ElectroGraph: Advancements and Philosophies for Plotting Relational Data in R

Andrew C. Thomas

June 27, 2009

ElectroGraph: Advancements and Philosophies for Plotting Relational Data in R

The ElectroGraph package for R has been assembled as a means for analyzing and presenting large relational data sets according to the properties of their graphs. While this includes standard measures for distances based on geodesic path lengths, there is also a great deal to be gained from the inclusion of alternative measures of distance such as social conductance , and the possible extension of this method to considering antagonistic relationships.

This guide contains installation instructions as well as directions for the use of the package to compare, analyze and display relational data of various types, primarily data that are more complicated than simple binary relations.

1 Installation Instructions

ElectroGraph will soon be available on the Comprehensive R Archive Network (or CRAN), so that the installation of ElectroGraph will be as simple as running the command

```
> install.packages("ElectroGraph")
```

For the time being, the source code can be downloaded from this location, and installed on UNIX-type systems from the command line:

```
% R CMD INSTALL ElectroGraph_0.1.tar.gz
```

2 Initializing an ElectroGraph object

To process a relational data set into an object that ElectroGraph can analyze, the `electrograph` function/constructor is used. The object to load can either be an n-by-n sociomatrix, or an n-by-k edge list, where k can take one of three values:

- $k = 2$: The two columns represent arcs of value 1 to enter into the system. The resulting system will have as many nodes as there are unique identifiers in the two columns. By default, the edges produced will be undirected. For example, the command sequence

```
sources <- c("Bob","Ted","Bob")
sinks <- c("Carol","Alice","Alice")
e.graph <- electrograph(cbind(sources,sinks))
```

  will create a four-node social network with three arcs.

- $k = 3$: The two columns represent source-sink pairs, and the third equals the value of this arc, which can be any real number (though caution is advised with negative values.) By default, the edges produced will be undirected. For example, the command sequence

```
sources <- c("Bob","Ted","Bob")
sinks <- c("Carol","Alice","Alice")
vals <- c(3,2,1)
e.graph <- electrograph(cbind(sources,sinks,vals))
```

  creates a four-node social network with three undirected edges of varying value.

- $k = 4$: The two columns represent source-sink pairs, and the third and fourth columns are the values for each arc in the dyad, for forward and reverse respectively. For example,

```
sources <- c("Bob","Ted","Bob")
sinks <- c("Carol","Alice","Alice")
val1 <- c(1,2,5)
val2 <- c(3,2,0)
e.graph <- electrograph(cbind(sources,sinks,val1,val2))
```

  creates an asymmetric directed sociomatrix, where among other relations, Bob pines for Alice but is not reciprocated.

2.1 Included Data Sets

There are two data sets included for demonstration purposes in the `ElectroGraph` package, both included in the data set `electro.frats`. They are the fraternities studied by **?** and Newcomb [1961] respectively.

The data set `bernard.killworth.b` is a weighted edgelist that indicates the number of times an observer noticed two people communicating with one another during a short period of time. The data set can be prepared as an electrograph object with the command

```
bk.graph <- electrograph(bernard.killworth.b)
```

The data set `newcomb` is a three-dimensional array, where each slice indicates a rank-order of each member's preferences towards the others during one week. The total period of observation is 15 weeks.

Because this encompasses 15 separate social interaction periods, it is recommended to load each week's observations into its own ElectroGraph object in this fashion:

```
newcomb.e <- list(NA)
for (kk in 1:15) newcomb.e[[kk]] <- electrograph(newcomb[,,kk])
```

Separate analyses are then conducted on each year's results.

2.2 Analyses

When an ElectroGraph object is initialized, several analyses are conducted automatically:

- The system is divided into disconnected components. The placement of each node into each component is given in the ElectroGraph element `component.vector`.
- The equivalent social conductance is calculated between each pair of nodes.
- Distance matrices are calculated from geodesics and social conductances.
- The distribution of geodesics is calculated, and centrality measures determined within.
- Betweenness centrality is calculated for each node and edge in the system to determine their relative importances.

The electro-social analyses are conducted if the option `perform.exam` is set as TRUE in `electrograph`, as it is by default. In the case of a large social network, say over 500 nodes with 2 edges per node, this may take a large amount of time to execute. In this case, it may be preferred to initialize the object without the extra analyses:

```
bk.graph <- electrograph(bernard.killworth.b, perform.exam=FALSE)
bk.graph <- electrograph.exam(bk.graph,sample.edges=TRUE)
```

3 Graph Plots

By default, an ElectroGraph object can be automatically plotted by the command

```
plot(bk.graph)
```

though depending on the object being plotted, this may not prove to be the most aesthetically pleasing presentation of the data.

There are several important features to consider when plotting graphs:

- Relative node position and distance. Are the geometric distances on the plot meant to reflect topological distance along a graph?
- Node shape, size and color. Are these indicative of isolated properties of the node, or perhaps other characteristics that only have meaning within a network ensemble?
- Edge color, thickness and presence. Which edges do we wish to show and which to omit for the sake of illumination?

There are several options available to be customized to suit these needs. Among them:

- `distance.mode`: By default, the distances between nodes are set to be `"shortest.path"`, the geodesic distances. Setting this option to be `"electro.social"` sets the plotting algorithm to use the inverse of social conductance for the ideal distance between points on a plot.
- `plot.mode`: ElectroGraph makes use of two force-energy direction algorithms to put an n-dimensional object into two-dimensional space: `"kamada.kawai"` (default) and `"fruchterman.reingold"`. The algorithms, with subsequent modifications for improved convergence, are given in the appendix.
- `ego.focus`: If specified, this lists a selection of nodes whose placement on the graph with respect to its neighbours takes priority over others. The energies and forces of these points in the placement algorithm are increased dramatically so as to give their relative placements preferential treatment.
- `source.sink.pair`: If selected, the electro-social current flow between these points will form the basis for the thickness of the edges and the inclusion of arrowheads.

- `previous.electrograph.plot.object`: If included, the points in the active plot will be placed close to those in the "previous" plot. Useful for animation purposes.

4 Dynamic Graph Plots

ElectroGraph contains two built-in methods for producing useful animations for display purposes.

4.1 Wedding Cake Plots of Valued Graphs

Because a single plot can be extremely dense with ties, the "wedding cake" method for plotting valued-edge graphs is presented to demonstrate various layers of network connectivity. The method is to take various lower and upper bounds for edges that should be displayed, plot the system with each set of bounds in sequence, then display the plots in order through static views or by using a movie-making program. The coordinates for these points are fixed at the outset according to whichever distance and plot methods are selected to operate on the raw data.

By entering a valued ElectroGraph object directly, we can produce a series of plots through

```
plot.wedding.cake(bk.graph)
```

that will produce a series of plots where the lower bound varies through the range of edges up to the maximum, and no upper bound is present; this is as if we continually remove the bottom layers from a tower structural representation of the model.

If we wish to take a tomographic scan of a 2-dimensional network projection, one possibility is to calculate the desired quantiles directly. For example, to take only 10% of a graph's edges at a time:

```
edge.vals <- bk.graph$grand.sociomatrix; edge.vals <- edge.vals[edge.vals>0]
lower.bound <- quantile(edge.vals,(1:90)/100)
upper.bound <- quantile(edge.vals,(11:100)/100)
plot.wedding.cake(bk.graph, lower.bound=lower.bound, upper.bound=upper.bound)
```

`plot.wedding.cake` will save a series of images to disk, currently only in the Portable Network Graphics (`.png`) format. The routine will also output a shell script for automatically creating the movie in the freeware program ImageMagick.

4.2 Plotting a Graphical Time Series with `animate.plot.series`

Given a series of ElectroGraph plots with identical node sets, ElectroGraph can produce a set of plots with animated transitions between each plot.

For this, we much save the outcome of each plot into an R object. Suppose we have the Newcomb data preloaded:

```
data(electro.frats)
newcomb.electro <- list(NA)
for (kk in 1:dim(newcomb)[3]) {
  newcomb.electro[[kk]] <- electrograph(newcomb[,,kk])
}
```

We can then pre-plot the objects using the following sequence:

```
newcomb.plot <- list(NA)
newcomb.plot[[1]] <- plot(newcomb.electro[[1]],distance.mode="electro.social", just=TRUE
for (kk in 2:length(newcomb.electro))
  newcomb.plot[[kk]] <- plot(newcomb.electro[[kk]],distance.mode="electro.social",
just.coordinates=TRUE, previous.elec=newcomb.plot[[kk-1]])
```

This sets up a series of plots of the Newcomb sequence where the nodes are aligned to be as close to their previous positions as possible.

To create a series of images, run the command

```
animate.plot.series(newcomb.plot)
```

Like the Wedding Cake plot, this will produce a series of images that can then be assembled into a movie using software such as ImageMagick; a shell script is provided that will perform the assembly assuming the software is installed.

A Force-Energy Graph-Drawing Algorithms

Kamada-Kawai Algorithm

The Kamada-Kawai plotting algorithm [Kamada and Kawai, 1989] begins with the premise that all pairs of points have a "preferred" distance from each other, as governed by a spring that pulls or pushes on the points when distance is not maintained; that is, that $\binom{N}{2}$ springs are present in the system. In the social network context, this distance is typically taken to

be the shortest geodesic distance between two nodes along the network. A spring between these individuals, with "spring constant" k_{ij}[1] exerts a restoring force on each proportional to the difference between their actual and preferred distance apart.

Letting d_{ij} be the geodesic distance, define the ideal plot difference $l_{ij} = Dd_{ij}$, where D holds the value of a unit distance. Kamada and Kawai recommend for a graph with width and height L_0, that D be defined as

$$D = \frac{L_0}{max_{i<j} d_{ij}}$$

so that the maximum distance between points, $max_{i<j} l_{ij}$, corresponds directly to the size of the plot.

The restoring force exerted by a spring between nodes i and j is

$$-k_{ij} \left(|p_i - p_j| - l_{ij} \right),$$

where $p_i = (x_i, y_i)$ is the position in two-dimensional space, and the total energy in the spring due to this deformation is

$$\frac{k_{ij}}{2} \left(|p_i - p_j| - l_{ij} \right)^2.$$

The trick to generating a pleasant looking graph is then to define an appropriate spring constant for each pair of nodes, such as one inversely proportional to the squared distance, or

$$k_{ij} = \frac{K}{d_{ij}^2}$$

where K is a constant to be chosen by the plotter.

The coordinates for each point $p_i = (x_i, y_i)$ are solved by finding the minimum of the total energy equation,

$$U = \sum_{i<j} \frac{k_{ij}}{2} \left(|p_i - p_j| - l_{ij} \right)^2,$$

with respect to the centering constraints $\sum_i x_i = \sum_i y_i = 0$. As they have no relationship to connected points in the graph, isolates are often placed at the periphery of the plot in random fashion. In ElectroGraph, isolates are considered to be separate components and are

[1] As the spring law that follows is known as Hooke's Law, k_{ij} is also known as Hooke's constant.

plotted distinctly from the main component.

Fruchterman-Reingold Algorithm

The work of TMJ and EM [1991] suggests an alternate method for deriving forces between nodes. In particular, it is assumed that all nodes have a repulsive force between them that decays with distance, and only nodes with a connection exhibit an attraction. Nodes are allowed to interact until they reach equilibrium distances from each other. The compelling factor to be considered is that all connected nodes should be evenly spaced as a means of picturing structure.

Following experimentation with various functional forms, TMJ and EM [1991] suggest setting an "ideal" distance for connected nodes equal to k. Given the observed distance d_{ij}, the force magnitudes are specified to be

$$F_a = \frac{d_{ij}^2}{k},$$

$$F_r = -\frac{k^2}{d_{ij}}.$$

Under this scheme, the forces balance when $d_{ij}^2/k = k^2/d_{ij}$, or when the observed distance d_{ij} equals the ideal distance k. In the case of unconnected nodes, there is no attractive force present between these pairs.

It should be noted that these algorithms will work in any integer dimension, and have been coded as such in ElectroGraph. However, plotting methods are limited in ElectroGraph to two dimensions, so I leave the creation of higher-dimensional plotting mechanisms as an exercise to the user.

Modifications to Consider in Weighted Graph Scenarios

As previously implemented, both algorithms have features that are desirable in some circumstances but less so in others. I attempt to bring the best of both worlds to the solution of graph positioning with the following implementations.

Combined Force-Energy Direction

Previous implementations of force-energy algorithms typically take one of two approaches when finding ideal positions, starting with a randomly determined configuration:

- All-energy. Take a random perturbation of a node's position, and compare the total energy of the new configuration to the old one. If the new energy is smaller, accept the new configuration. This has been improved with the introduction of simulated annealing techniques, in which a larger energy can be accepted with nonzero probability as a function of the "temperature" of a system, which is slowly lowered throughout the process.

- All-force. A node is chosen and all resulting forces on it are added up; the node is then moved in the direction of the force. This approach benefits from the directionality of forces but does not have the global-energy-minimum properties of the first approach.

We can combine these approaches to quickly find an ideal position that finds an energy minimum more likely to be a global minimum. For each node in the system, perform the following steps:

1. Exchange its position with another node. Calculate the resulting energy difference and accept the swap if the new energy configuration is smaller in magnitude, or if an acceptance step is satisfied in a simulated annealing setting.

2. Determine the total force on a node, as well as the rate of change of the force in the direction of influence. From here, estimate the point where the force equals zero from one step of Newton's method.

3. Add a random perturbation to this point proportional to the temperature of the system.

4. Accept this point if the energy is lower, or if an acceptance step is satisfied in a simulated annealing setting.

This approach has the benefit of directed searches as well as finding global energy minima.

References

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

Newcomb, T. (1961). The acquaintance process.

TMJ, F. and EM, R. (1991). Graph drawing by force-directed placement. *Software – Practice and Experience*, **21** 1129–64.