

Network Visualization with `ggplot2`

by Sam Tyner, François Briatte and Heike Hofmann

Abstract This paper explores three different approaches to visualize networks by building on the grammar of graphics framework implemented in the `ggplot2` package. The goal of each approach is to provide the user with the ability to apply the flexibility of `ggplot2` to the visualization of network data, including through the mapping of network attributes to specific plot aesthetics. We will show that these additions contribute to the study of networks by making visualization of networks intuitive, both for users who are familiar with `ggplot2` and for those who are not.

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 `statnet` by ?, `igraph` by Csardi and Nepusz (2006), `sna` by Butts (2014), and `network` by Butts et al. (2014) (see also 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 heavily on 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 “traditional” 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 traditional data structures, they could find the learning curve for these packages burdensome. The packages we have written have one primary purpose: to create beautiful network visualizations by providing a wrapper 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, (cf. Tamassia, 2013) but rather it is on implementing existing graph drawing capabilities in the `ggplot2` framework, using the more traditional data frame structure.

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 by Horikoshi and Tang (2015) for visualizing statistical models, the package `GGally` by Schloerke et al. (2014), which encompasses various complementary visualization techniques to `ggplot2`, and the `ggbio` and `ggtree` Bioconductor packages by Yin et al. (2012) and Yu et al. (submitted), 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 the most 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 section `Brief introduction to networks` introduces the basic terminology of networks and illustrates their ubiquity in natural and social life. The next section `Three implementations of network visualizations` then discusses the logic behind each of the three approaches that we offer. The section `Examples` extends that discussion through several examples ranging from simple to complex networks, for which we provide the code corresponding to each approach alongside

¹In order to give an indication of how large the user base of `ggplot2` is, we looked at its usage statistics from January 1, 2015 to December 31, 2015 (see <http://cran-logs.rstudio.com/>). Over this period, the `ggplot2` package was downloaded over 1.75 million times from CRAN, which amounts to over 6,200 downloads per day. Over 700 R packages list `ggplot2` as a dependency or as a suggestion.

²Version 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.

its graphical result. We follow with some considerations of runtime behavior in plotting networks in the section [Some considerations of speed](#) before closing with a discussion.

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 terms “nodes” and “actors” to refer to vertices, as well as the terms “ties” and “relationships” to refer to edges. 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 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, 1998).

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), is made up 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.

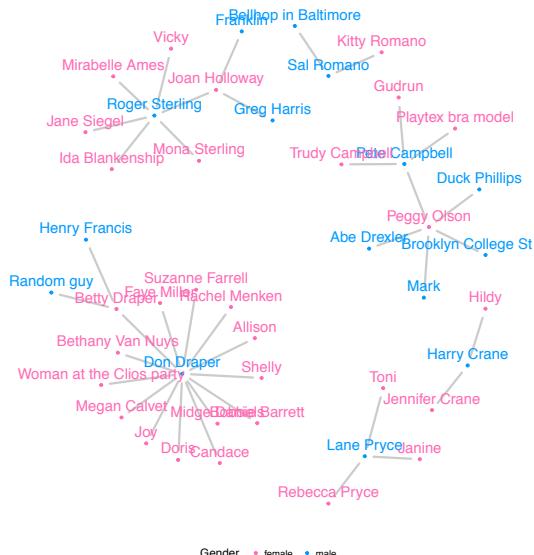


Figure 1: Graph of the characters in the show *Mad Men* who are linked by a romantic relationship.

Figure 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 can help 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.

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. 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 structural component in an independent **geom** and layers them to create the visualization. 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.

ggnet2

The **ggnet2** function is an improved version of the **ggnet** function. Both functions are part of the **GGally** package (Schlöerke et al., 2014). A detailed description of the **ggnet2** function is available from within the package as a vignette.

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 user. 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’. As a consequence, 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** (with the exception of hypergraphs or multiplex graphs), or any data structure that can be coerced to an object of that class, 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). 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 global value, such as ‘15’ or ‘dashed’ a vector of global values, or the name of an edge or vertex attribute, in which case the values of that attribute are used as the mapping aesthetic.

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 `%<attribute>%` and

`%v%` operators respectively.³ 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.

geomnet

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 `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, but only needs to specify the essential elements necessary for calculating a layout. 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` and `layout.par`.

In network analysis there are usually two sources of information: one data set consisting of a description of the actors, represented as the vertices in the network and vertex attributes, and another data set detailing the relationship between these actors, i.e. it consists of the edge list and any additional edge attributes. We make two important assumptions about the way this information is encoded for use in `geomnet`. First, we assume that the vertices are labelled using an ‘id’ column. Second, we assume that the edges are described as connection between vertices via their respective ‘id’ labels in a ‘from’ and a ‘to’ column. Thus, 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 are using the convention that all of the vertex information is merged into the edge data set using the ‘from’ as the reference column in a full join operation. Generally, there will be some vertices that are sinks in the network because they only show up in the ‘to’ column. We can easily accommodate for these by adding artificial edges in the data set that have missing information for the ‘to’ column.

Fortunately, R provides functionality that allows for an easy way of producing the required result: `merge` (in base R), `join` (in `plyr` see Wickham, 2011) and `full_join` (in `dplyr`, see Wickham and Francois, 2015) can be used. In `merge` the argument `all` needs to be set to `TRUE`; in `join`, the parameters have to be set to `type = "full"` and `all = TRUE`. These operations are equivalent to a ‘full join’ in SQL terminology.

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.

Parameters and aesthetics

Parameters that are currently implemented in `geom_net` are:

- **layout:** the `layout` parameter takes a character value corresponding to the possible layouts in the `sna` package that are available within the `gplot.layout.*()` family of functions. The default layout 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 to `geom_net`. If the user wishes to create small multiples using `ggplot2` facets, they can use `fiteach`, a logical value specifying whether

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

each panel's data should be fit separately (default) or whether the same layout should be used for all of the small multiples.

- **vertices:** any of `ggplot2`'s aesthetics relating to points: `colour`, `size`, `shape`, `alpha`, and `stroke` are available and used for specifying the appearance of nodes in the network.
- **edges:** for edges we distinguish between two different sets of aesthetics: aesthetics that only relate to line attributes, such as `linewidth` and `line_type`, 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 `selfies` 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:** `label` can be used as either a logical parameter or as a data variable, in which case it is assumed that the associated variable consists of the character strings to be used for labeling the nodes. Again, 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`. With the help of the package `ggrepel` by [Slowikowski \(2016\)](#) 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_repel`, respectively.

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 (see [Wickham, 2009](#), p. 67, Table 4.6).

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

- `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`.

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

Both the `arrow.gap` and the `by` arguments are further illustrated in the remainder of this paper.

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 are going to 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)
```

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

⁴One 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).

corresponding to vertex. This network is shown in Figure 2. The code to produce each one of the networks is shown below:

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

# plot with ggnet2 (Figure 2a)
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")

# plot with geomnet (Figure 2b)
set.seed(12252016)
ggplot(data = blood$edges, aes(from_id = from, to_id = to)) +
  geom_net(colour = "darkred", layout = "circle", label = TRUE, size = 15,
           directed = TRUE, vjust = 0.5, labelcolour = "grey80",
           arrowsize = 1.5, linewidth = 0.5, arrowgap = 0.05,
           selfies = TRUE, ecolour = "grey40") +
  theme_net()

# plot with ggnetwork (Figure 2c)
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 `selfies = TRUE`.

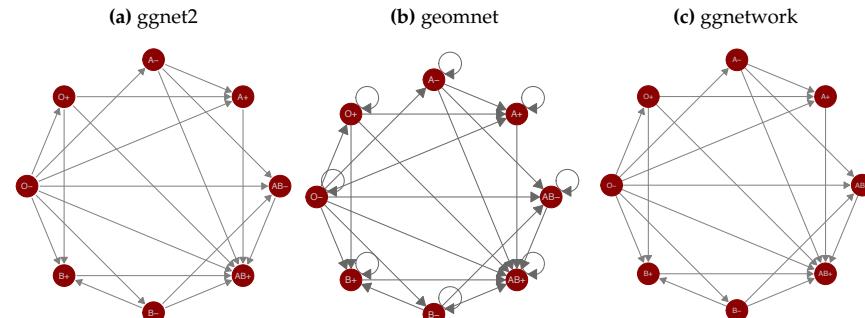


Figure 2: Network of blood donation possibilities in humans by ABO and RhD blood types.

colour and size aesthetics in Figure 2 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.

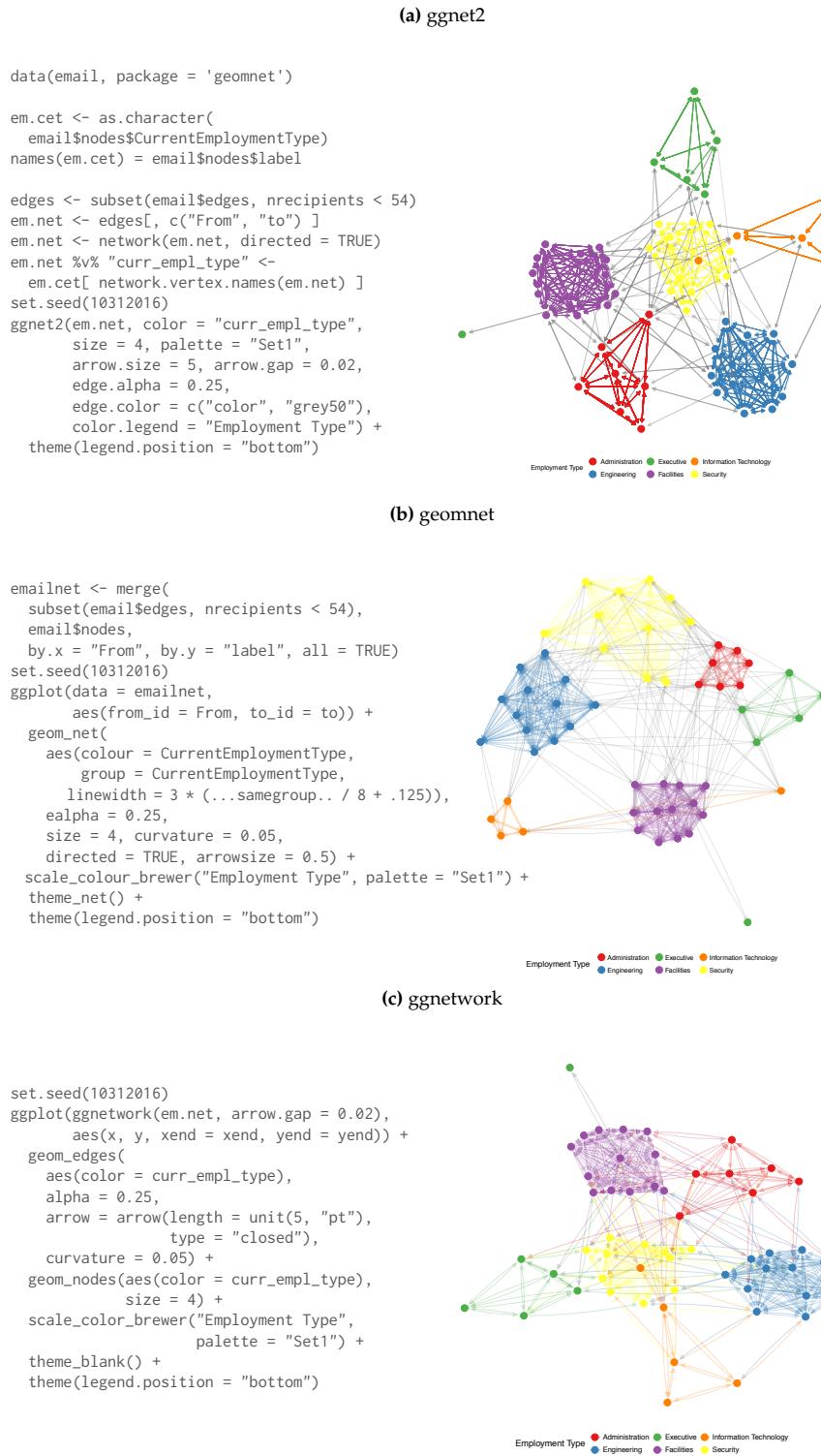


Figure 3: Email network within a company over a two week period.

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.

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 3), then we aggregate emails by day and use small multiples to allow a comparison of day-to-day behavior (shown in Figure 4).

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.

In Figure 3 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 accordingly. 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 3, 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 3, 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 shows email networks associated with each day of the work week. The code for these visualization is below.

```
# ggnet2 code for the email network faceted by day as shown in fig.4a

# data preparation
em.day <- subset(email$edges, nrecipients < 54)[, c("From", "to", "day") ]
em.day <- lapply(unique(em.day$day),
                 function(x) subset(em.day, day == x)[, 1:2 ])
```

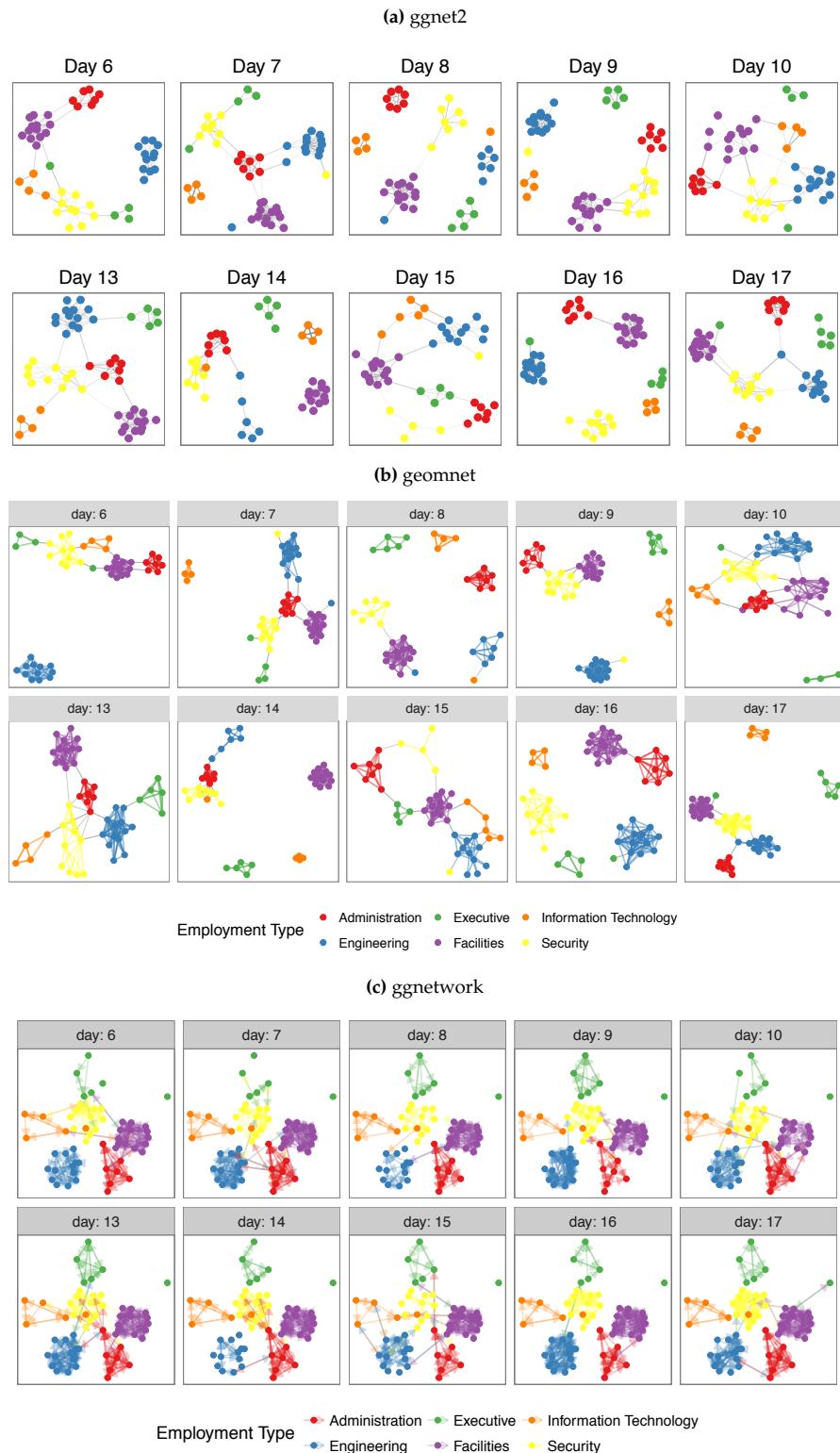


Figure 4: The same email network as in figure 3 faceted by day of the week.

```

em.day <- lapply(em.day, network, directed = TRUE)
for (i in 1:length(em.day)) {
  em.day[[ i ]] %>% "curr_empl_type" <-
    em.cet[ network.vertex.names(em.day[[ i ]]) ]
  em.day[[ i ]] %>% "day" <- unique(email$edges$day)[ i ]
}

# plot ggnet2
g <- list(length(em.day))
set.seed(7042016)
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") +
    ggtitle(paste("Day", em.day[[ i ]] %>% "day")) +
    theme(panel.border = element_rect(color = "grey50", fill = NA),
          aspect.ratio = 1)
}
gridExtra::grid.arrange(grobs = g, nrow = 2)

# geomnet code for the email network facetted by day as shown in fig.4b

# data step: making sure that there is one entry for each person on each day
employee <- data.frame(expand.grid(label = unique(email$nodes$label),
                                     day = unique(email$edges$day)))
employee <- merge(employee, email$nodes, by = "label")
emailnet <- merge(subset(email$edges, nrecipients < 54), employee,
                  by.x = c("From", "day"), by.y = c("label", "day"),
                  all = TRUE)

# creating the plot
set.seed(7042016)
ggplot(data = emailnet, aes(from_id = From, to_id = to)) +
  geom_net(aes(colour = CurrentEmploymentType,
              group = CurrentEmploymentType,
              linewidth = 2 * (...samegroup.. / 8 + .125)),
            fiteach = TRUE, ealpha = 0.5, size = 1.5) +
  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"))

# ggnetwork code for the email network facetted by day as shown in fig.4c

# create the network and aesthetics
edges <- subset(email$edges, nrecipients < 54)
edges <- edges[, c("From", "to", "day") ]
em.net <- network(edges[, 1:2])
set.edge.attribute(em.net, "day", edges[, 3])
em.net %>% "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"),
       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 that the main difference in the visualizations of Figure 4 stems from 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). In `geomnet` this is controlled via 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.

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.

The inheritance structure is plotted in figure 5. In 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.

Romantic relationships in Mad Men

The following code creates the network example given in Figure 1 in the introduction. We changed the vertex size and edge color for all vertices and edges, included vertex labels, and colored the vertices according to the character's gender.

The code for all three approaches is shown below:

```
# make the data available
data(madmen, package = 'geomnet')

# code for geom_net
# data step: merge edges and nodes
MMnet <- merge(madmen$edges, madmen$vertices,
                by.x = "Name1", by.y = "label", all = TRUE)

# create plot
set.seed(10052016)
ggplot(data = MMnet, aes(from_id = Name1, to_id = Name2)) +
  geom_net(aes(colour = Gender), size = 4, label = TRUE,
           repel = TRUE, ecolour = "grey60") +
  scale_colour_manual(values = c("#FF69B4", "#0099ff")) +
  xlim(c(-0.05, 1.05)) +
  theme_net() +
  theme(legend.position = "bottom")
```

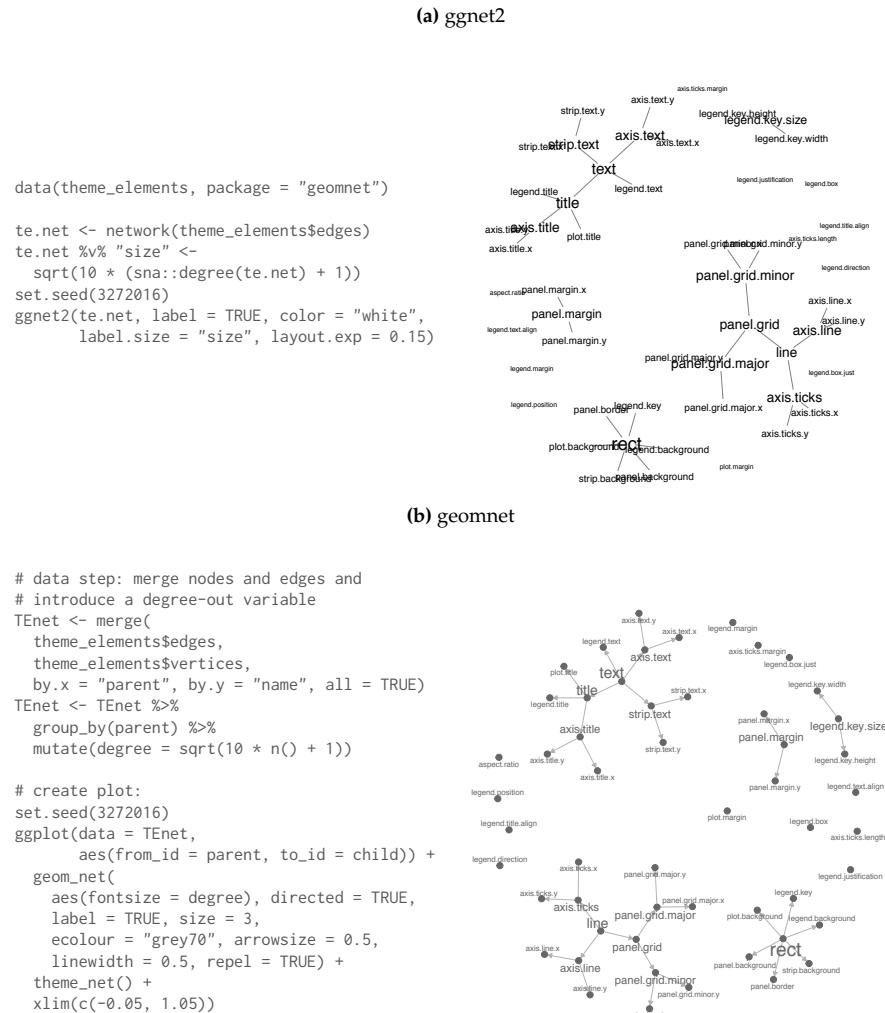


Figure 5: Inheritance structure of ggplot2 theme elements. This is a recreation of the graph found at <http://docs.ggplot2.org/current/theme.html>.

```

# data step for both ggnet2 and ggnetwork
# create undirected network
mm.net <- network(madmen$edges[, 1:2], directed = FALSE)
# gender vertex attribute
rownames(madmen$vertices) <- madmen$vertices$label
mm.net %>% "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" ],
       label = TRUE, label.color = mm.col[ mm.net %v% "gender" ],
       size = 4, vjust = -0.6)
  
```

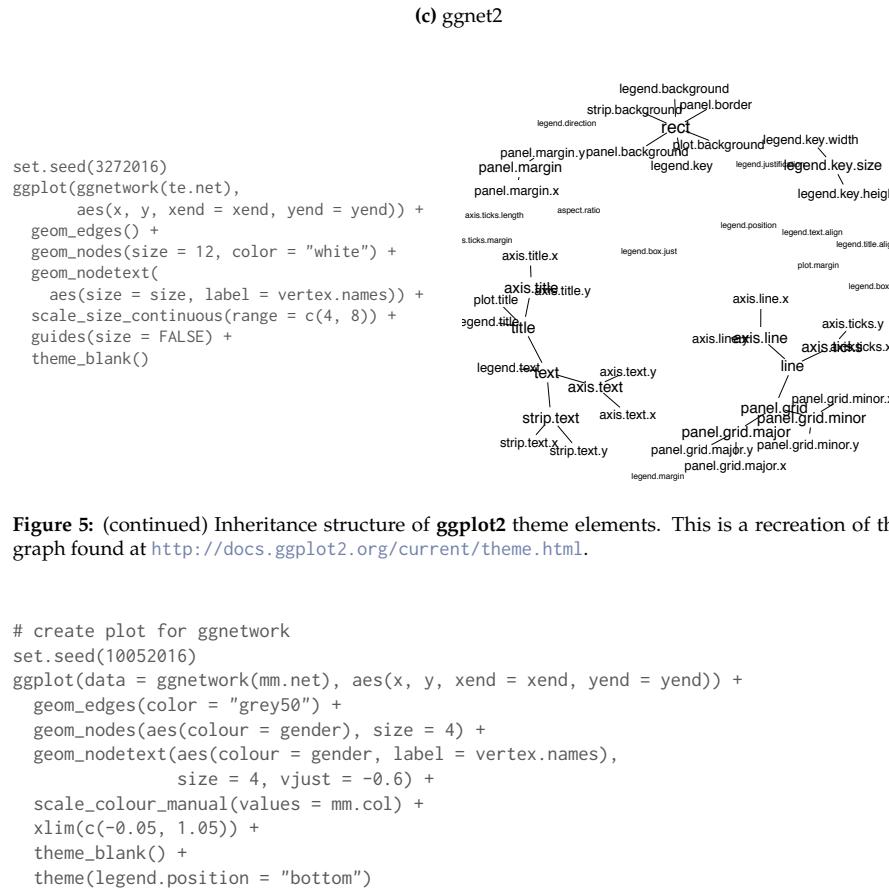


Figure 5: (continued) Inheritance structure of `ggplot2` theme elements. This is a recreation of the graph found at <http://docs.ggplot2.org/current/theme.html>.

```

# create plot for ggnetwork
set.seed(10052016)
ggplot(data = ggnetwork(mm.net), aes(x, y, xend = xend, yend = yend)) +
  geom_edges(color = "grey50") +
  geom_nodes(aes(colour = gender), size = 4) +
  geom_nodetext(aes(colour = gender, label = vertex.names),
                size = 4, vjust = -0.6) +
  scale_colour_manual(values = mm.col) +
  xlim(c(-0.05, 1.05)) +
  theme_blank() +
  theme(legend.position = "bottom")

```

The `gcookbook` package (Chang, 2013) gives another, directed form of the Mad Men network. It also encapsulates the romantic relationships between characters, but it additionally includes all advances made by one character but rejected by the other. For example, Peggy Olson made advances toward her boss Don Draper, but Don refused her. This is represented a directed edge going from Peggy to Don, but not from Don to Peggy. If the advance is reciprocated, such as between Sal Romano and the Bellhop, there are two directed edges between the two vertices. This network is shown in figure 6. It is much more densely connected than the previous Mad Men example. This network allows us to see more of the drama in the show. For instance, we can see that Roger made advances towards Betty, his business partner's wife, which is much more scandalous than what appears in the first network.

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.

The network of football games is given in figure 7. Here, the `line` type 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

(a) ggnet2

```
# data step for ggnet2 and ggnetwork
# create directed network
data(mm_directed, package = "geomnet")
mm.dir <- network(mm.directed$edges,
                   directed = TRUE)
# gender vertex attribute
rownames(mm.directed$vertices) <-
  mm.directed$vertices$label
mm.dir %>% "gender" <-
  as.character(
    mm.directed$vertices[ network.vertex.names(mm.
      )])
set.seed(1121991)
ggnet2(
  mm.dir, mode = "fruchtermanreingold", size = 3,
  color = mm.col[ mm.dir %>% "gender" ],
  label = TRUE,
  label.color = mm.col[ mm.dir %>% "gender" ],
  hjust = -0.1, legend.position = "bottom",
  layout.exp = 0.15, arrow.size = 7.5,
  arrow.gap = 0.02
)
```

(b) geomnet

```
MM2net <- merge(
  mm.directed$edges,
  mm.directed$vertices,
  by.x = "Name1", by.y = "label", all = TRUE)
set.seed(1121991)
ggplot(data = MM2net,
       aes(from_id = Name1, to_id = Name2)) +
  geom_net(
    aes(colour = Gender), directed = TRUE,
    label = TRUE, ecolour = "grey50",
    linewidth = 0.5, size = 2.5, repel = TRUE,
    layout = "fruchtermanreingold") +
  scale_colour_manual(
    values = c("#ff69b4", "#0099ff")) +
  xlim(c(-0.1, 1.1)) +
  theme_net() +
  theme(legend.position = "bottom")
```

and changing of the line types make the structure of the game network easier to view. There is one 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.

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 Newman, 2010, p. 53-54 and p. 123-127).

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), i.e. 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.

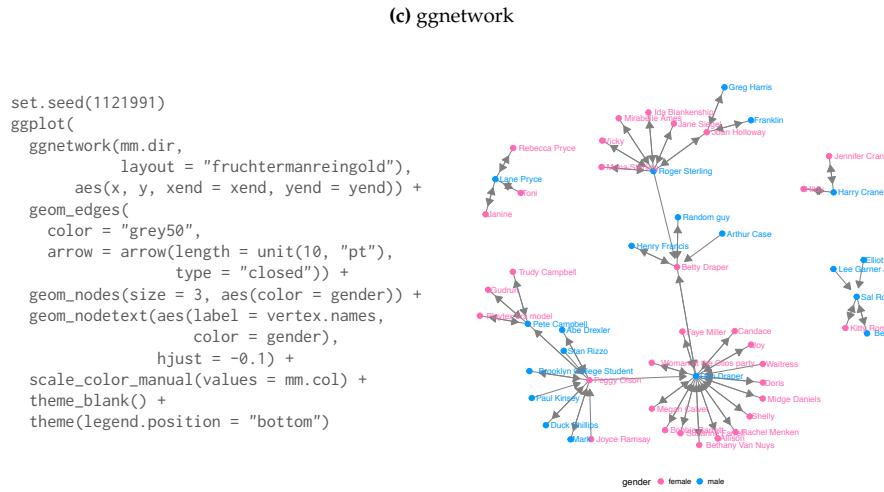
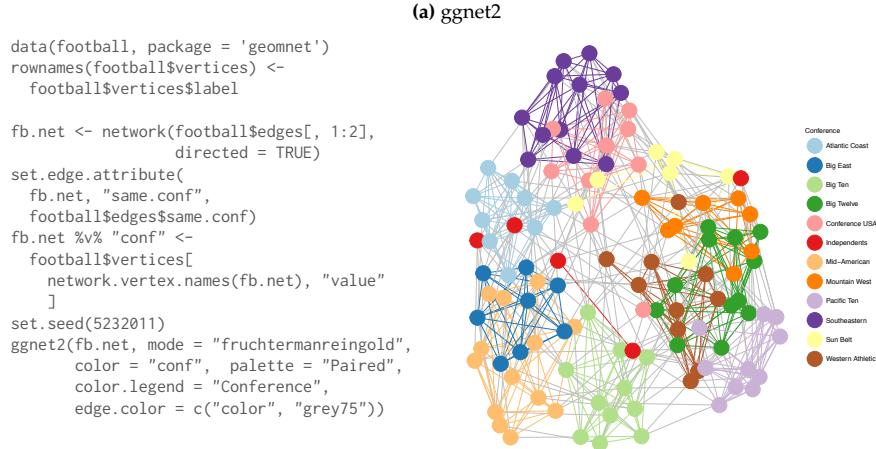


Figure 6: (continued) A directed network of relationships in Mad Men.



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 8.

```
# 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)
##   Lady Event
## 1     1     1
## 2     1     2
## 3     1     3
## 4     1     4
## 5     1     5
## 6     1     6
```

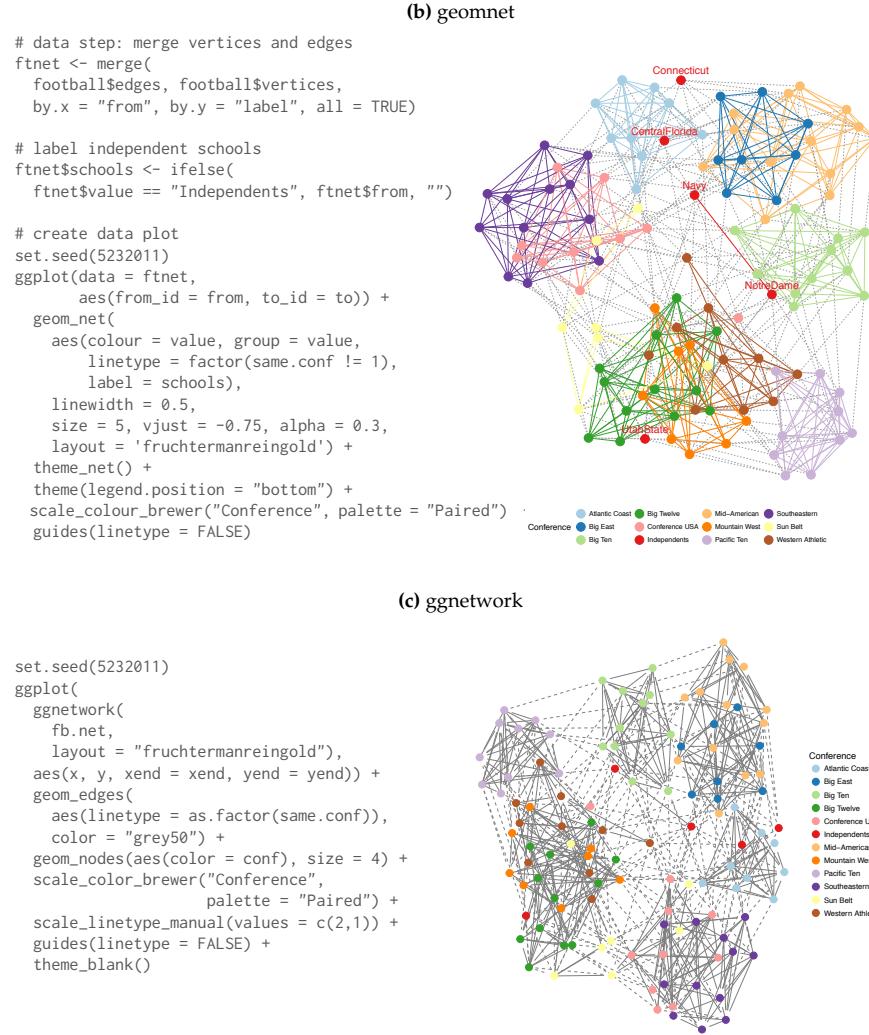


Figure 7: (continued) The network of regular season Division I college football games in the season of fall 2000. The vertices and their labels are colored by conference.

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.

Bike sharing in Washington D.C.

The data shows trips taken with bikes from the bike share company Capital Bikeshare⁵ 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 9 shows four renderings of this data. 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:

```
# code for geomnet
# data preparation
data(bikes, package = 'geomnet')
tripnet <- merge(bikes$trips, bikes$stations, by.x = "Start.station",
                  by.y = "name", all = TRUE)

tripnet$Metro = FALSE
idx <- grep("Metro", tripnet$Start.station)
tripnet$Metro[idx] <- TRUE

# plot the bike sharing network shown in Figure 7b
set.seed(1232016)
ggplot(aes(from_id = Start.station, to_id = End.station), data = tripnet) +
  geom_net(aes.linewidth = n / 15, colour = Metro,
           label = 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$trips[, 1:2 ], directed = FALSE)
network::set.edge.attribute(bikes.net, "n", bikes$trips[, 3 ] / 15)
bikes.net %v% "station" <- grepl("Metro", network.vertex.names(bikes.net))
bikes.net %v% "station" <- 1 + as.integer(bikes.net %v% "station")
```

⁵<https://secure.capitalbikeshare.com/>

(a) **geomnet**

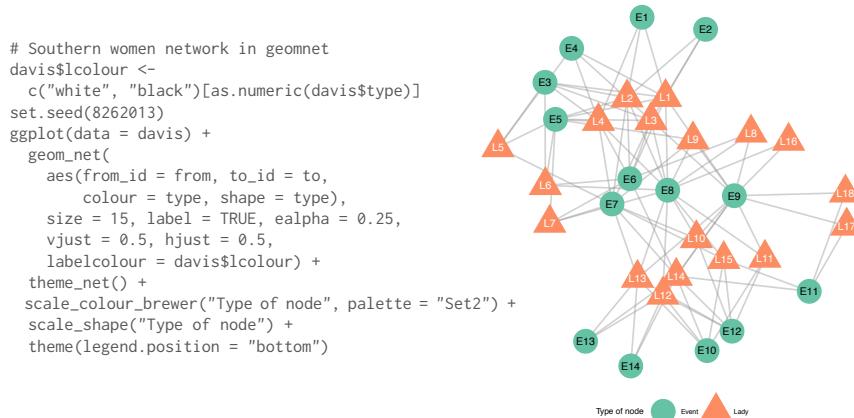
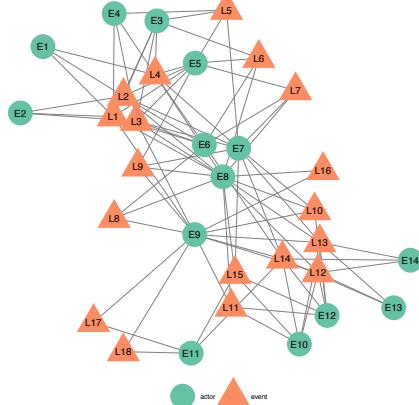


Figure 8: Graph of the Southern women data. Women are represented as orange triangles, events as green circles.

```
# 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",
       size = 15, label = TRUE) +
  theme(legend.position="bottom")
```



(b) ggnet2

```
# Southern women network in ggnetwork
set.vertex.attribute(bip, "mode",
                     c(rep("event", 14), rep("woman", 18)))
set.seed(8262013)
ggplot(data = ggnetwork(bip),
        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")
```

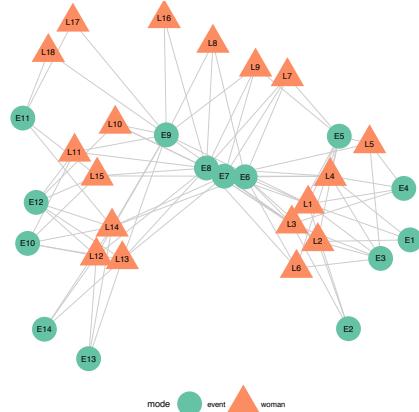


Figure 8: Graph of the Southern women data. Women are represented as orange triangles, events as green circles.

```
rownames(bikes$stations) <- bikes$stations$name
bikes.net %>% "lon" <-
  bikes$stations[ network.vertex.names(bikes.net), "long" ]
bikes.net %>% "lat" <-
  bikes$stations[ network.vertex.names(bikes.net), "lat" ]
bikes.col <- c("grey40", "darkorange")

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

# Fruchterman-Reingold placement
set.seed(1232016)
ggplot(data = ggnetwork(bikes.net),
```

```

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")

```

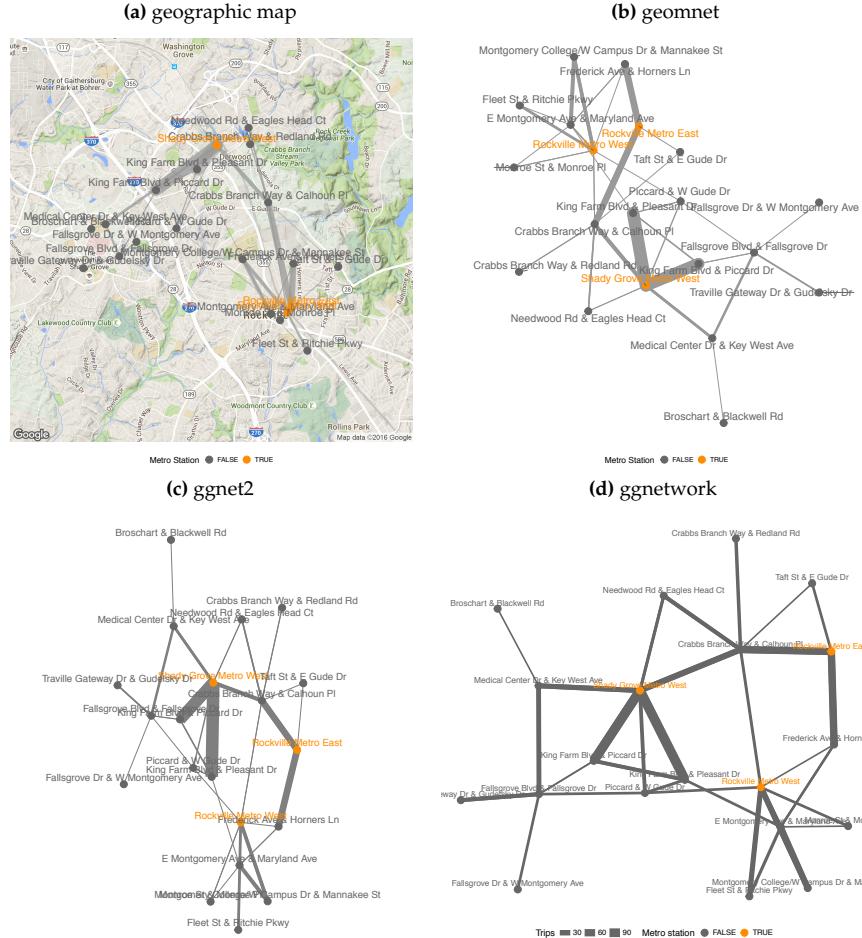


Figure 9: Network of bike trips using a geographically true representation (top left) overlaid on a satellite map, a Kamada-Kawai layout in **geomnet** (top right), a Fruchterman-Reingold layout in **ggnet2** (bottom left) and **ggnetwork** (bottom right). Metro stations are shown in orange. In both the Kamada-Kawai and the Fruchterman-Reingold layouts, metro stations take a much more central position than in the geographically true representation.

To plot the geographically correct bike network layout in **geomnet**, we use the ‘layout = NULL’ option and provide the latitude and longitude coordinates of the bike stations from the company’s data. Because all three approaches result in the same picture, we only show one of these in Figure 9a. 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 = NULL, label = TRUE,
           vjust = -0.5, ealpha = 0.5,
           aes(from_id = Start.station,
               to_id = End.station,
               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) +
  theme(legend.position = "bottom")

```

We can also make use of the option ‘layout = 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.

Protein interactions in yeast

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. Despite its size, each one of the approaches in the `ggplot2` framework can be drawn in a few hundred milliseconds. A more in-depth discussion on plotting speed follows in the next section. The proteins and the interactions between them are plotted in the networks shown in Figure 10. In this visualization, we demonstrate the layout capabilities of our functions by changing the layout to `random` and setting the distribution to “uniang”, which is a “Gaussian donut” layout (Butts, 2014). Indeed, we see a near circular area in the middle of the graph because the layout parameter has forced all of the vertices to the exterior, leaving a “donut hole” in the middle. Because all of the approaches result in nearly identical figures, only one network is shown. The code for all three approaches is given below:

```

# plot with ggnet2
data(protein, package = 'geomnet')
set.seed(10052016)
ggnet2(network(protein$edges[,1:2]), size = 2, color = "magenta",
       mode = "random", layout.par = list(dist = "uniang"),
       edge.alpha = 0.05)

# plot with geom_net
set.seed(10052016)
ggplot(data = protein$edges, aes(from_id = from, to_id = to)) +
  geom_net(alpha = 0.25, ealpha = 0.05, size = 2, colour = "magenta",
            ecolour = "grey70", linewidth = 0.5,
            layout = "random", layout.par = list(dist = "uniang")) +
  theme_net()

# plot with ggnetwork
set.seed(10052016)
ggplot(ggnetwork(protein$edges[, 1:2], layout = "random", dist = "uniang"),
       aes(x, y, xend = xend, yend = yend)) +
  geom_edges(color = "grey70", lwd = 0.5, alpha = 0.05) +
  geom_nodes(alpha = 0.25, color = "magenta", size = 2) +
  theme_blank()

```

Some considerations of speed

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 plotting time of the protein network shown in Figure 10, 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 online supplement.

We plotted the protein interaction network of Figure 10 100 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 11. 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 12 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

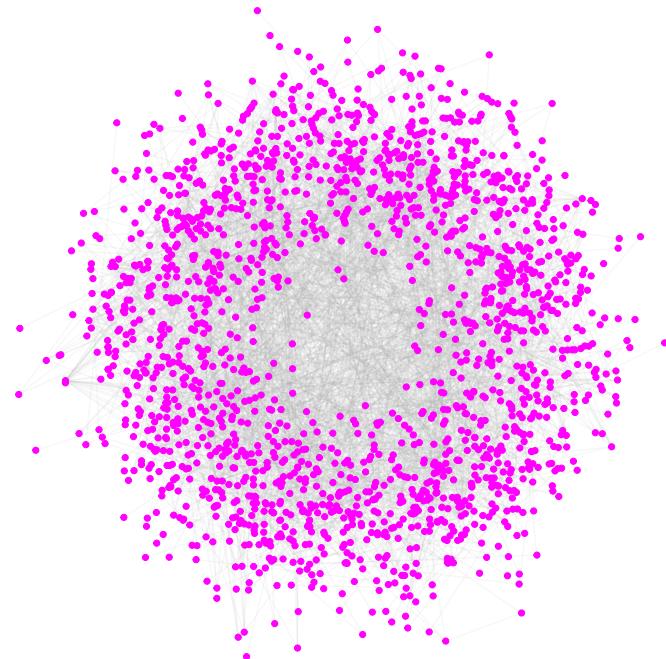


Figure 10: Protein-protein interaction network in *S. cerevisiae*. The layout is random and follows a Gaussian donut distribution.

generally among the fastest. Our three approaches generally feature in between **igraph** and **network** with **gnet2** being as fast or faster than **igraph** plotting, followed by **ggnetwork** and **geomnet**, which

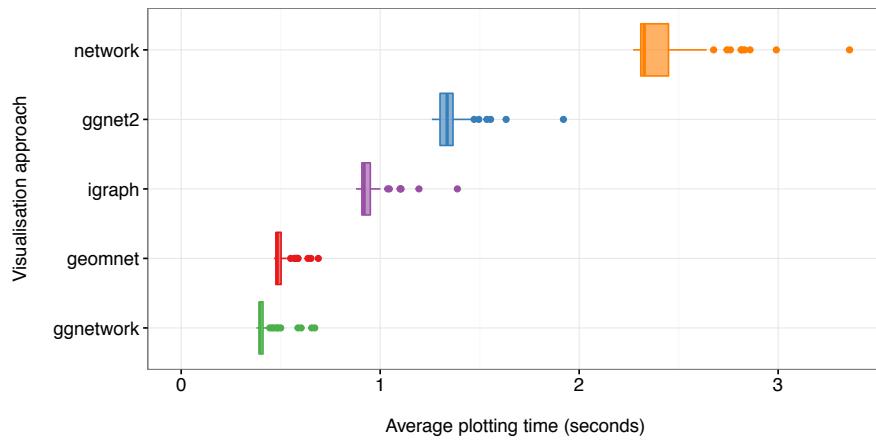


Figure 11: Comparison of the times needed for calculating and rendering the previously discussed protein interaction network in the three **ggplot2** approaches and the standard plotting routines of the **network** and **igraph** packages based on 100 evaluations each.

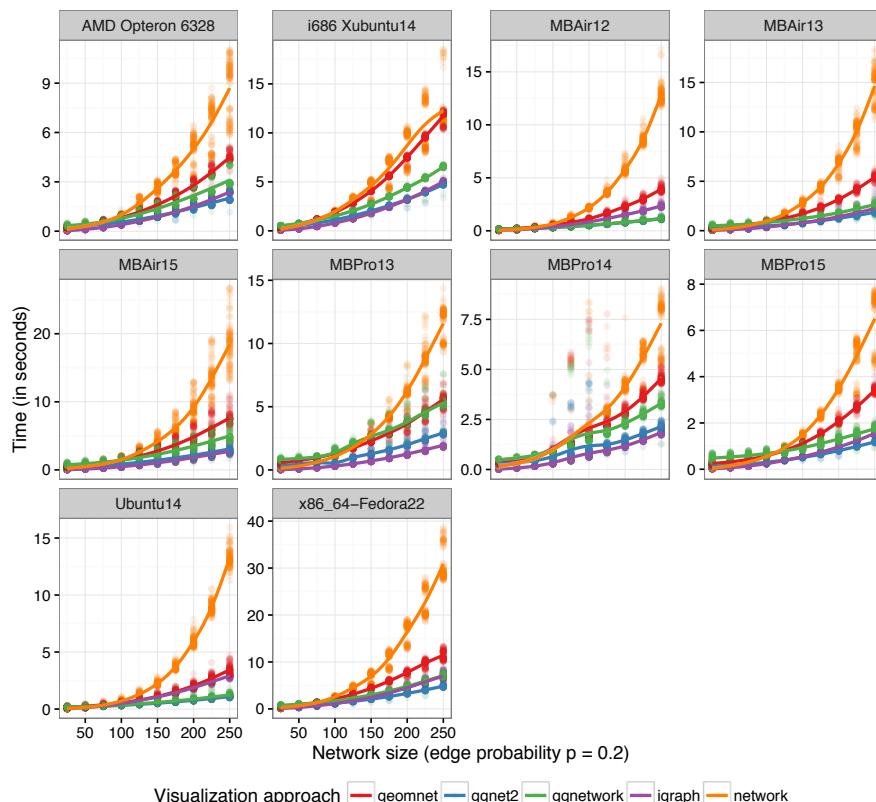


Figure 12: Plotting times of random undirected networks of different sizes under each of the available visualization approaches using their default settings. Note that each panel is scaled independently to highlight relative differences in the visualization approaches rather than speed of different hardware.

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.

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.

These 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. **sna**, unlike the **network** package, supports the use of weights in its edge list. A larger range of layouts is available through **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.

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 packages are in the way that network information is passed into the functions. For packages **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. These users might actually be looking to **ggplot2** to create high-quality visualizations that also have many additional visualization features open to them.

Users who are already familiar with **ggplot2** and some of the other ‘Hadley-verse’ packages, and who find themselves dealing with data with a network structure 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`. 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 now presents itself in form of the `arrow.gap` parameter, which allows to re-track the tip of an arrow on a directed edge. The implementation of a **ggplot2** geom for networks within **geomnet** inspired the creation of the aliased geoms of the **ggnetwork** package.

Supplementary material

Software: **ggnetwork** 0.5.1 and **geomnet** 0.0.1 were used to create the visualizations. **ggnet2** is part of **GGally** 1.0.1.

Reproducibility: All the code and data used in the examples is available in the electronic supplement, which is hosted and kept maintained at <https://github.com/sctyner/ggnet-paper/tree/master/revision/supplementary%20code>.

Bibliography

- S. Bender-deMoll. *ndtv: Network Dynamic Temporal Visualizations*, 2016. URL <https://CRAN.R-project.org/package=ndtv>. R package version 0.10.0. [p⁹]
- M. Bojanowski. *intergraph: Coercion Routines for Network Data Objects*, 2015. URL <http://mbojan.github.io/intergraph>. R package version 2.0-2. [p³]
- S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, and S. Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025–1028, April 2010. [p²]
- C. T. Butts. network: a Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2), 2008. [p¹]
- C. T. Butts. *sna: Tools for Social Network Analysis*, 2014. URL <http://CRAN.R-project.org/package=sna>. R package version 2.3-2. [p¹, 21]
- C. T. Butts, M. S. Handcock, and D. R. Hunter. *network: Classes for Relational Data*. Irvine, CA, 2014. URL <http://statnet.org/>. R package version 1.10.2. [p¹, 4]
- W. Chang. *gcookbook: Data for “R Graphics Cookbook”*, 2012. URL <https://CRAN.R-project.org/package=gcookbook>. R package version 1.0. [p²]
- W. Chang. *R graphics cookbook*. O'Reilly, Sebastopol, CA, 2013. ISBN 978-1449316952. [p², 14]
- K. Cook, G. Grinstein, and M. Whiting. Vast challenge 2014. <http://hcil2.cs.umd.edu/newvarepository/benchmarks.php>, jun 2014. [p⁹]
- G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*:1695, 2006. URL <http://igraph.org>. [p¹]
- A. Davis, B. B. Gardner, and M. R. Gardner. *Deep South: A social anthropological study of caste and class*. The University of Chicago Press, Chicago, IL, 1941. [p¹⁵]
- T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. [p³, 6]
- M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(12):7821–7826, jun 2002. [p², 14]
- M. Horikoshi and Y. Tang. *Data Visualization Tools for Statistical Analysis Results*, 2015. URL <http://CRAN.R-project.org/package=ggfortify>. R package version 0.0.4. [p¹]
- H. Jeong, S. P. M. A.-L. Barabási, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411:41–42, may 2001. [p²¹]
- B. Junker and F. Schreiber. *Analysis of Biological Networks*. Wiley Series in Bioinformatics. Wiley, 2008. ISBN 9780470253465. URL <https://books.google.com/books?id=2DloLXaXSNgC>. [p¹]
- D. Kahle and H. Wickham. ggmap: Spatial visualization with ggplot2. *The R Journal*, 5(1):144–161, 2013. URL <http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf>. [p¹]
- T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, apr 1989. [p⁴]
- M. E. J. Newman. *Networks : an introduction*. Oxford University Press, Oxford New York, 2010. ISBN 978-0199206650. [p², 15]
- T. Opsahl. *Structure and Evolution of Weighted Networks*. University of London (Queen Mary College), London, UK, 2009. URL <http://toreopsahl.com/publications/thesis/>. [p¹⁵]
- C. Prell. *Social Network Analysis: History, Theory and Methodology*. SAGE Publications, 2011. ISBN 9781446290132. URL <https://books.google.com/books?id=wZYQAgAAQBAJ>. [p¹]
- B. Schloerke, J. Crowley, D. Cook, H. Hofmann, H. Wickham, F. Briatte, M. Marbach, and E. Thoen. GGally: Extension to ggplot2., 2014. R package version 0.5.0. [p¹, 3]

- K. Slowikowski. *ggrepel: Repulsive Text and Label Geoms for 'ggplot2'*, 2016. URL <https://CRAN.R-project.org/package=ggrepel>. R package version 0.5. [p5]
- R. Tamassia, editor. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013. [p1]
- D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998. [p2]
- D. J. Watts. The "new" science of networks. *Annual Review of Sociology*, 30:243–270, 2004. [p1]
- H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN 978-0-387-98140-6. URL <http://had.co.nz/ggplot2/book>. [p1, 5, 12]
- H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29, 2011. URL <http://www.jstatsoft.org/v40/i01/>. [p4]
- H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2015. URL <http://CRAN.R-project.org/package=dplyr>. R package version 0.4.3. [p4]
- L. Wilkinson. *The Grammar of Graphics*. NY: Springer, New York, 1999. [p1]
- T. Yin, D. Cook, and M. Lawrence. ggbio: an R package for extending the grammar of graphics for genomic data. *Genome Biology*, 13(8):R77, 2012. [p1]
- G. Yu, D. Smith, H. Zhu, Y. Guan, and T. T.-Y. Lam. ggtree: an R package for visualization and annotation of phylogenetic tree with different types of meta-data. *Methods in Ecology and Evolution*, submitted. [p1]

Samantha Tyner
Department of Statistics and Statistical Laboratory
Iowa State University
United States
sctyner@mail.iastate.edu

François Briatte
European School of Political Sciences
Catholic University of Lille
France
francois.briatte@univ-catholille.fr

Heike Hofmann
Department of Statistics and Statistical Laboratory
Iowa State University
United States
hofmann@mail.iastate.edu