# 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. By incorporating networks in the **ggplot2** framework, these approaches (1) allow users to enhance networks with additional information on edges and nodes, (2) give access to the strengths of **ggplot2**, such as layers and facets, and (3) convert network data objects to the more familiar data frames.

## Introduction

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, see also). 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, 2016). 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 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 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]

---

[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**.

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 (2016), 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. (2017), 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. 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 structure and capabilities of 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 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 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, 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), 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.

---

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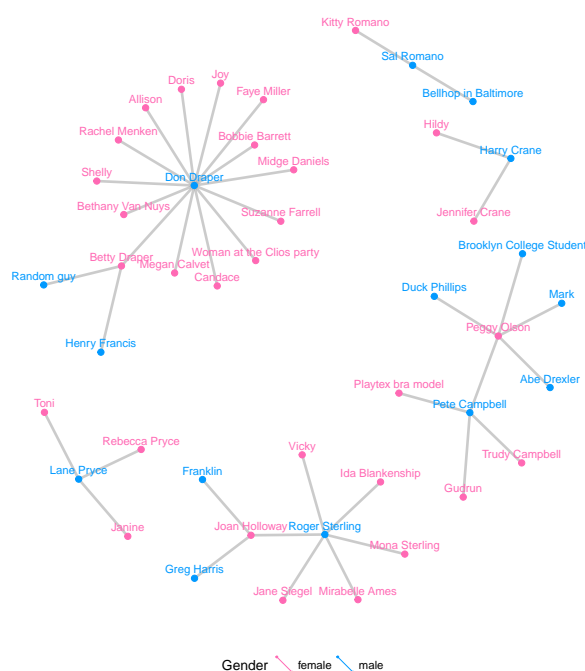
**Figure 1:** Graph of the characters in the show Mad Men who are linked by a romantic relationship.

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 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.

## 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 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 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 1 using ggnet2 is presented below.

```
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$edges[, 1:2], directed = FALSE)
mm.net # glance at network object
##  Network attributes:
##    vertices = 45
##    directed = FALSE
##    hyper = FALSE
##    loops = FALSE
##    multiple = FALSE
##    bipartite = FALSE
##    total edges= 39
##      missing edges= 0
##      non-missing edges= 39
##
##  Vertex attribute names:
##      vertex.names
##
## No edge attributes
# create node attribute (gender)
rownames(madmen$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)
```

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 the code above, 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 above, 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 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 above. 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.

### geomnet

```
# 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")
```

### 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, 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

---

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

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 the code above 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
## Betty Draper Betty Draper female
## Don Draper     Don Draper   male
## Harry Crane   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.

## 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 above 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 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 (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_label`, respectively. Using `repel` can be extremely useful when the networks are dense or the labels are long, as in Figure 1, helping to solve a common problem with many network visualizations.

## ggnetwork

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

```
# 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")
```

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, 2016, 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`. Three of the four availble aliased geoms are used above 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 each other, and allows the user to visualize the network structure as a function of a specific edge attribute, like a temporal attribute.

## 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)
```

---

[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).

| Functionality | ggnet2<br><br>(**GGally**) | geom_net<br><br>(**geomnet**) | geom_nodes,<br>geom_edges, etc<br>(**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 code-geom_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 |

**Table 1:** Comparing the three different package side-by-side.

```
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 corresponding *to* vertex. This network is shown in Figure 2. The code to produce each one of the networks is shown above Figure 2. 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 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.
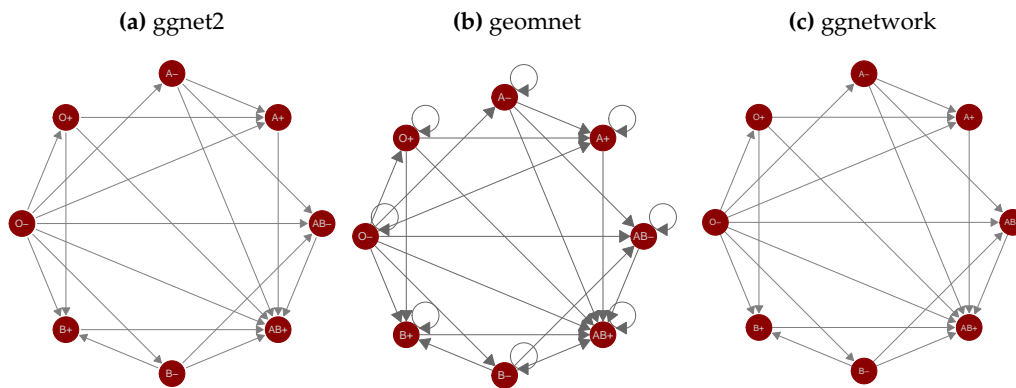
**(a)** ggnet2      **(b)** geomnet      **(c)** ggnetwork

**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.

### 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.

**(a)** 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")}
```
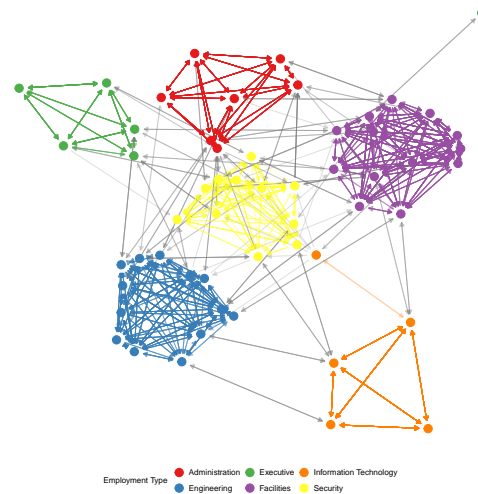
**(b)** geomnet

```
# 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")
```
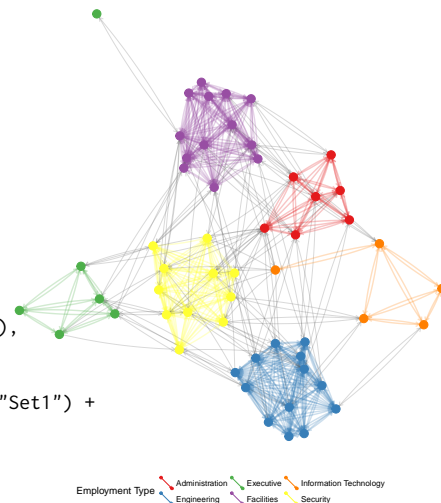
**(c)** ggnetwork

```
# 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")
```
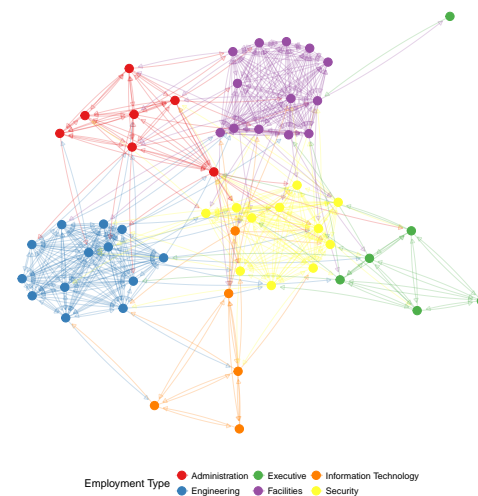
**Figure 3:** Email network within a company over a two week period.

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. 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 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 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 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 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(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 ])
```
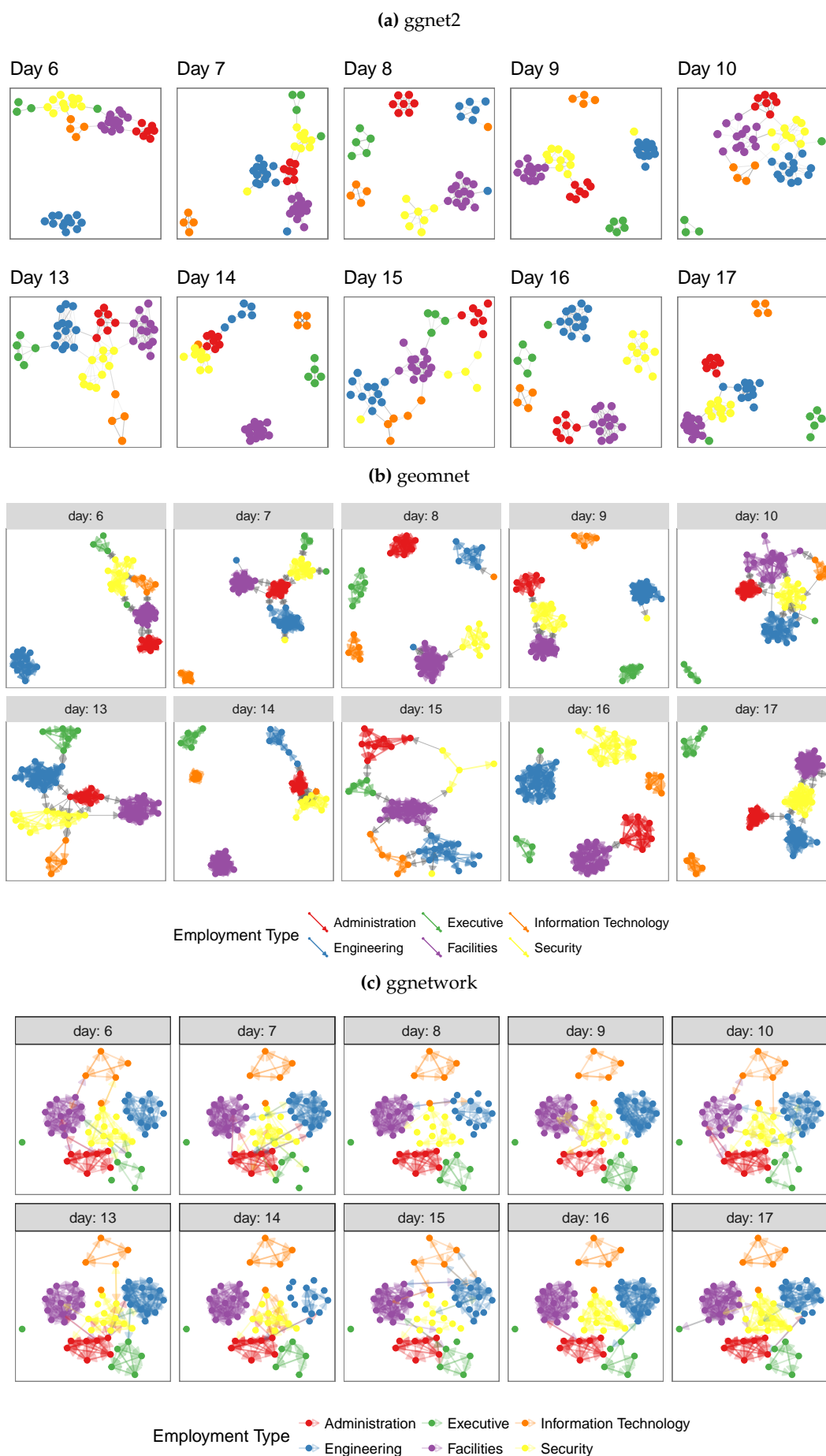
**(a)** ggnet2



**(b)** geomnet



**(c)** ggnetwork



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

```
# 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(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(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: 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.

**ggplot2 theme elements**

This example comes from the `theme()` help page in the **ggplot2** documentation (Wickham, 2016). 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 5. 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
```

**(a)** 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")
```

**(b)** 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 5:** Inheritance structure of **ggplot2** theme elements. This is a recreation of the graph found at http://docs.ggplot2.org/current/theme.html.

**(c)** ggnetwork

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

```
head(TEnet)
## Source: local data frame [6 x 3]
## Groups: from_id [2]
##
##   from_id        to_id  degree
##    <fctr>       <fctr>   <dbl>
## 1    text        title 6.403124
## 2    text legend.text 6.403124
## 3    text    axis.text 6.403124
## 4    text   strip.text 6.403124
## 5    line    axis.line 5.567764
## 6    line  axis.ticks 5.567764
```

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 5(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.

### 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 6. Here, the `linetype` aesthetic corresponds to games that occur between teams in the same conference or different conferences.

```
#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"))
```

**(a)** ggnet2
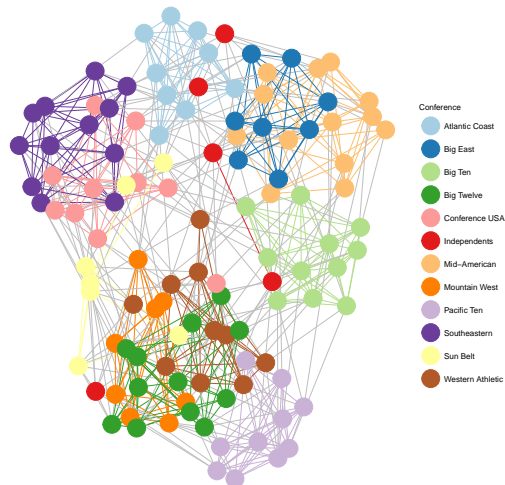


These lines are dotted and solid, respectively. We have also assigned a different color to each conference, so that the vertices and their labels are colored according to their conference. Additionally, in the first two implementations, the edges between two teams in the same conference share that conference 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 6(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.

**(b)** 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)
```
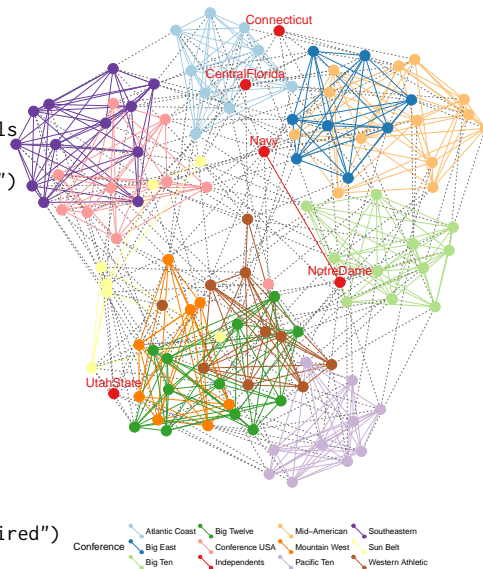


**(c)** ggnetwork

```
# 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()
```
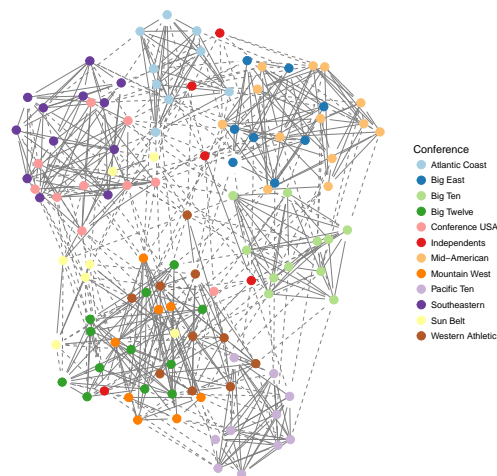


**Figure 6:** (continued) The network of regular season Division I college football games in the season of fall 2000. The vertices and their labels are colored by conference.

**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). 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 7. 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 *L*1 apart from the first event, *E*1.

```
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 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 8 shows four renderings of this data. The first is a geographically true

---

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

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
## 1                           <NA> NA 39.10210 -77.20032
## 2 Crabbs Branch Way & Redland Rd 11 39.10771 -77.15207
## 3   Needwood Rd & Eagles Head Ct 14 39.10771 -77.15207
## 4            Rockville Metro East 51 39.10771 -77.15207
## 5            Rockville Metro West  8 39.10771 -77.15207
## 6         Shady Grove Metro West 36 39.10771 -77.15207
##  Metro
## 1 FALSE
## 2 FALSE
## 3 FALSE
## 4 FALSE
## 5 FALSE
## 6 FALSE
```

Because all three approaches result in the same picture, we only show one of these in Figure 8a. 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.

## 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 9. 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 9, 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 9 100 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 10. 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 11 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.

## 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 2(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

## Acknowledgements

# Bibliography

B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2016. URL https://CRAN.R-project.org/package=gridExtra. R package version 2.2.1. [p39]

S. Bender-deMoll. *Ndtv: Network Dynamic Temporal Visualizations*, 2016. URL https://CRAN.R-project.org/package=ndtv. R package version 0.10.0. [p39]

M. Bojanowski. *Intergraph: Coercion Routines for Network Data Objects*, 2015. URL http://mbojan.github.io/intergraph. R package version 2.0-2. [p30]

F. Briatte. *Ggnetwork: Geometries to Plot Networks with 'ggplot2'*, 2016. URL https://github.com/briatte/ggnetwork. R package version 0.5.1. [p27, 29]

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, 2010. [p28]

C. T. Butts. network: a Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2), 2008. [p27]

C. T. Butts. *Sna: Tools for Social Network Analysis*, 2014. URL http://CRAN.R-project.org/package=sna. R package version 2.3-2. [p27]

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. [p27, 31]

W. Chang. *Gcookbook: Data for "R Graphics Cookbook"*, 2012. URL https://CRAN.R-project.org/package=gcookbook. R package version 1.0. [p28]

W. Chang. *R Graphics Cookbook*. O'Reilly, Sebastopol, CA, 2013. ISBN 978-1449316952. [p28]

K. Cook, G. Grinstein, and M. Whiting. VAST Challenge 2014. http://hcil2.cs.umd.edu/newvarepository/benchmarks.php, 2014. [p37]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.org. [p27]

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. [p47]

T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. [p30, 34]

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(12):7821–7826, 2002. [p28, 44]

M. S. Handcock, D. R. Hunter, C. T. Butts, S. M. Goodreau, and M. Morris. Statnet: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11, 2008. URL http://www.jstatsoft.org/v24/i01. [p27]

M. Horikoshi and Y. Tang. *Ggfortify: Data Visualization Tools for Statistical Analysis Results*, 2016. URL http://CRAN.R-project.org/package=ggfortify. R package version 0.4.1. [p28]

H. Jeong, S. P. M. A.-L. Barabási, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411:41–42, 2001. [p50]

B. H. Junker and F. Schreiber. *Analysis of Biological Networks*. Wiley Series in Bioinformatics. John Wiley & Sons, 2008. ISBN 9780470253465. URL https://books.google.com/books?id=2DloLXaXSNgC. [p27]

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. [p28]

T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31(1):7–15, 1989. [p32]

M. E. J. Newman. *Networks : An Introduction*. Oxford University Press, Oxford New York, 2010. ISBN 978-0199206650. [p28, 47]

T. Opsahl. *Structure and Evolution of Weighted Networks*. University of London (Queen Mary College), London, UK, 2009. URL http://toreopsahl.com/publications/thesis/. [p47]

C. Prell. *Social Network Analysis: History, Theory and Methodology*. SAGE Publications, 2011. ISBN 9781446290132. URL https://books.google.com/books?id=wZYQAgAAQBAJ. [p27]

B. Schloerke, J. Crowley, D. Cook, H. Hofmann, H. Wickham, F. Briatte, M. Marbach, and E. Thoen. *GGally: Extension to Ggplot2.*, 2016. R package version 1.3.0. [p28, 29]

K. Slowikowski. *Ggrepel: Repulsive Text and Label Geoms for 'ggplot2'*, 2016. URL https://CRAN.R-project.org/package=ggrepel. R package version 0.5. [p33]

R. Tamassia, editor. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013. [p27]

Y. Tang, M. Horikoshi, and W. Li. Ggfortify: Unified interface to visualize statistical result of popular r packages. *The R Journal*, 2016. URL http://CRAN.R-project.org/package=ggfortify. [p28]

S. Tyner and H. Hofmann. *Geomnet: Network Visualization in the 'ggplot2' Framework*, 2016a. URL http://github.com/sctyner/geomnet. R package version 0.2.0. [p27]

S. Tyner and H. Hofmann. *ggCompNet: Compare Timing of Network Visualizations*, 2016b. URL https://CRAN.R-project.org/package=ggCompNet. R package version 0.1.0. [p50]

D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998. [p28]

D. J. Watts. The "New" Science of Networks. *Annual Review of Sociology*, 30:243–270, 2004. [p27]

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/. [p51]

H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2016. ISBN 978-3-319-24277-4. URL http://ggplot2.org. [p27, 34, 42]

H. Wickham. *Tidyverse: Easily Install and Load 'tidyverse' Packages*, 2017. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.1.1. [p51]

L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag, New York, 1999. [p27]

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. [p28]

G. Yu, D. K. Smith, H. Zhu, Y. Guan, and T. T.-Y. Lam. Ggtree: An R package for visualization and annotation of phylogenetic trees with their covariates and other associated data. *Methods in Ecology and Evolution*, 8(1):28–36, 2017. ISSN 2041-210X. URL https://doi.org/10.1111/2041-210x.12628. [p28]

*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*
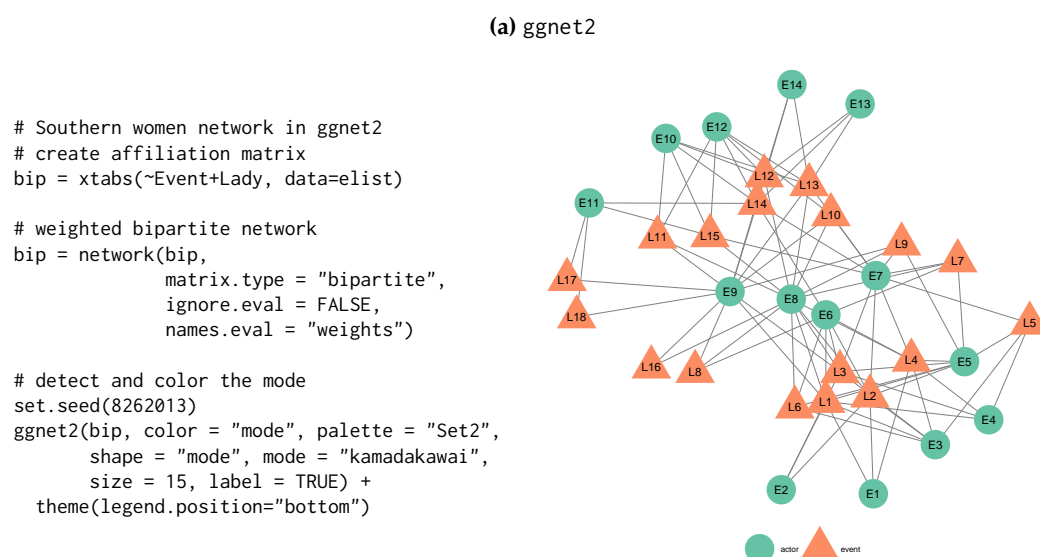*Iowa State University*
*United States*
hofmann@mail.iastate.edu

**(a)** 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 7:** Graph of the Southern women data. Women are represented as orange triangles, events as green circles.

**(b) 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")
```
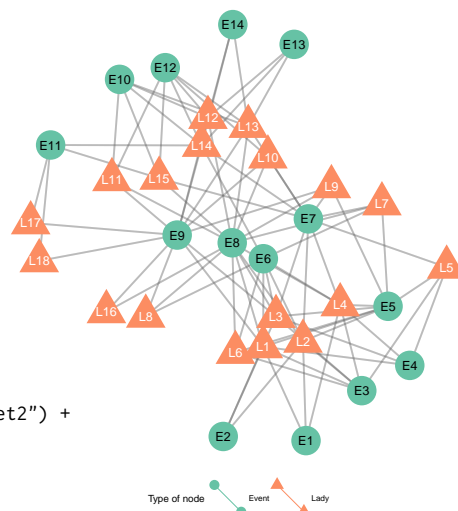
**(c) ggnetwork**

```
# Southern women network in ggnetwork. Use data f
# 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")
```
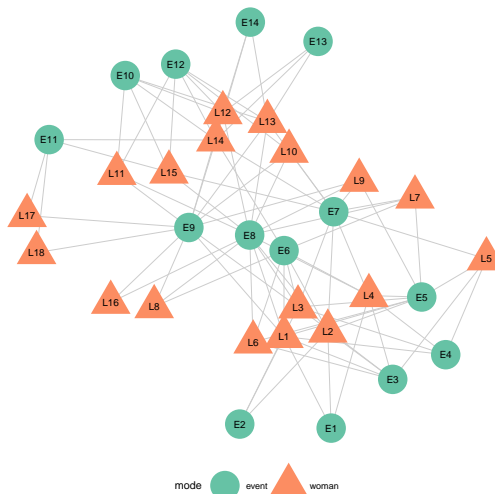
**Figure 7:** Graph of the Southern women data. Women are represented as orange triangles, events as green circles.
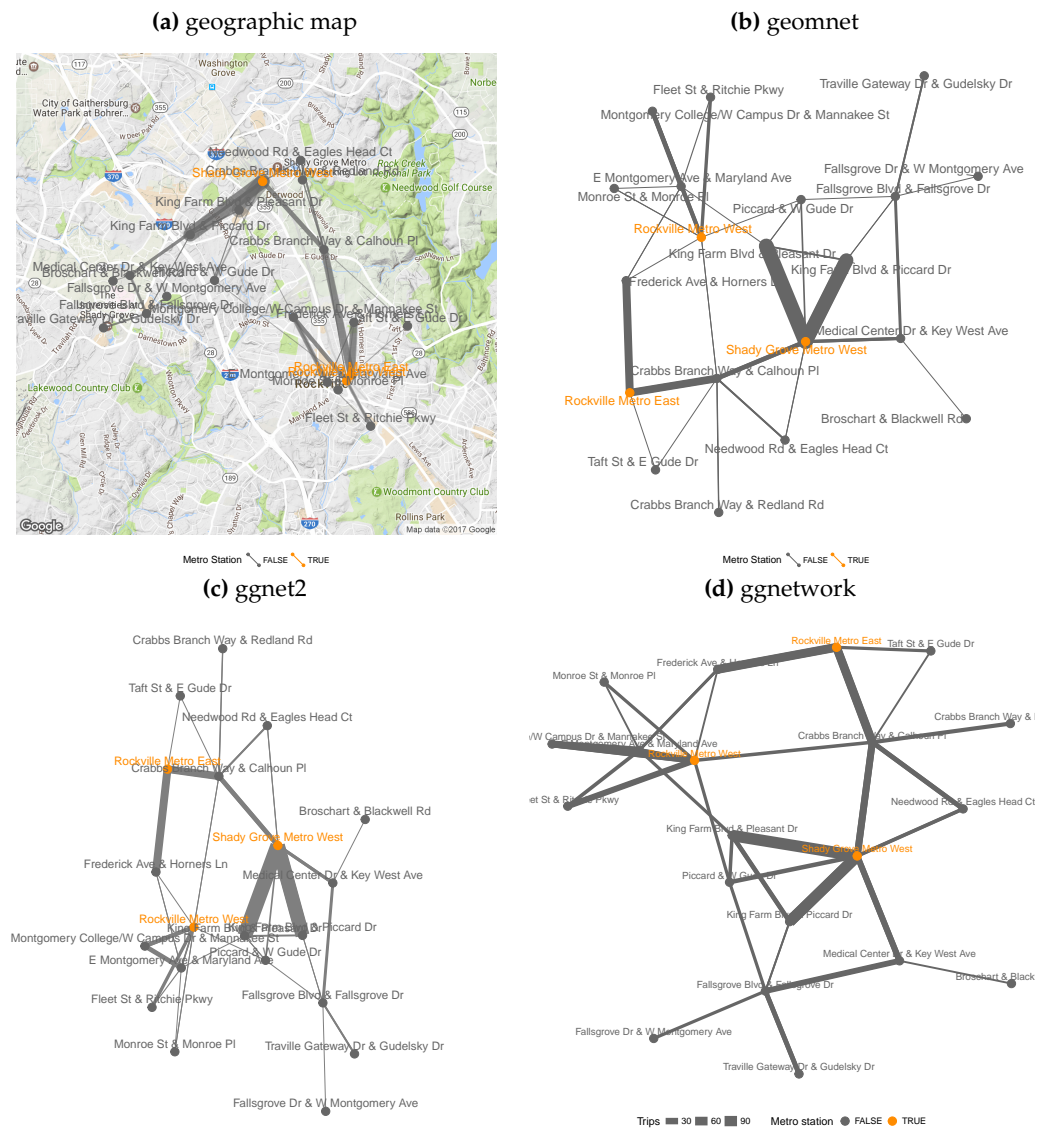
**Figure 8:** Network of bike trips using a geographically true representation(top left) overlaid on a satellite map, a Kamada-Kawai layout in **geomnet** (top right), a Fruchterman-Reingold layout in ggnet2 (bottom left) and **ggnetwork** (bottom right). Metro stations are shown in orange. In both the Kamada-Kawai and the Fruchterman-Reingold layouts, metro stations take a much more central position than in the geographically true representation.
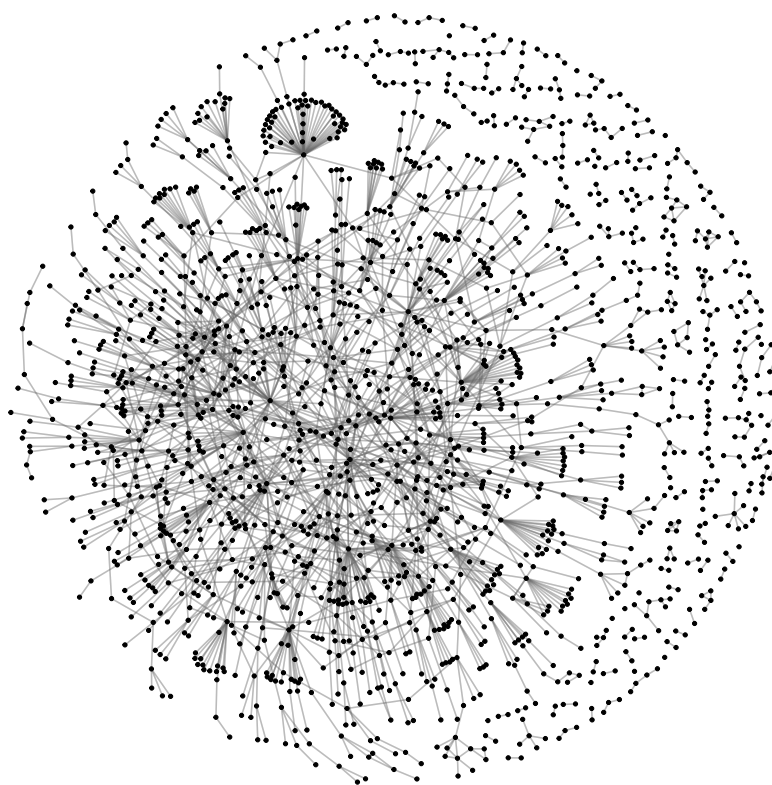
**Figure 9:** Protein-protein interaction network in *S. cerevisiae*. A Fruchterman-Reingold algorithm allowed to run for 50,000 iterations produced the coordinates for the nodes.
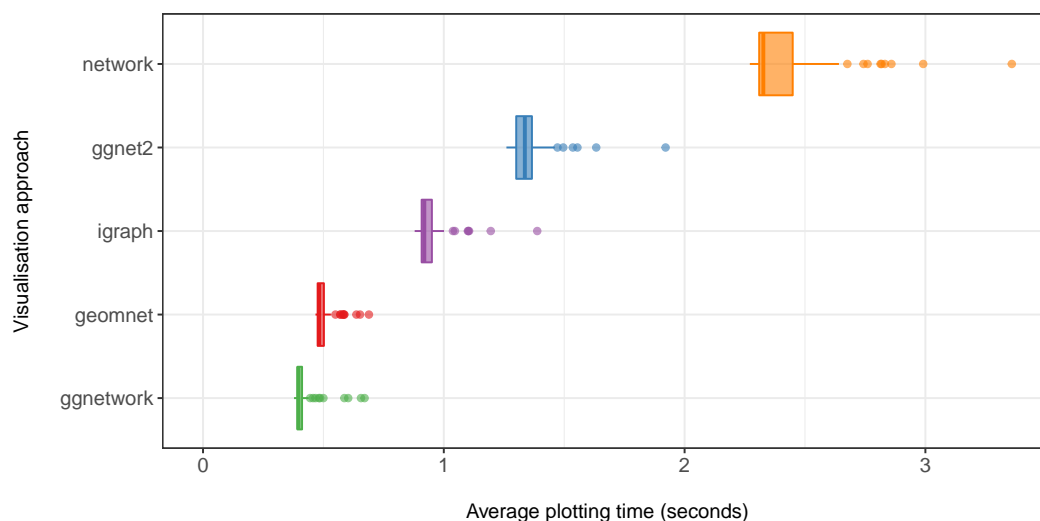


**Figure 10:** Comparison of the times needed for calculating and rendering the previously discussed protein interaction network in the three **ggplot2** approaches and the standard plotting routines of the **network** and **igraph** packages based on 100 evaluations each.
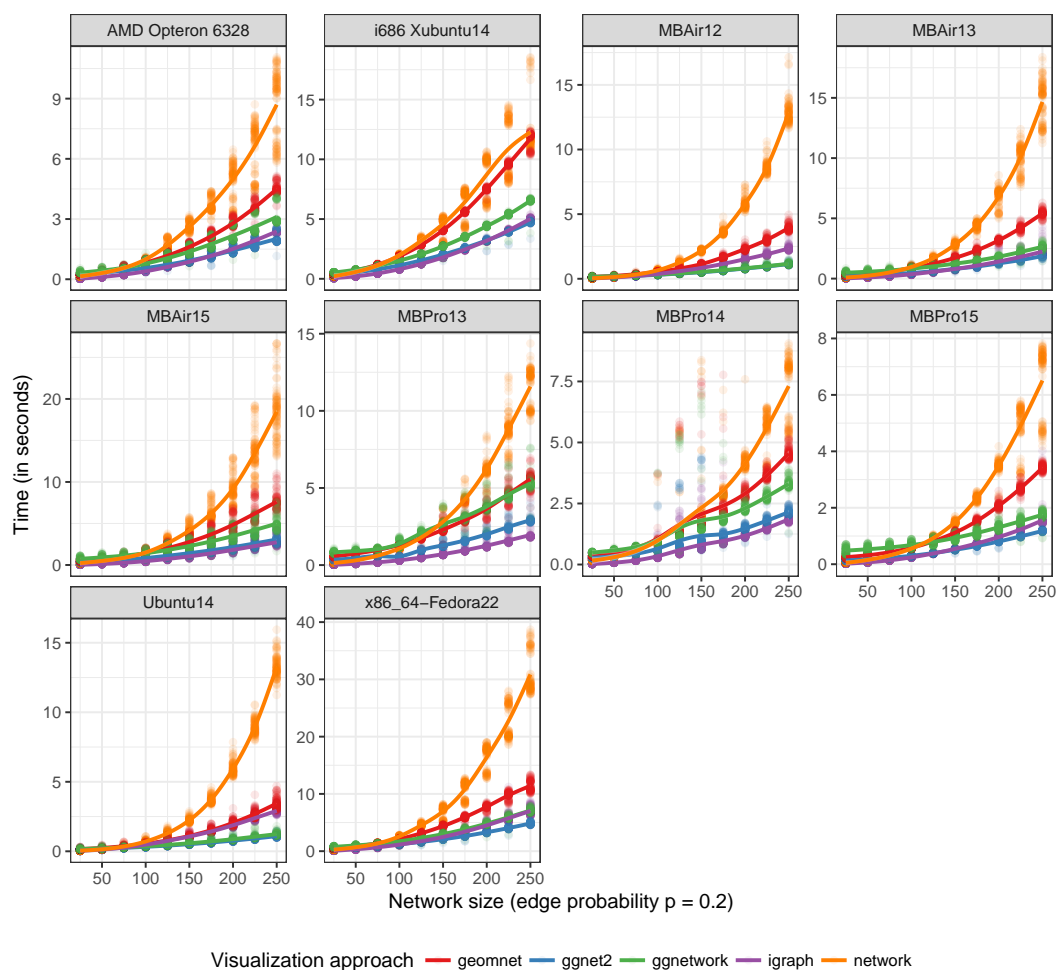
**Figure 11:** Plotting times of random undirected networks of different sizes under each of the available visualization approaches using their default settings. Note that each panel is scaled independently to highlight relative differences in the visualization approaches rather than speed of different hardware.