centrality in protein networks. *Nature*, 411:41–42.

Junker, B. and Schreiber, F. (2008). *Analysis of Biological Networks.* Wiley Series in Bioinformatics. Wiley.

Kahle, D. and Wickham, H. (2013). ggmap: Spatial visualization with ggplot2. *The R Journal*, 5(1):144–161.

Kamada, T. and Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15.

Liiv, I. (2010). Seriation and Matrix Reordering Methods: An Historical Overview. *Statistical Analysis and Data Mining*, 3(2):70–91.

Loy, A., Follett, L., and Hofmann, H. (2016). Variations of Q–Q Plots: The Power of Our Eyes! *The American Statistician*, 70(2):202–214.

Loy, A. and Hofmann, H. (2015). Are You Normal? The Problem of Confounded Residual Structures in Hierarchical Linear Models. *Journal of Computational and Graphical Statistics*, 24(4):1191–1209.

Majumder, M., Hofmann, H., and Cook, D. (2013a). Validation of visual statistical inference, applied to linear models. *Journal of the American Statistical Association*, 108(503):942–956.

Majumder, M., Hofmann, H., and Cook, D. (2013b). Validation of Visual Statistical Inference, Applied to Linear Models. *Journal of American Statistical Association*, 108(503):942–956.

Michell, L. and Amos, A. (1997). Girls, pecking order and smoking. *Social Science & Medicine*, 44(12):1861–1869.

Moody, J., McFarland, D., and Bender-deMoll, S. (2005). Dynamic network visualization. *American Journal of Sociology*, 110(4):1206?1241.

Newman, M. E. J. (2010). *Networks : an introduction.* Oxford University Press, Oxford New York.

Opsahl, T. (2009). *Structure and Evolution of Weighted Networks.* University of London (Queen Mary College), London, UK.

Prell, C. (2011). *Social Network Analysis: History, Theory and Methodology.* SAGE Publications.

R Core Team (2016). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.

Ringe, N., Victor, J. N., and Cho, W. T. (2017). *The Oxford Handbook of Political Networks*, chapter Legislative Networks. Oxford.

Ripley, R., Boitmanis, K., and Snijders, T. A. (2013). *RSiena: Siena - Simulation Investigation for Empirical Network Analysis*. R package version 1.1-232.

Ripley, R., Boitmanis, K., Snijders, T. A., and Schoenenberger, F. (2016). *RSienaTest: Siena - Simulation Investigation for Empirical Network Analysis*. R package version 1.1-294.

Ripley, R. M., Snijders, T. A., Boda, Z., Vörös, A., and Preciado, P. (2017). Manual for rsiena. Technical report, University of Oxford: Department of Statistics; Nuffield College; University of Groningen: Department of Sociology, https://www.stats.ox.ac.uk/~snijders/siena/RSiena_Manual.pdf.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22:400–407.

Schloerke, B., Crowley, J., Cook, D., Briatte, F., Marbach, M., Thoen, E., Elberg, A., and Larmarange, J. (2016). *GGally: Extension to ggplot2*. R package version 1.1.0.

Schweinberger, M. (2012). Statistical modelling of network panel data: Goodness of fit. *British Journal of Mathematical and Statistical Psychology*, 65(2):262–281.

Slowikowski, K. (2017). *ggrepel: Repulsive Text and Label Geoms for 'ggplot2'*. R package version 0.7.0.

Snijders, T., Steglich, C., and Schweinberger, M. (2007). *Longitudinal Models in the Behavioral and Related Sciences*, chapter Modeling the Coevolution of Networks and Behavior. Lawrence Erlbaum Associates.

Snijders, T. A. (2001). The statistical evaluation of social network dynamics. *Sociological Methodology*, 31(1):361–395.

Snijders, T. A. (2005). *Models and Methods in Social Network Analysis*, chapter Models for Longitudinal Network Data. New York: Cambridge University Press.

Snijders, T. A. (2016). *Siena Algorithms*. University of Oxford: Department of Statistics, https://www.stats.ox.ac.uk/~snijders/siena/Siena_algorithms.pdf.

Snijders, T. A. (2017). Stochastic actor-oriented models for network dynamics. *Annual Review of Statistics and Its Application*, 4:343–63.

Snijders, T. A., van de Bunt, G. G., and Steglich, C. E. (2010a). Introduction to stochastic actor-based models for network dynamics. *Social Networks*, 32(1):44 – 60. Dynamics of Social Networks.

Snijders, T. A. B. (1996). Stochastic actor-oriented models for network change. *Journal of Mathematical Sociology*, 21(1-2):149–172.

Snijders, T. A. B., Koskinen, J., and Schweinberger, M. (2010b). Maximum likelihood estimation for social network dynamics. *The Annals of Applied Statistics*, 4(2):567–588.

Snijders, T. A. B., Koskinen, J., and Schweinberger, M. (2010c). Maximum likelihood estimation for social network dynamics. *The Annals of Applied Statistics*, 4(2):567–588.

Swan, M. (2013). The quantified self: Fundamental disruption in big data science and biological discovery. *Big Data*, 1(2):85–99.

Tamassia, R., editor (2013). *Handbook of Graph Drawing and Visualization*. CRC Press.

Tang, Y., Horikoshi, M., and Li, W. (2016). ggfortify: Unified interface to visualize statistical result of popular r packages. *The R Journal*.

Tyner, S. and Hofmann, H. (2016a). *geomnet: Network Visualization in the 'ggplot2' Framework*. R package version 0.2.0.

Tyner, S. and Hofmann, H. (2016b). *ggCompNet: Compare Timing of Network Visualizations*. R package version 0.1.0.

Vander Plas, S. and Hofmann, H. (2015). Clusters beat trend!? testing feature hierarchy in statistical graphics. *Journal of Computational and Graphical Statistics*, page submitted.

Wasserman, S. S. (1980). A stochastic model for directed graphs with transition rates determined by reciprocity. *Sociological Methodology*, 11:392–412.

Watts, D. and Strogatz, S. (1998a). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

Watts, D. J. (2004). The “new” science of networks. *Annual Review of Sociology*, 30:243–270.

Watts, D. J. and Strogatz, S. H. (1998b). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–2.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York.

Wickham, H. (2011). The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29.

Wickham, H. (2017). *tidyverse: Easily Install and Load 'tidyverse' Packages.* R package version 1.1.1.

Wickham, H., Cook, D., and Hofmann, H. (2015). Visualizing statistical models: Removing the blindfold. *Statistical Analysis and Data Mining*, 8(4):203–225.

Wilkinson, L. (1999). *The Grammar of Graphics.* NY: Springer, New York.

Yin, G. G. and Zhang, Q. (2010). *Continuous-Time Markov Chains and Applications: A Two-Time-Scale Approach.* Springer, New York, 2nd edition. ISBN 978-1-4614-4345-2.

Yin, T., Cook, D., and Lawrence, M. (2012). ggbio: an R package for extending the grammar of graphics for genomic data. *Genome Biology*, 13(8):R77.

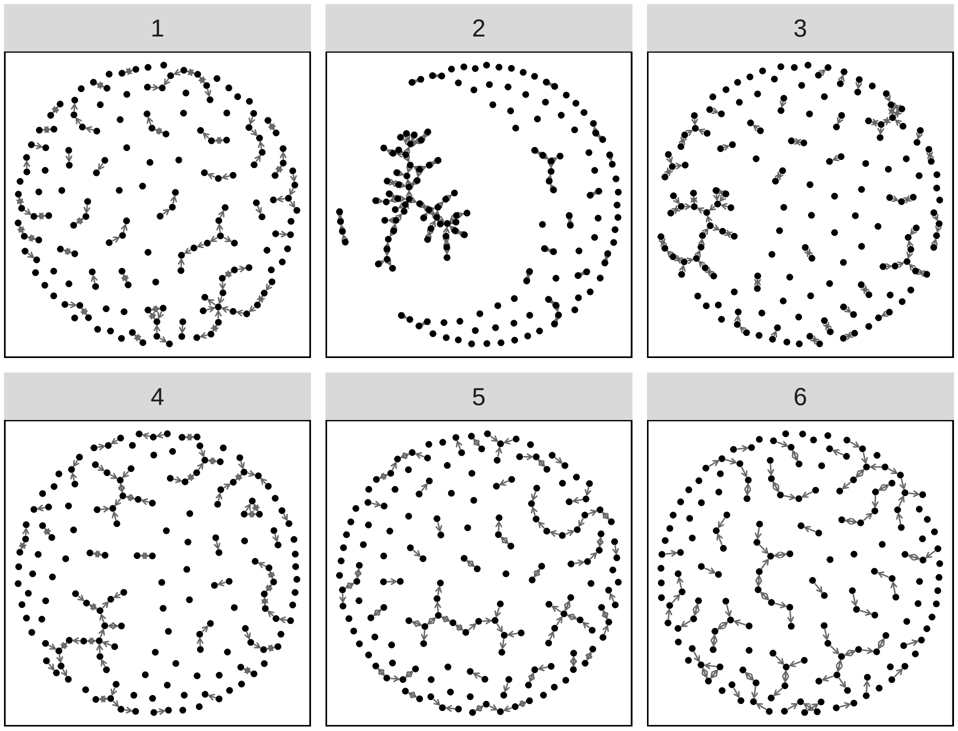
Yu, G., Smith, D., Zhu, H., Guan, Y., and Lam, T. T.-Y. (submitted). ggtree: an R package for visualization and annotation of phylogenetic tree with different types of meta-data. *Methods in Ecology and Evolution.*

Zhao, Y., Cook, D., Hofmann, H., Majumder, M., and Chowdhury, N. R. (2013). Mind Reading: Using an Eye-Tracker to See How People are Looking at Lineups. *International Journal of Intelligent Technologies and Applied Statistics*, 6(4):393–413.

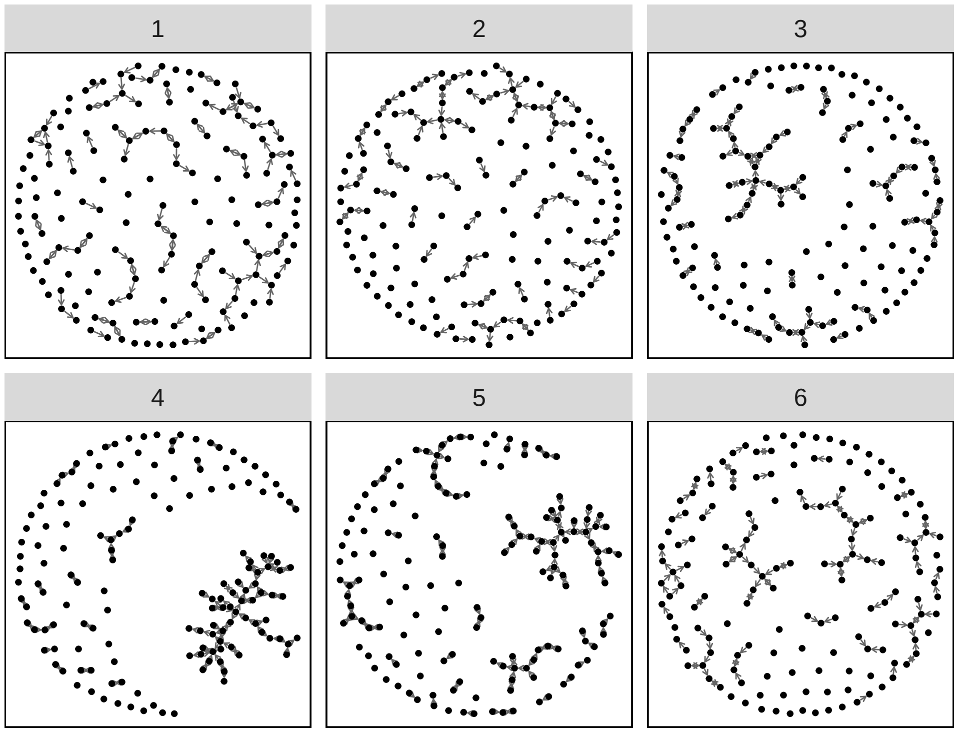
APPENDIX. ADDITIONAL MATERIAL - CHAPTER 3

The lineups shown to participants in our study from Chapter 3 are shown below. The title of the plot encodes the type of lineup as “model_type_difficulty_rep”. The values of ‘model’ correspond to the models M1, M3, M4, M5, M6, and M7, and the parameter being altered in the lineup, $\beta_1, \ldots, \beta_6$. The model codes are: dens, recip, jttp, jtts, simttb, samettp, and bigmod, which correspond to (M1, β_1), (M1, β_2), (M3, β_3), (M4, β_4) , (M5, β_5), (M6, β_6), and (M7, goodness of fit only). The values of ‘type’ correspond to the lineup types: ‘gof’ is a goodness of fit lineup (contains the data), ‘neg’ is lineup type -1, and ‘pos’ is lineup type 1. The ‘difficulty’ is value is 1,2,3, or 9, for easy, medim, hard, and goodness of fit (difficulty is not applicable). Finally, the ‘rep’ value is the replicate of the lineup condition, 1,2,3.

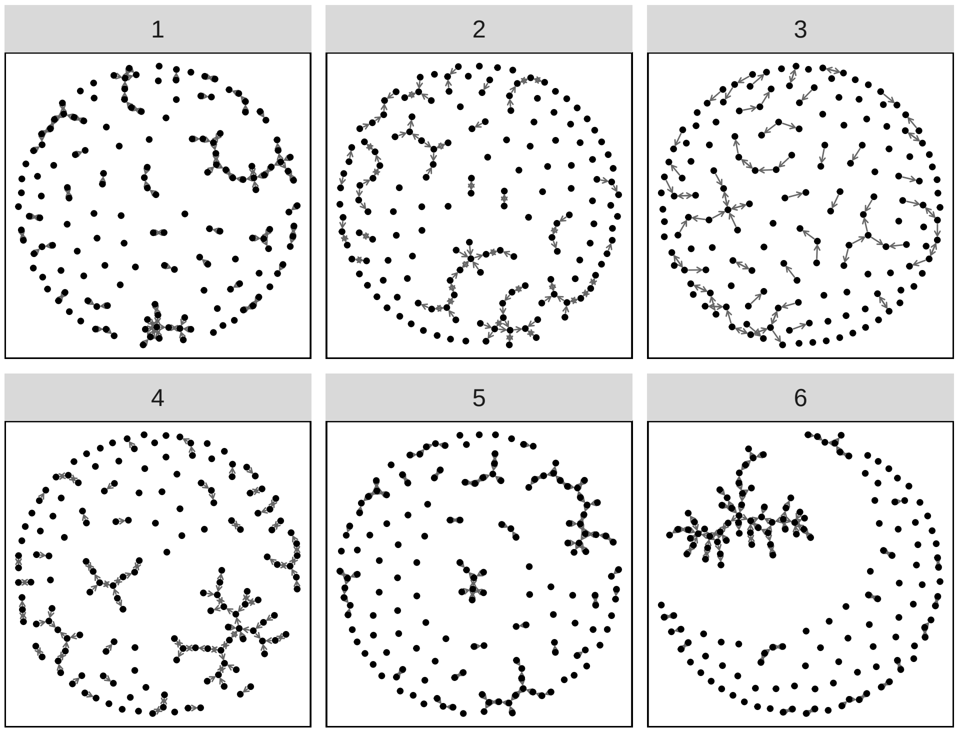
bigmod_gof_9_1



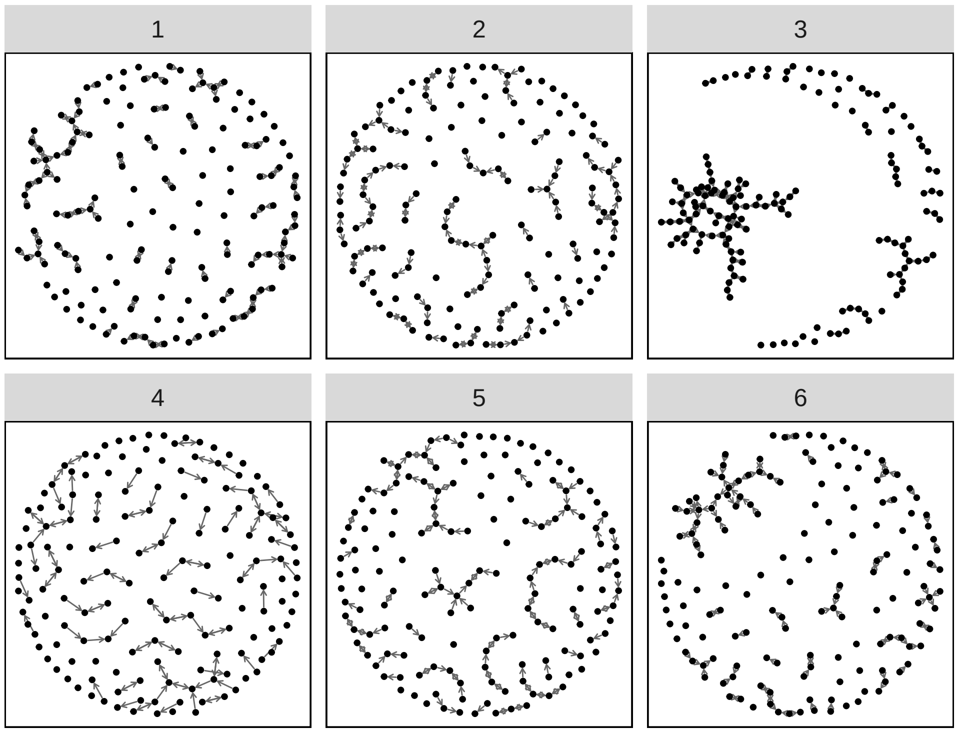
bigmod_gof_9_2

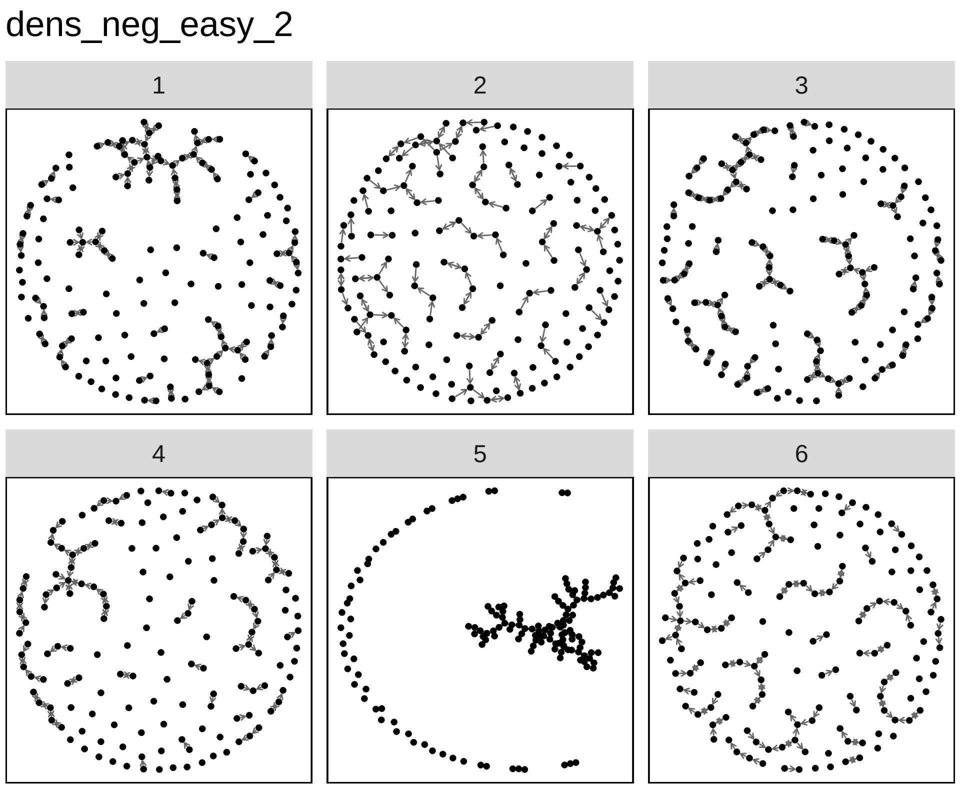


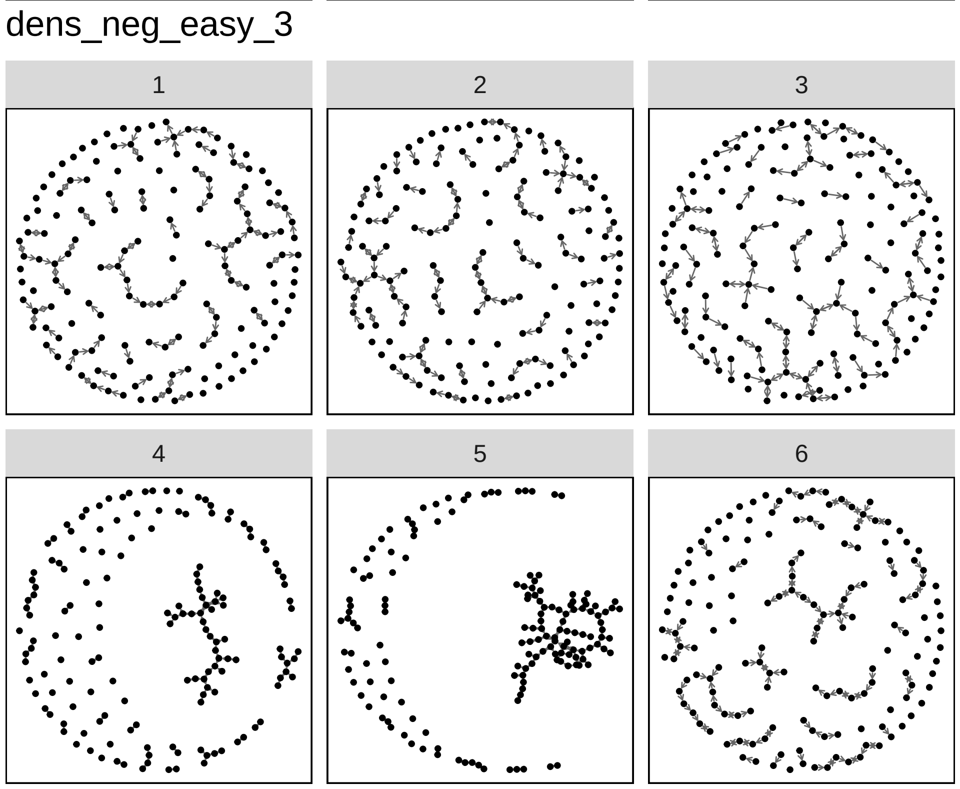
bigmod_gof_9_3



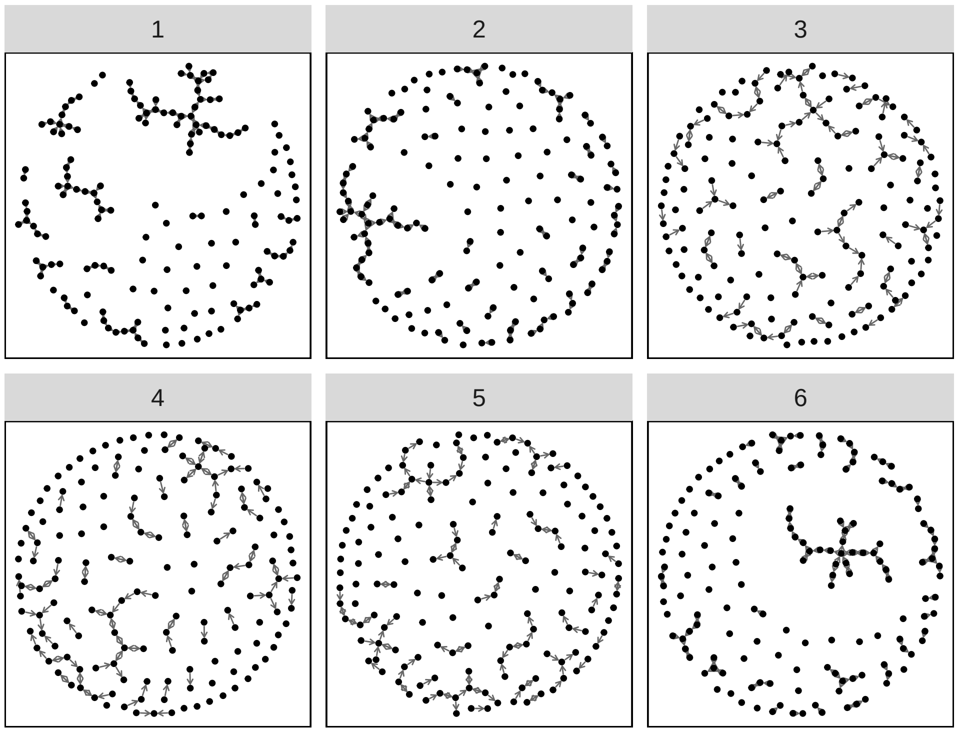
dens_neg_easy_1



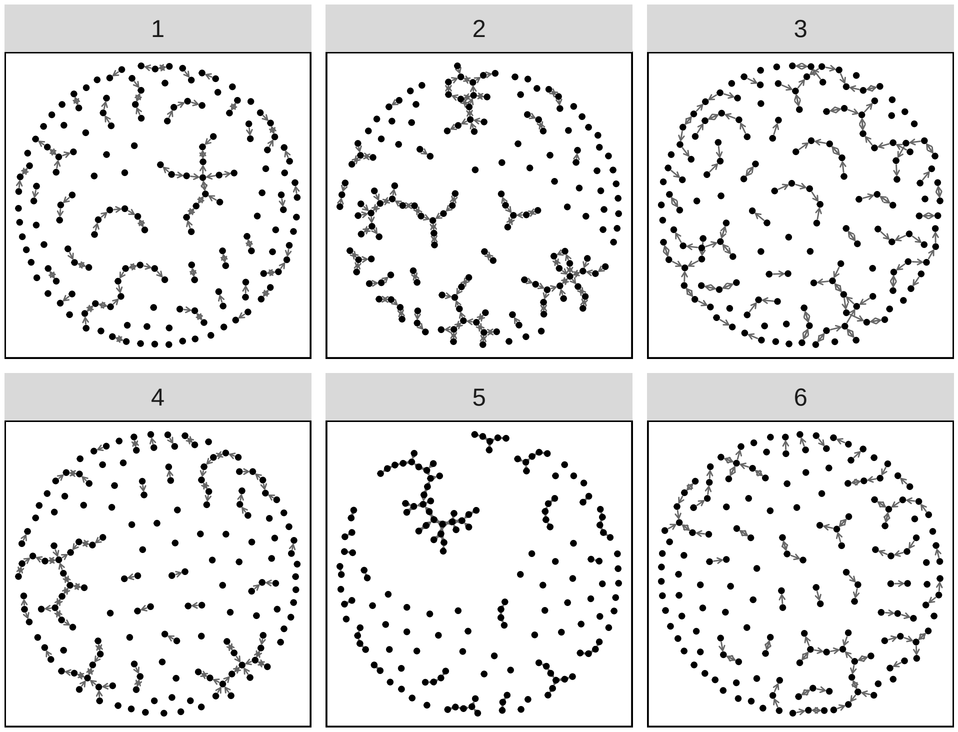

dens_neg_easy_2
1
2
3
4
5
6

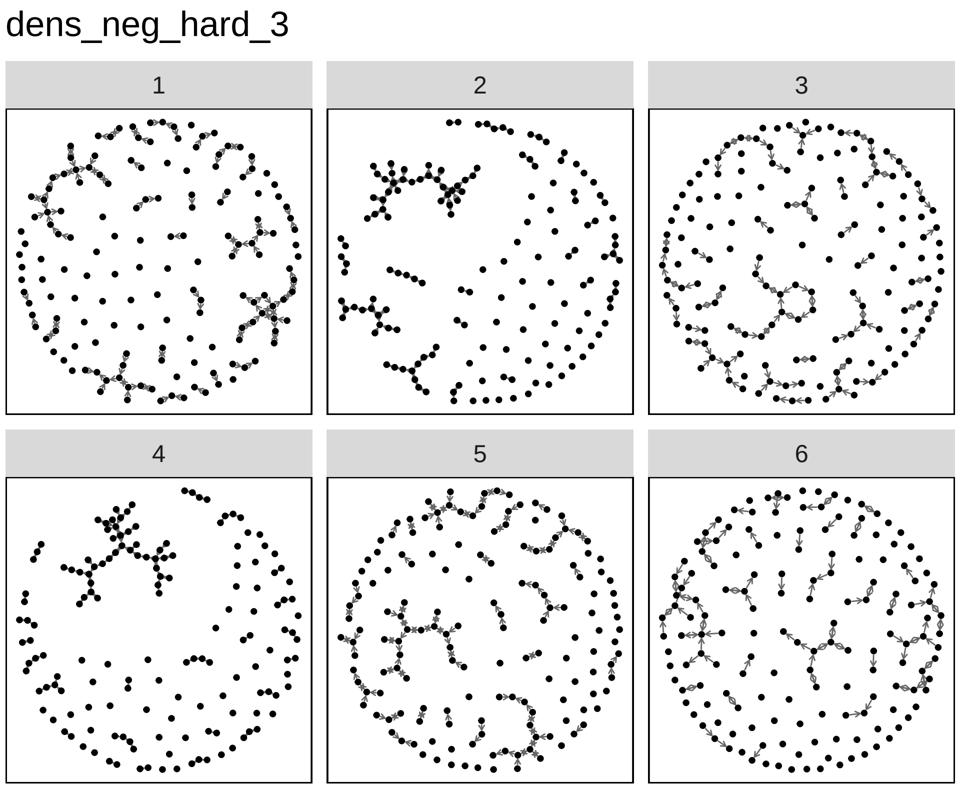

dens_neg_easy_3
1
2
3
4
5
6

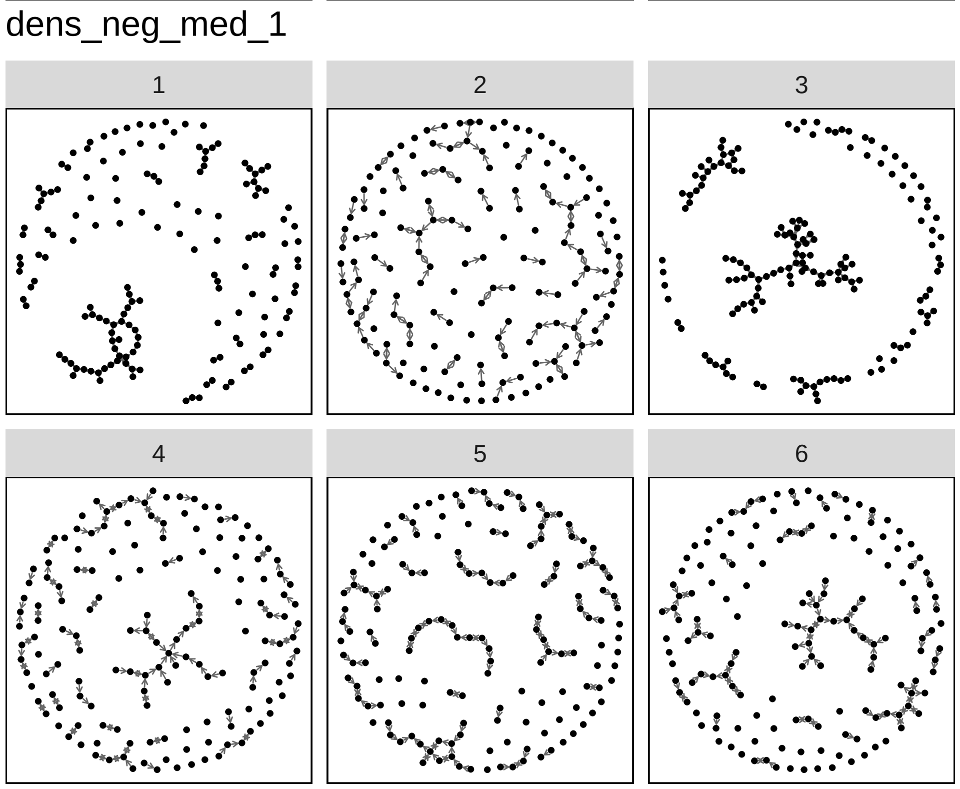
dens_neg_hard_1



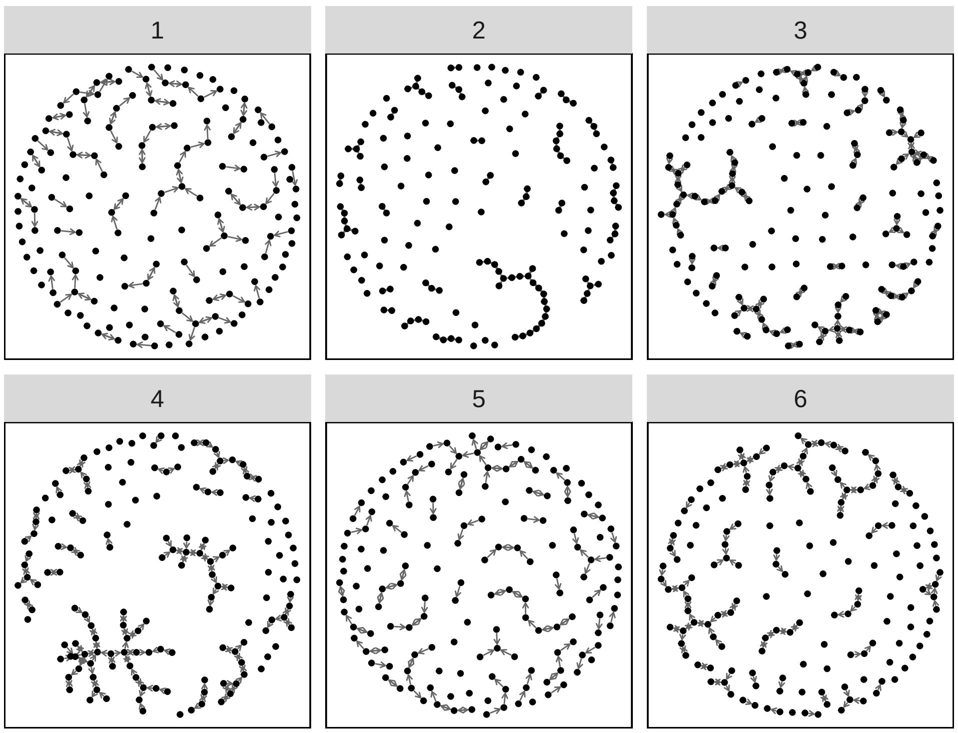
dens_neg_hard_2



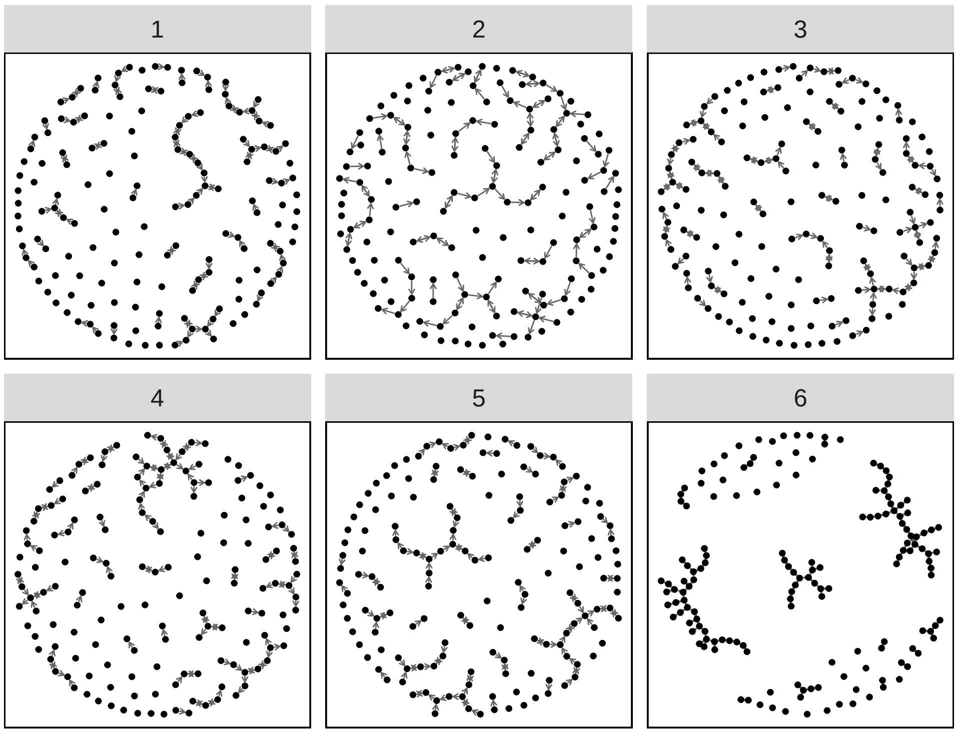

dens_neg_hard_3
1
2
3
4
5
6

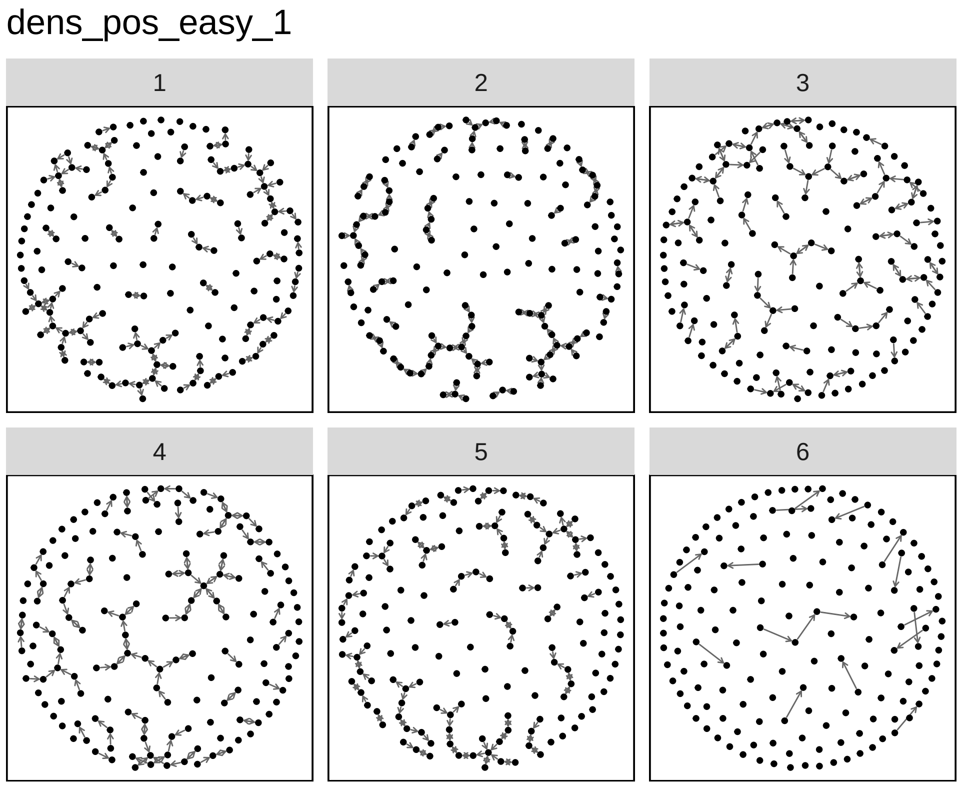

dens_neg_med_1
1
2
3
4
5
6

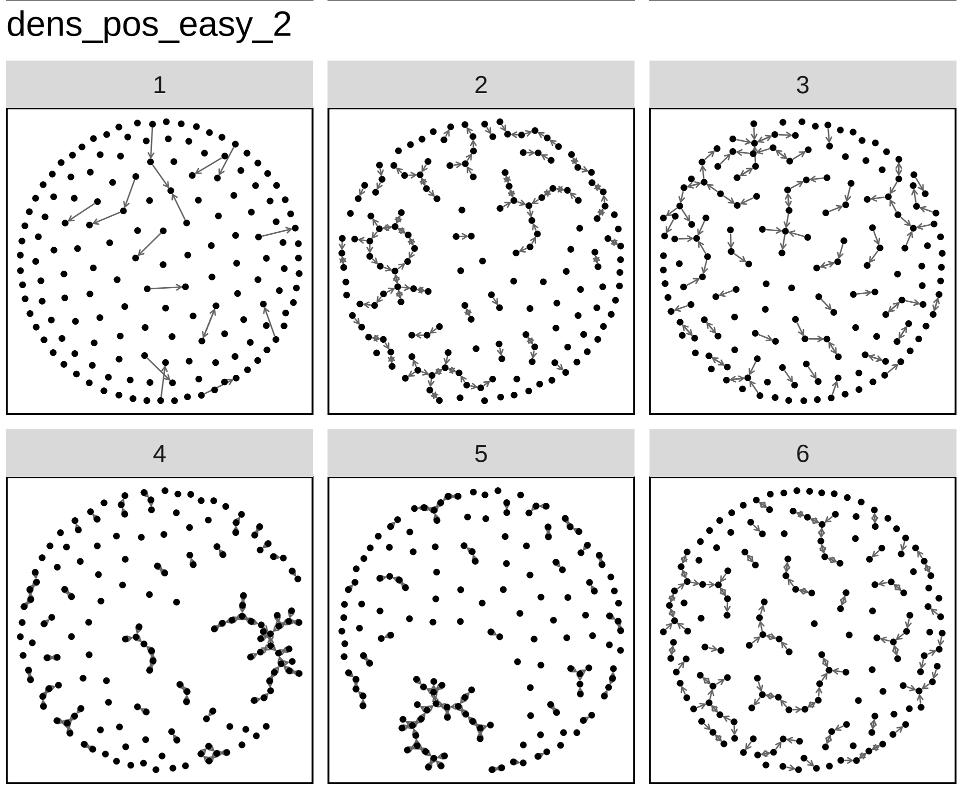
dens_neg_med_2



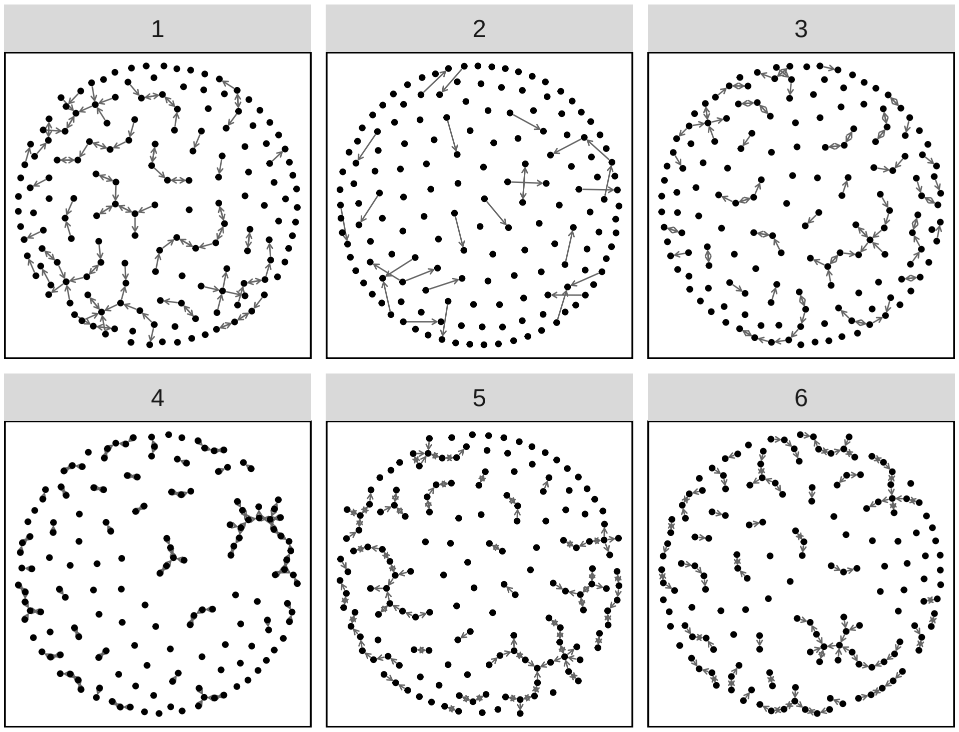
dens_neg_med_3



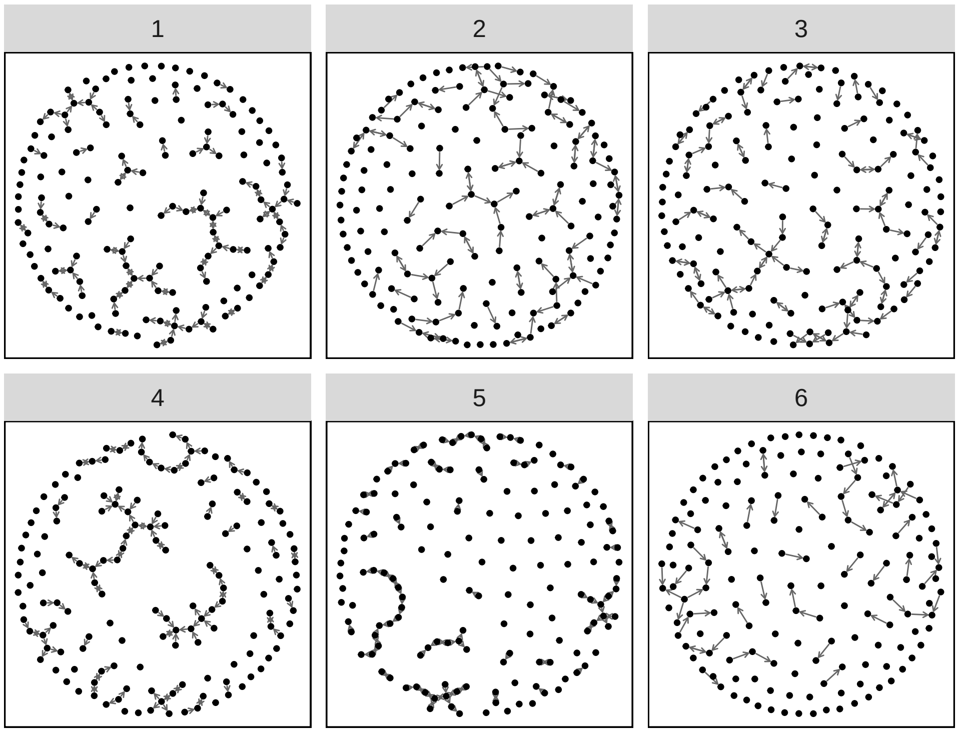

dens_pos_easy_1
1
2
3
4
5
6


dens_pos_easy_2
1
2
3
4
5
6

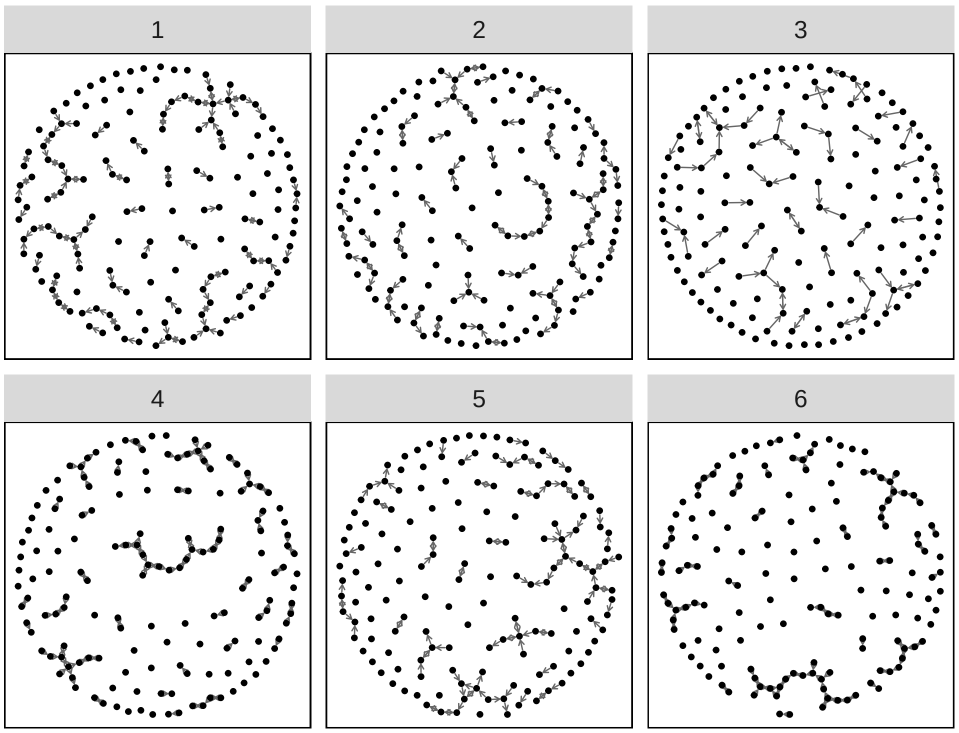
dens_pos_easy_3



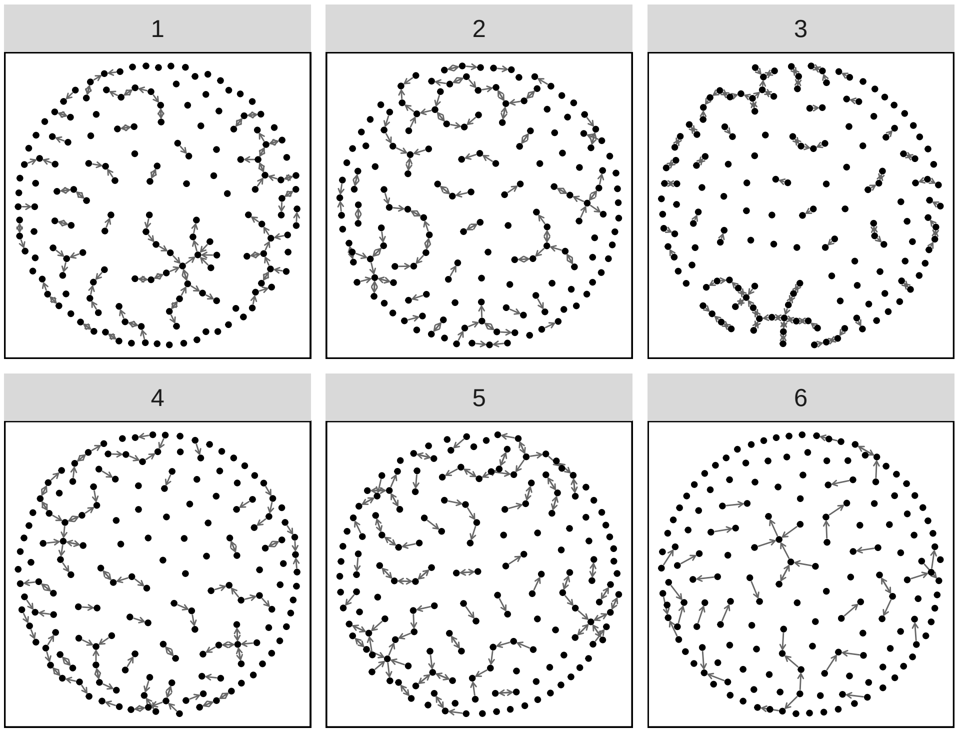
dens_pos_hard_1



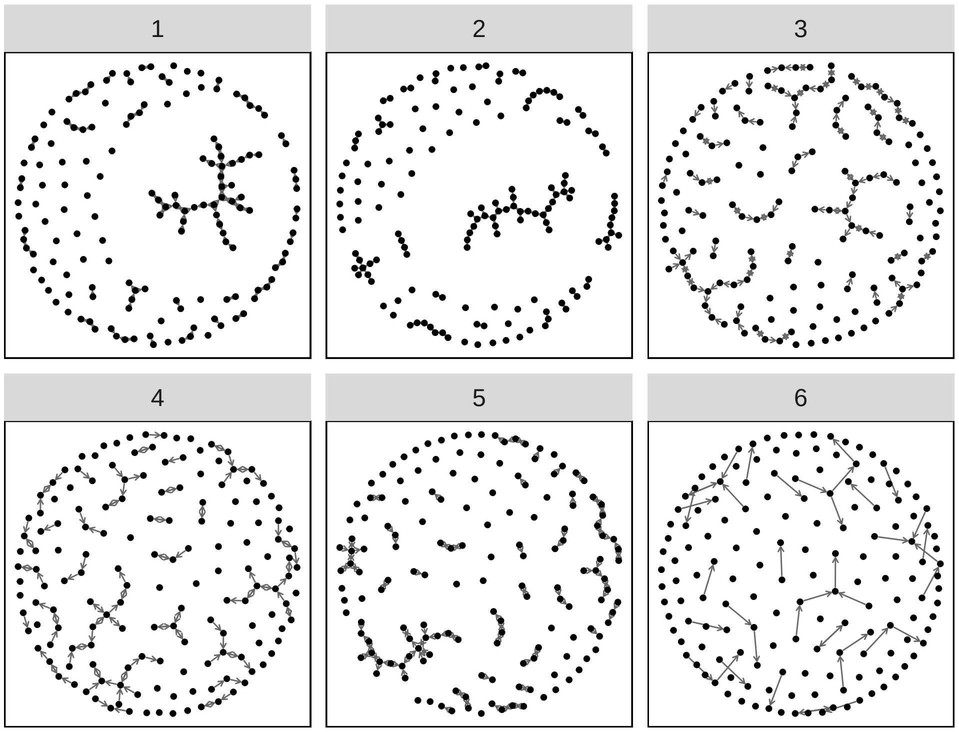
dens_pos_hard_2



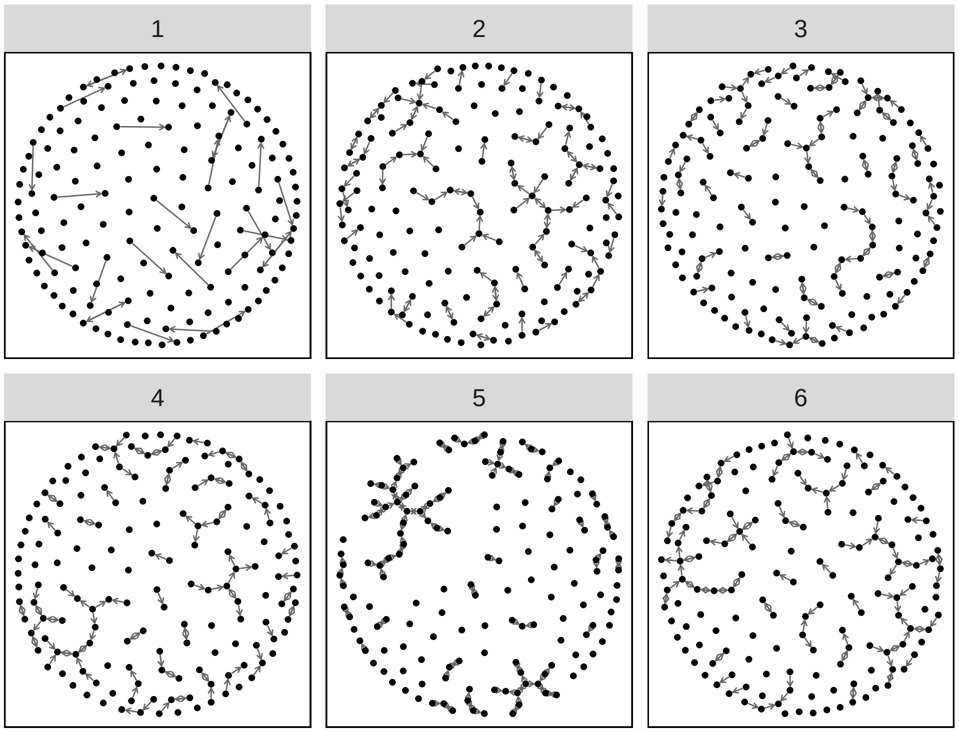
dens_pos_hard_3

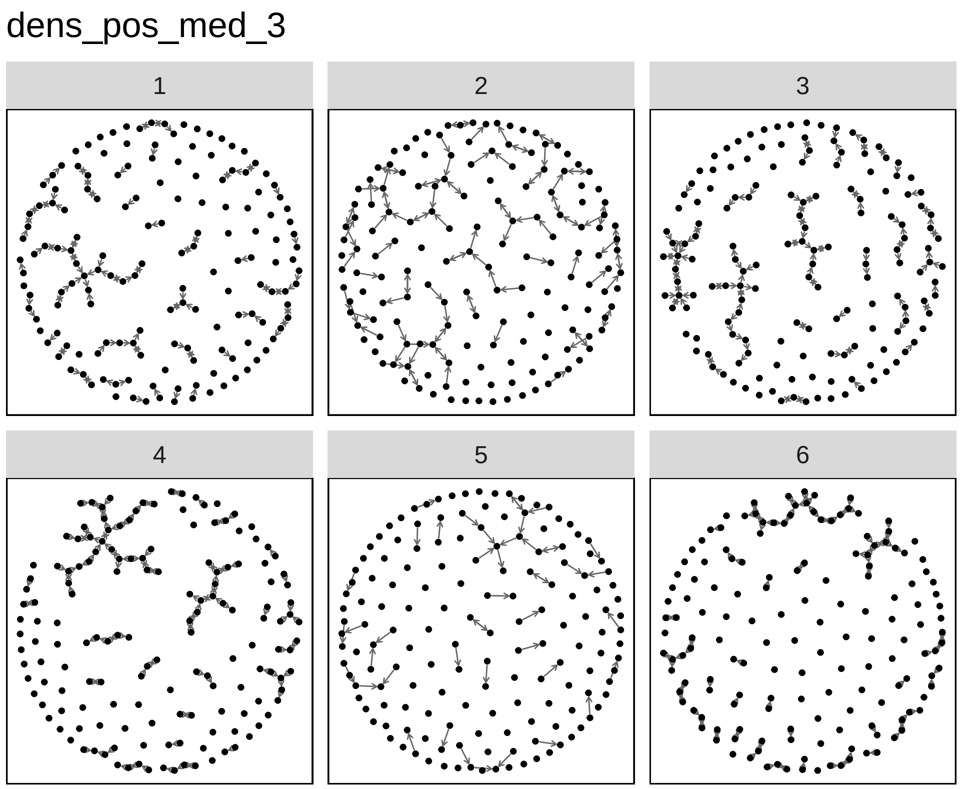


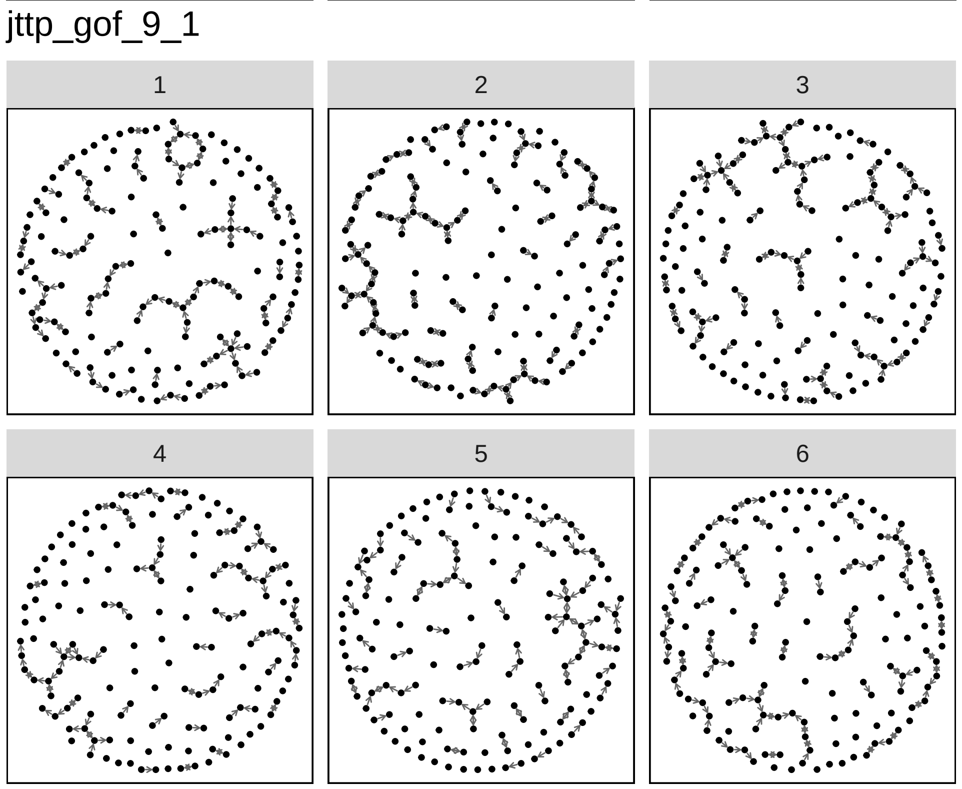
dens_pos_med_1



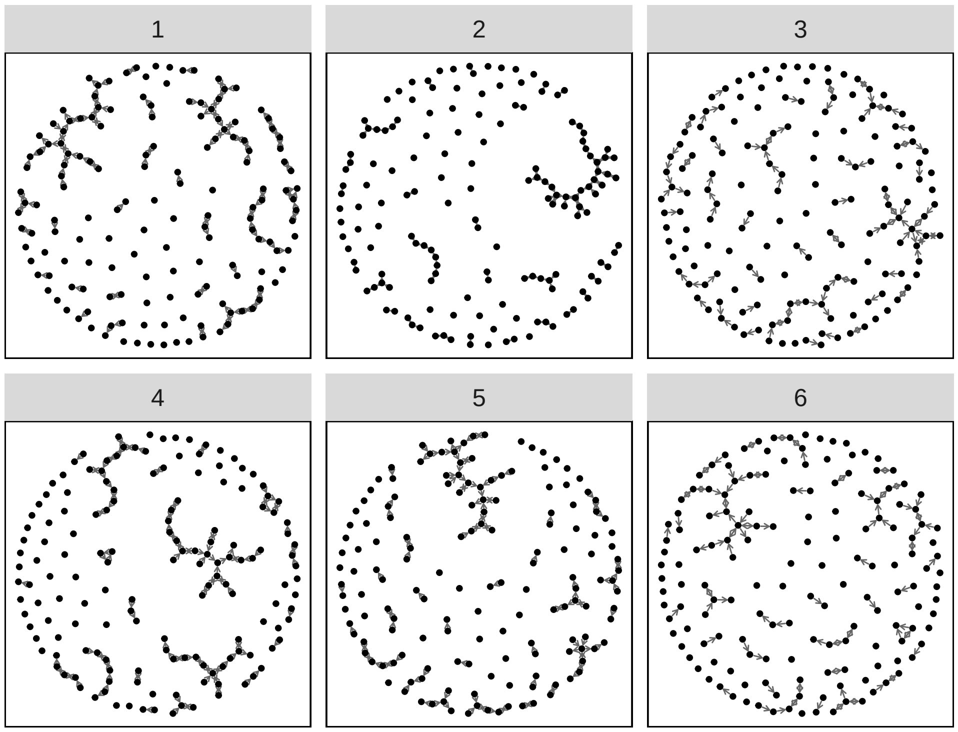
dens_pos_med_2



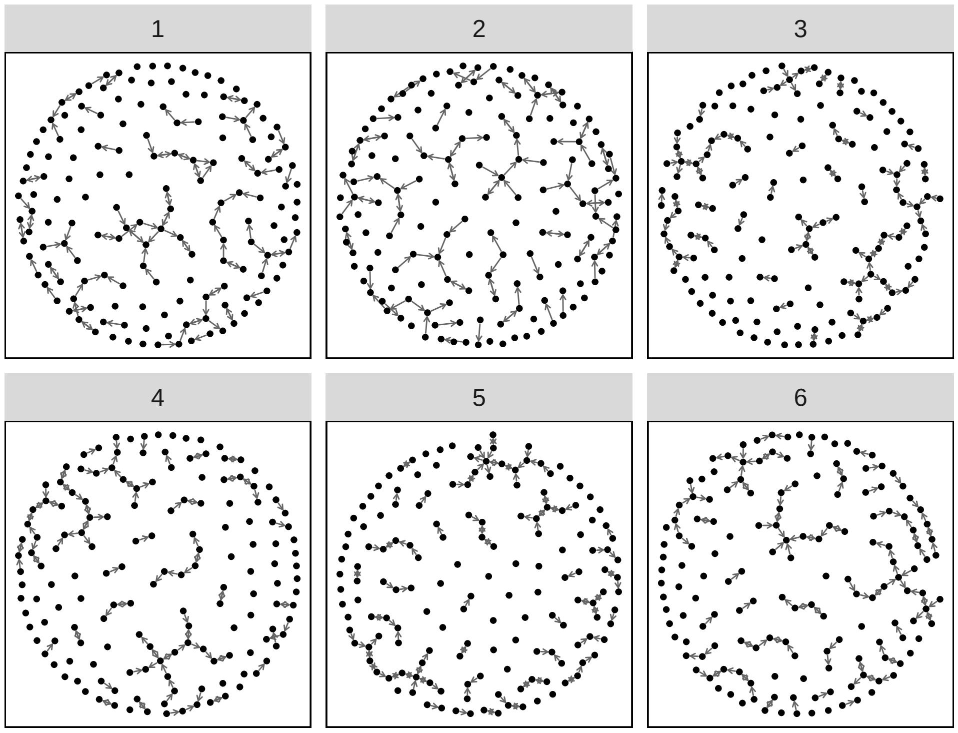

dens_pos_med_3
1
2
3
4
5
6


jttp_gof_9_1
1
2
3
4
5
6

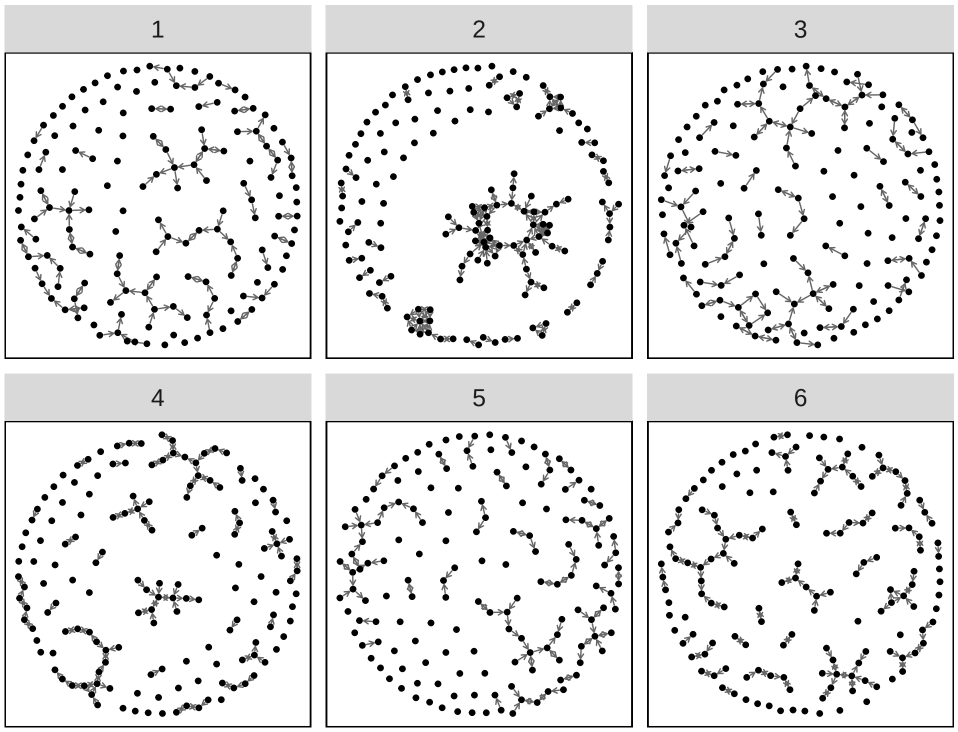
jttp_gof_9_2



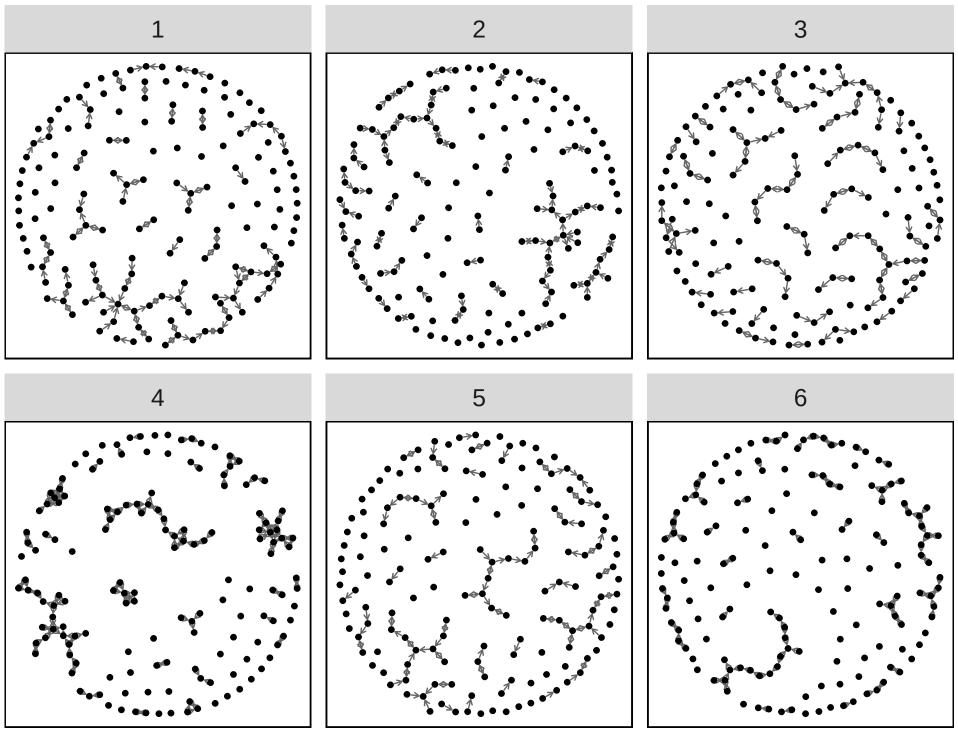
jttp_gof_9_3



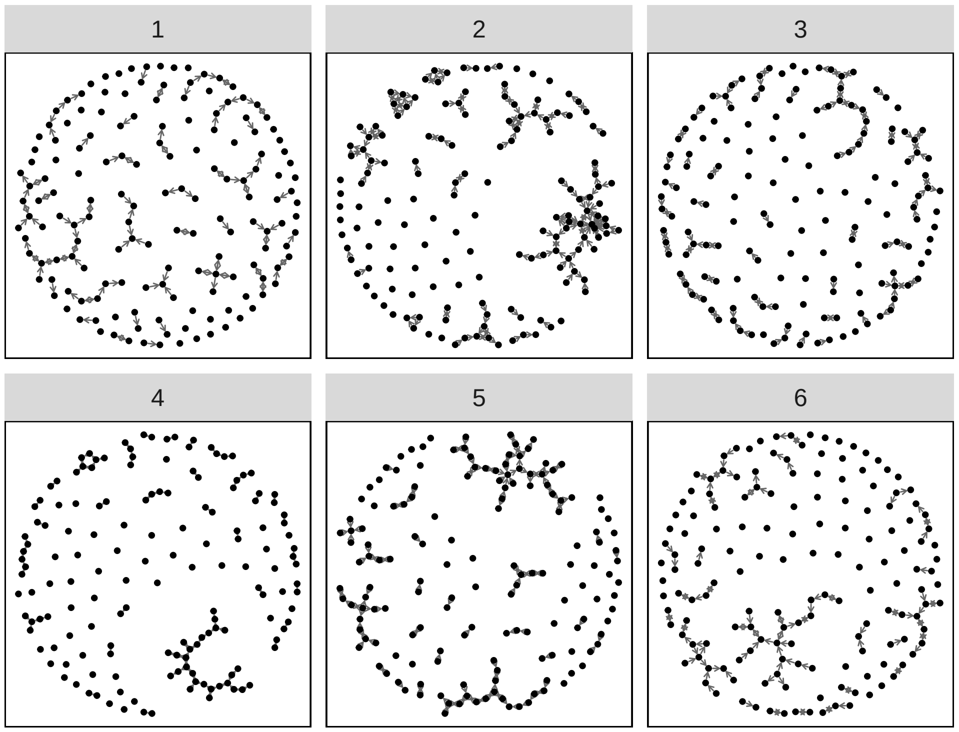
jttp_neg_easy_1



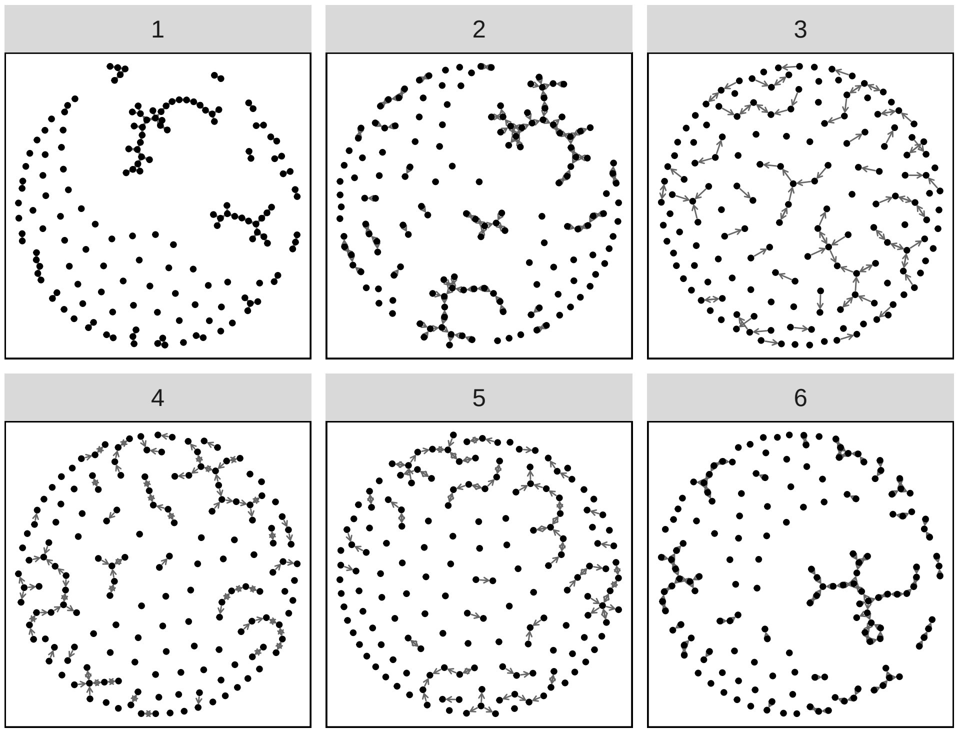
jttp_neg_easy_2



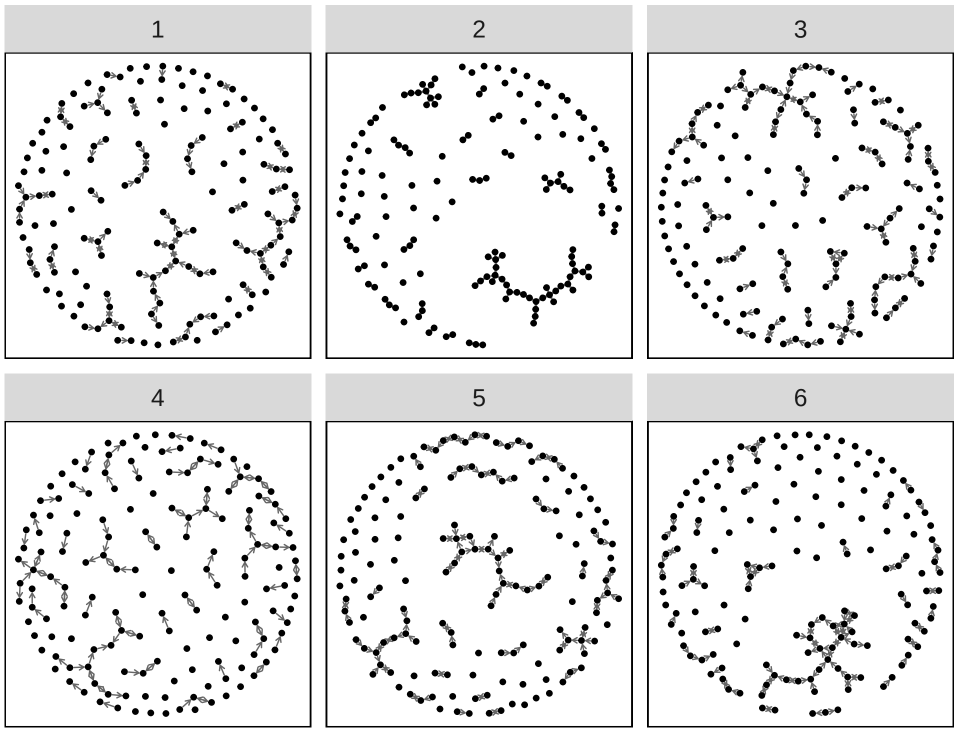
jttp_neg_easy_3



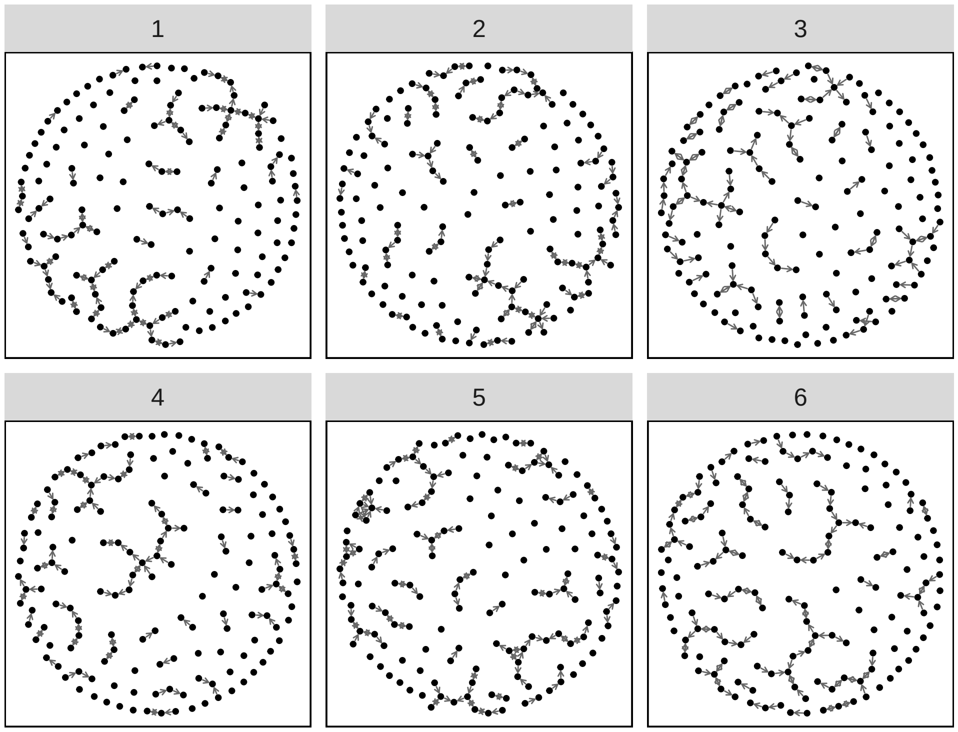
jttp_neg_hard_1

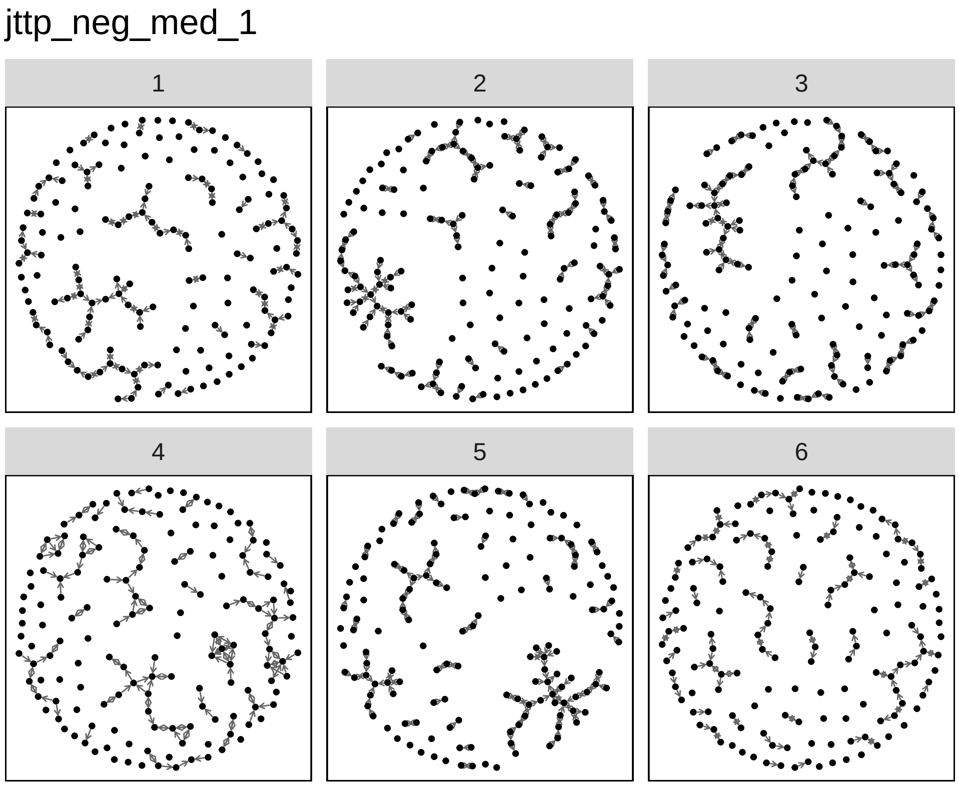


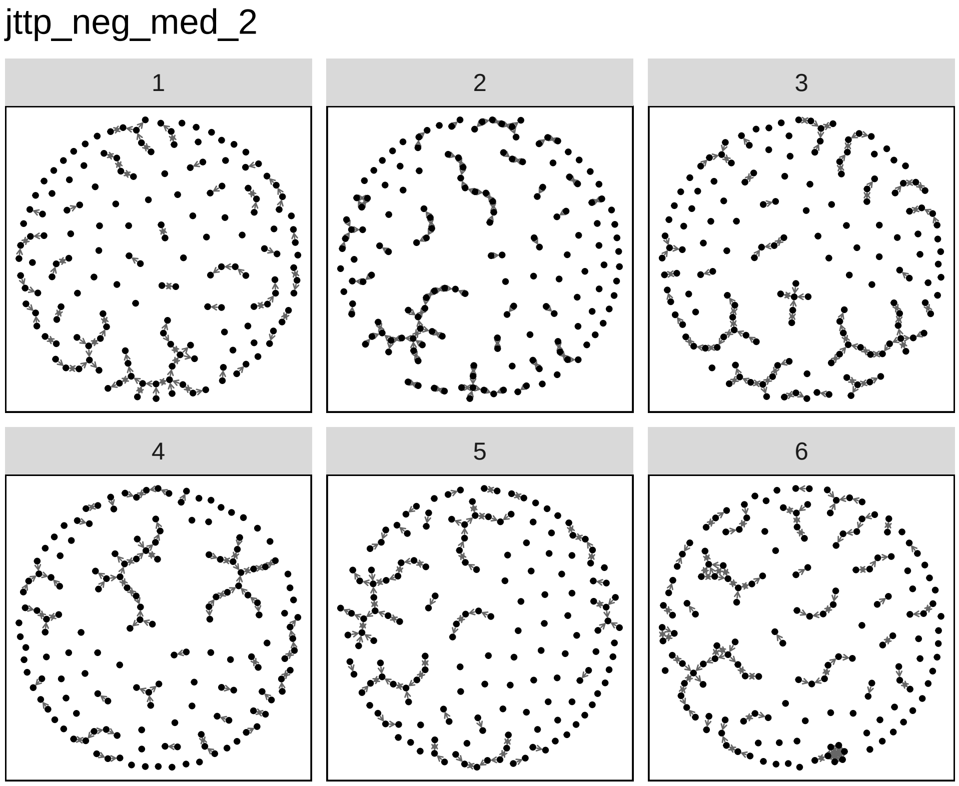
jttp_neg_hard_2



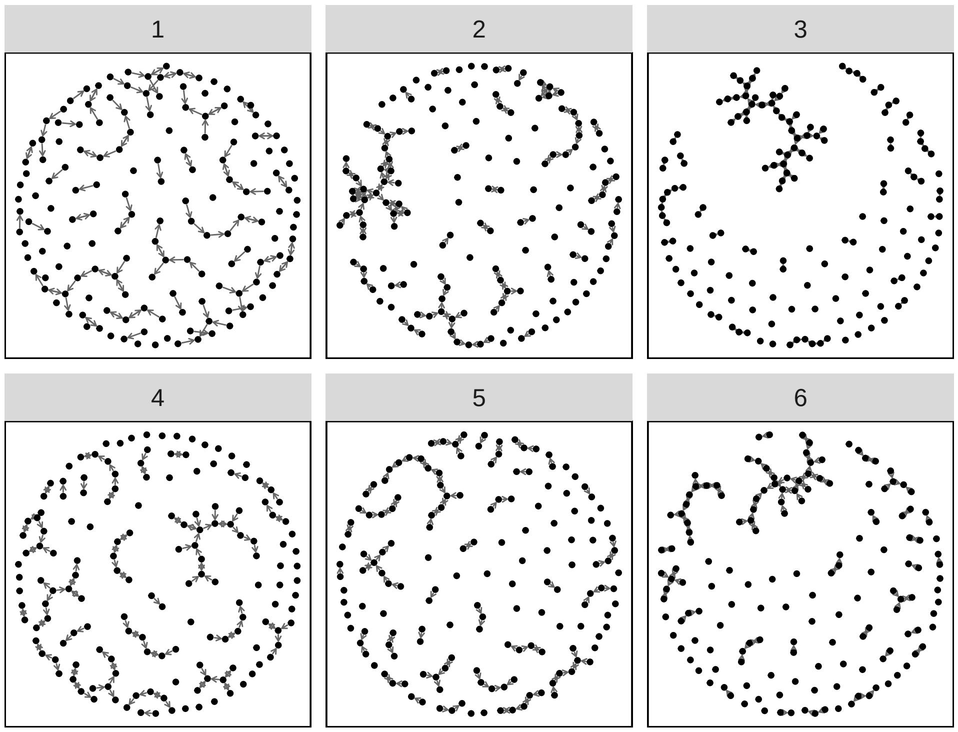
jttp_neg_hard_3



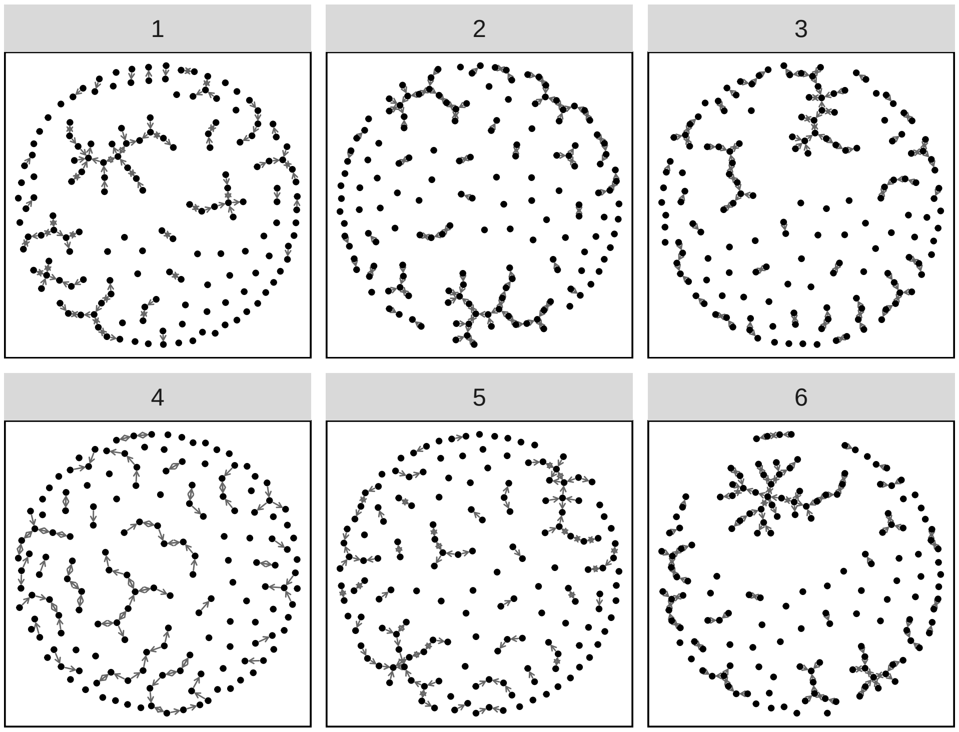

jttp_neg_med_1
1
2
3
4
5
6


jttp_neg_med_2
1
2
3
4
5
6

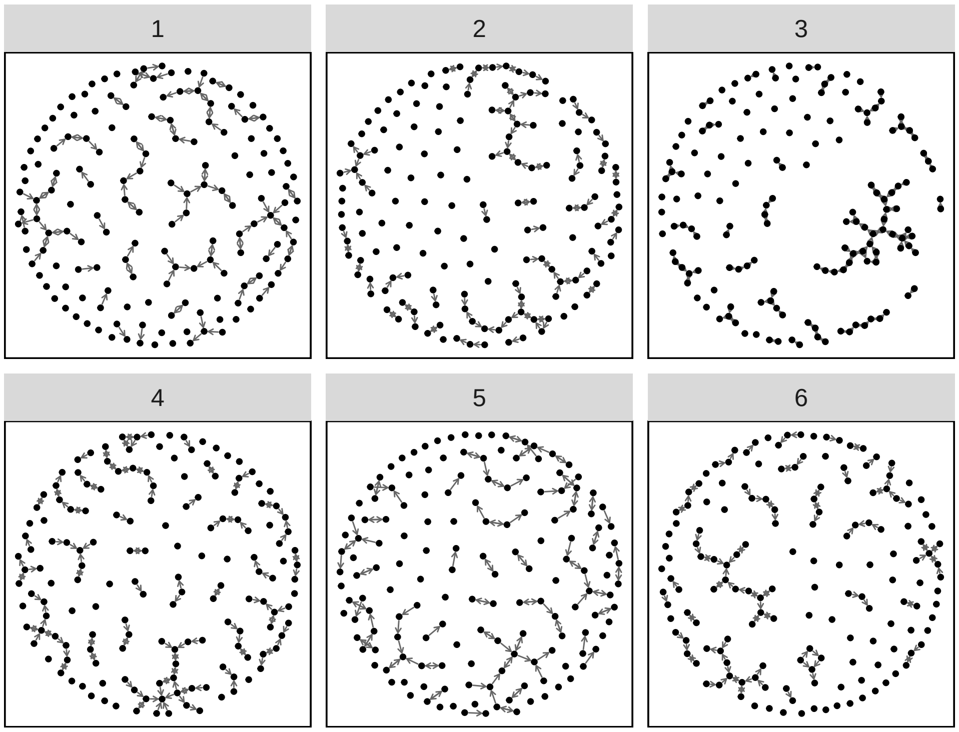
jttp_neg_med_3



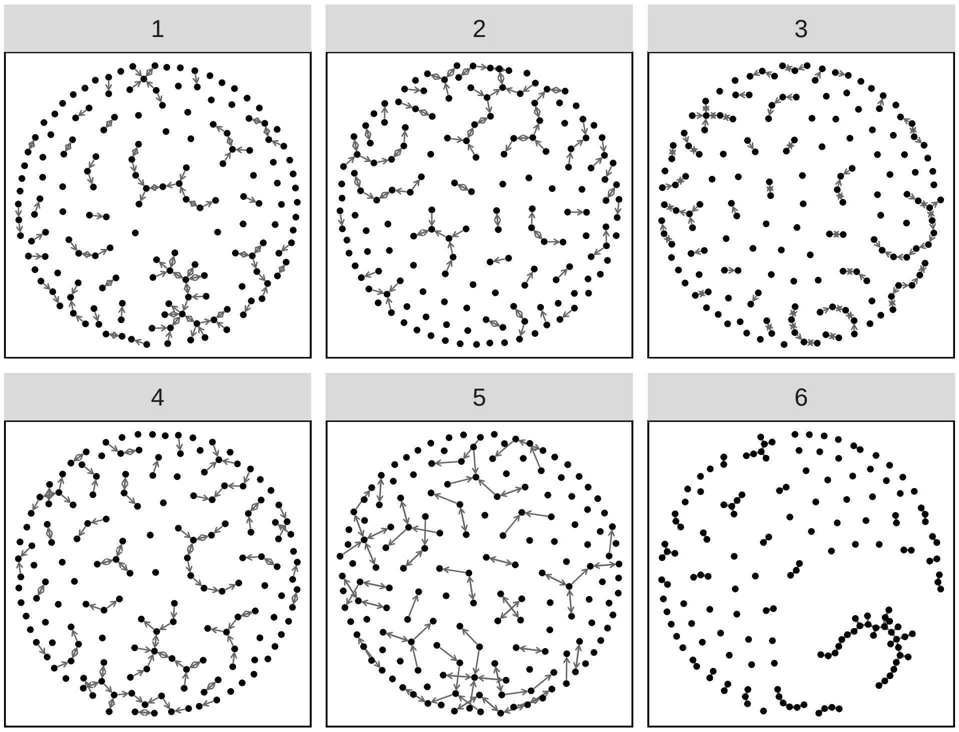
jttp_pos_easy_1



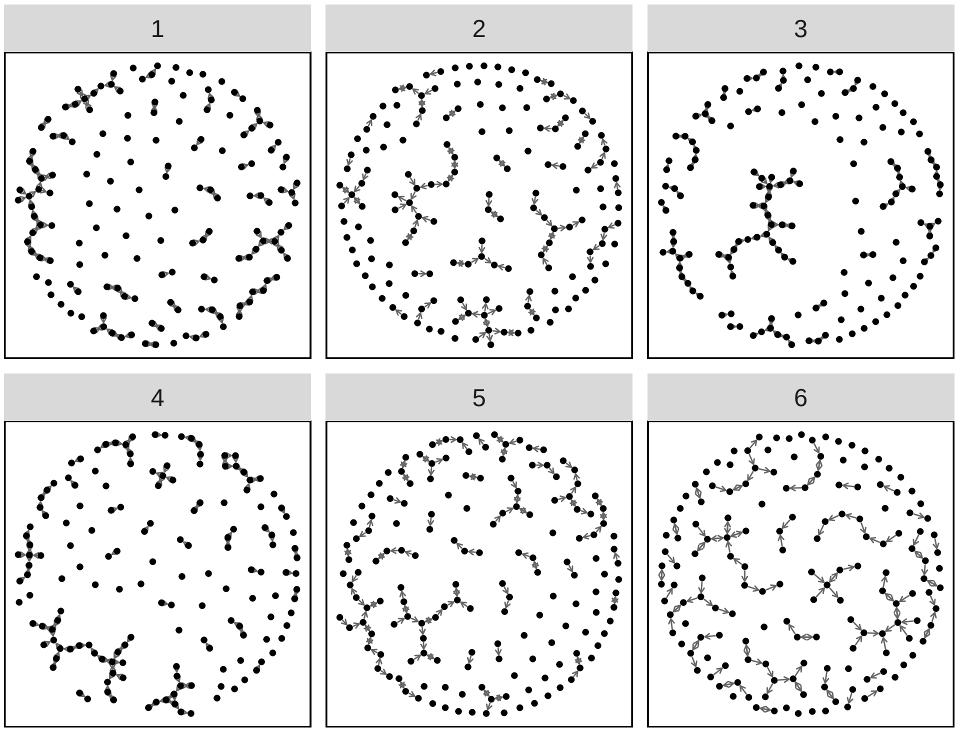
jttp_pos_easy_2



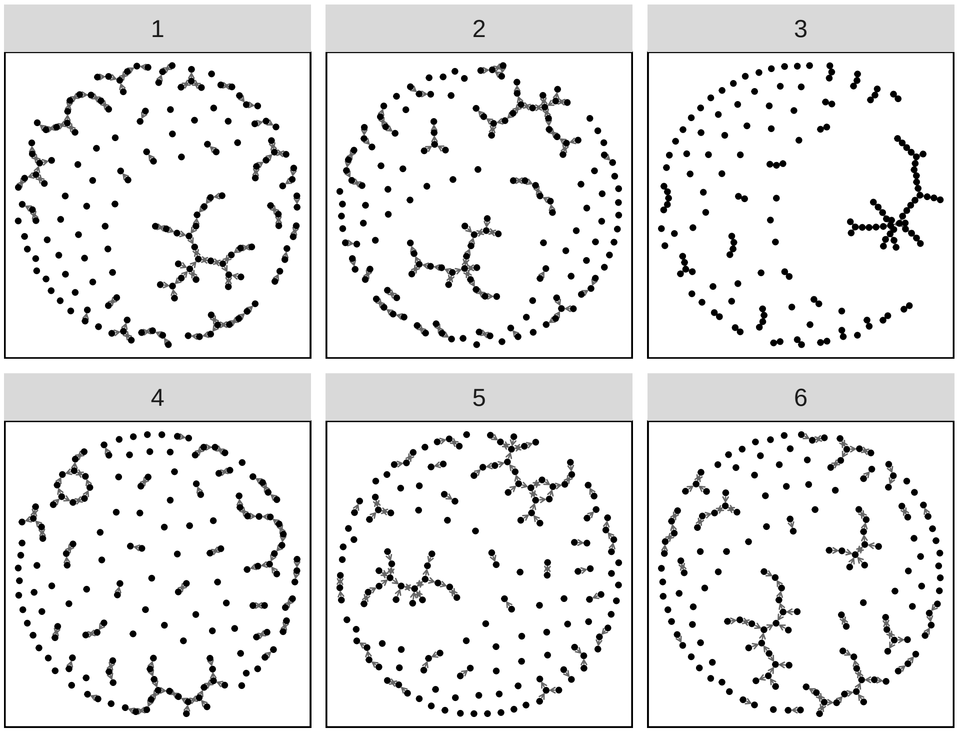
jttp_pos_easy_3

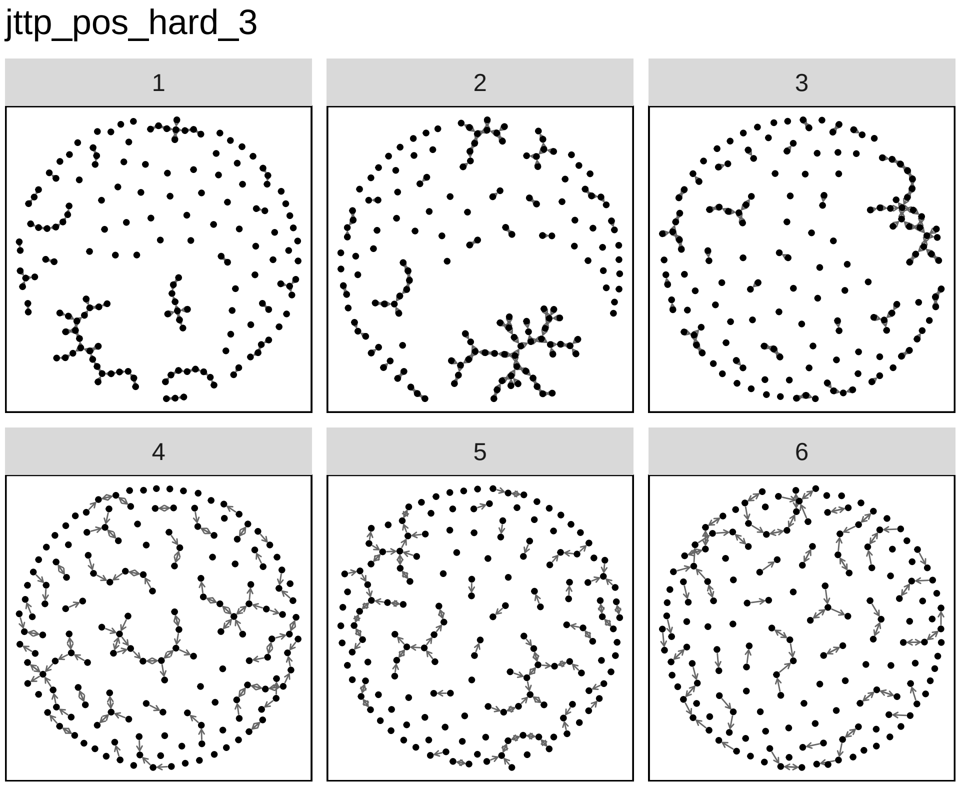


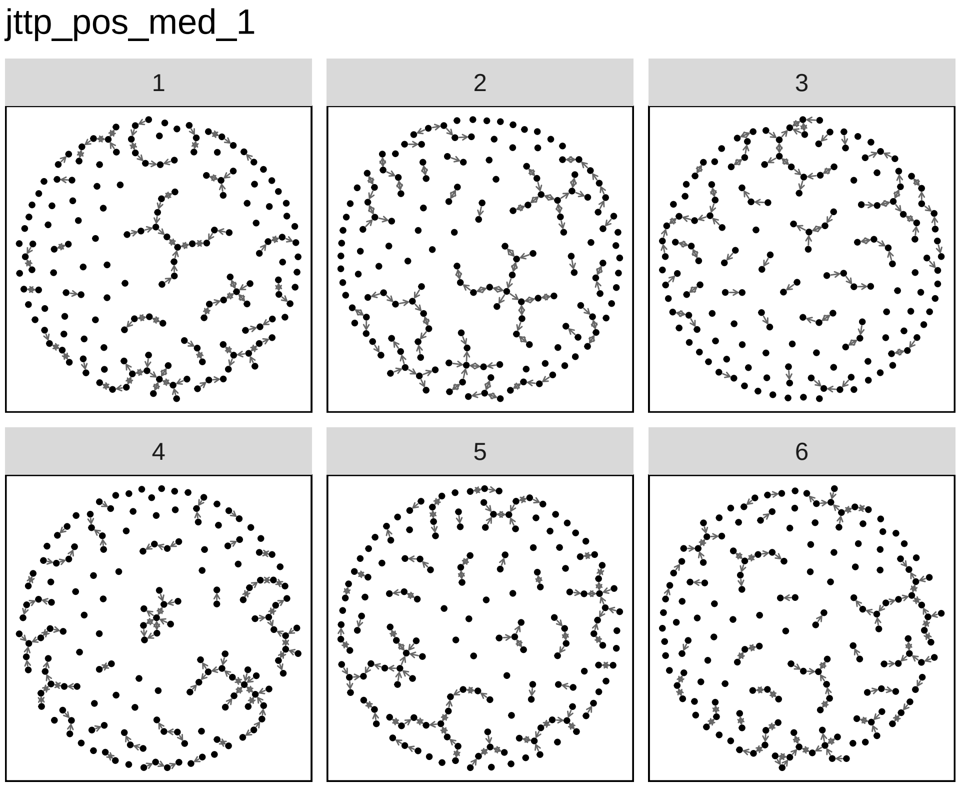
jttp_pos_hard_1



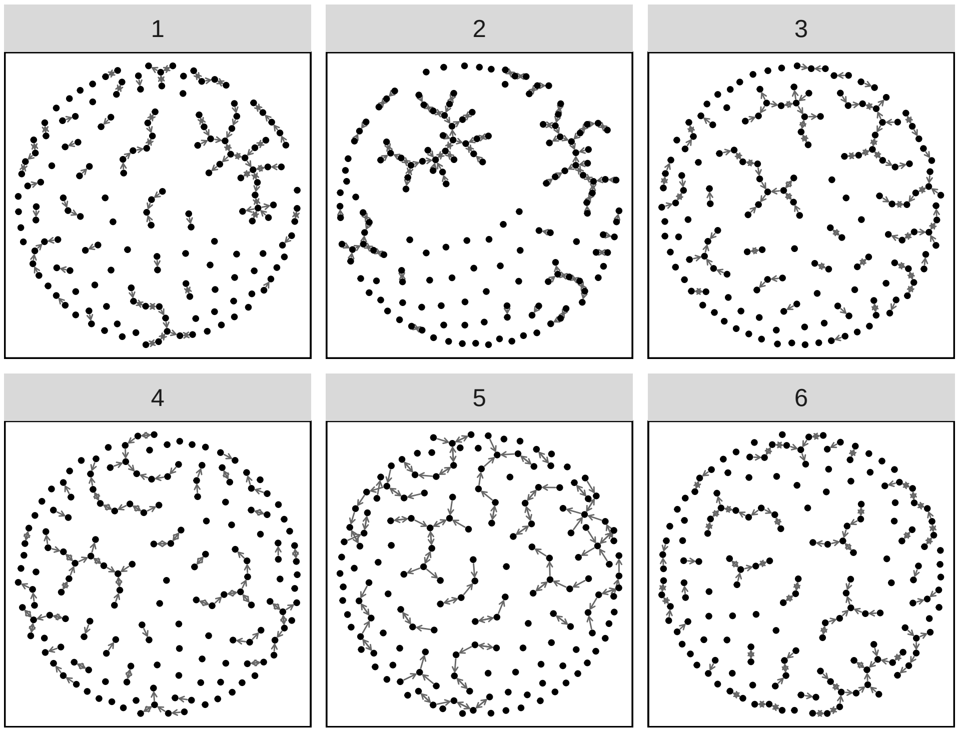
jttp_pos_hard_2



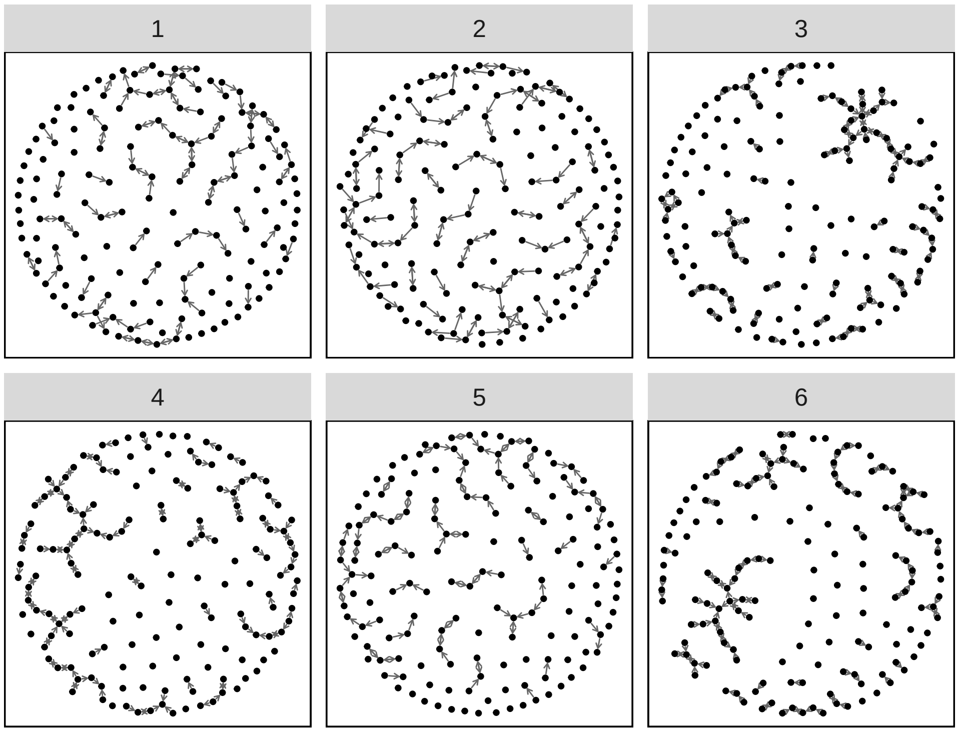

jttp_pos_hard_3
1
2
3
4
5
6


jttp_pos_med_1
1
2
3
4
5
6

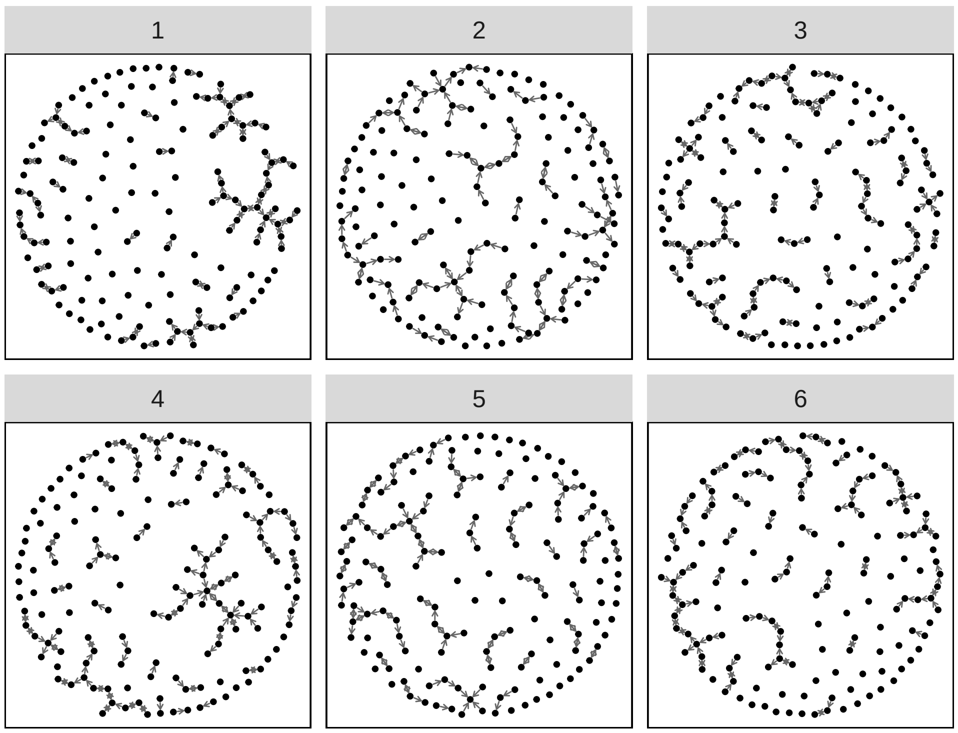
jttp_pos_med_2



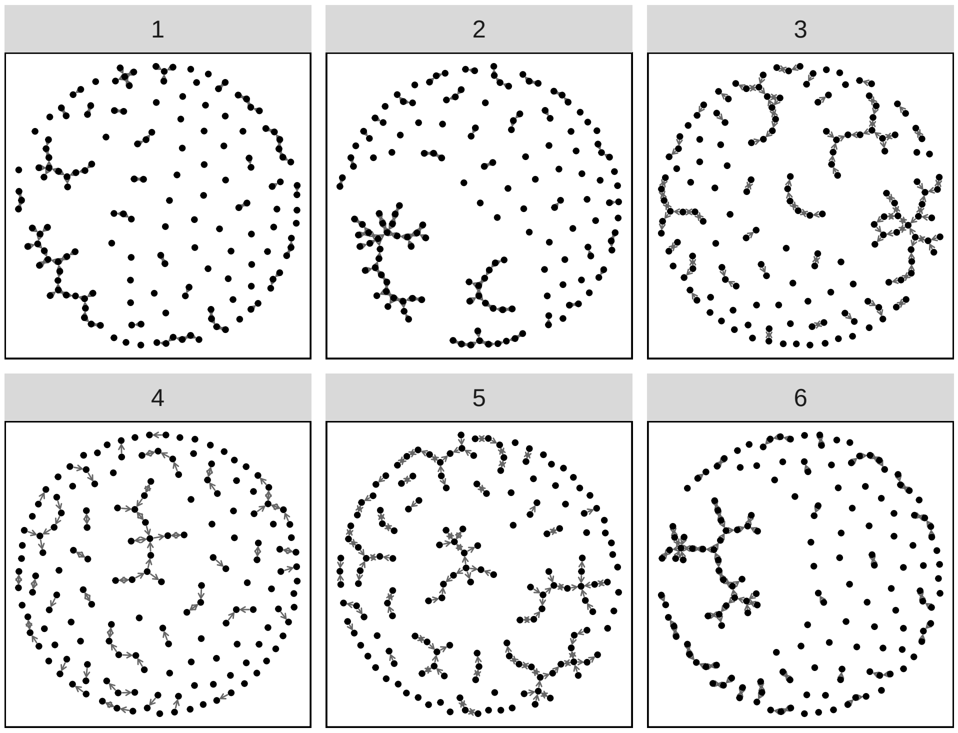
jttp_pos_med_3



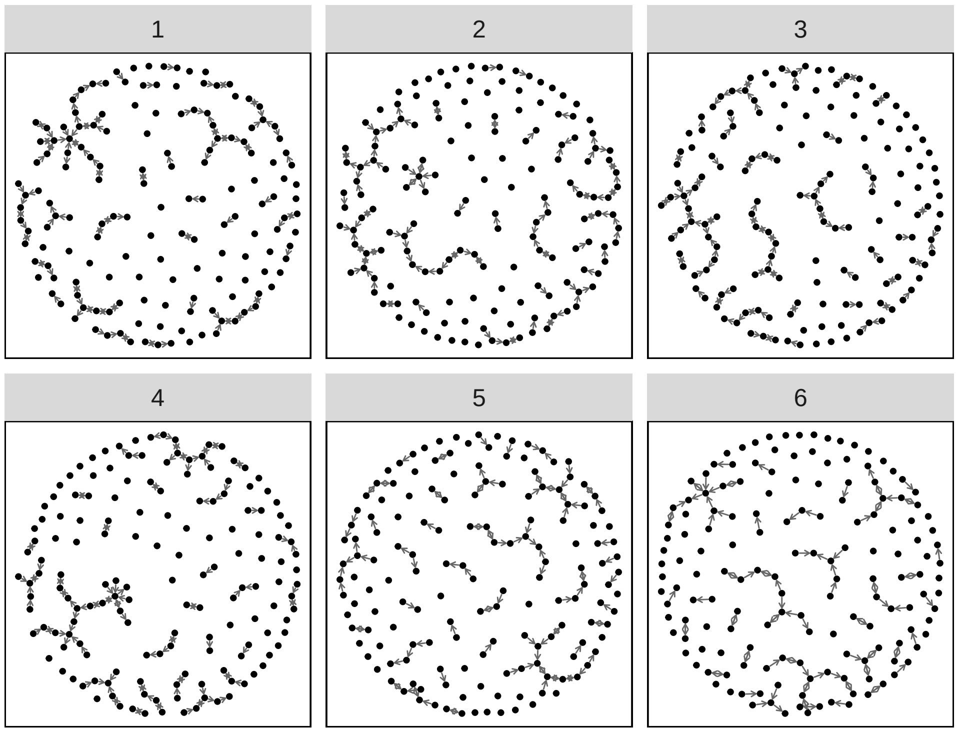
jtts_gof_9_1



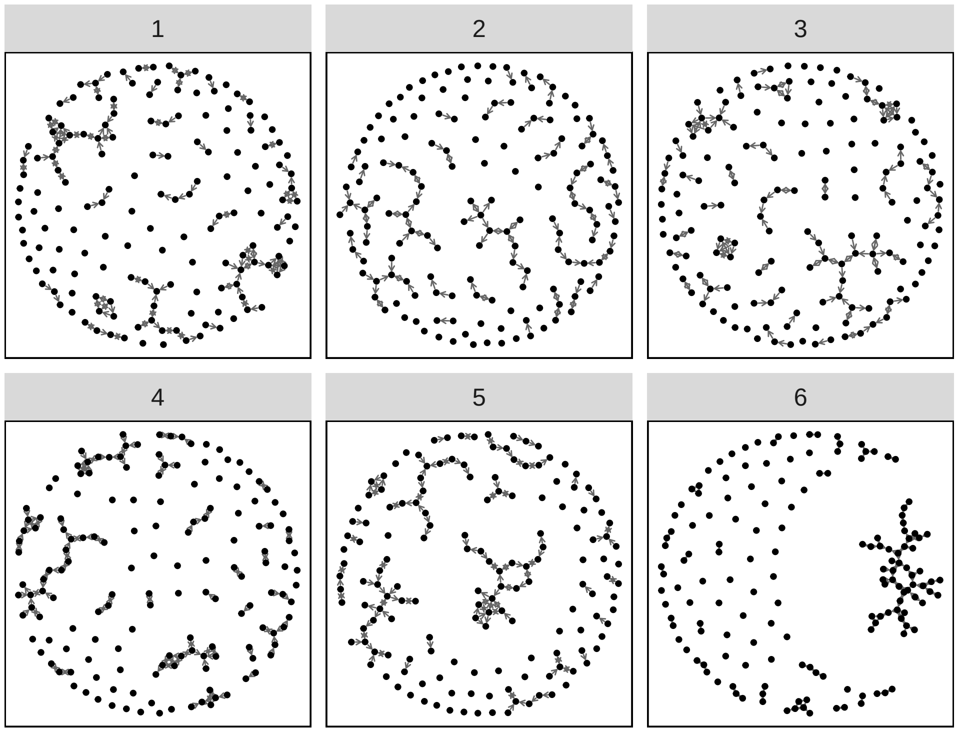
jtts_gof_9_2



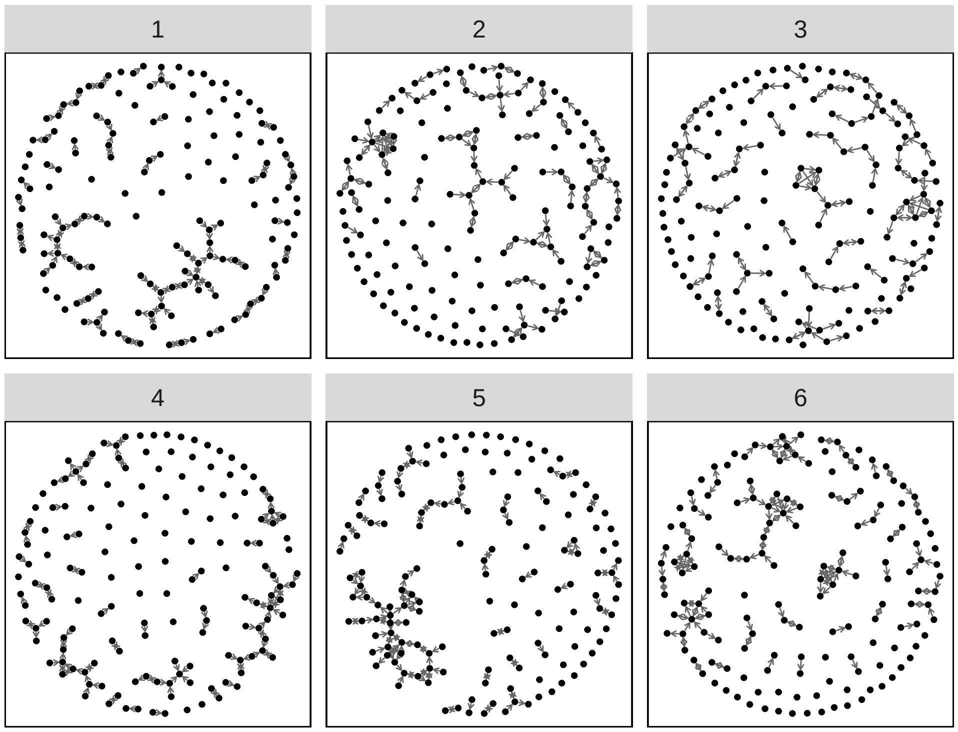
jtts_gof_9_3



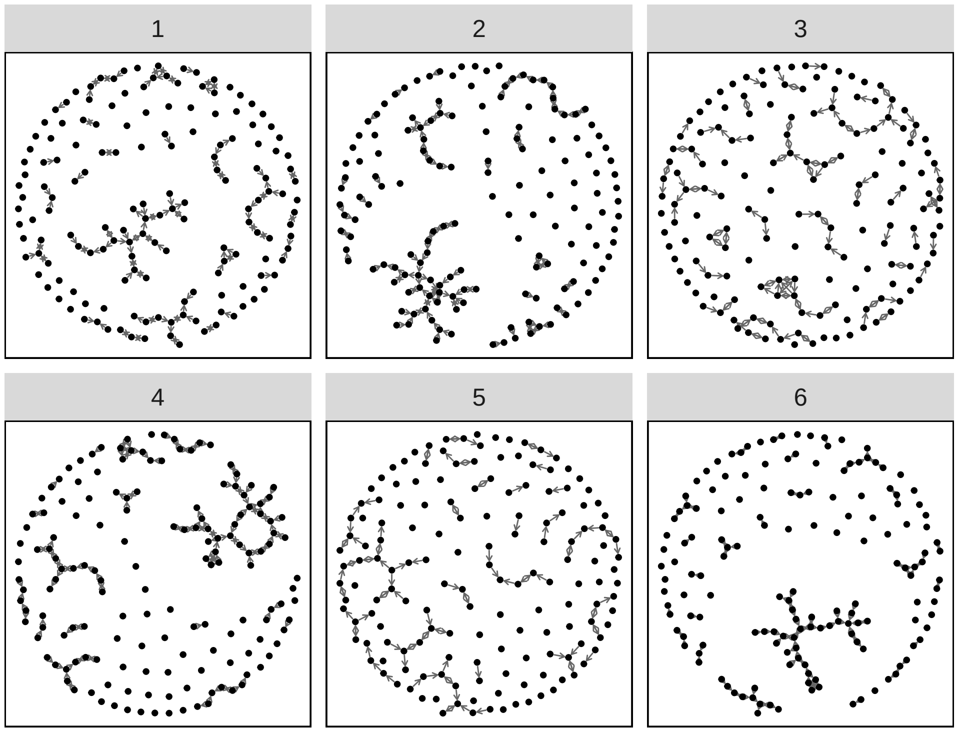
jtts_neg_easy_1



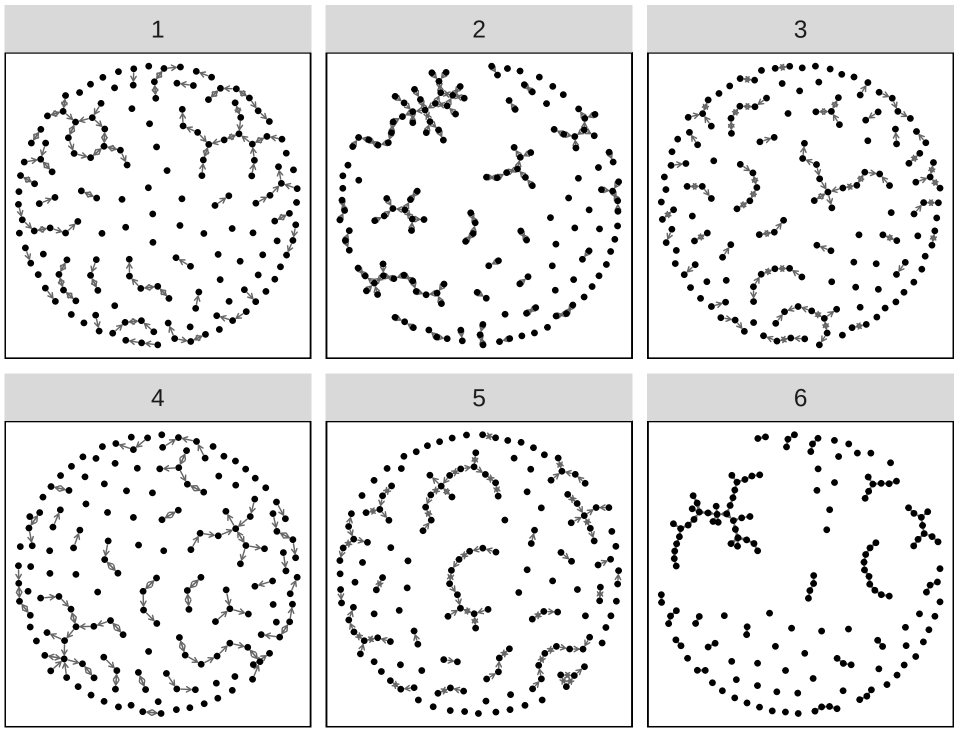
jtts_neg_easy_2



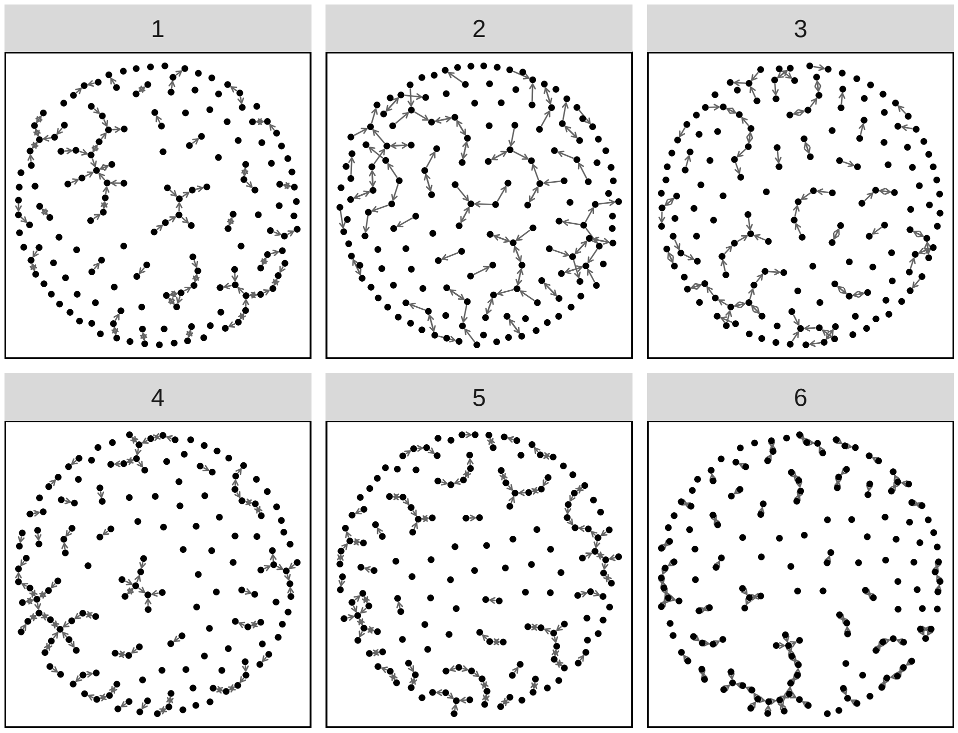
jtts_neg_easy_3

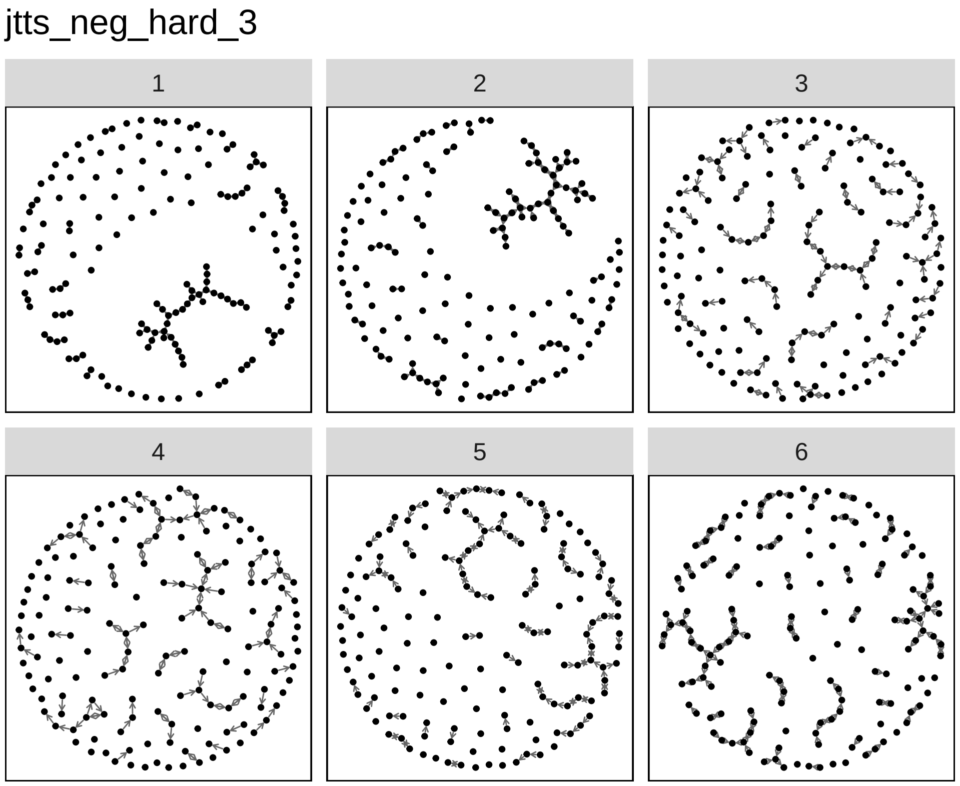


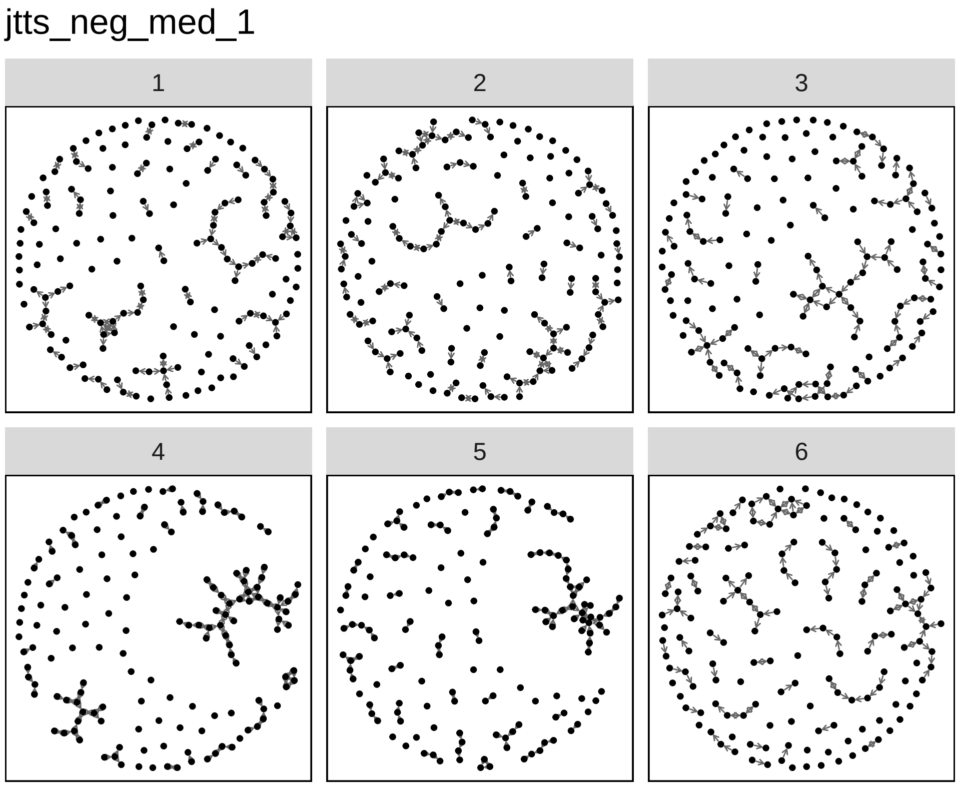
jtts_neg_hard_1



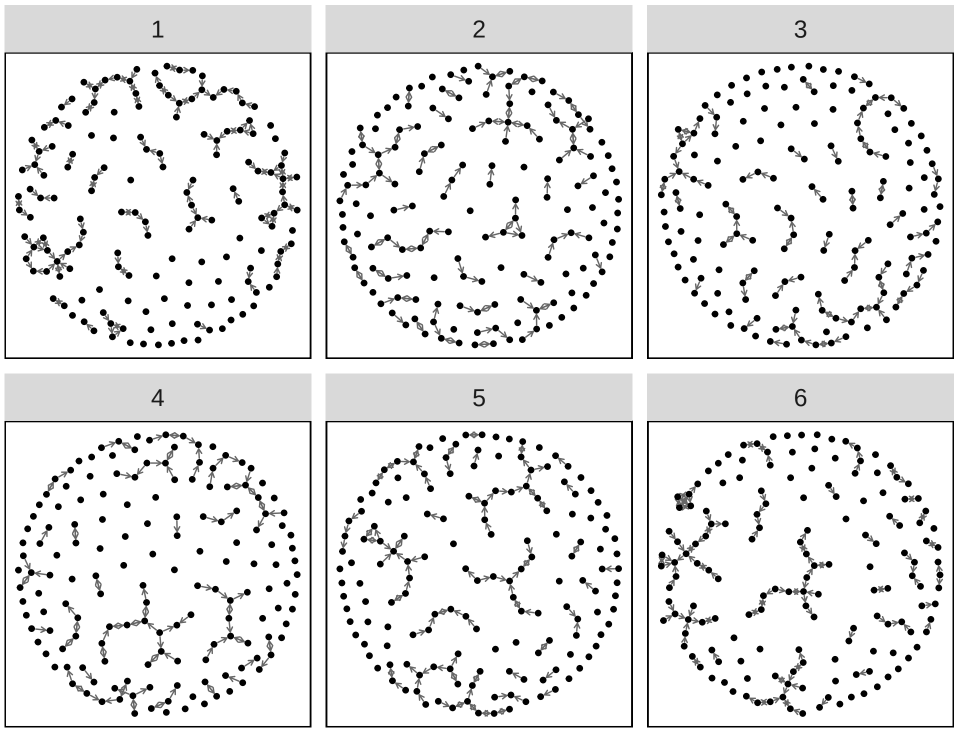
jtts_neg_hard_2



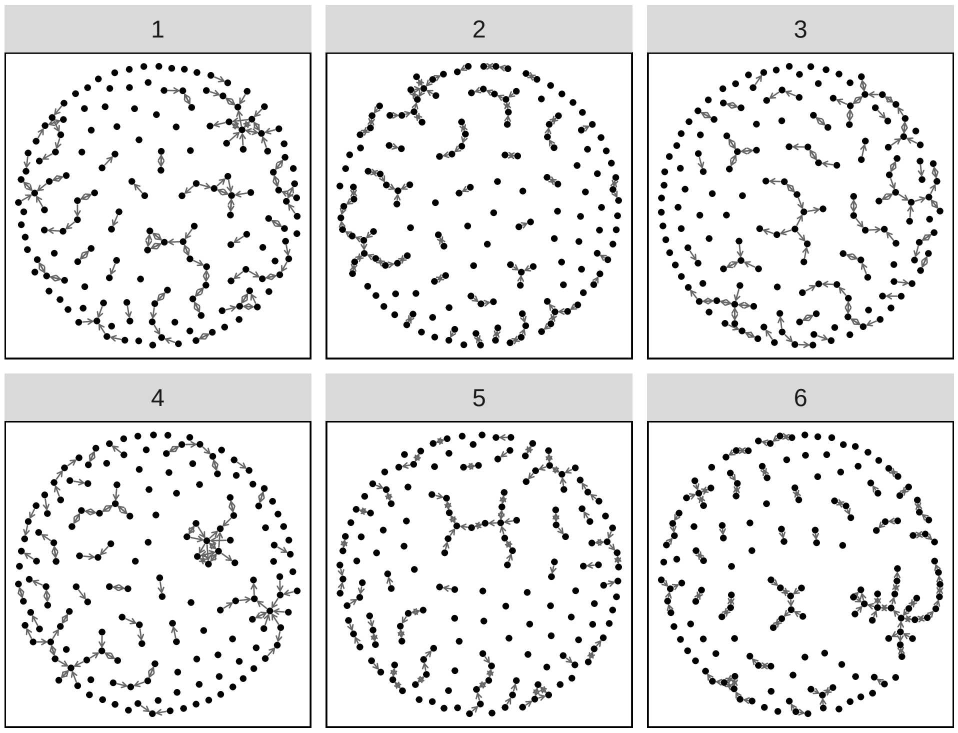

jtts_neg_hard_3
1
2
3
4
5
6


jtts_neg_med_1
1
2
3
4
5
6

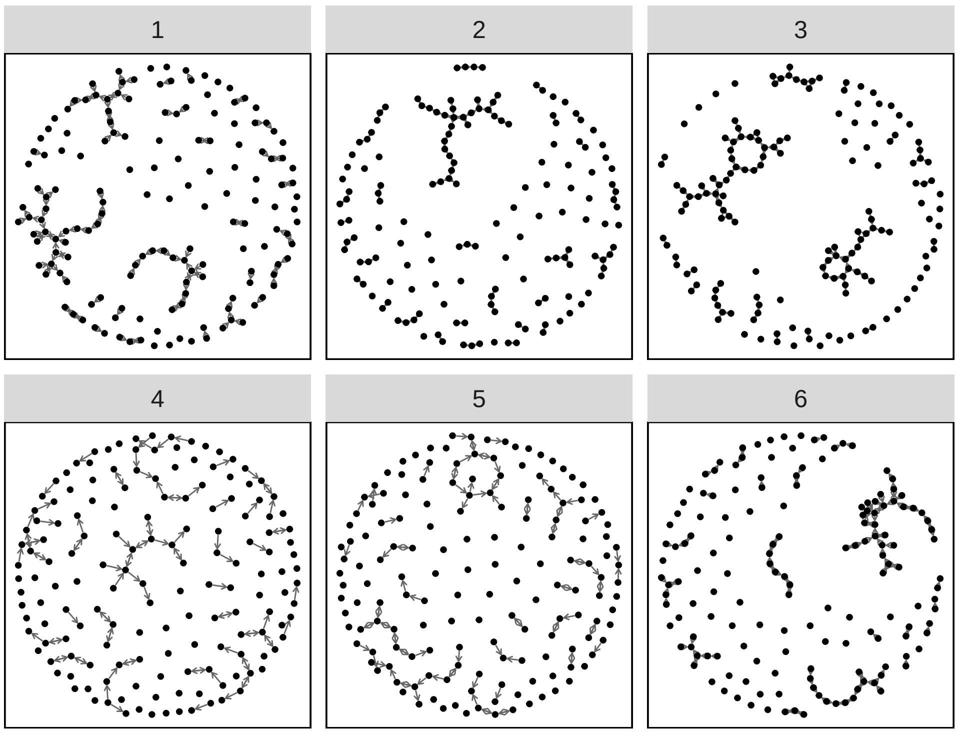
jtts_neg_med_2



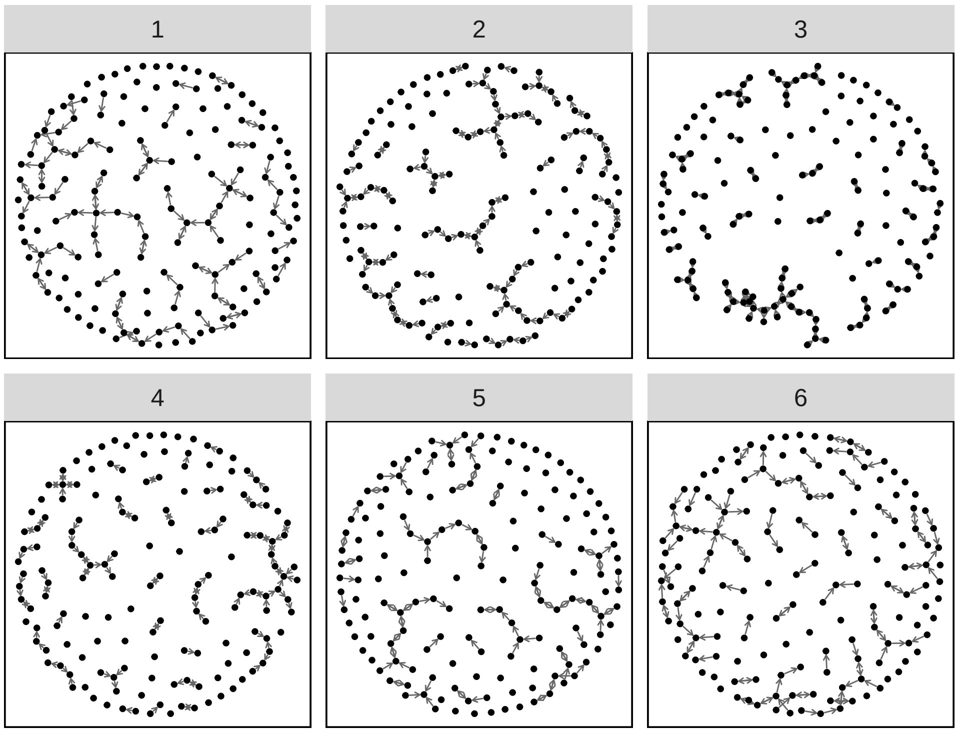
jtts_neg_med_3



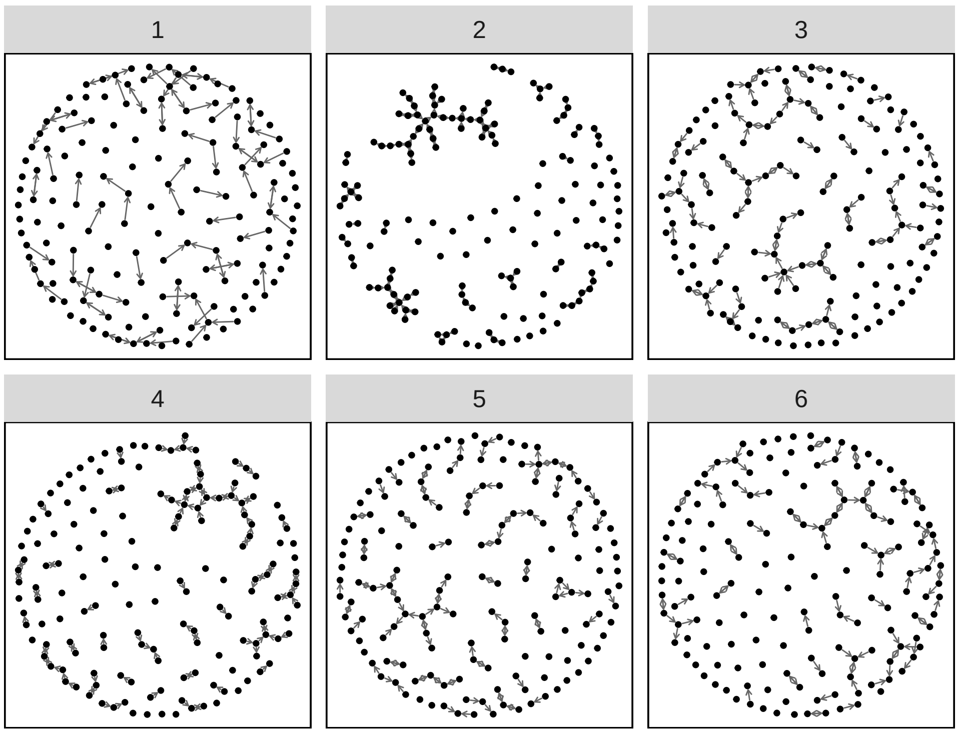
jtts_pos_easy_1



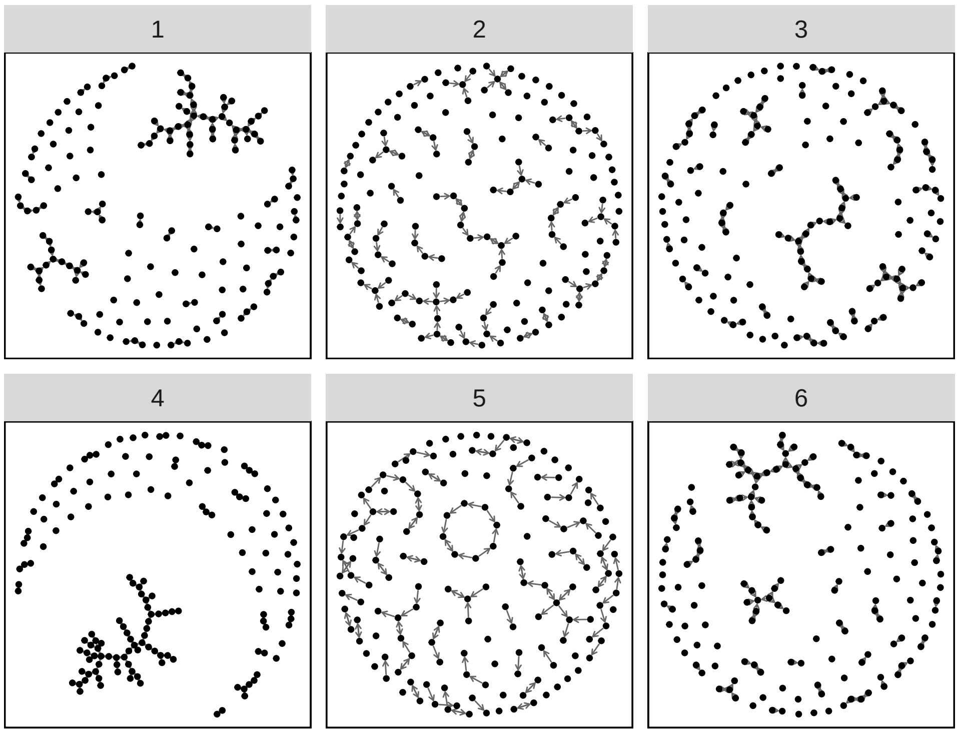
jtts_pos_easy_2



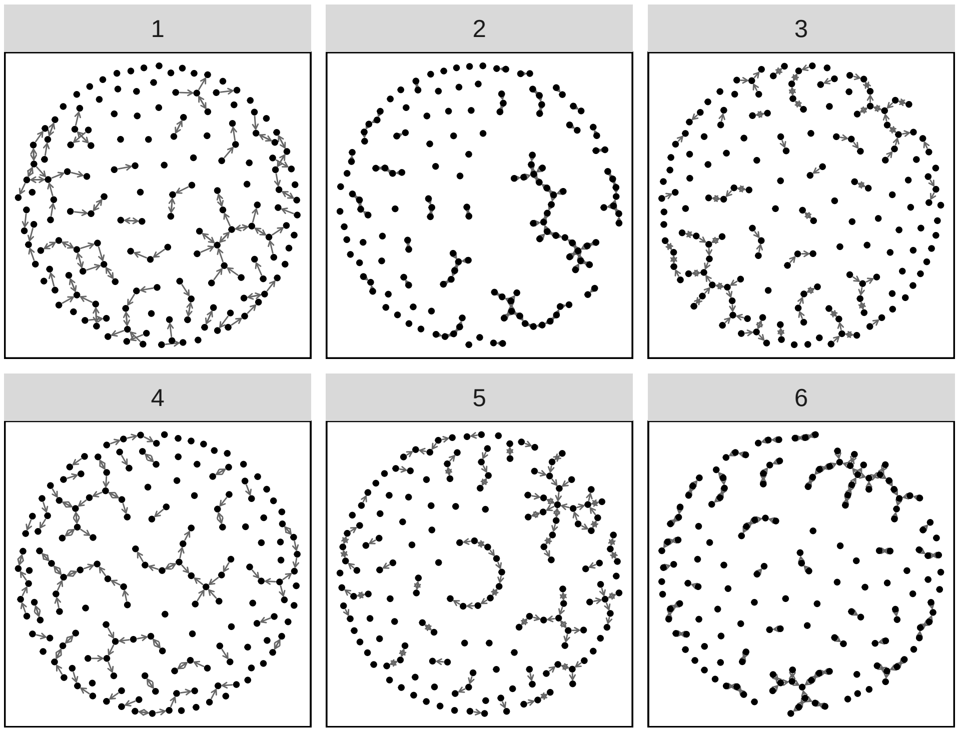
jtts_pos_easy_3



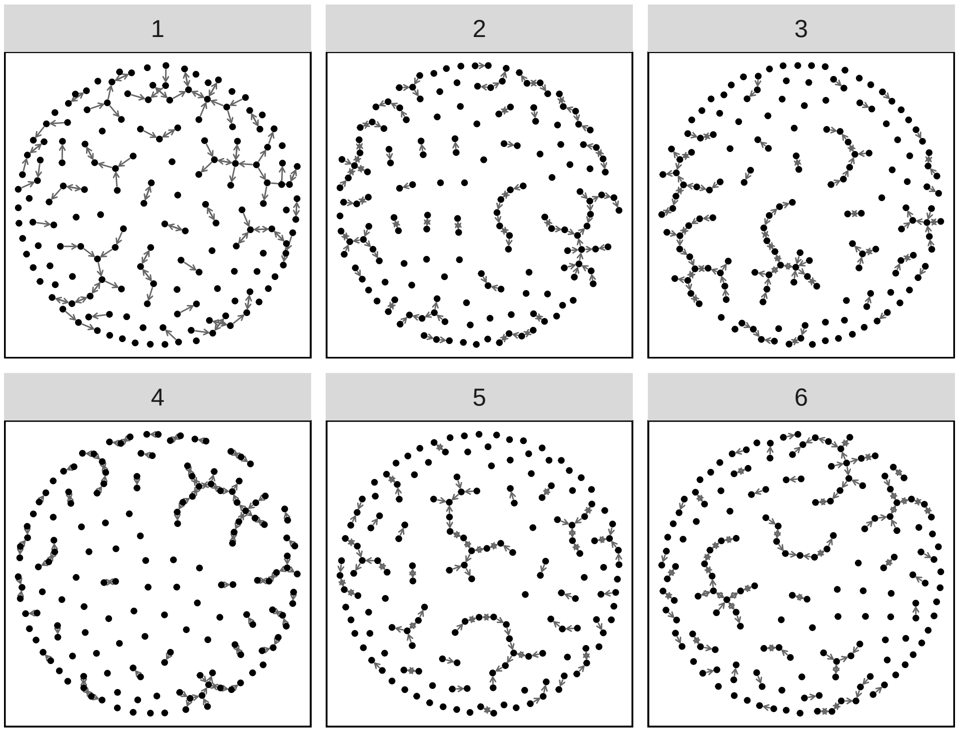
jtts_pos_hard_1



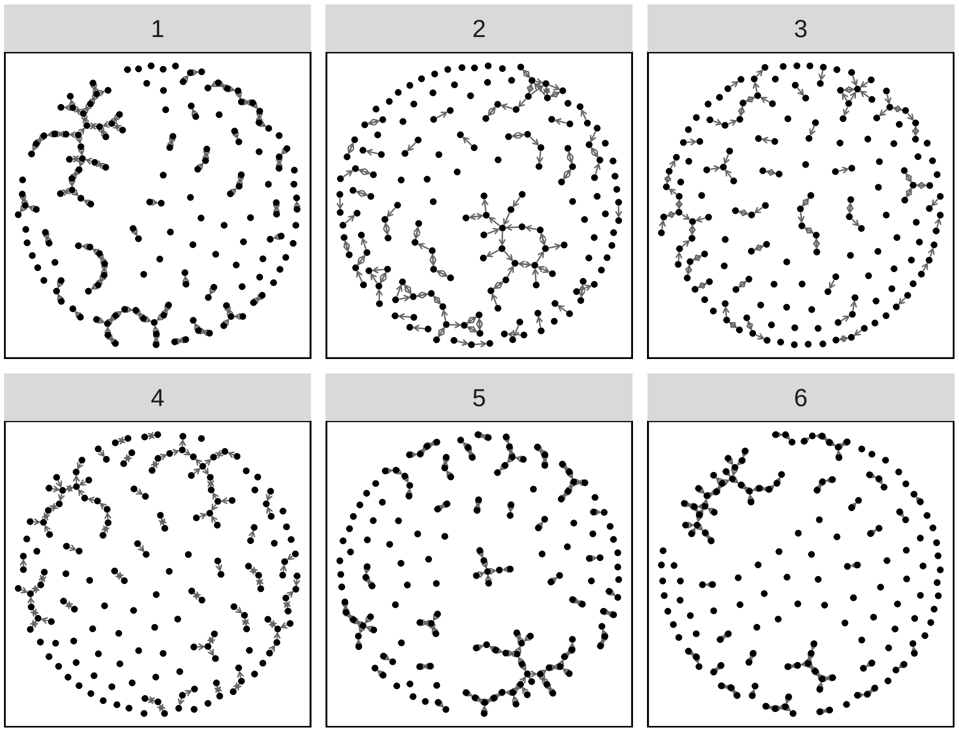
jtts_pos_hard_2



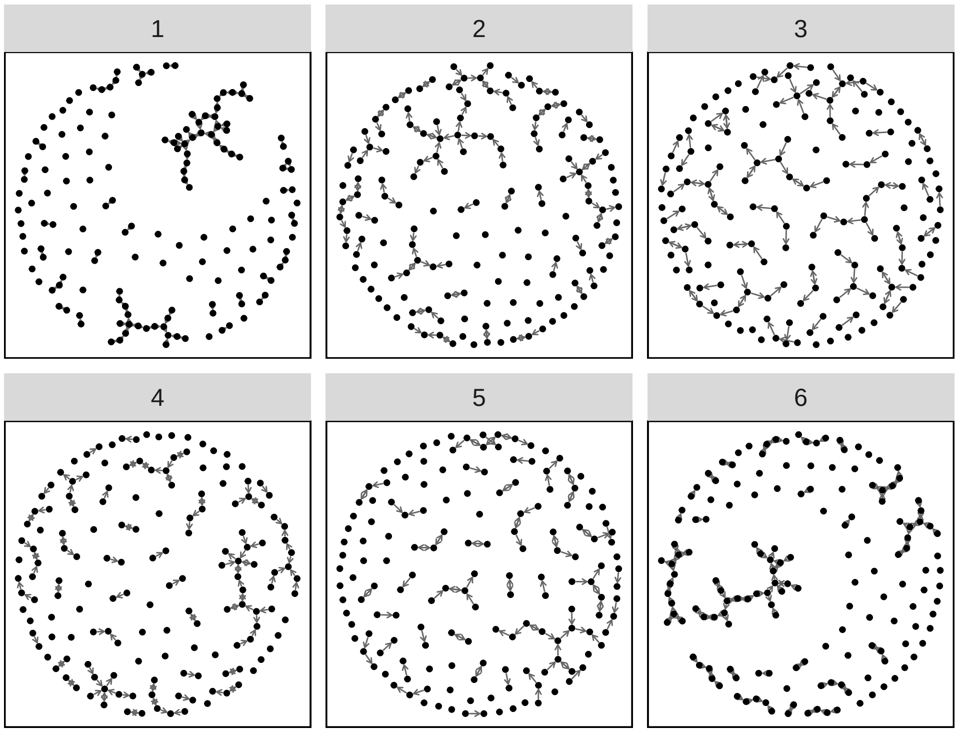
jtts_pos_hard_3



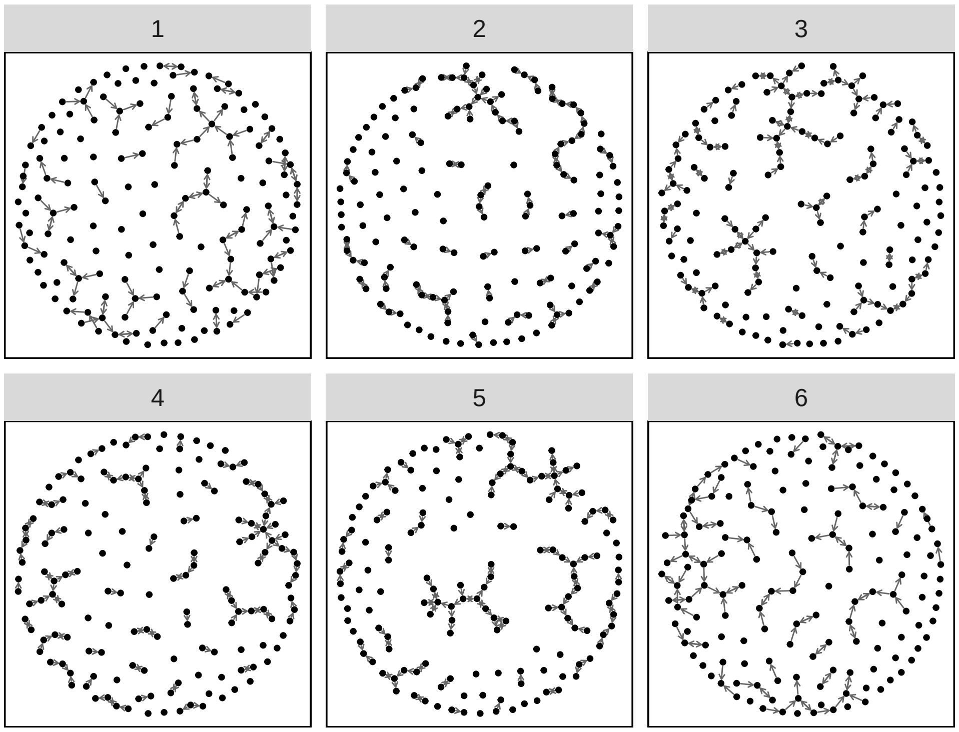
jtts_pos_med_1



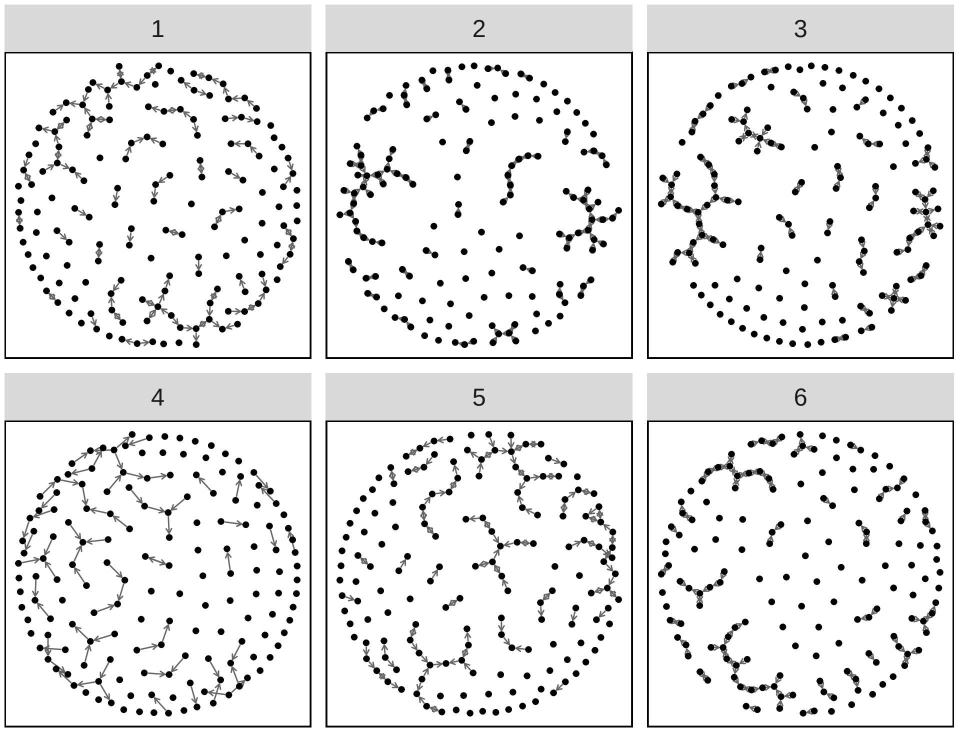
jtts_pos_med_2

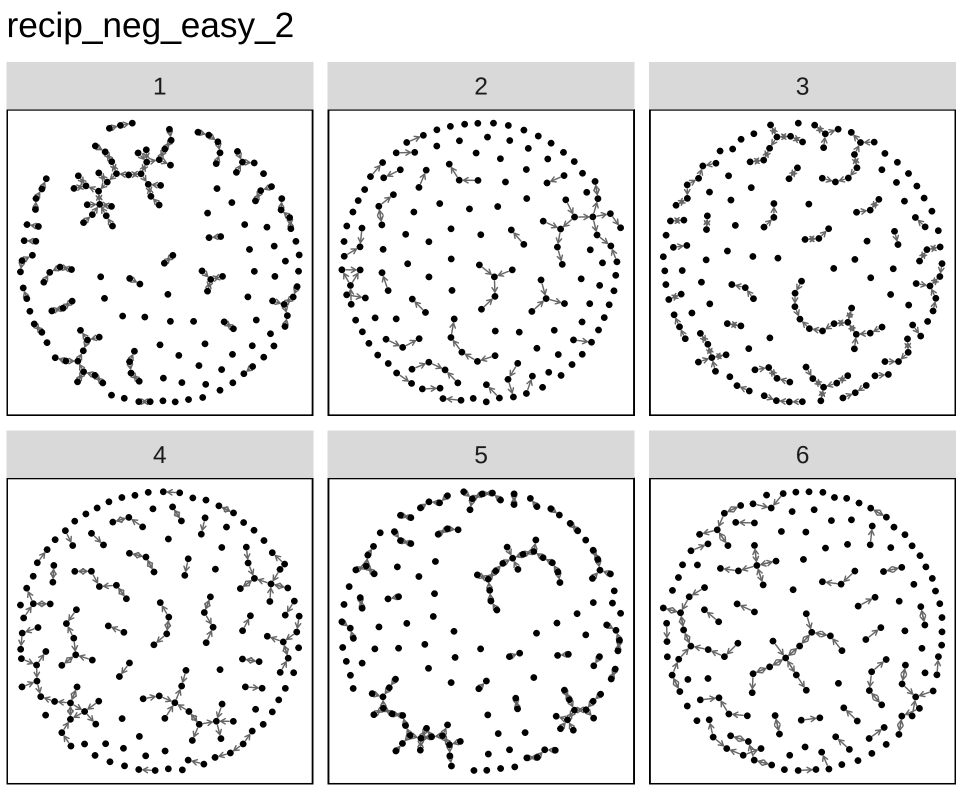


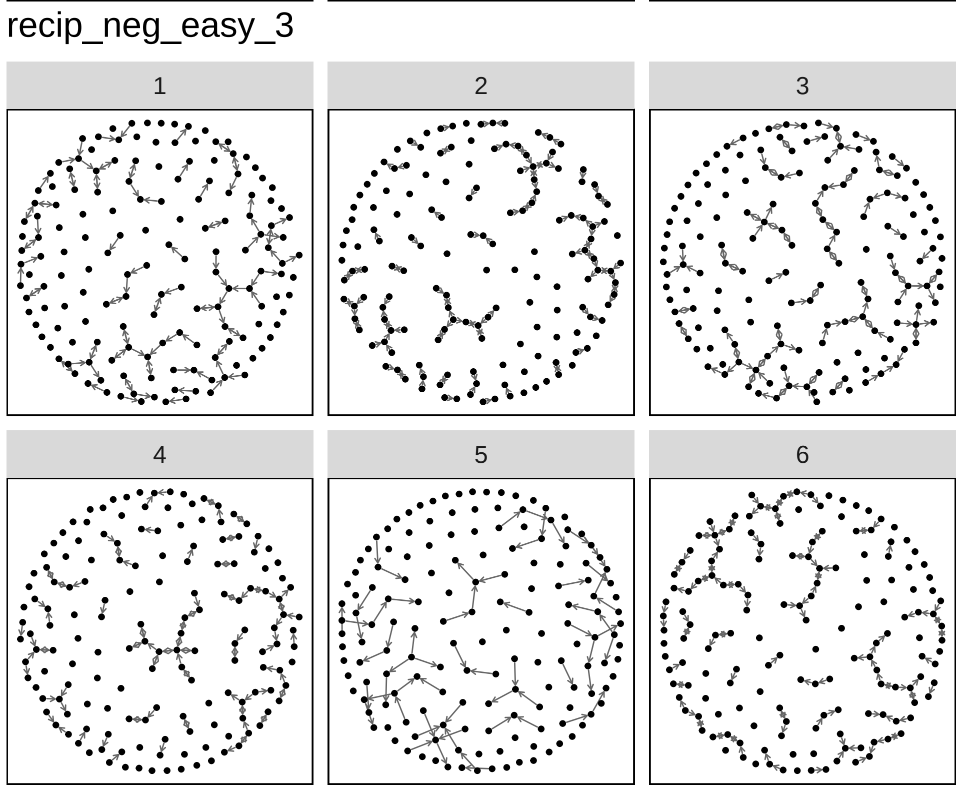
jtts_pos_med_3



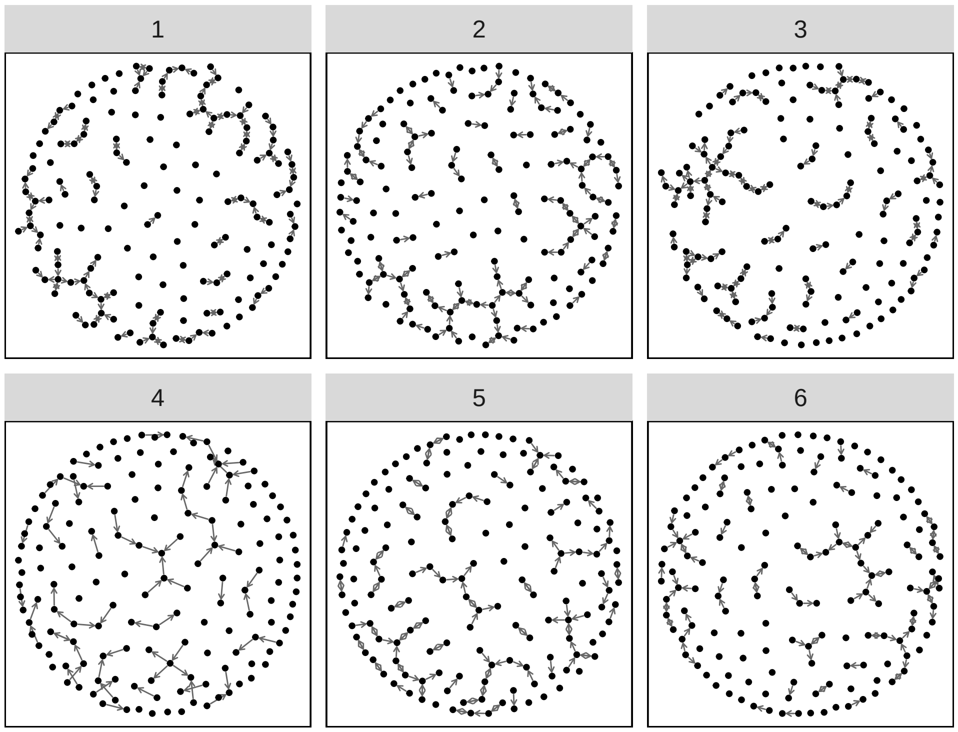
recip_neg_easy_1



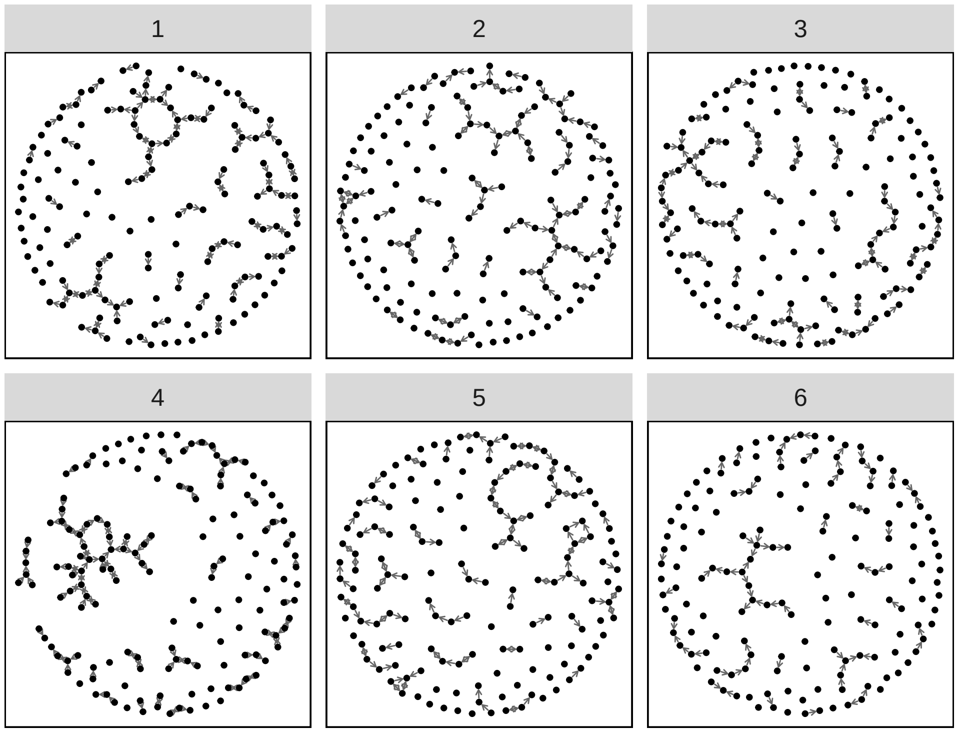

recip_neg_easy_2
1
2
3
4
5
6

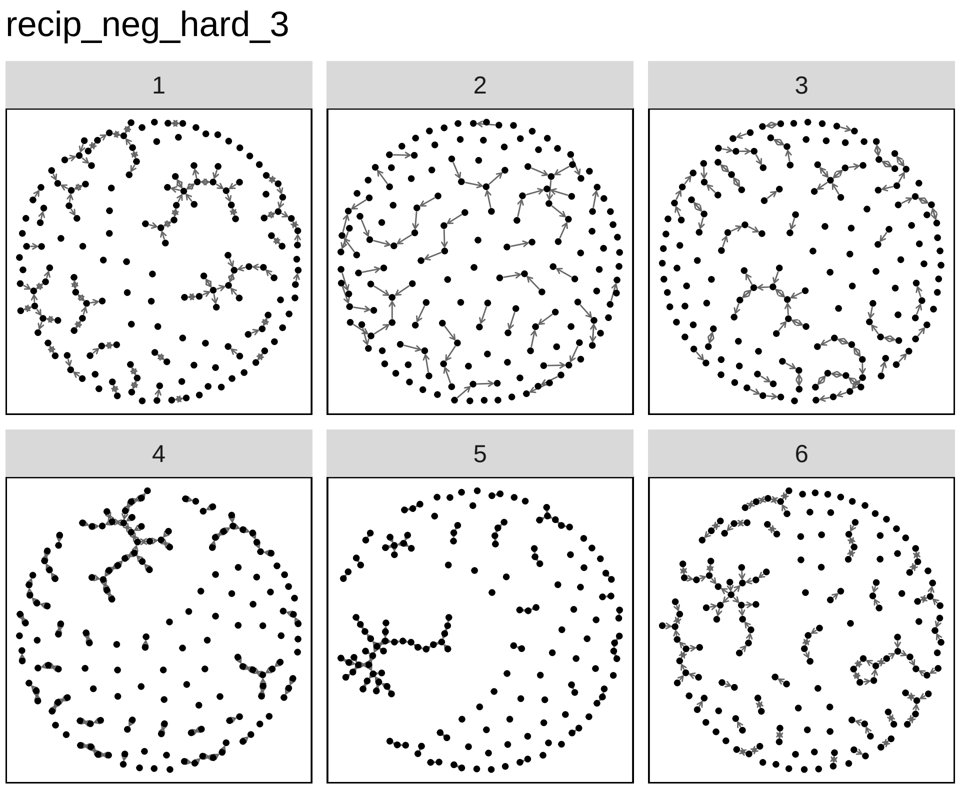

recip_neg_easy_3
1
2
3
4
5
6

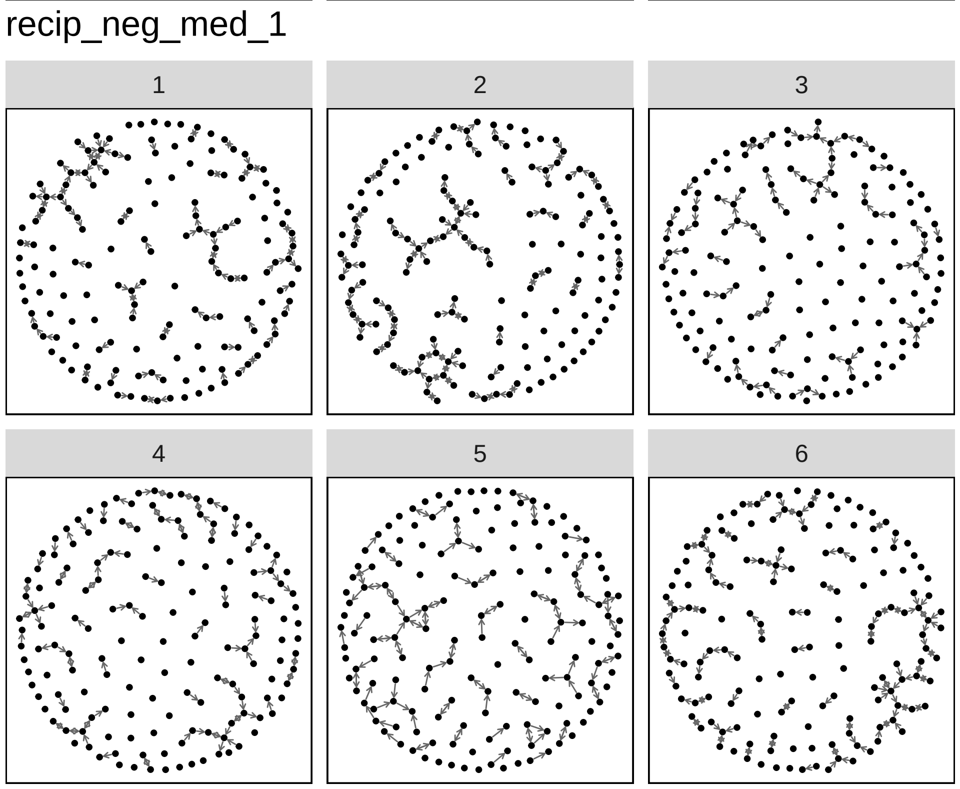
recip_neg_hard_1



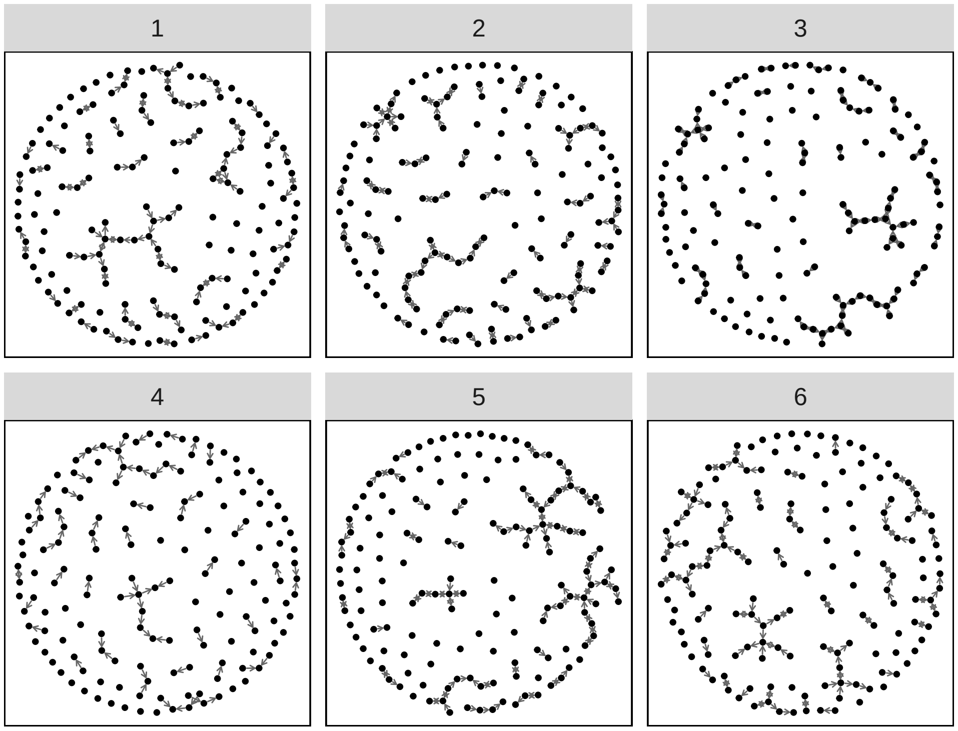
recip_neg_hard_2



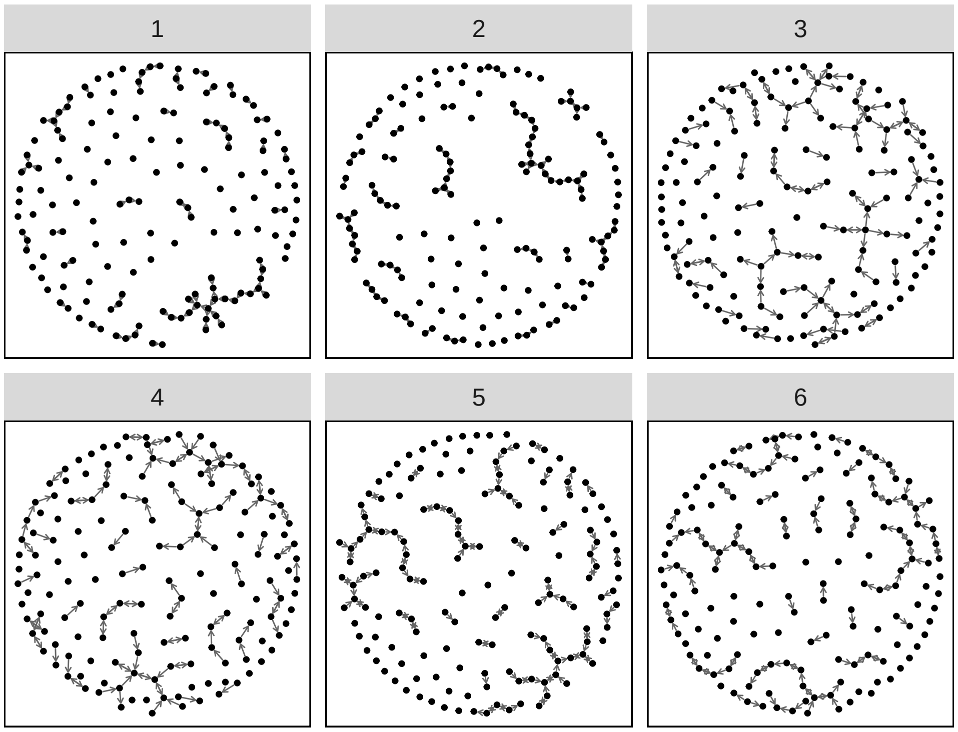

recip_neg_hard_3
1
2
3
4
5
6


recip_neg_med_1
1
2
3
4
5
6

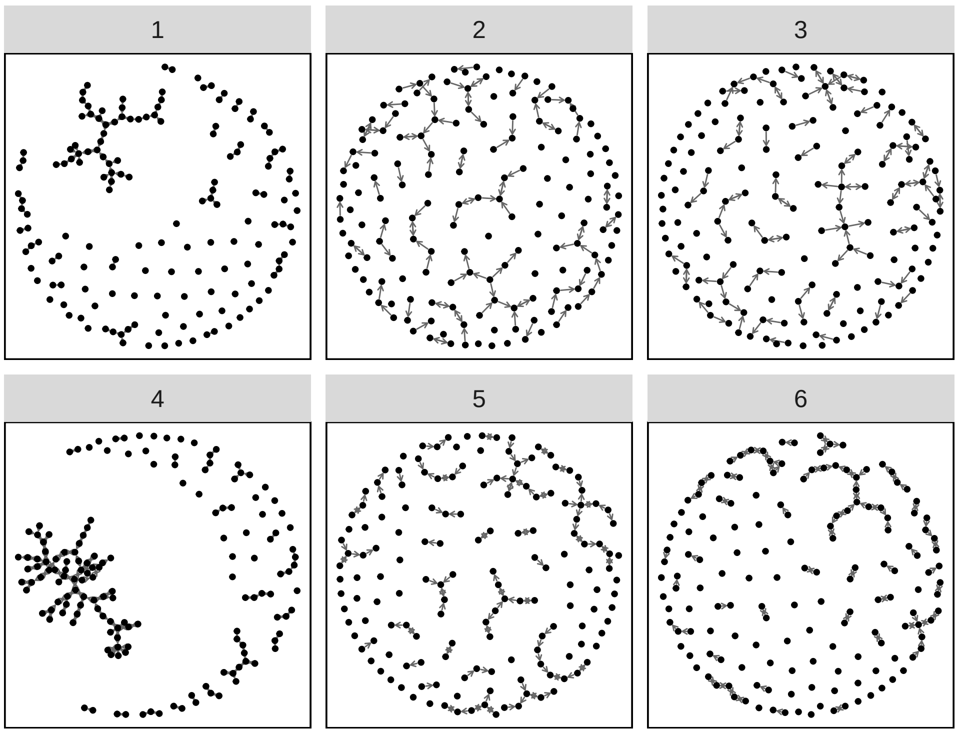
recip_neg_med_2



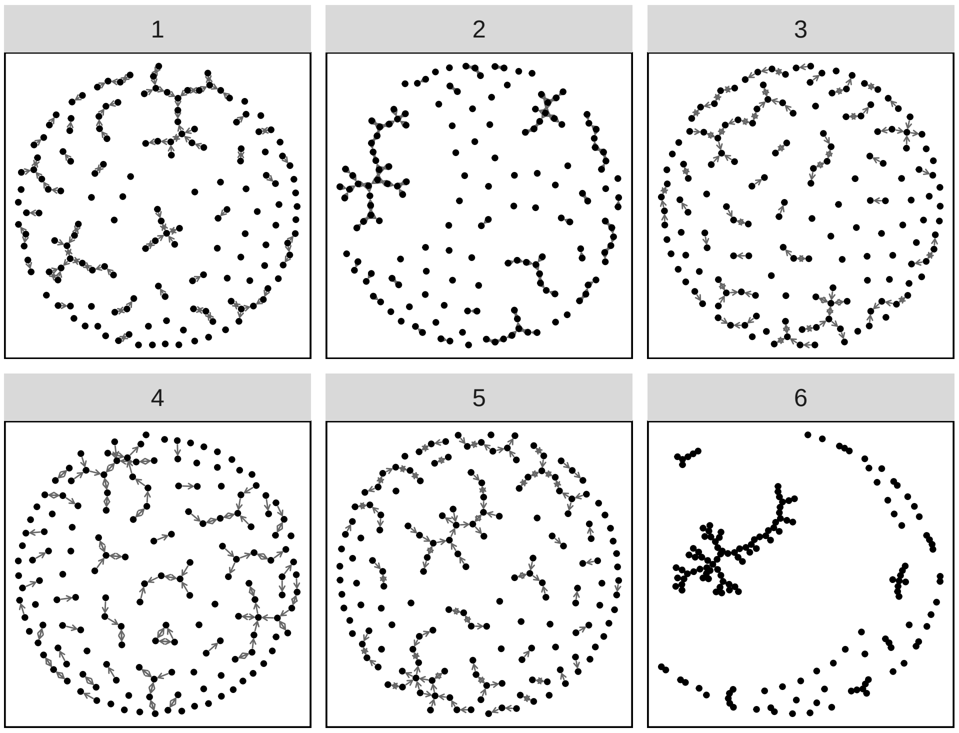
recip_neg_med_3



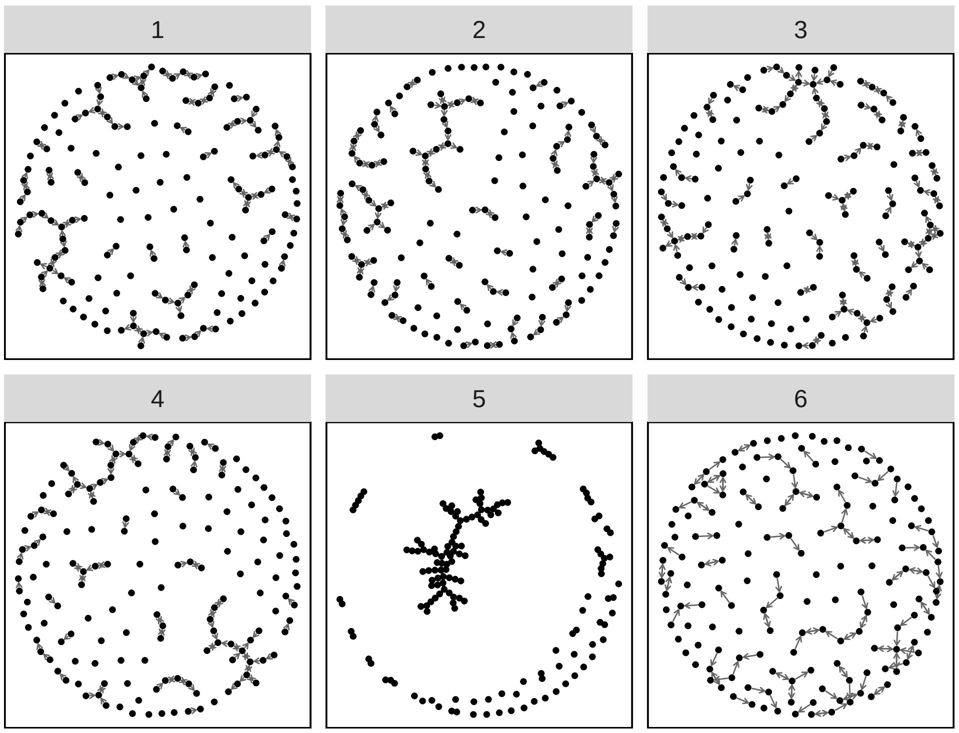
recip_pos_easy_1



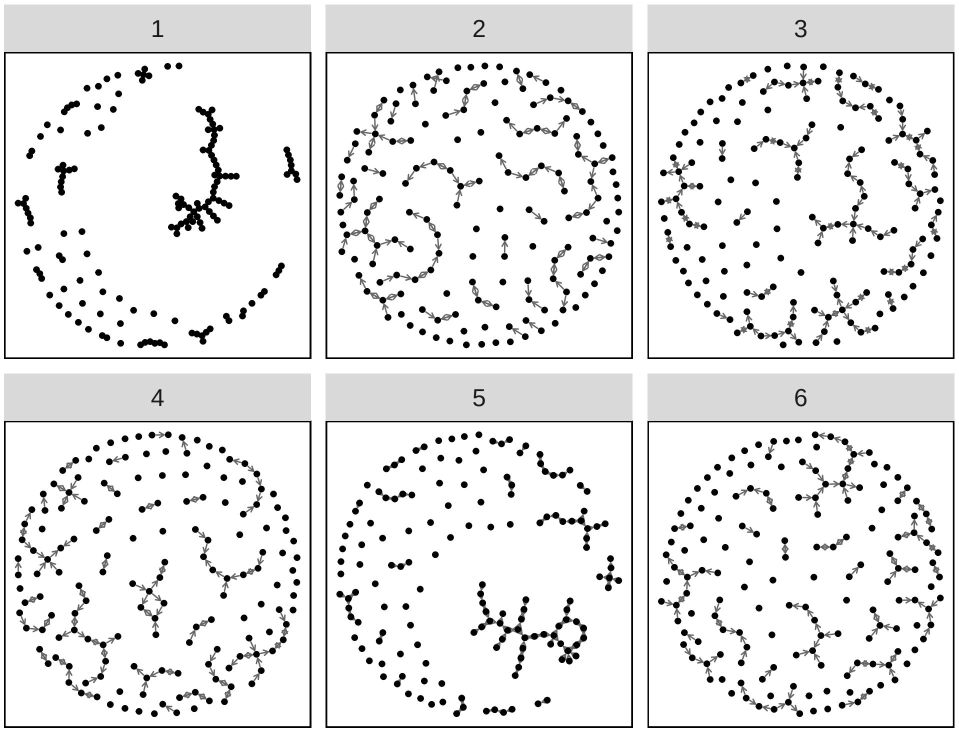
recip_pos_easy_2



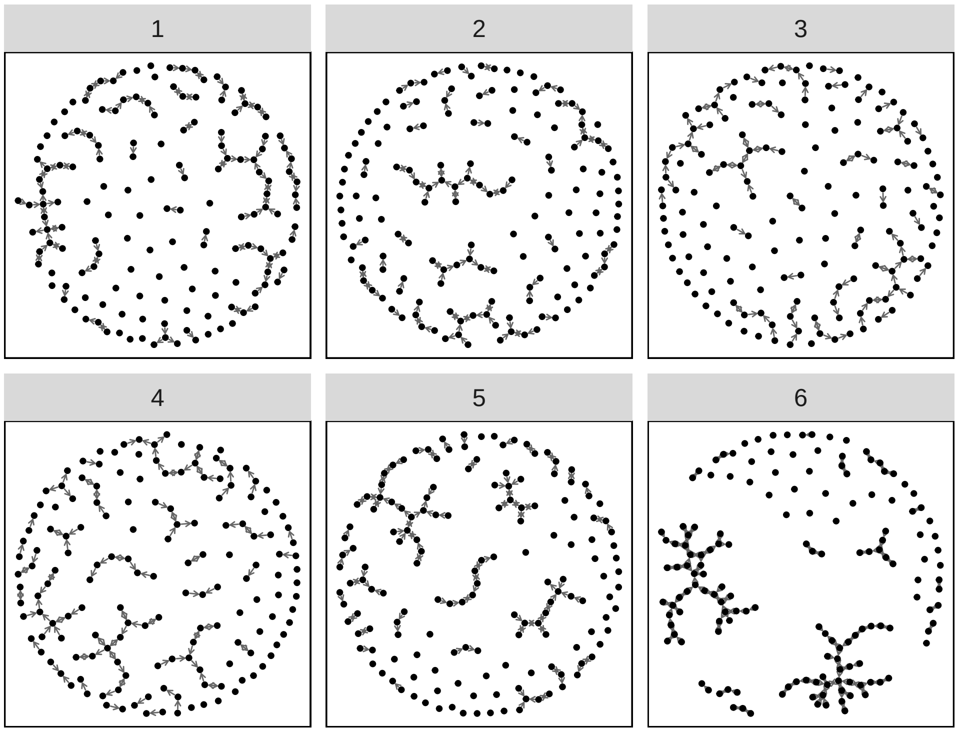
recip_pos_easy_3



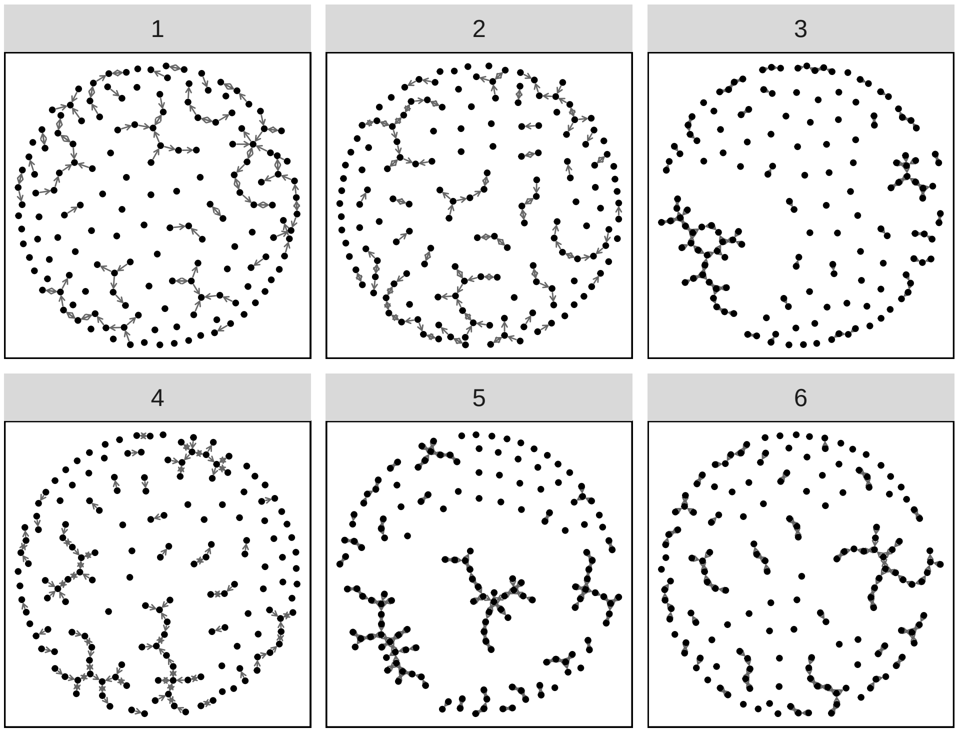
recip_pos_hard_1



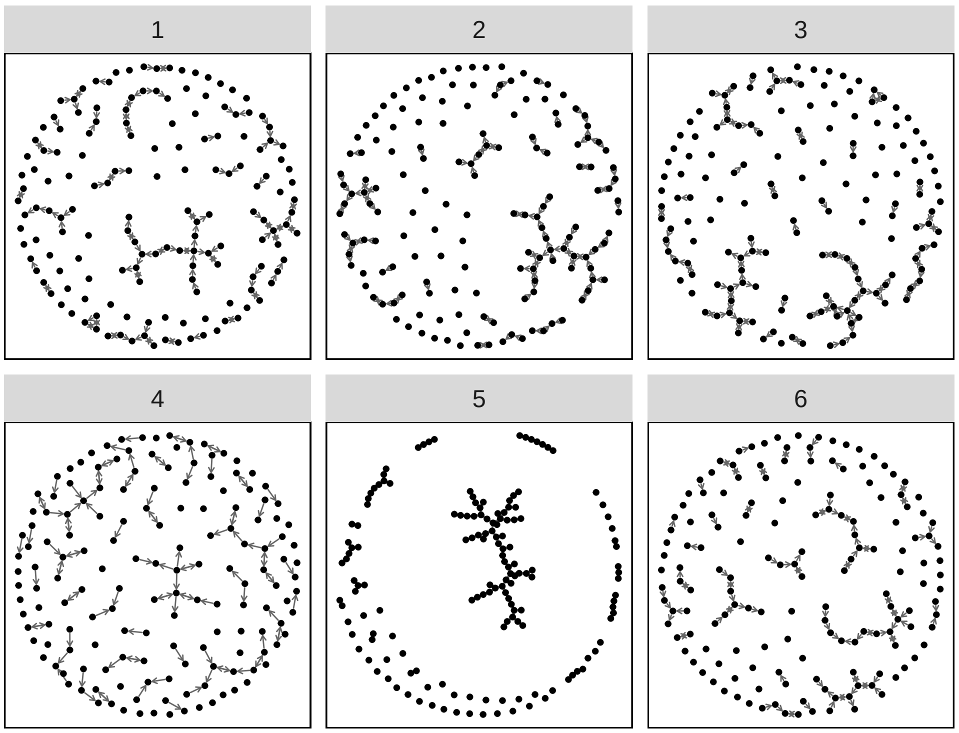
recip_pos_hard_2



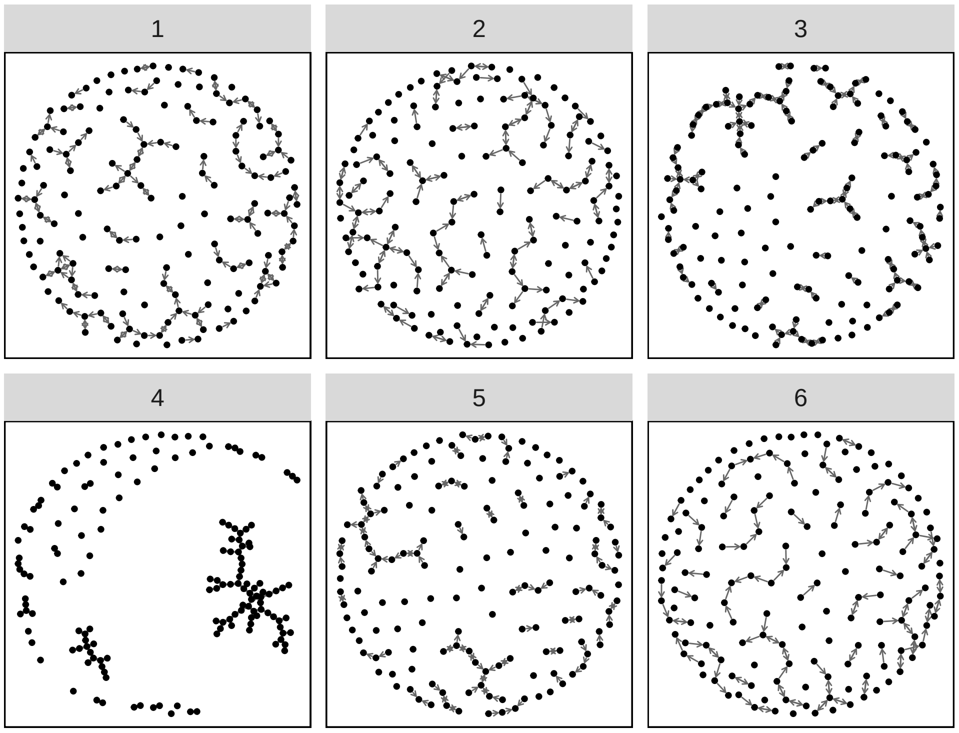
recip_pos_hard_3



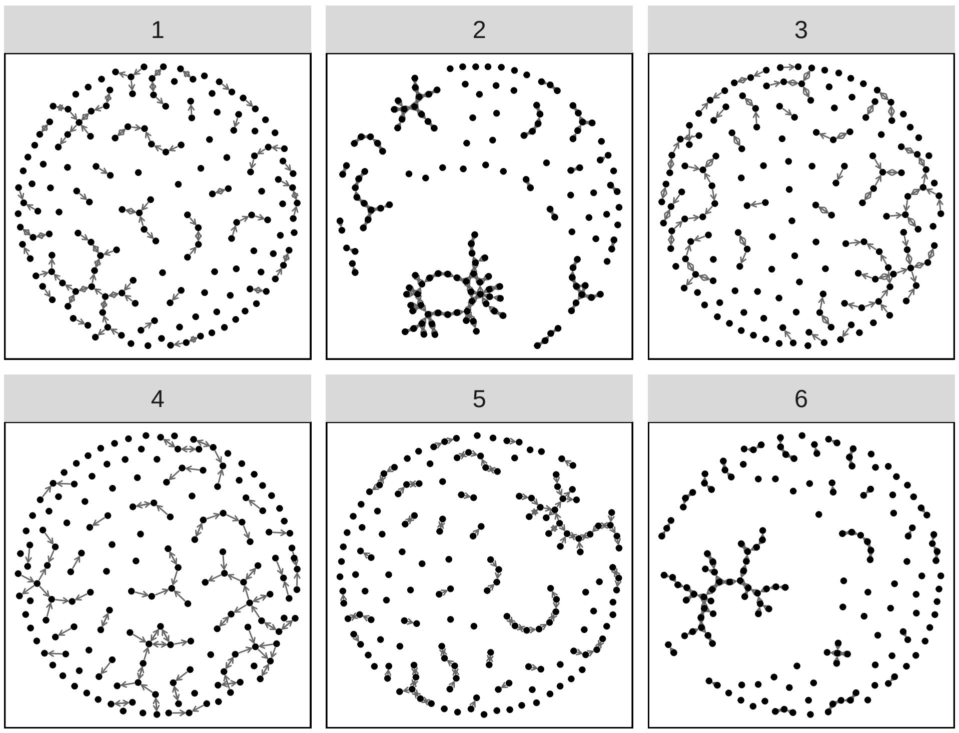
recip_pos_med_1



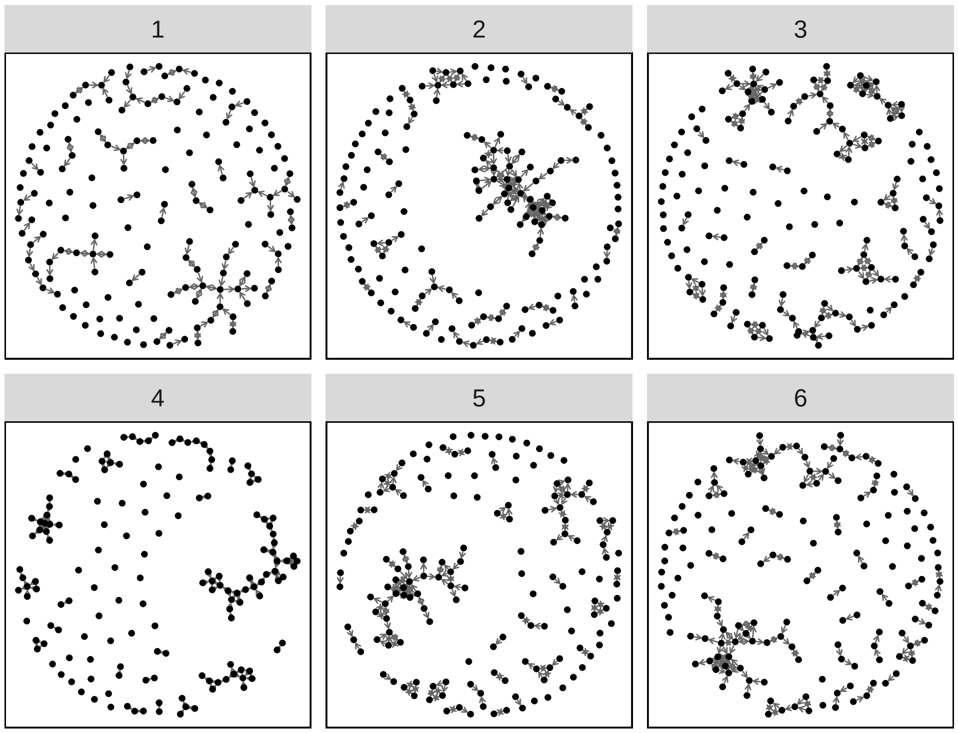
recip_pos_med_2



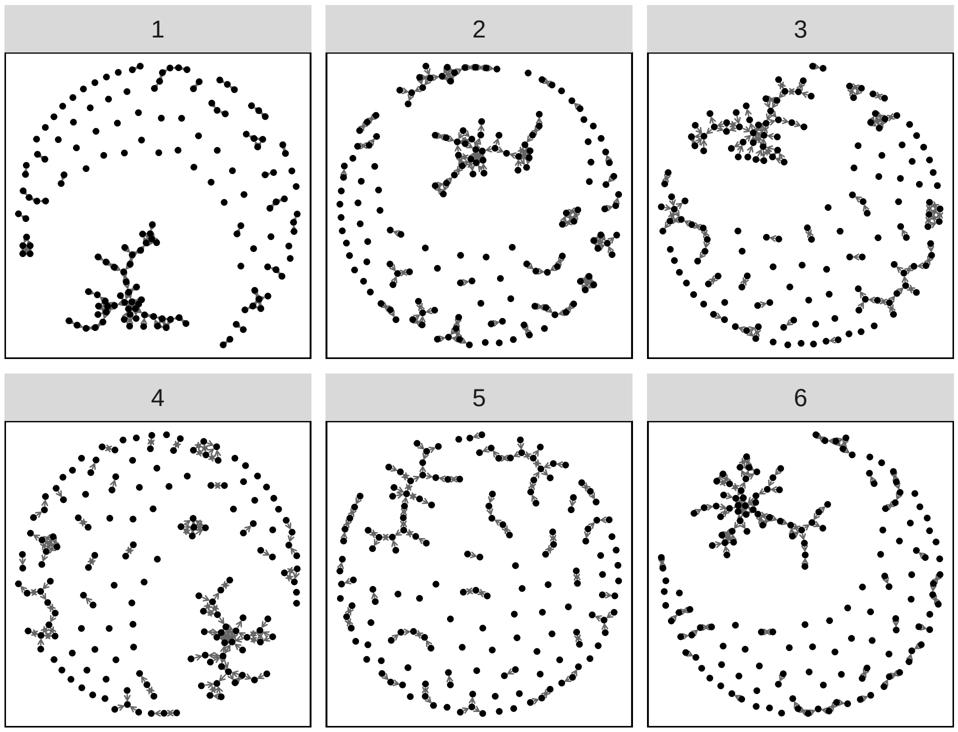
recip_pos_med_3



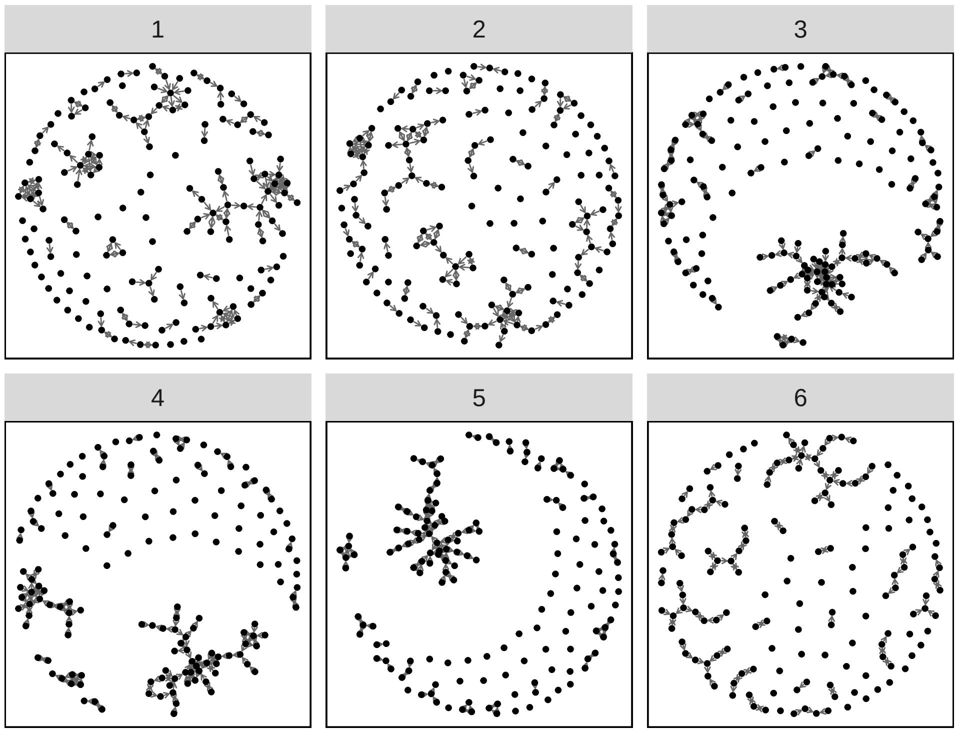
samettp_neg_easy_1



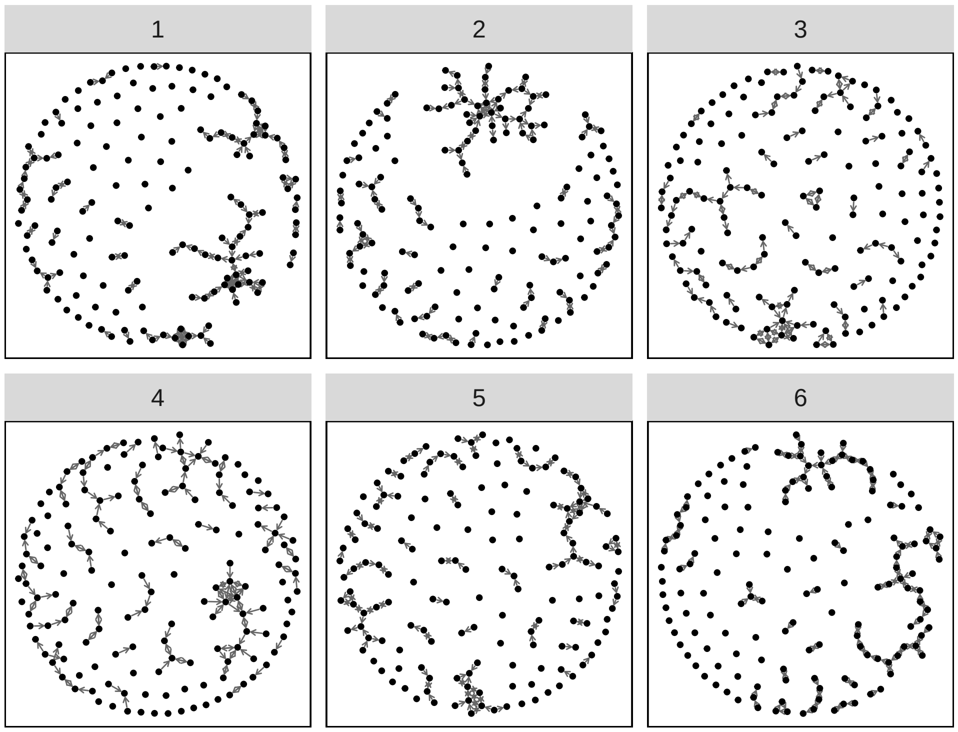
sametp_neg_easy_2



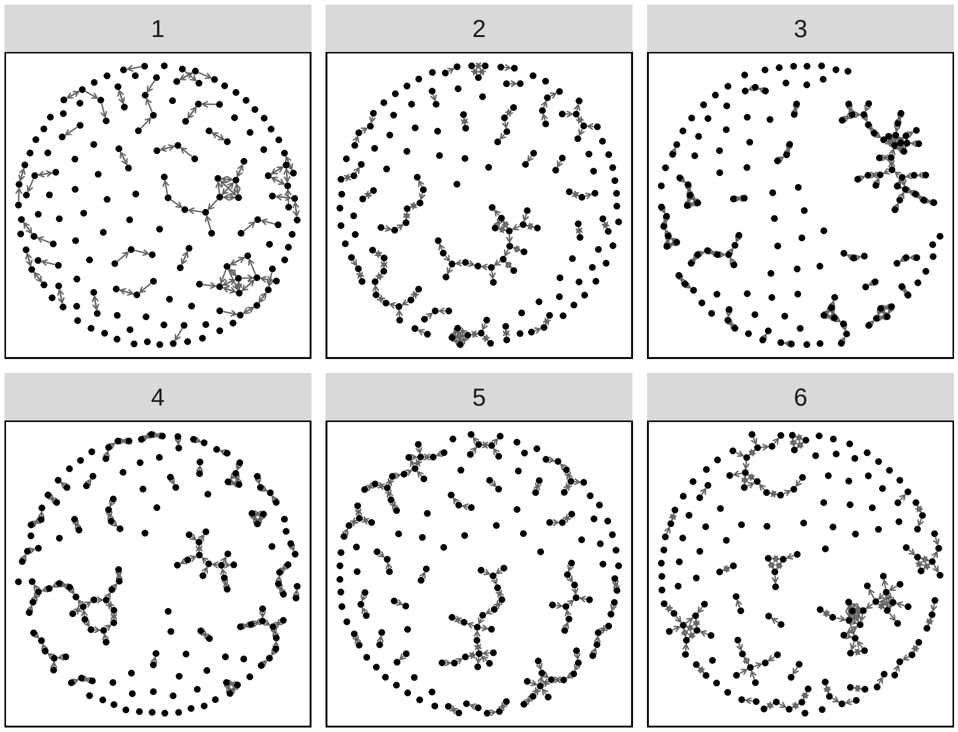
sametp_neg_easy_3



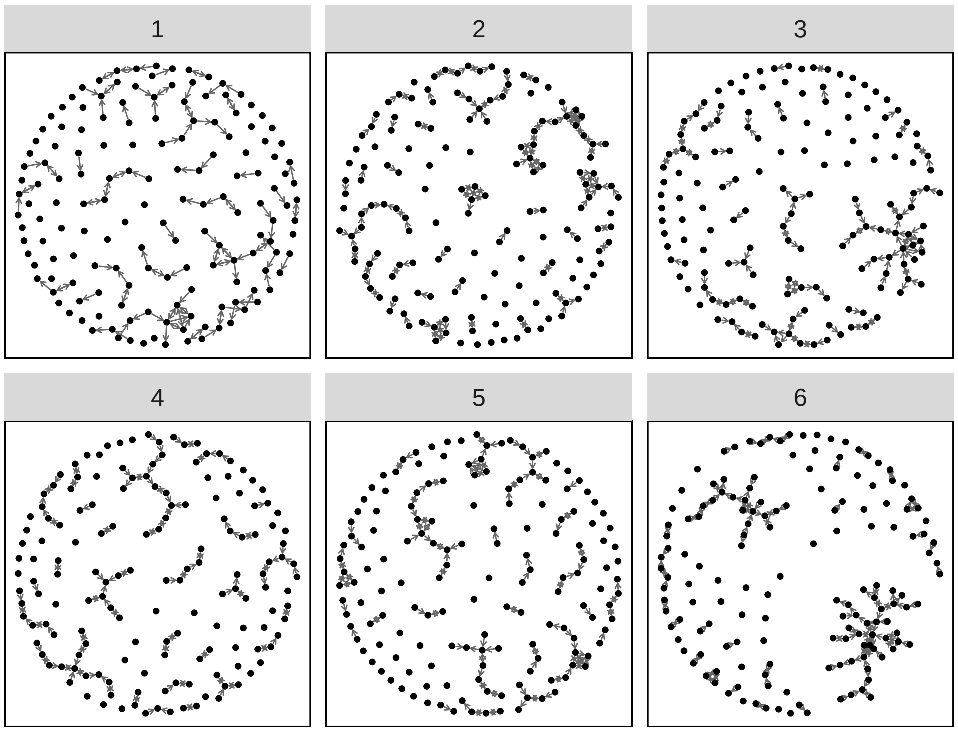
samettp_neg_hard_1



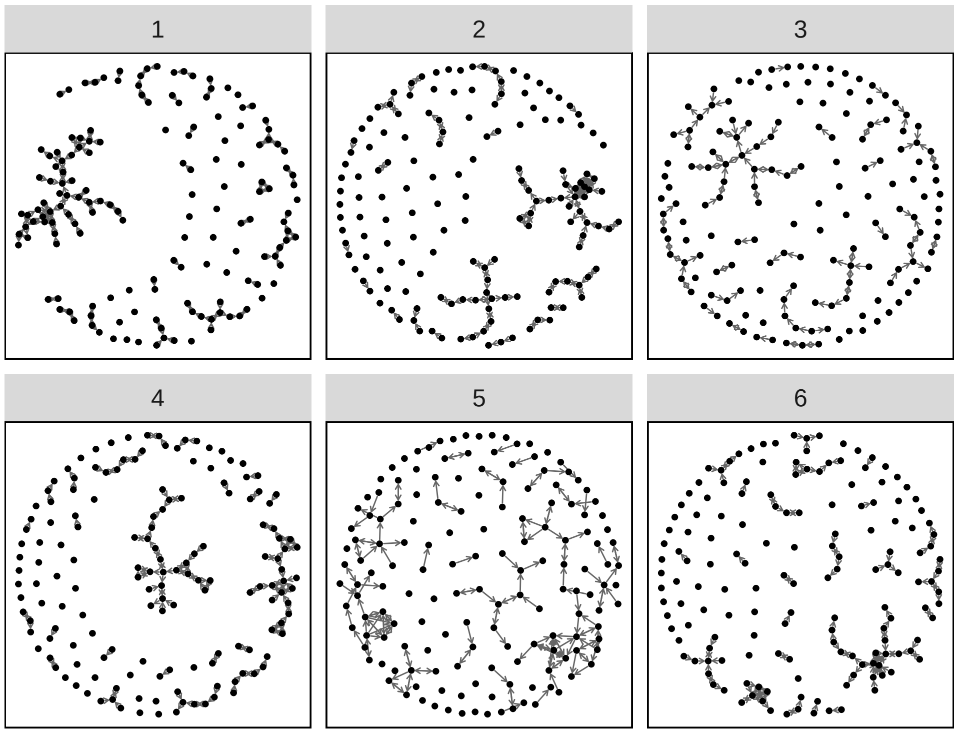
samettp_neg_hard_2



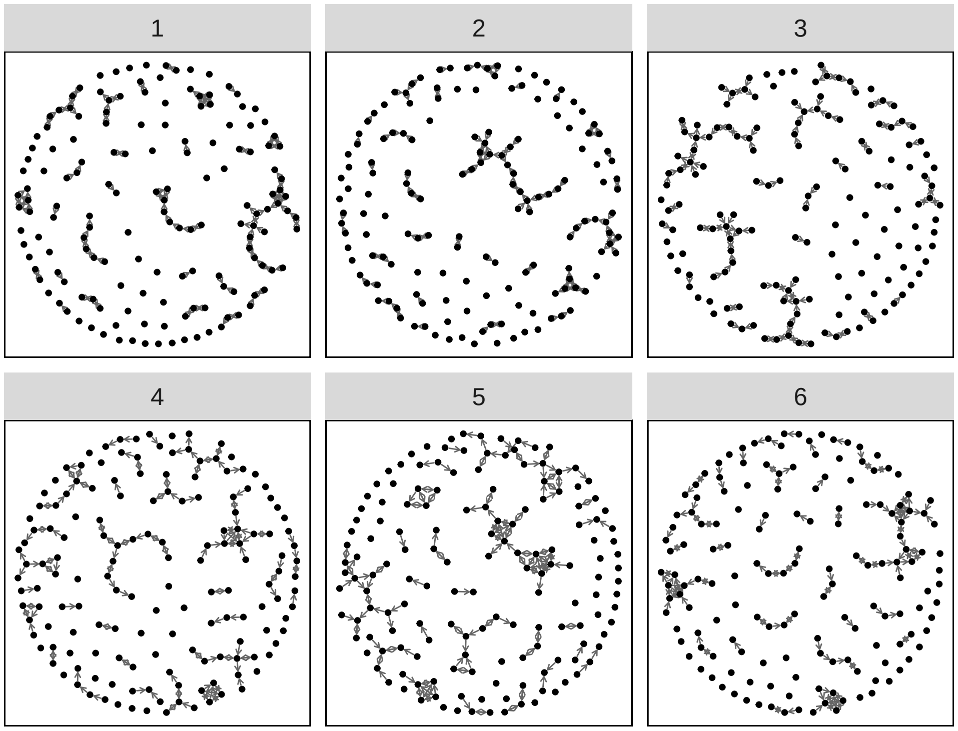
sametp_neg_hard_3



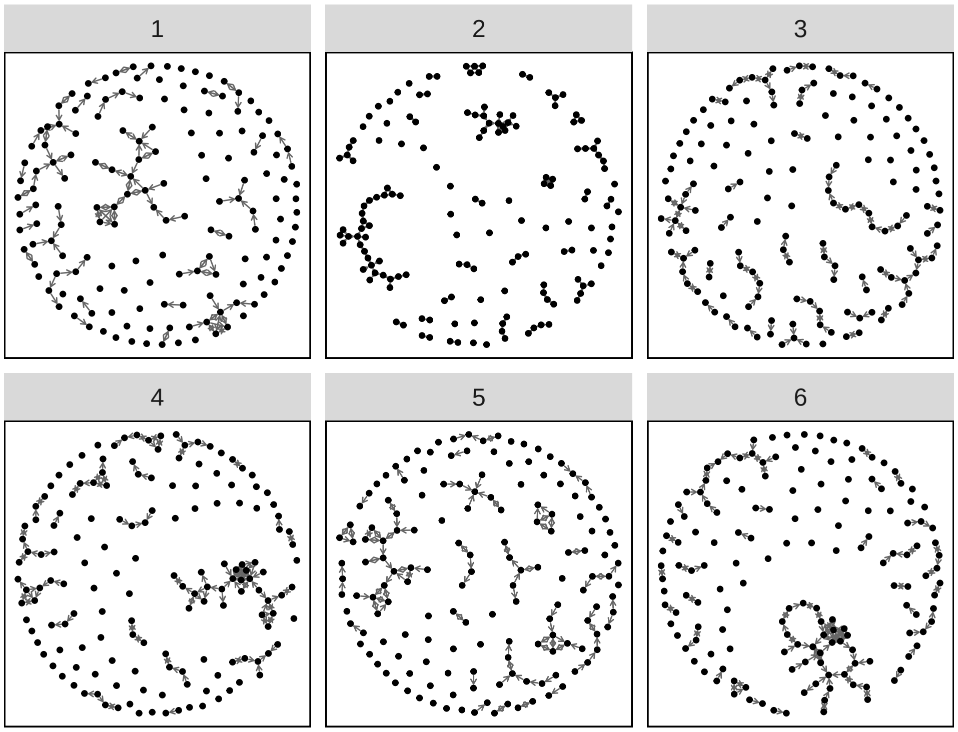
samettp_neg_med_1



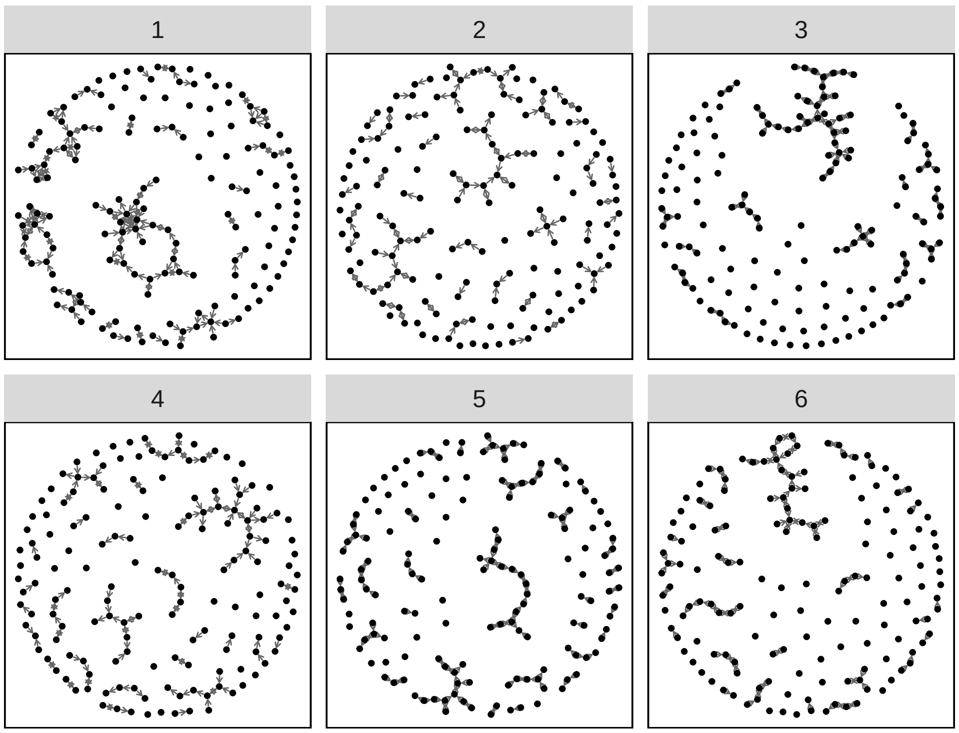
samettp_neg_med_2



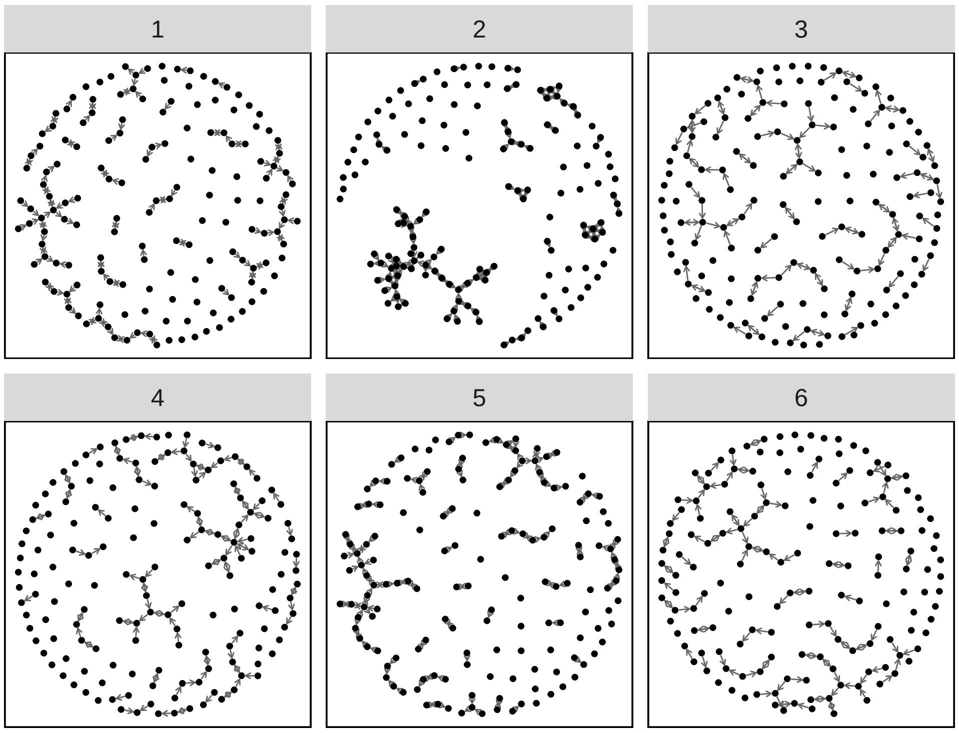
samettp_neg_med_3



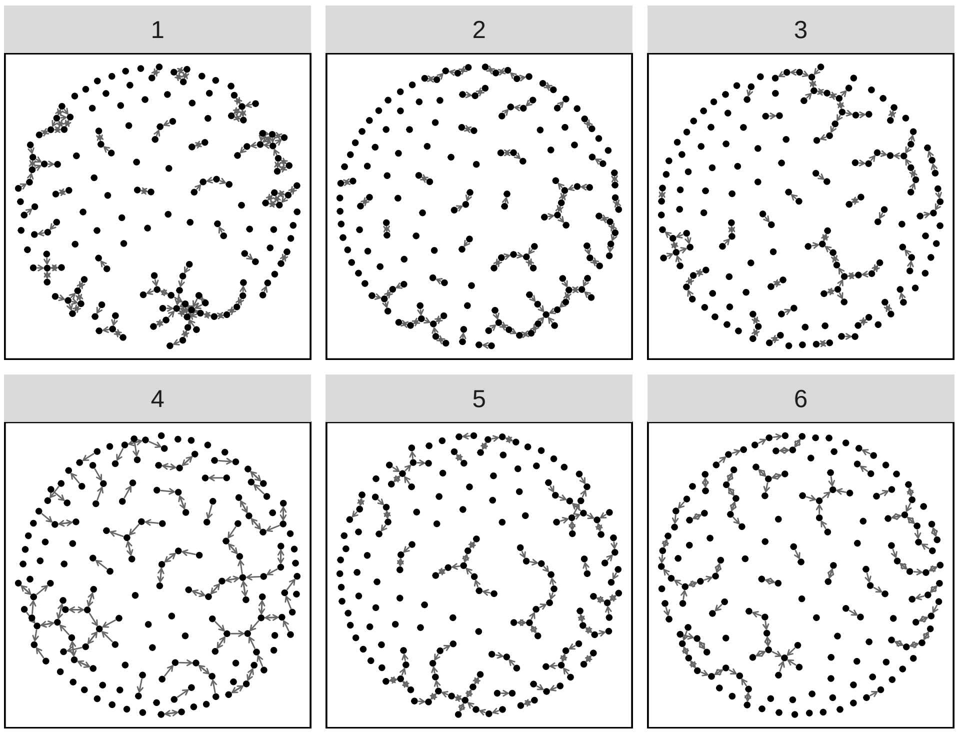
samettp_pos_easy_1



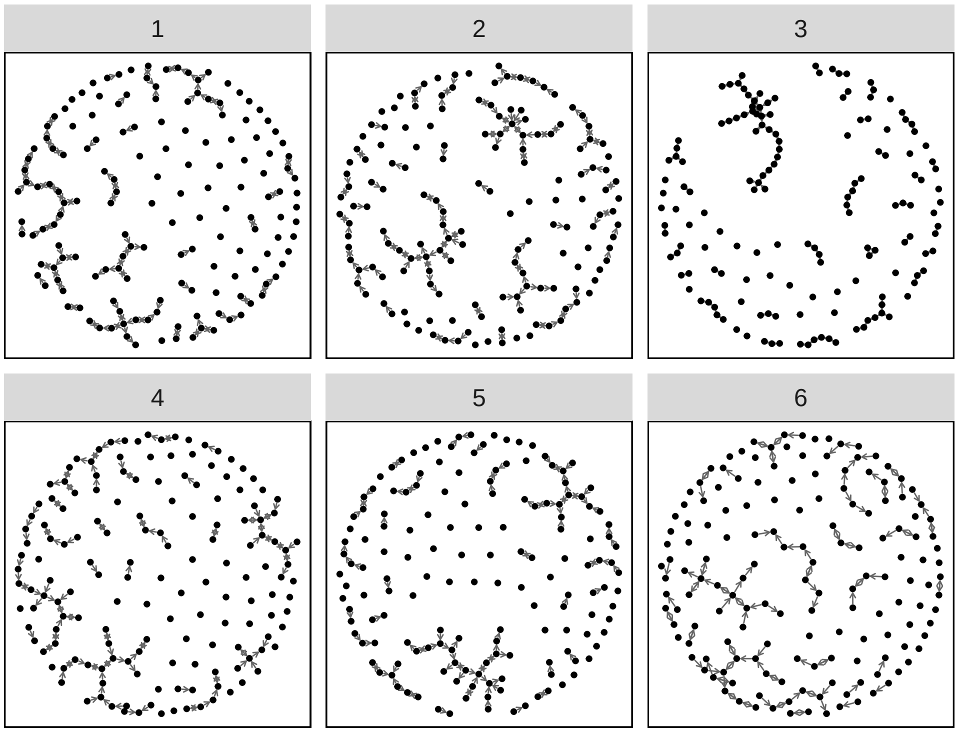
samettp_pos_easy_2



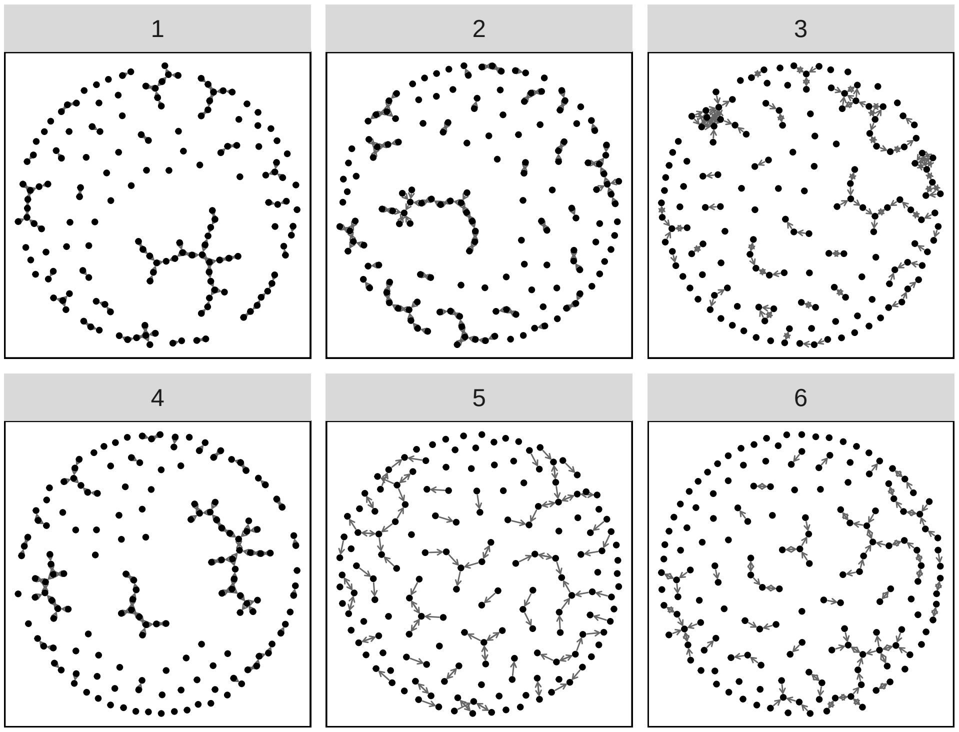
sametp_pos_easy_3



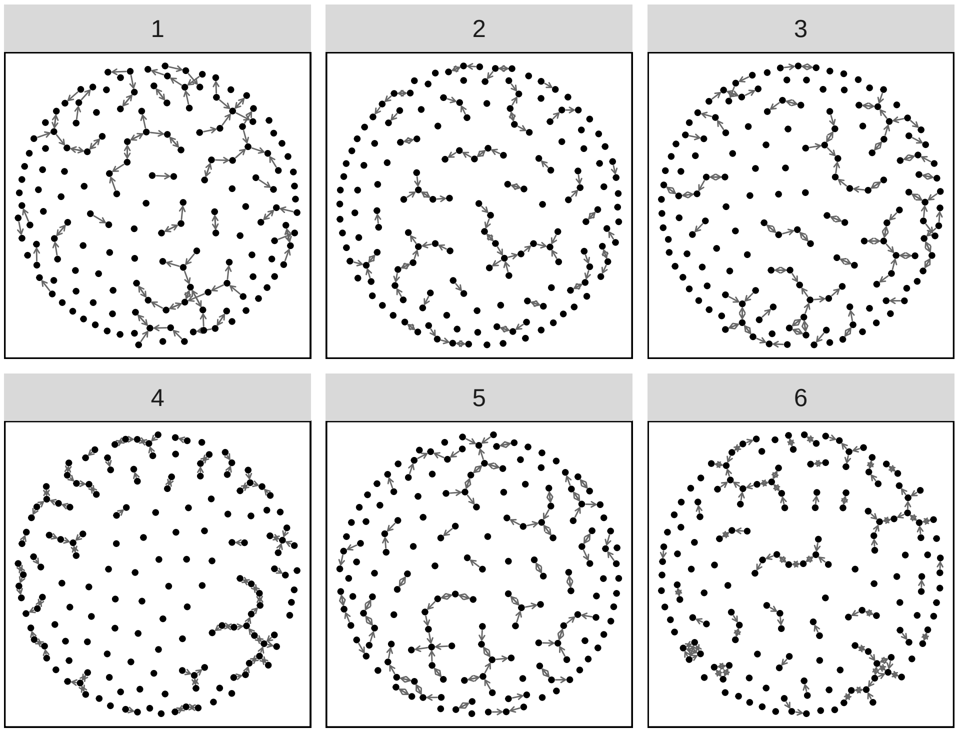
sametp_pos_hard_1



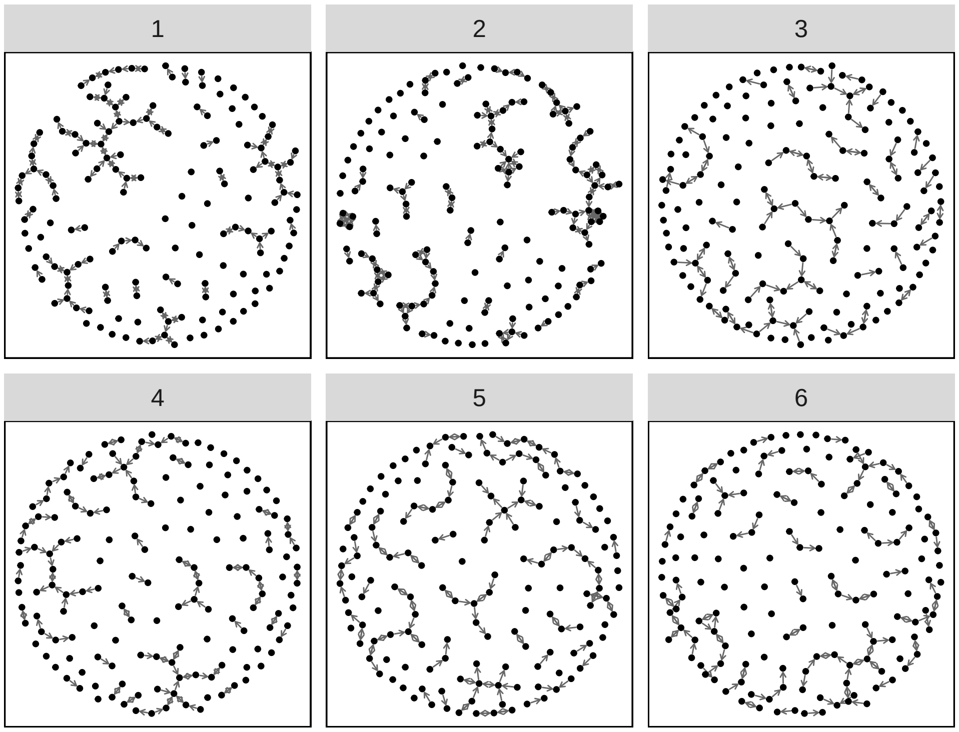
sametp_pos_hard_2



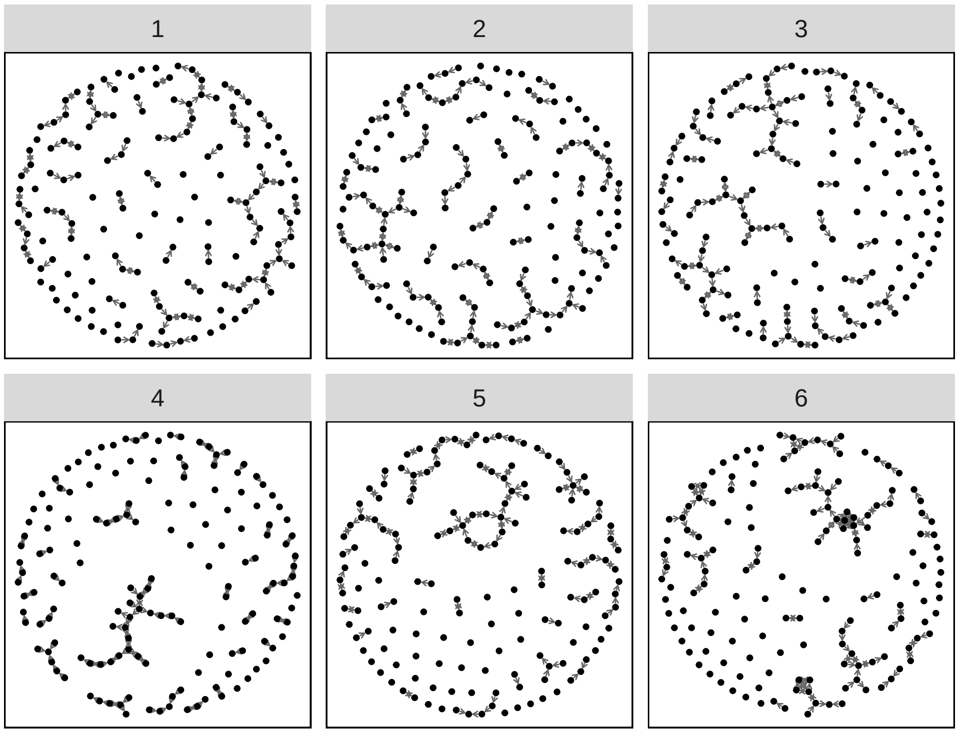
sametp_pos_hard_3



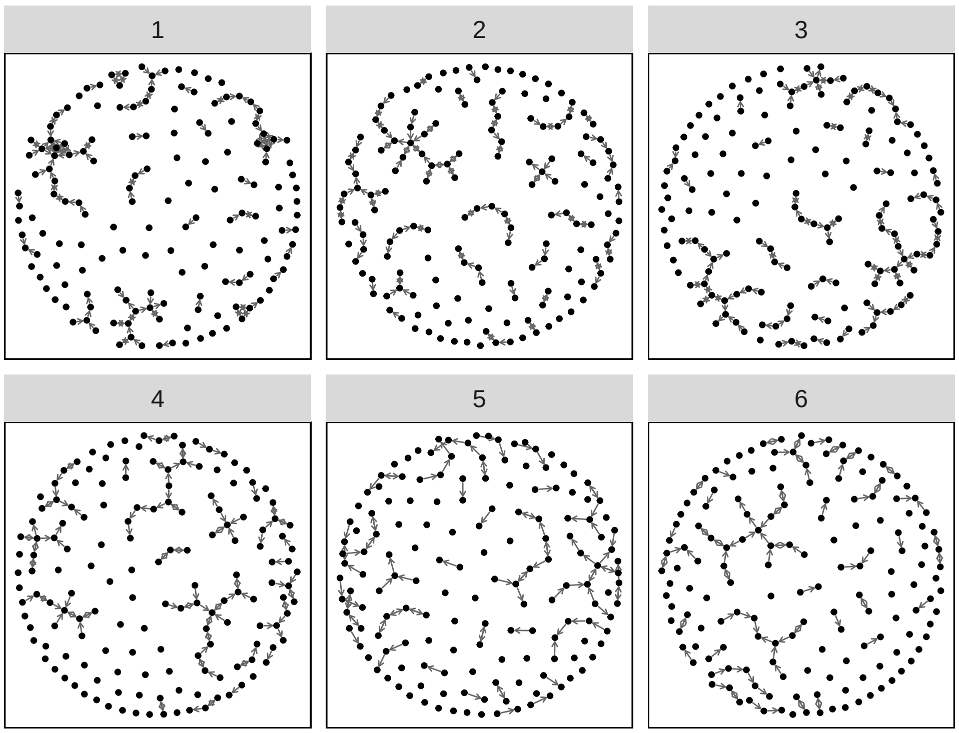
sametp_pos_med_1



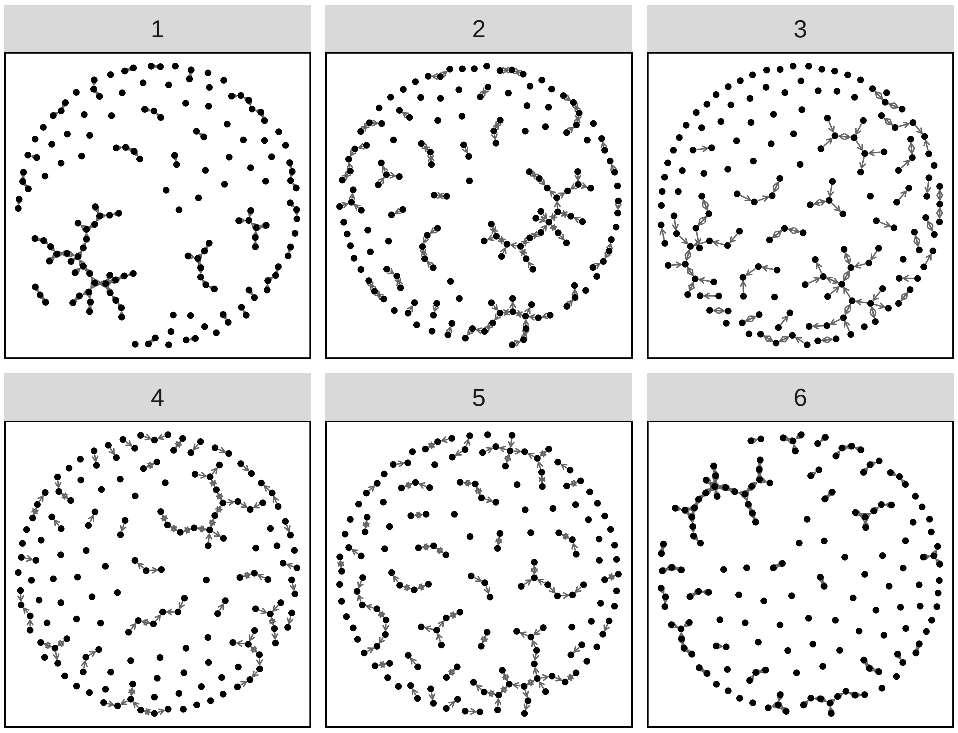
sametp_pos_med_2



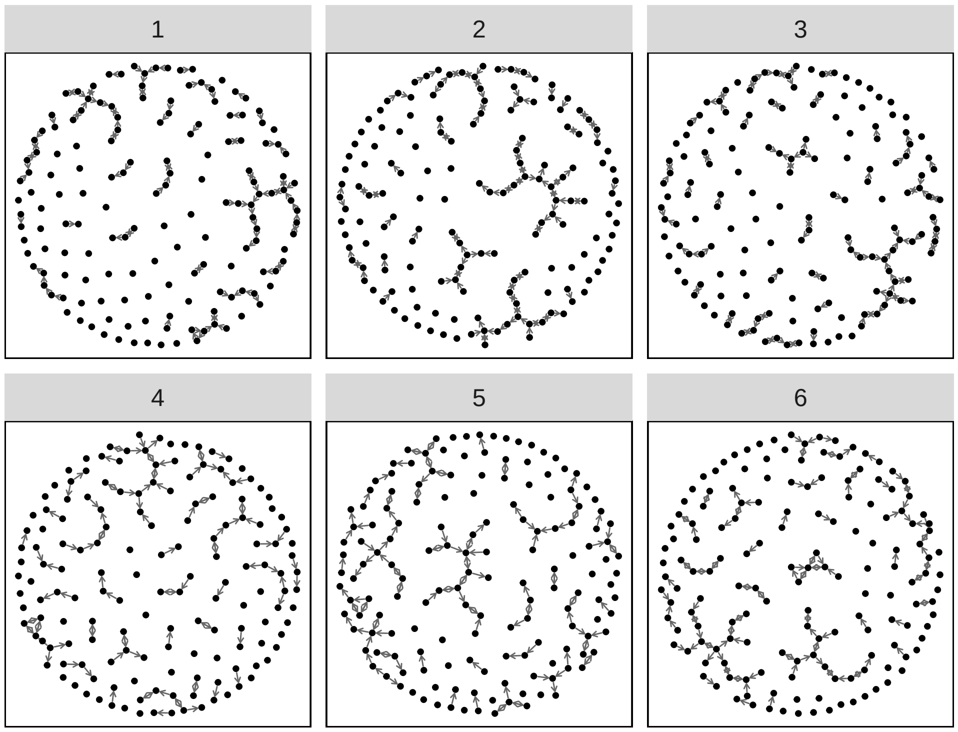
samettp_pos_med_3



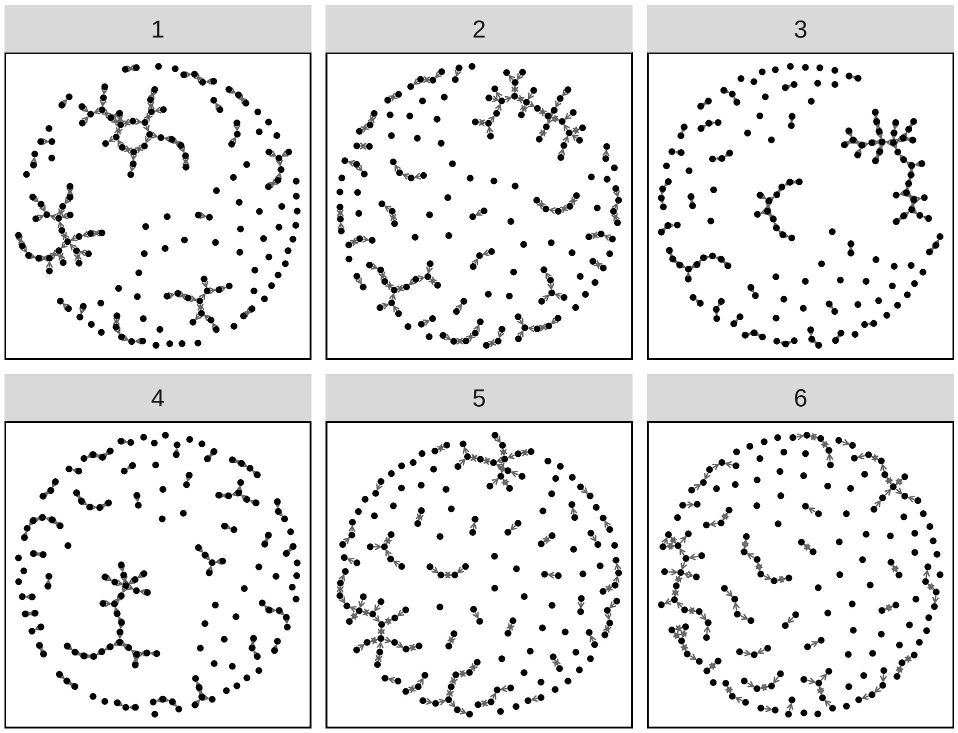
simttb_gof_9_1

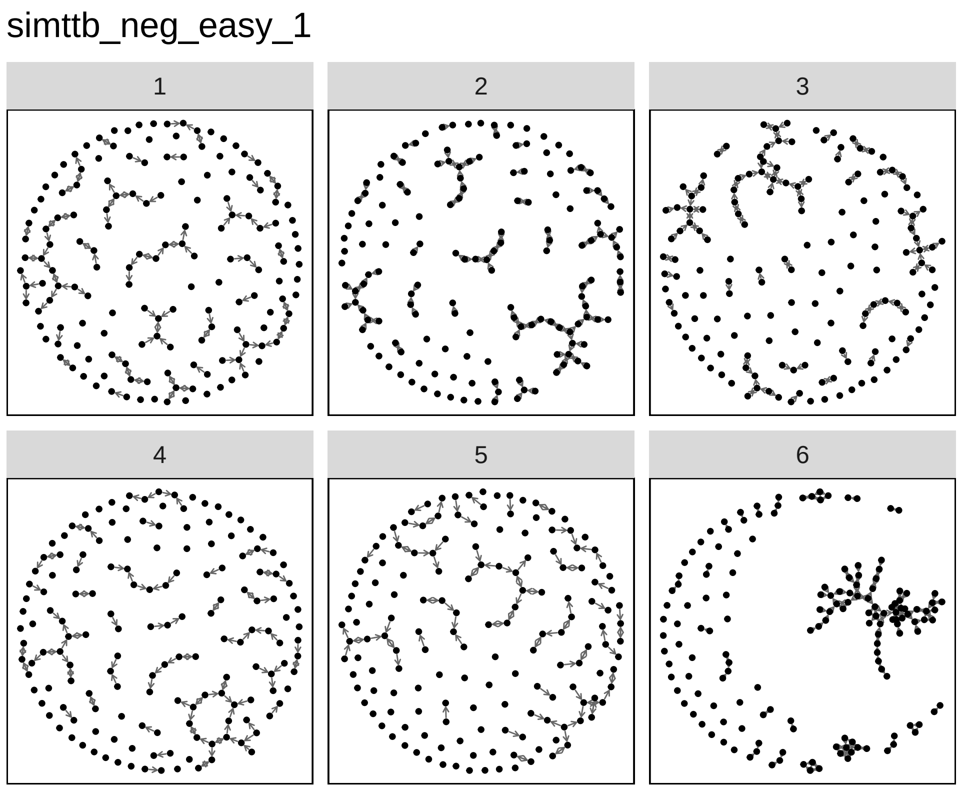


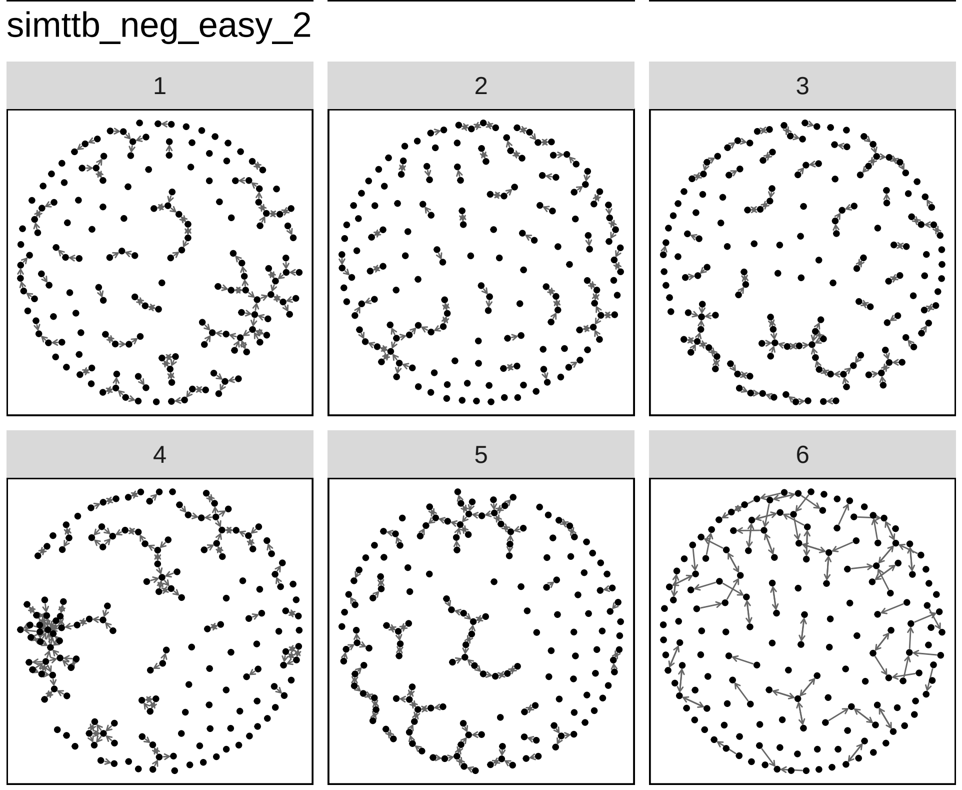
simttb_gof_9_2



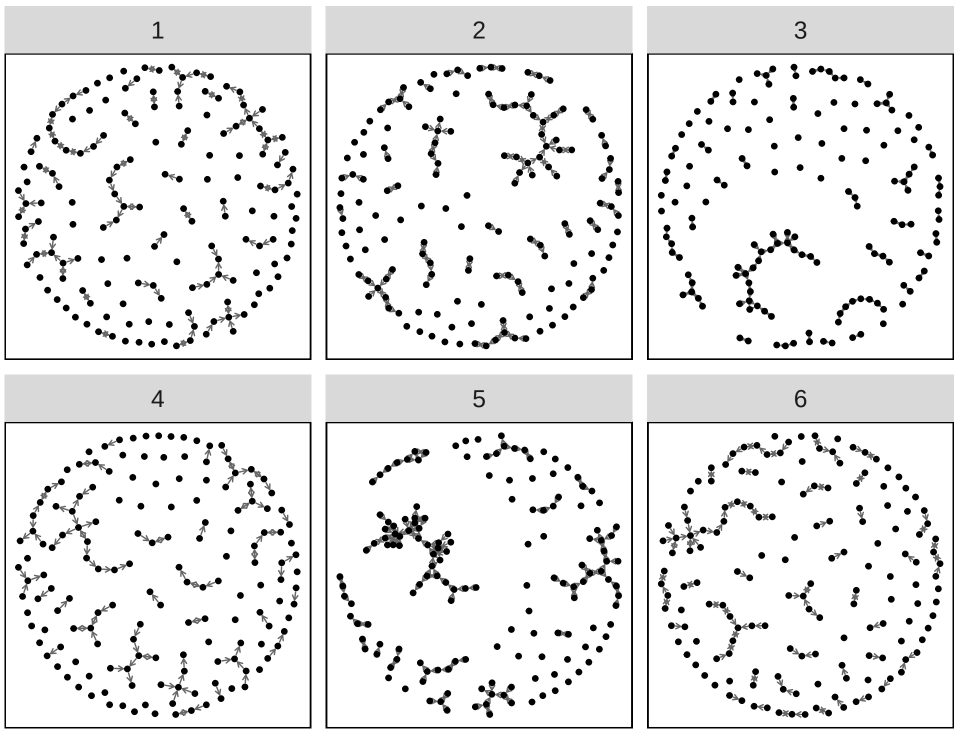
simttb_gof_9_3



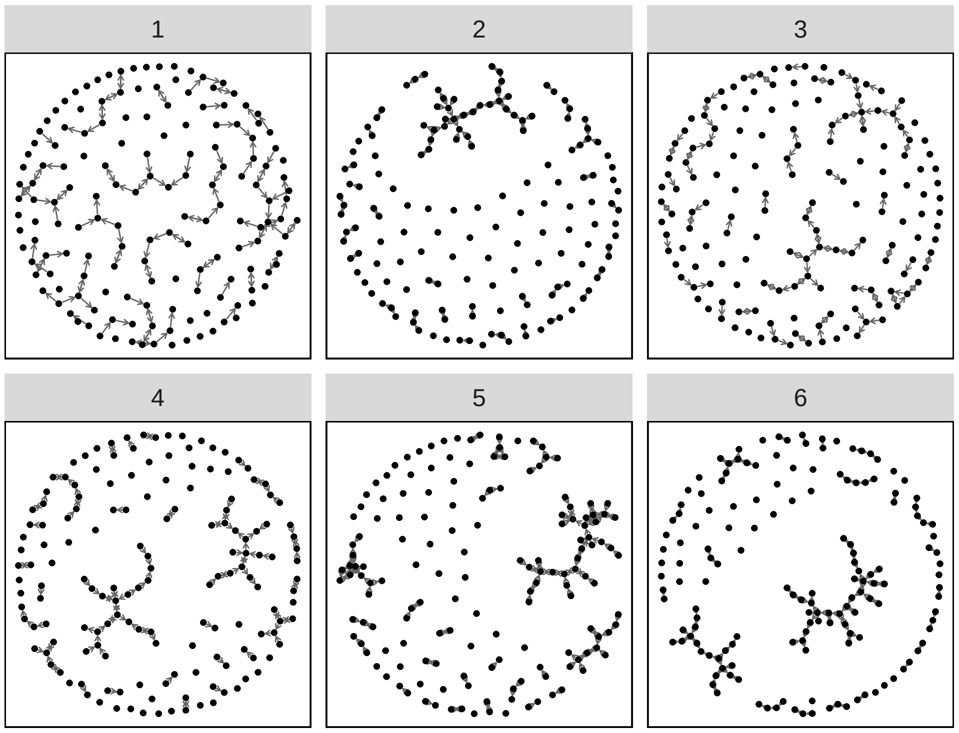

simttb_neg_easy_1
1
2
3
4
5
6


simttb_neg_easy_2
1
2
3
4
5
6

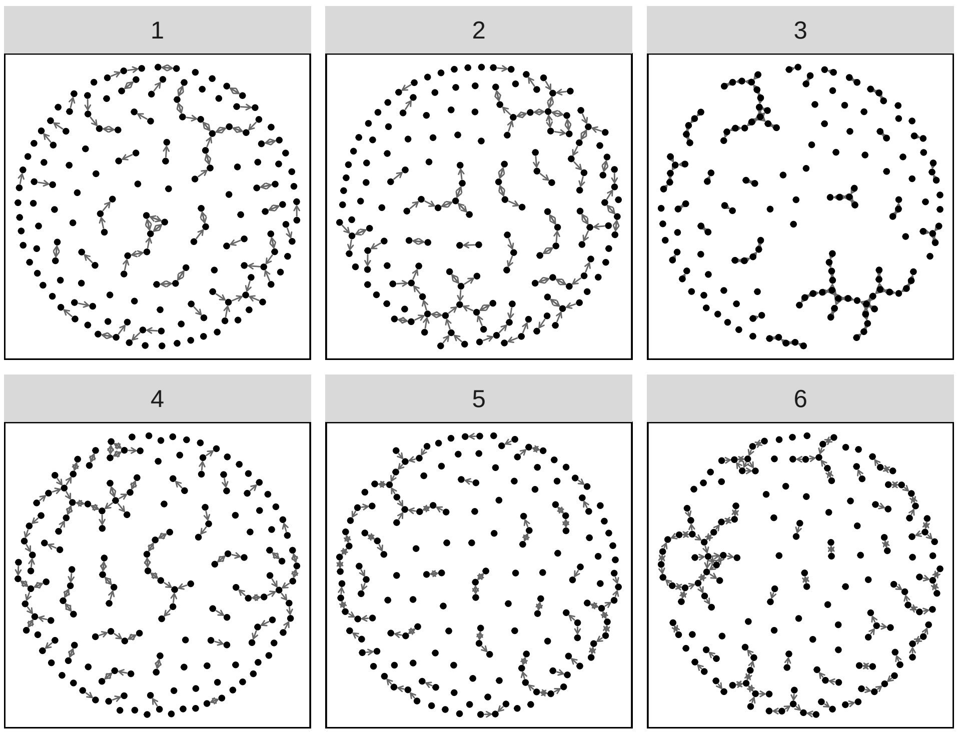
simttb_neg_easy_3



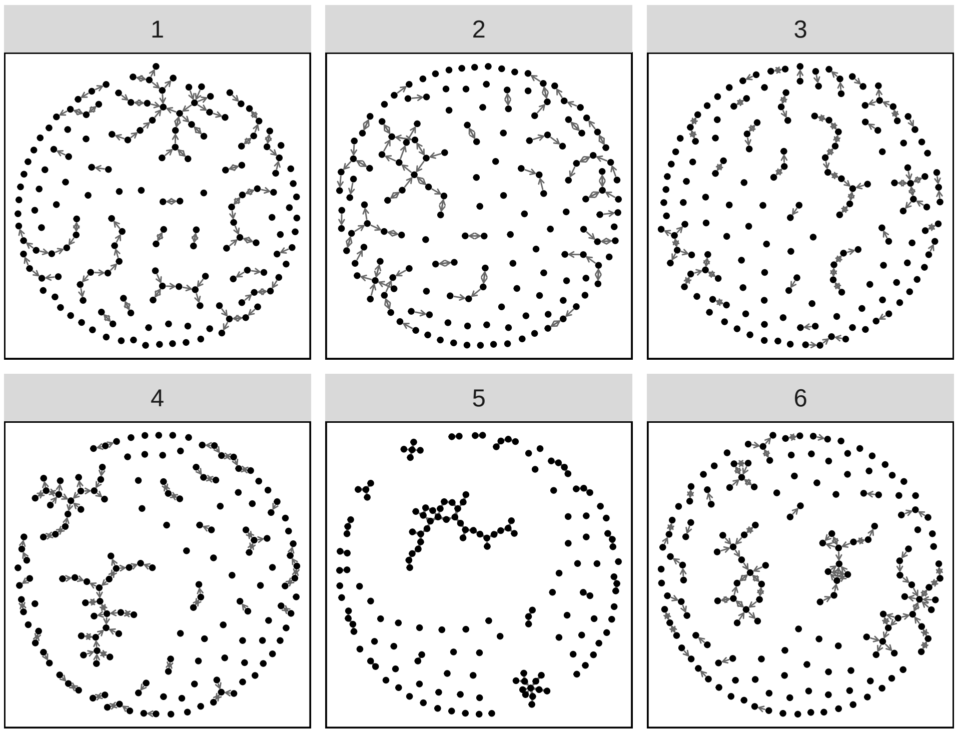
simttb_neg_hard_1



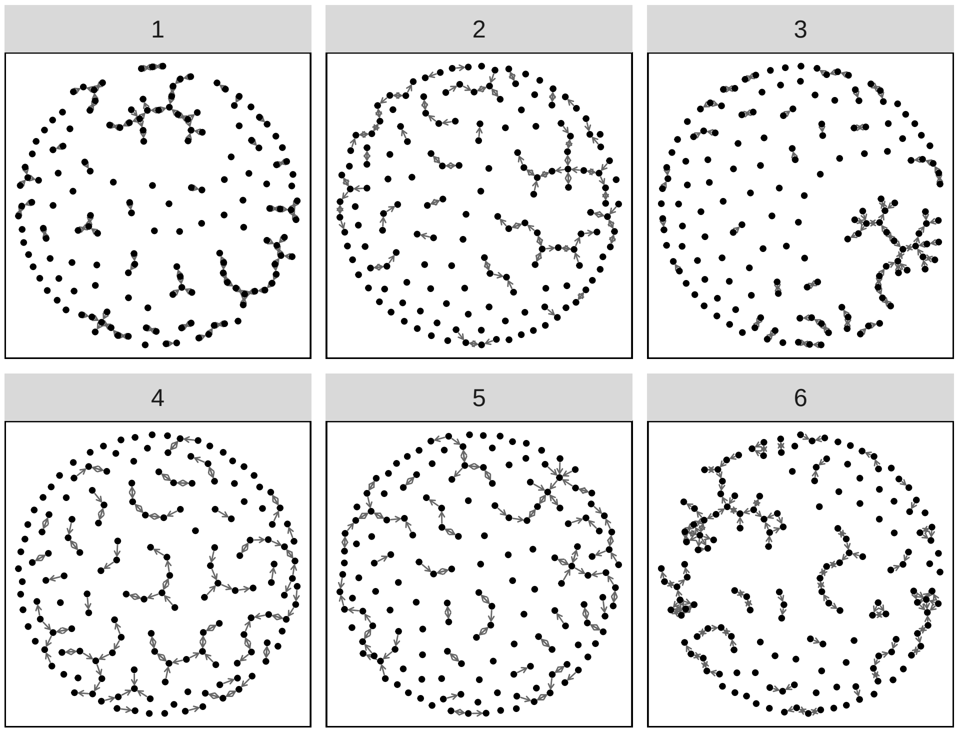
simttb_neg_hard_2



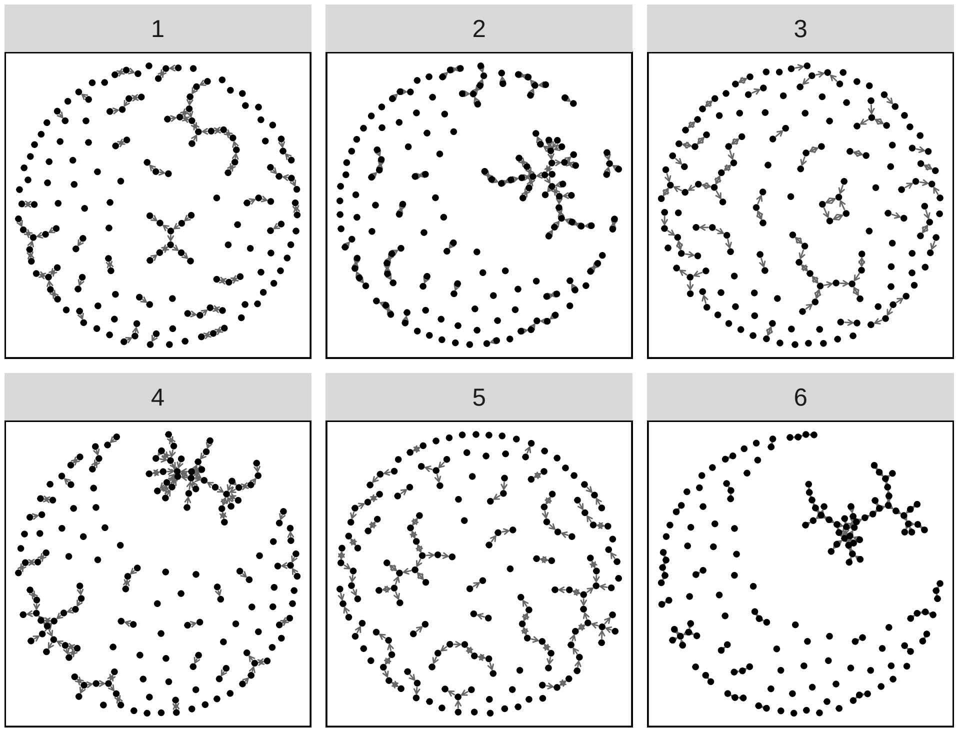
simttb_neg_hard_3



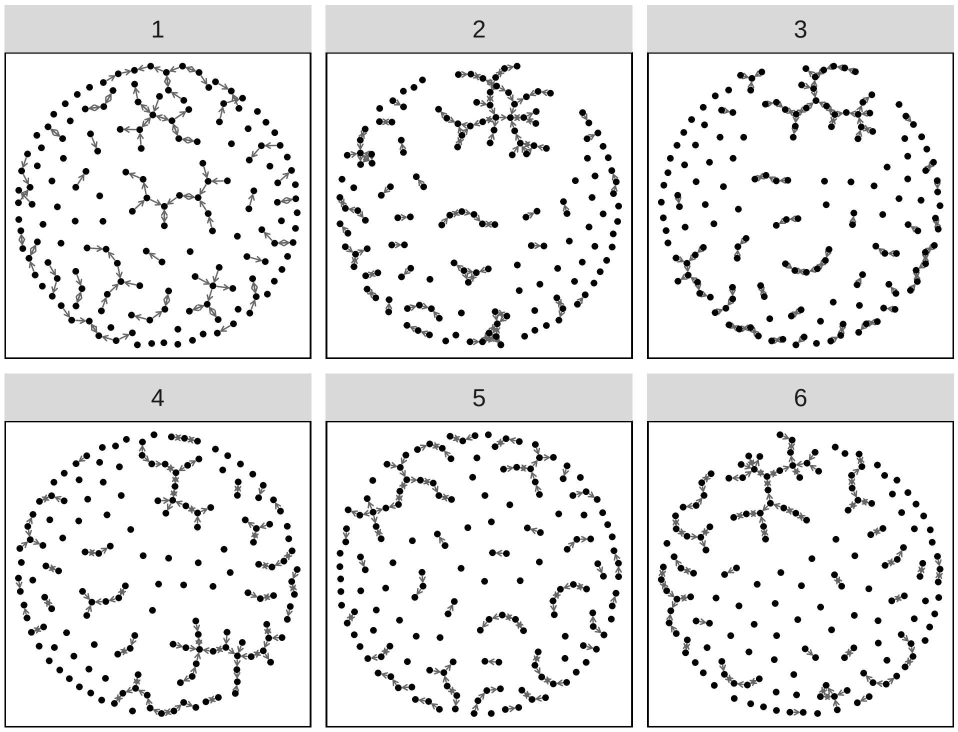
simttb_neg_med_1



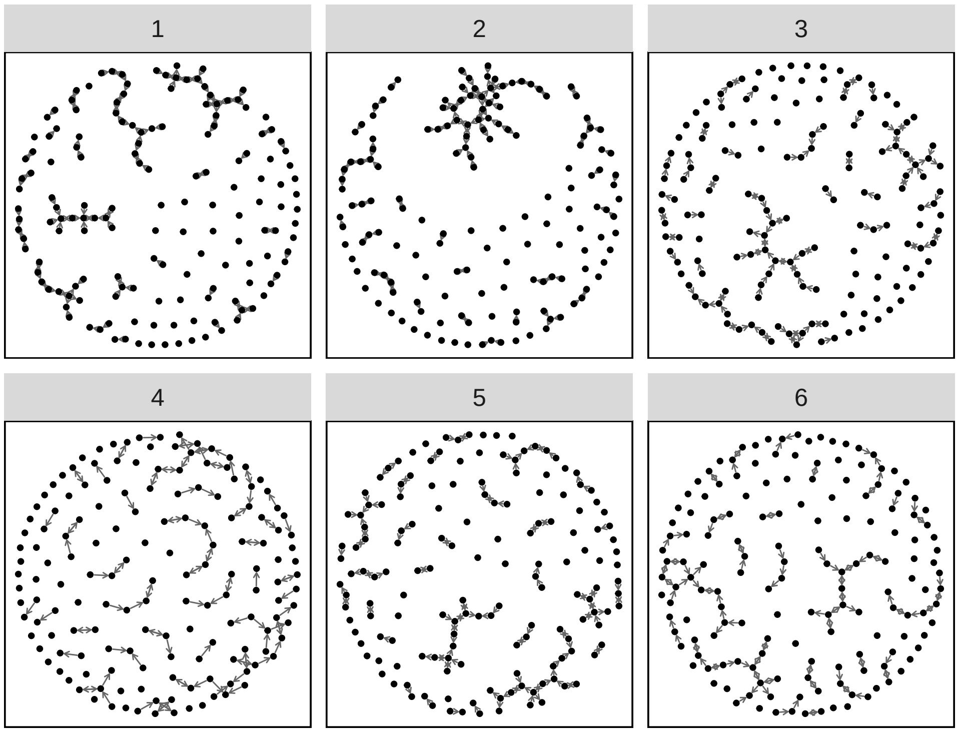
simttb_neg_med_2



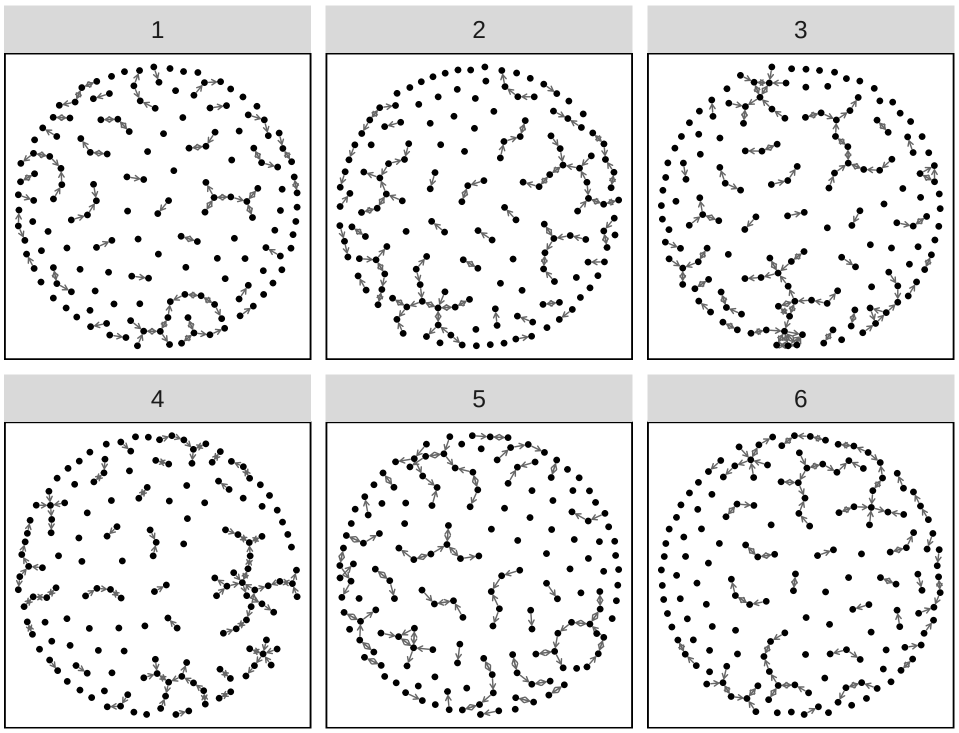
simttb_neg_med_3



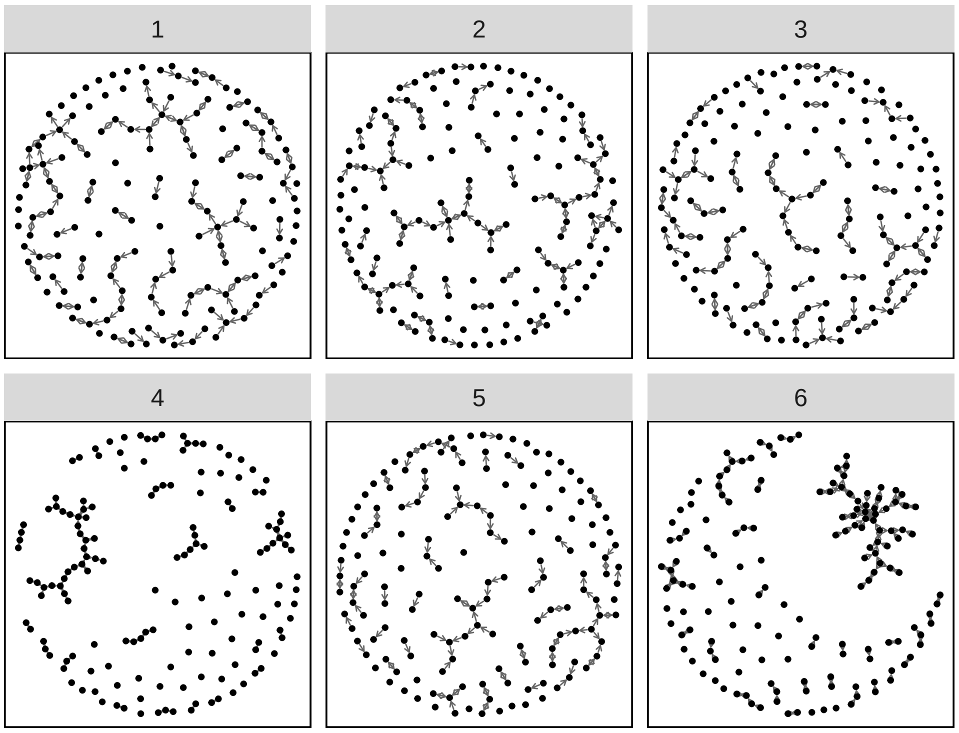
simttb_pos_easy_1



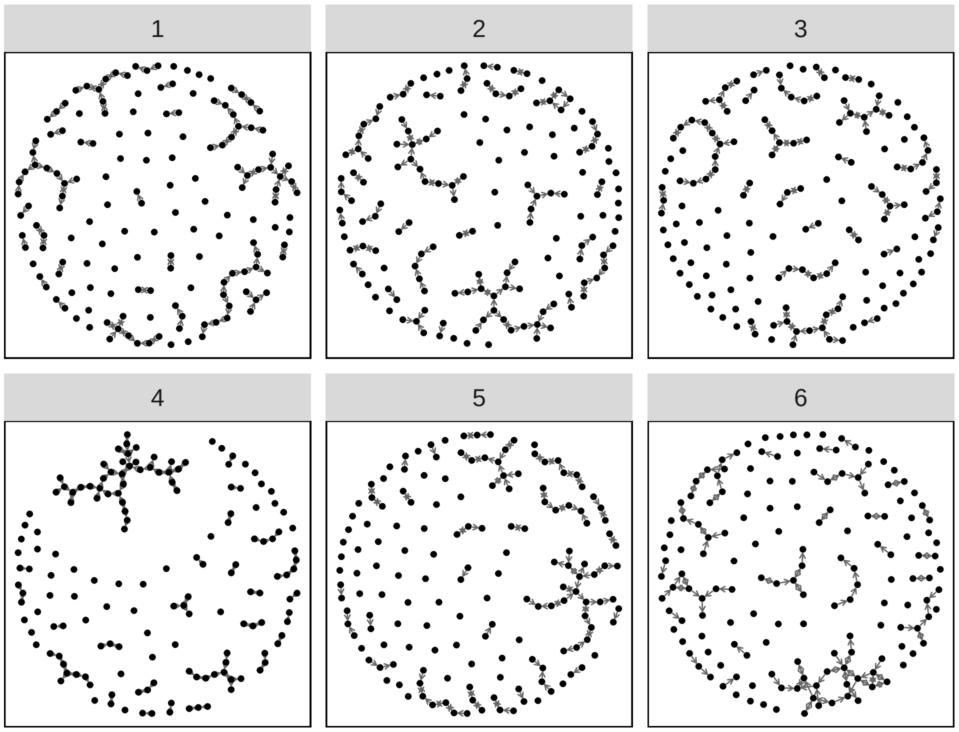
simttb_pos_easy_2



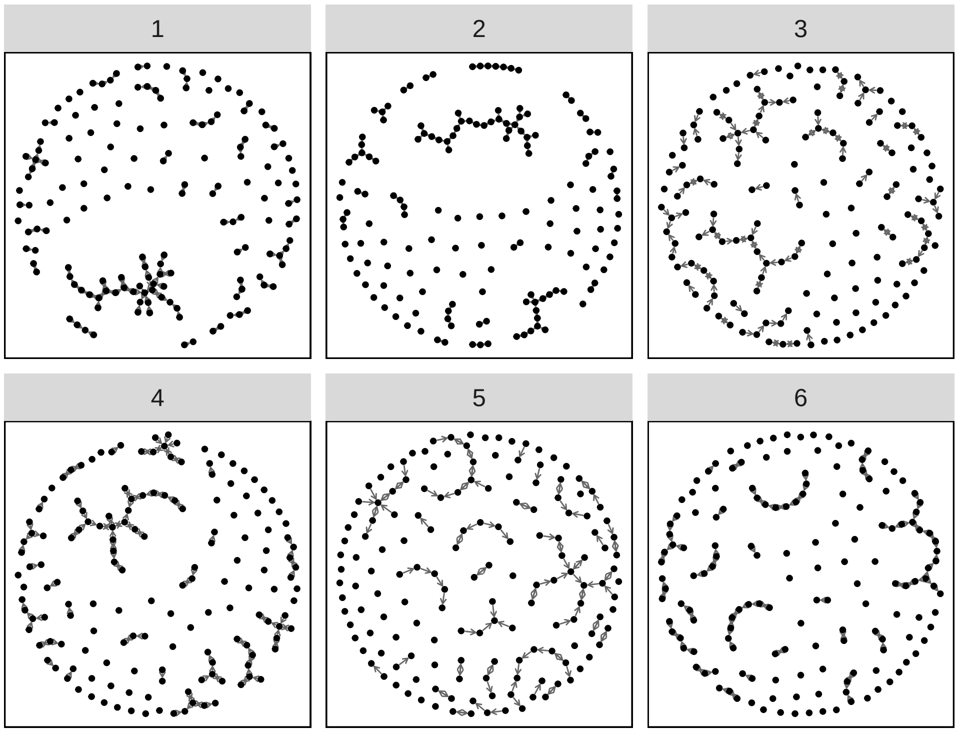
simttb_pos_easy_3



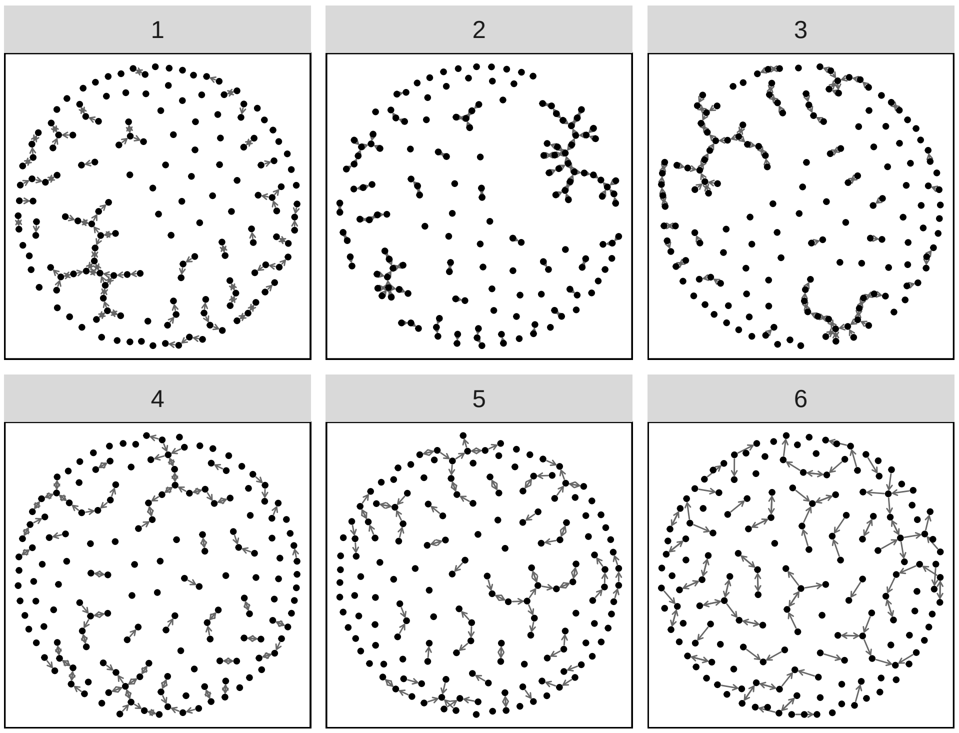
simttb_pos_hard_1



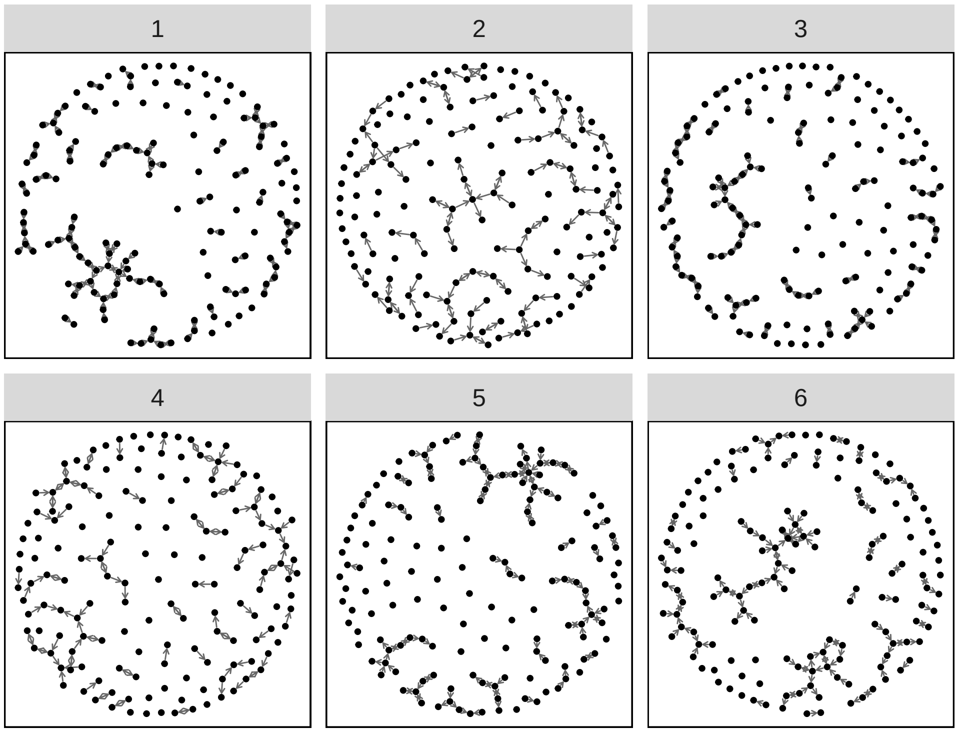
simttb_pos_hard_2



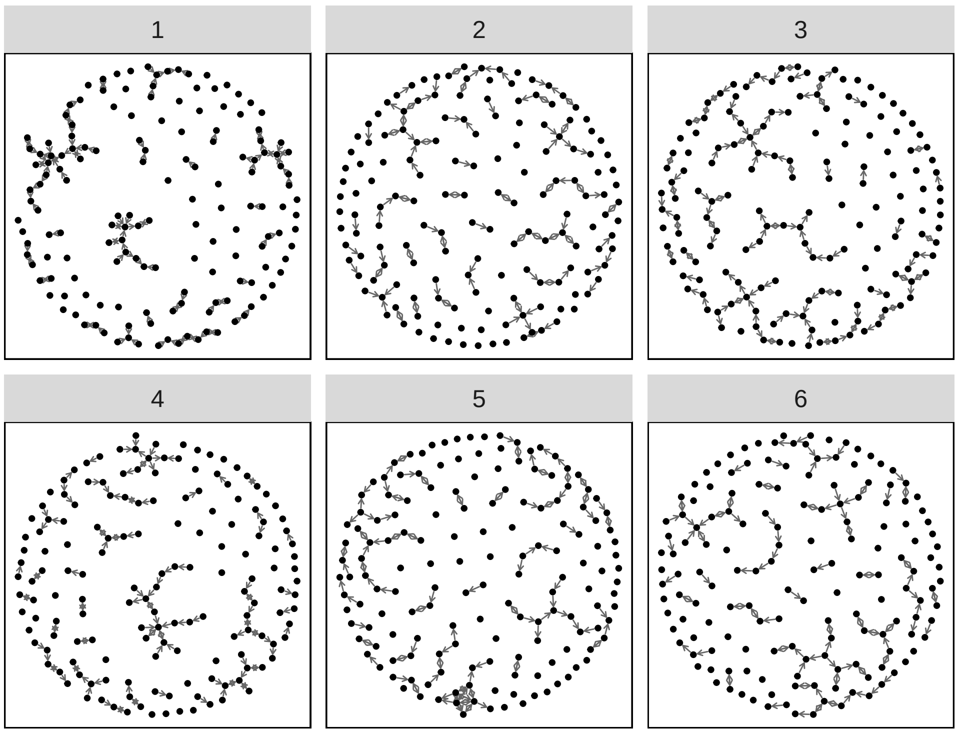
simttb_pos_hard_3



simttb_pos_med_1



simttb_pos_med_2



simttb_pos_med_3