ElectroGraph: Advancements and Philosophies for Plotting Relational Data in R

Andrew C. Thomas*

January 26, 2010

The ElectroGraph package for R collects a number of network analysis algorithms and presentation styles. 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 (also known as the inverse of Ohmic or "resistance" distance, as seen in Klein and Randic [1993] among others), and the extension of this method to considering antagonistic relationships (as in Thomas [2009]).

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

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

though the package and this manual can also be obtained from the CRAN URL.

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:

*Visiting Assistant Professor, Department of Statistics, Carnegie Mellon University. Contact email: act@acthomas.ca.

- $k = 2$: 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

  ```
  ssinks <- c("Carol","Alice","Alice")
  ources <- c("Bob","Ted","Bob")
  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 Tie Fidelities and "Enemy" Ties

ElectroGraph has the capacity to include not only positively real-valued ties, but to have ties that are imperfect carriers of information. The range for a tie fidelity is $(-1, 1)$, so that a tie with fidelity 1 is considered a perfect transmitter of information (or a regular friend), fidelity -1 is considered a perfect inverter of information (or an enemy), and values in between have varying degree of amity or enmity respectively. Standard geodesic measures are not affected by fidelities, only current-based measures. The concept of network tie fidelity is defined in Thomas [2009].

To include fidelity measures for each tie in the data set, use the "fidelities" option, which takes either a length-k column vector, k-by-2 matrix or n-by-n matrix depending on the input type for the dyads:

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

vals <- c(3,2,1)
fid2 <- c(1,1,-1)
e.graph <- electrograph(cbind(sources,sinks,vals), fidelities=fid2)
```

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 (geodesic) shortest paths between all nodes in each component are calculated using the Floyd-Warshall algorithm [Floyd, 1962]. This is controlled by the option `solve.for.shortest.paths`, and is set to `TRUE` by default. This can be slow on large (1000+ nodes) graphs.
- The Ohmic social conductance is calculated between each pair of nodes in each component, and by connection the Ohmic/resistance distance. This is controlled by

the option `ohmic.properties`, and is set to `TRUE` by default. Note that this can be very slow on large graphs (500+ nodes), considerably slower than finding the lengths of shortest paths.

These items are the basis for several summary statistics that can be obtained for each node. If we have an ElectroGraph object `e.graph`, then the command `summary(e.graph)` will produce the following quantities for each node:

- Out-degree and in-degree, defined as the column- and row-sums of the sociomatrix respectively. These terms are of course identical in a symmetric sociomatrix.
- The fraction of possible transitive triples and three-cycles that are formed at each node, if the sociomatrix is binary, generalized from the clustering statistic of Watts and Strogatz [1998]; labelling the sociomatrix Y, the terms take the form

$$T_i = \frac{\sum_{j \neq i} \sum_{k \neq j,i} Y_{ij} Y_{ik} Y_{jk}}{\sum_{j \neq i} \sum_{k \neq j,i} Y_{ij} Y_{ik}},$$

$$L_i = \frac{\sum_{j \neq i} \sum_{k \neq j,i} Y_{ij} Y_{ki} Y_{jk}}{\sum_{j \neq i} \sum_{k \neq j,i} Y_{ij} Y_{ki}}.$$

- Harmonic closeness centralities for each node, defined as

$$C_{1/C,out}(i) = \sum_j \frac{1}{d(i,j)}$$

$$C_{1/C,in}(i) = \sum_j \frac{1}{d(j,i)}$$

  where $d(i,j)$ is the geodesic distance from node i to node j. These quantities are only displayed if the geodesic matrix has been calculated.
- Ohmic closeness centralities for each node, defined as

$$C_{\Omega,out}(i) = \sum_j \frac{1}{d_\Omega(i,j)}$$

$$C_{\Omega,in}(i) = \sum_j \frac{1}{d_\Omega(j,i)}$$

where $d_{\Omega(i,j)}$ is the resistance distance defined in Klein and Randic [1993] (among others), or as extended to the asymmetric case as in Thomas [2009]. $1/d_{\Omega(i,j)}$ can also be expressed as G^{eq}_{ij}, or the equivalent conductance from node i to j. These quantities are only displayed if the Ohmic procedures have been run.

- Ohmic betweenness centralities for each node, defined according to the scheme used to estimate the type of signal being used – fixed power, voltage or current:

$$C_P = \sum_a \sum_{b \neq a} \frac{1}{\sqrt{G^{eq}_{ab}}} \sum_{j \neq i} I^{ab}_{ij}$$

$$C_V = \sum_a \sum_{b \neq a} \sum_{j \neq i} I^{ab}_{ij}$$

$$C_P = \sum_a \sum_{b \neq a} \frac{1}{G^{eq}_{ab}} \sum_{j \neq i} I^{ab}_{ij}$$

I^{ab}_{ij} is the current we would observe along edge (i,j) if the sociomatrix edge strengths are interpreted as electrical conductances, and a potential difference of 1 Volt were applied across the node pair (a,b) – this is defined fully in Thomas [2009]. These quantities are only displayed if the Ohmic procedures have been run.

2.3 Included Data Sets

There are three data sets included for demonstration purposes in the `ElectroGraph` package. Two are the fraternities studied by Bernard et al. [1980] and Newcomb [1961] respectively, available in the command

```
data(electro.frats)
```

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 can then be conducted on each year's results.

The third data set is a ten-node network synthesized for this guide, termed the Alphabet class: ten people with 14 friendships and two antagonistic connections. This can be loaded with the command

```
data(alphabet.class)
```

and immediately load this as an ElectroGraph object with

```
alphabet.graph <- electrograph(alphabet.class)
```

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:

- `connectivity.mode`: By default, the distances between nodes are set to be `"sociomatrix"`, which sets the ideal distances as the inverses of the unprocessed sociomatrix terms, when they are less than There are three other options: `"shortest.path"`, which sets the preferred distance to be the geodesic distances; `"ohmic"` sets the plotting algorithm to use the inverse of the equivalent Ohmic conductance for the ideal distance between points on a plot; and `"ohmic.socio"` sets the preferred distance to be the inverse Ohmic conductance for non-zero dyads, ignoring the indirect Ohmic conductance between node pairs that do not have a direct connection.
- `force.mode`: ElectroGraph makes use of two force-energy direction algorithms to put an n-dimensional object into two-dimensional space. The default selection is `"fruchterman.reingold"` [Fruchterman and Reingold, 1991], which will work with all four connectivity modes; the other option is `"kamada.kawai"` [Kamada and Kawai, 1989], which will work with either `"shortest.path"` or `"ohmic"`. The workings of these algorithms is given in the appendix.
- `source.sink.pair`: If selected, the Ohmic current flow from the source to the sink will form the basis for the thickness of the edges and the inclusion of arrowheads. (ElectroGraph does not support arrowheads explicity for directed edges.)
- `previous.electrograph.plot.object`: If included, the points in the active plot will be placed close to those in the "previous" plot. This command is useful for animation purposes.

Figure 1 demonstrates several of these plots in sequence.

3.1 Separating Components

ElectroGraph will place each disconnected component into its own separate unit in the resulting figure. Examples of this can be seen in Figure 2.

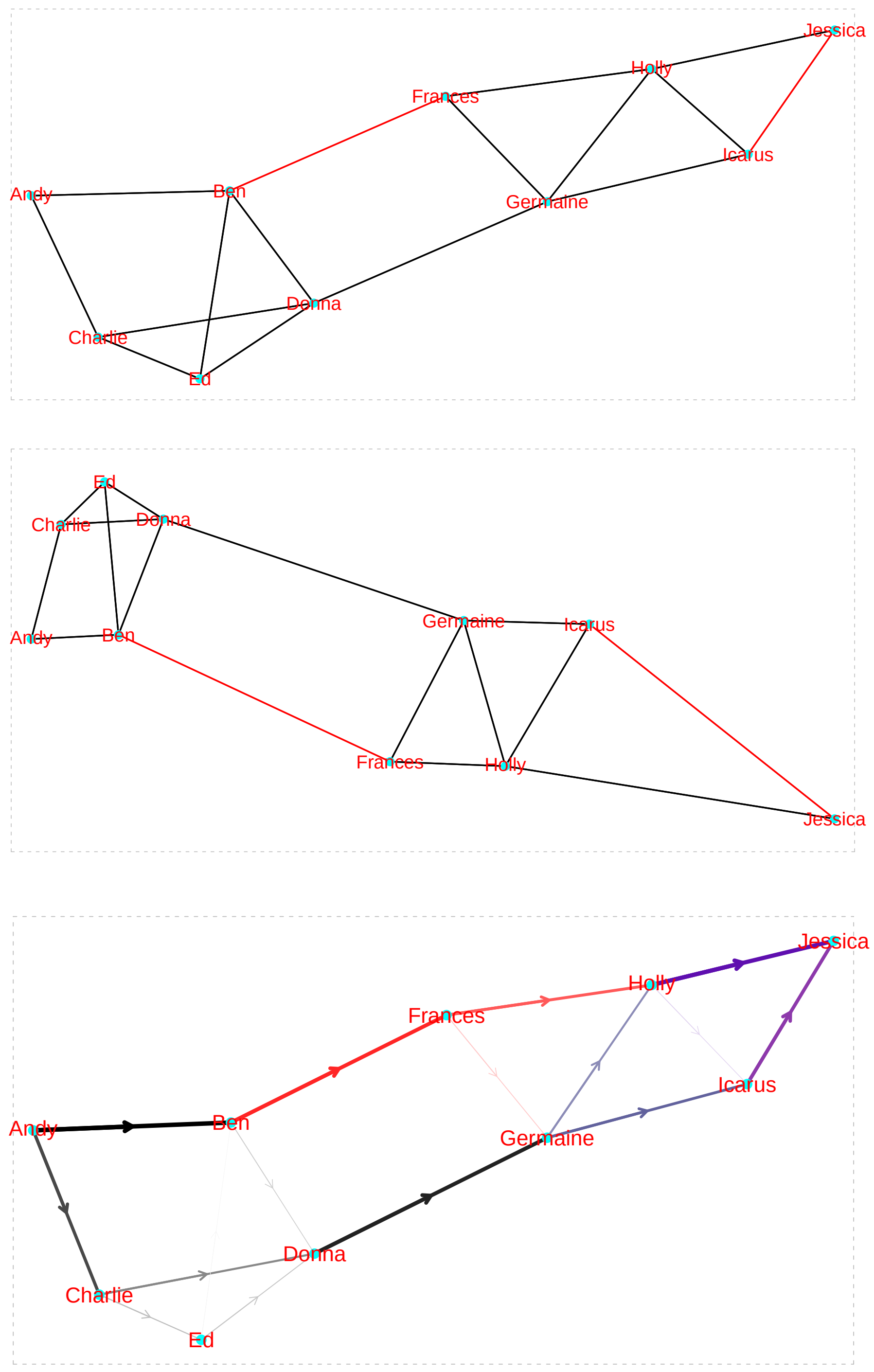

Figure 1: Three forms for the plotting of network data with ElectroGraph, demonstrating on the synthetic data set `alphabet.class`. Top: standard plotting with sociomatrix terms defining the total connectivity. Middle: Plotting with Ohmic conductances defining the connectivity between dyads. Bottom: a demonstration of the Ohmic current flow from one node to another in the presence of antagonistic ties.

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.

5 Force-Energy Graph-Drawing Algorithms

The plotting methods in ElectroGraph use either Fruchterman-Reingold [Fruchterman and Reingold, 1991] or Kamada-Kawai [Kamada and Kawai, 1989], the two most common methods for placing graph objects in a two-dimensional space for easy viewing. These algorithms take one of several choices for the “ideal” distance between any pair of nodes: basing forces on the sociomatrix (`sociomatrix`), the geodesic path length (`shortest.path`), the Ohmic conductance (`ohmic`), or the Ohmic conductance only where there are ties present (`ohmic.socio`).

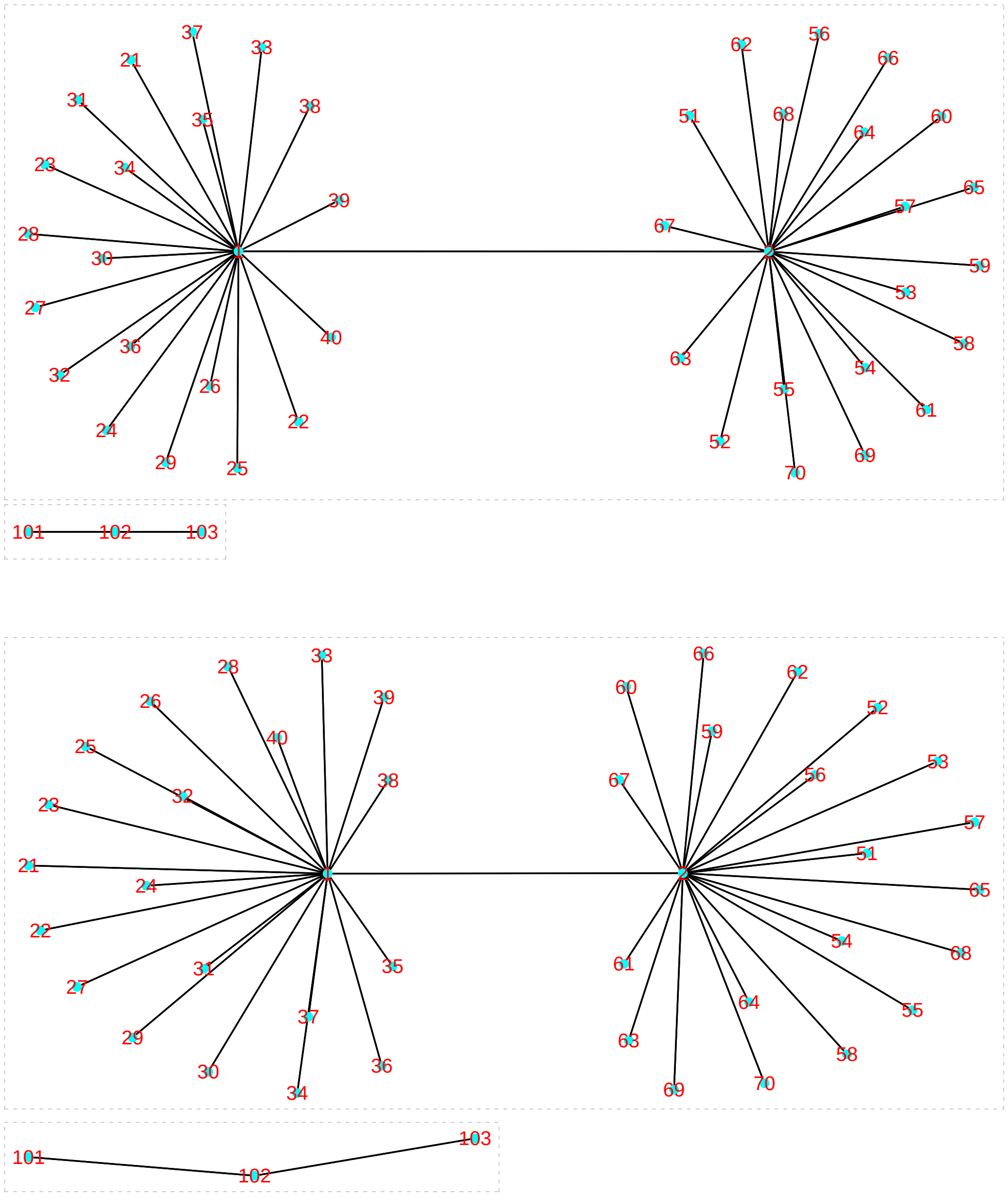

Figure 2: Two identical binary networks plotted under the defaults Fruchterman-Reingold (top) and Kamada-Kawai (bottom) force-directed placement schemes. Because the Fruchterman-Reingold scheme was designed to induce repelling forces between clusters, there is considerably more separation between the two stars than in the Kamada-Kawai version of this network. To compare the relative distances, note that each graph has an associated three-node disconnected component, also with binary ties; the repulsive effect stretches the prime tie (1,2) far more in the former case.

Here I review the original definitions of the algorithms, noting how the regular use of weighted tie strengths impacts these algorithms, and how thinking of forces in terms of *connectivity* or *closeness* rather than distance allows for more generalization.

5.1 Fruchterman-Reingold Algorithm

In this implementation, Fruchterman and Reingold [1991] assume that all nodes have a repulsive force between them that decays with distance, and that only nodes with a connection exhibit an attraction to each other. Forces are designed to balance for a pair of nodes at their "ideal" distance in the absence of any other effects; in practice, the repulsive forces that come about through the addition of neighbouring nodes cause connected communities to appear closer to each other, and that non-members are pushed farther away. In this way, the Fruchterman-Reingold algorithm is optimal for quick community detection; this is enhanced further if the `ohmic.socio` option is used, which enhances the connectivity for node pairs with alternative indirect paths.

Following experimentation with various functional forms, Fruchterman and Reingold [1991] suggest setting an "ideal" distance for connected nodes equal to k. Given the ideal distance d_{ij}, the force magnitudes are specified to be $F_a = \frac{d_{ij}^2}{k}$ and $F_r = -\frac{k^2}{d_{ij}}$. 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. However, this specification is insufficient for graphs with differentially weighted edges.

The implementation in ElectroGraph is slightly different to allow for this. Letting a_{ij} be the connectivity between two nodes (from the sociomatrix, inverse geodesic distance or Ohmic conductance), and d_{ij} be the distance between the two nodes, the net attractive force is set to be

$$F_{ij} = a_{ij}^3 d_{ij}^2 - \frac{1}{d_{ij}}$$

so that the nodes that have no connectivity always exert a repulsive force.

5.2 Kamada-Kawai Algorithm

This implementation [Kamada and Kawai, 1989] begins with the premise that all pairs of points have an ideal separation that must be respected whether or not the node pairs have a non-zero connectivity. The forces between nodes are then governed by a spring that pulls or pushes on the points when the ideal separation distance is not maintained; that is, that $\binom{N}{2}$ springs are present in the system.

In the social network context, this separation 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.

The connectivity factor to be respected is a_{ij}, though in this case it can only be the inverse geodesic path length `shortest.path` or the Ohmic conductance `ohmic`. Setting the spring constant to be a_{ij}^2 specifies that stronger forces apply between more highly connected nodes.

The attractive force is then equal to

$$F_{ij} = (d_{ij} - \frac{1}{a_{ij}})a_{ij}^2,$$

so that the force is repulsive if the distance between the nodes is less than that prescribed by the connectivity factor.

Dimensionality

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

5.3 Solutions for Ideal Plot Points

These two methods can be expressed either in terms of the cancellation of forces or the minimization of energies (which are equivalent conditions); for ease of solution, the implementation in ElectroGraph takes the force-directed approach. Namely, the following Newton-like algorithm is used:

- Take a node and assess the forces acting on it.
- Move the node a tiny amount in the direction the force would carry it, and reassess the forces applied to it.

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

- If the total force has decreased, move the node to a position where the force would be zero if the force were a linear function. If the total force has increased, move the node proportionally to the amount of force being applied in order that the next step will permit the finding of a local minimum.
- Repeat for each point in the graph.
- If the maximum distance moved is greater than a predetermined "tolerance" distance, repeat this procedure until the maximum distance moved in any one iteration is less than the tolerance distance.

This is a blending of the approaches used by [Kamada and Kawai, 1989] and [Fruchterman and Reingold, 1991], though with any optimization algorithm for highly multi-modal data, it can stand to be improved and tinkered beyond the current standard.

6 Future Developments

As with all worthwhile software projects, there are inevitably suggestions from the community that can provide substantial improvements, as well as features whose implementation would be useful to a select few. With that in mind, there are a few areas that can stand to be improved in the next iteration, which will take considerable thinking on the part of the network graphics community to implement properly.

6.1 Animation

ElectroGraph's animation capabilities are proof-of-concept in nature and not meant to be especially sophisticated. The wedding cake plot, for example, is provided to show that it is not necessary to "threshold" a graph in order to plot it nicely [Thomas and Blitzstein, 2009], and can certainly be enhanced beyond the current display method.

The package rSoNIA [Bender-deMol et al., 2007] has been designed to implement the animation of social network data using the algorithms and methods of the `statnet` suite [Handcock et al., 2008]. The method `animate.plot.series` was designed to mimic this functionality with the enhancements provided in the ElectroGraph approach, but can stand either to be improved on its own or integrated with the rSoNIA methodology depending on the stated needs of users.

6.2 Combined Force-Energy Direction Methods

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, for a distance proportional to the force. This approach benefits from the directionality of forces but does not have the global-energy-minimum properties of the first approach.

The implementation currently in ElectroGraph is an improvement on the all-force method, but there are cases when the benefits of simulated annealing may be useful as well [Kirkpatrick et al., 1983; Davidson and Harel, 1996]. For that, I propose a class of algorithm that incorporates elements of tempering methods with force-direction. Namely, two features should be present:

1. "Swap": Select a node and 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-rejection step is satisfied in a simulated annealing setting, where the acceptance probability varies with the temperature of the system.
2. "Step": Rather than simply step in the direction of the node to an estimated position of zero force, add a random perturbation proportional to the temperature of the system.

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

At the present time, this is not implemented in ElectroGraph for the reason that it does not appear to be *necessary* in any of the applications I have considered, and requires considerably more computing time to execute. This application may be worthwhile for cases I have not studied, however, and its addition is most certainly possible.

Acknowledgements

I am grateful to Joseph Blitzstein, Steve Fienberg, Claudia Mueller-Birn and Pavel Krivitsky for their comments in helping to improve the presentation of this work, as well as the lab group of Nicholas Christakis at Harvard Medical School where I first presented many of these ideas for their consideration. Research with the Christakis group was supported by grant PO1-AG031093 from the NIA.

References

Bender-deMol, S., Morris, M. and Moody, J. (2007). Prototype Packages for Managing and Animating Longitudinal Network Data: dynamicnetwork and rSoNIA. *Journal of Statistical Software*, **24** 1–36.
URL http://www.jstatsoft.org/v24/i07.

Bernard, H., Killworth, P. and Sailer, L. (1980). Informant accuracy in social network data IV. *Social Networks*, **2** 191–218.

Davidson, R. and Harel, D. (1996). Drawing Graphs Nicely Using Simulated Annealing. *ACM Transactions on Graphics*, **15** 301331.

Floyd, R. W. (1962). Algorithm 97: Shortest Path. *Communications of the ACM*, **5** 345.

Fruchterman, T. and Reingold, E. (1991). Graph Drawing by Force-Directed Placement. *Software – Practice and Experience*, **21** 1129–64.

Handcock, M. S., Hunter, D. R., Butts, C. T., Goodreau, S. M. and Morris, M. (2008). statnet: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data. *Journal of Statistical Software*, **24**.

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

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, **220** 671680.

Klein, D. and Randic, M. (1993). Resistance distance. *Journal of Mathematical Chemistry*, **12** 81–95.

Newcomb, T. M. (1961). *The Acquaintance Process.* Holt, Rinehart and Winston.

Thomas, A. C. (2009). Ohmic Circuit Intepretations of Network Distance and Centrality. Unpublished manuscript.

Thomas, A. C. and Blitzstein, J. K. (2009). The Thresholding Problem: Uncertainties Due to Dichotomization of Valued Ties. Unpublished manuscript.

Watts, D. and Strogatz, S. (1998). Collective Dynamics of "Small-World" Networks. *Nature*, **393** 440–442.