Visualization of Social Networks in Stata by Multi-dimensional Scaling *

Rense Corten
Department of Sociology/ICS
Utrecht University
The Netherlands
r.corten@uu.nl

April 12, 2010

1 Introduction

Social network analysis (SNA) studies patterns of interaction between social entities (Wasserman and Faust, 1994; Scott, 2000). In the past few decades, SNA has emerged as a major research paradigm in the social sciences (including economics), and has also attracted attention in other fields (Newman et al., 2006). While dedicated software for SNA exists (e.g., UCInet (Borgatti et al., 1999) or Pajek (Bagatelj and Mrvar, 2009)), Stata currently lacks readily available facilities for SNA. In this paper, we illustrate how methods for SNA can be developed in Stata, using *network visualization* as an example.

Visualization is one of the oldest methods in SNA and is still one of its most important and widely applied tools to uncover patterns of relations (Freeman, 2000). We describe a procedure for network visualization using Stata's built-in procedures for multidimensional scaling and describe an implementation as a Stata command. While we believe that network visualization in itself can be highly useful, the example also illustrates how SNA problems can be handled in Stata more generally.

2 Methods

2.1 Some terminology

Network visualization is concerned with showing binary relations between entities. Adopting the terminology of graph theory, we refer to these entities as *vertices*. Relations between vertices may be *directed*, if they can be understood as flowing *from* one vertex *to* another, or *undirected* if no such direction can be identified. We refer to directed relations as *arcs* and to undirected relations as *edges*. A typical representation of a network of relations is an *adjacency matrix*, as shown in Figure 1 for a network of 10 vertices. In this matrix, every cell represents a relation from a vertex (row) to another vertex (column); for undirected networks, this matrix is symmetric. Vertices that have no edges or arcs are called *isolates*. The number of edges connected to a vertex is called the *degree* of the vertex. Lastly, the *distance* between two vertices is defined as the shortest path between them. If there is no path between two isolates, we define the distance between them as infinite.

*I thank Jeroen Weesie for highly useful comments and suggestions. This paper is currently under review at the *Stata Journal*.

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	0	1	0	0	0	0	0
2	1	0	0	1	0	1	0	1	0	0
3	1	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	1	0	0	0
5	1	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	1	0	0	0
7	0	0	0	1	0	1	0	0	0	0
8	0	1	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	0

Figure 1: An adjacency matrix, $N = 10$

	col 1	col 2
1	2	1
2	3	1
3	4	2
4	5	1
5	6	2
6	7	6
7	7	4
8	8	2
9	9	8
10	10	.

Figure 2: Edgelist based on the adjacency matrix in Figure 1

2.2 Data structure

One particular obstacle in analyzing network data in conventional statistics packages such as Stata is the specific structure of *relational data*. Whereas in conventional datasets, one line in the data typically represents an individual entity, observations in relational datasets represent relations *between* entities.

We assume that data are available as a list of edges or arcs. That is, for a network of k relations, we have a $k \times 2$ data matrix, in which every row represents an edge (if the network is undirected) or arc (if the network is directed) between two vertices in the cells. The use of edgelist and arclists is an often more economical way to store network data than an adjacency matrix, especially for networks that are relatively sparse.

We extend the traditional edgelist and arclist formats by allowing the use of missing values. We use missing values to include isolates in the list (Fig. 2). In Figure 1, vertex 10 is isolated; in Figure 2, its vertex number appears in one column but is accompanied by a missing value in the other column.The order of appearance might be reversed. Thus, a network consisting of k edges and N vertices of which h isolates can be represented by a $(k + h) \times 2$ matrix.

2.3 Procedure

The main task in network visualization is to determine the positions of the vertices in a (typically two-dimensional) graphical layout. Obviously, the optimal placement of vertices depends on the purpose of the analysis, but it is often desirable to place vertices that have a central position in the network also centrally in the graphic and to represent larger distance in the network by a larger distance in the two-dimensional graph. Various algorithms have been proposed to solve this problem, of which those by Kamada and Kawai (1989) and Fruchterman and Reingold (1991) are probably most widely used. Instead, we use multidimensional scaling (MDS) to compute

coordinates for the vertices, which has the advantage of being available in Stata by default. The use of MDS for network visualization has a long history in SNA and was first used as such by Laumann and Guttman (1966).

Assuming that we have a relational dataset formatted as an edgelist, we propose to visualize the network by the following procedure:

1. Reshape the data into an adjacency matrix;
2. Compute the matrix of shortest paths (the distance matrix);
3. Compute coordinates for the vertices, arranged on a circle, in a random order;
4. Compute coordinates for the vertices by `mds`, using the "modern" method, and using the coordinates circle layout obtained in the previous step as starting positions;
5. Draw the graphic by combining the `twoway` plot types `pcspike` or `pcarrow` and `scatter`.

In our implementation, steps 1–3 are performed in Mata. The calculation of the distance matrix (step 2) involves calculating higher powers of the adjacency matrix, and can be rather time-consuming for larger networks. More efficient procedures for obtaining distances in a network are feasible, but not implemented in our example.

We choose Stata's iterative "modern" `mds` method for step 4 because it allows for the specification of starting positions and appears to provide better results in our tests. In particular, the modern method performs better than the "classic" method with regard the placement of vertices that have identical distances to all other vertices (for example, vertices in the periphery of a "star"). Experimentation furthermore suggests that starting with a circular layout provides the best results. [1]

3 Implementation: the `netplot` command

Syntax:

`netplot` *var1 var2* [*if*][*in*] [`, type(mds|circle) label arrows iterations(#)`]

The command `netplot` produces a graphical representation of a network stored as an extended edgelist or arclist in `var1` and `var2`.

`type(`*string*`)` specifies the type of layout. Valid values are:
`circle`: vertices are arranged on a circle
`mds`: positions of vertices are calculated using multidimensional scaling. This is the default; omitting `type()` is equivalent to specifying `type(mds)`.

`label` specifies that vertices are to be labeled using their identifiers in `var1` and `var2`

`arrows` specifies that arrows rather than lines are drawn between vertices. Arrows run from the vertex in `var1` to the vertex in `var2`. This option is useful for arclists that represent directed relations.

`iterations(#)` specifies the maximum number of iterations in the multidimensional scaling procedure. The default is `iterate(1000)`.

4 Examples

To illustrate the process outlined above we use a well-known dataset known as "Padgett's Florentine Families," which contains information on relations among 16 families in 15th century Florence, Italy (Padgett and Ansell, 1993). The part of the data we use represent marital relations between the families. These relations are by nature *undirected*. The data are described below:

[1] Internally, our program issues the command `mdsmat` *distance matrix*`, noplot method(modern) initialize(from(`*circle matrix*`)) iterate(#)`.

```
.
. desc
Contains data from padgett_marital02_undir.dta
  obs:            21                          Padgett marital data with
                                                undirected ties
 vars:             2                          22 Jan 2010 17:37
 size:           588 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
from            str12  %12s                   family 1 name
to              str12  %12s                   family 2 name
-------------------------------------------------------------------------------
Sorted by:  from  to
. list, sepby(from)

     +-------------------------+
     |       from          to  |
     |-------------------------|
  1. |                  Pucci  |
     |-------------------------|
  2. |    Albizzi     Guadagni  |
  3. |    Albizzi       Medici  |
     |-------------------------|
  4. |  Barbadori       Medici  |
     |-------------------------|
  5. |   Bischeri     Guadagni  |
  6. |   Bischeri      Peruzzi  |
  7. |   Bischeri      Strozzi  |
     |-------------------------|
  8. | Castellani    Barbadori  |
  9. | Castellani      Strozzi  |
     |-------------------------|
 10. |     Ginori      Albizzi  |
     |-------------------------|
 11. |   Guadagni Lamberteschi  |
     |-------------------------|
 12. |     Medici   Acciaiuoli  |
 13. |     Medici     Salviati  |
 14. |     Medici   Tornabuoni  |
     |-------------------------|
 15. |      Pazzi     Salviati  |
     |-------------------------|
 16. |    Peruzzi   Castellani  |
 17. |    Peruzzi      Strozzi  |
     |-------------------------|
 18. |    Ridolfi       Medici  |
 19. |    Ridolfi   Tornabuoni  |
     |-------------------------|
 20. |    Strozzi      Ridolfi  |
     |-------------------------|
 21. | Tornabuoni     Guadagni  |
     +-------------------------+
.
```

Note that the data are in this case formatted as strings, simply using the family names as identifiers for the vertices of the network.

The first example shows the most basic usage of `netplot`, using the command `netplot from to`, which produces a network plot of the data based as resulting from multidimensional scaling (Fig. 3).

In many analyses it is useful to be able to identify specific vertices in the network, which is facilitated by adding labels to the plot using the option `label` (Fig. 4). We can now observe that this network has a cohesive core formed by the Medici, Ridolfi and Tornabuoni families, and that the isolated vertex is the Pucci family. The rotation of the graph and the different location of the isolated vertex as compared to Figure 3 is caused by the randomized initial positions used in the

MDS procedure.[2]

Sometimes it is not necessary to have the relatively complicated plot as produced by multidimensional scaling. Then, a simple view on the data can be produced by the `circle` option (Fig. 5).

For our final example with these data, we assume that the data are *directed*. That is, we assume that each line in the data represents a relation *from* the vertex in "from" *to* the vertex in "to." Imagine, for instance, that the data now represent whether a family has ever sold goods to another family. Such situations can be visualized using the option `arrows`, which draws arrows instead of lines between the vertices (Fig. 6). The graph in this example was slightly adjusted afterwards using the Graph Editor, by reducing the sizes of the markers in order to make the arrowheads better visible.

As a final example, we draw a plot of a somewhat larger network of 100 vertices. The data for this example were simulated using the "preferential attachment" algorithm proposed by Albert and Barabási (1999) to construct the network (Fig. 7).[3]

This example highlights two limitations of `netplot`. First, as Figure 7 shows, vertex placement can be suboptimal: several vertices (such those labeled 63, 58, and 68) in the figure are placed too far from neighboring vertices, leading to many crossings of edges and an overall unpleasing result. The reason is that, in this particular tree-like network structure, there are many vertices (such as those mentioned) that have the exact same distance to all other vertices, which makes placement by MDS difficult. Second (not visible in the figure), the procedure becomes considerably more time-consuming with this number of vertices. We discuss this issue in more detail in the next section.

5 Performance

To get of rough idea of the performance of `netplot` in terms of computation time we conduct two simulated tests. First, we draw plots of networks of increasing network size, keeping network *density* constant at .5. Note that this leads to an exponentially increasing number of edges in the network. To draw the plots, we use `neplot` without any options. The input networks are randomly generated Erdös-Rényi graphs (Erdös and Rényi, 1959). [4]

For the second test, we again draw plots with increasing network size, but keep *average degree* constant rather than density. This implies a linear increase in the number of edges in the network. We use an average degree of 3.

In both test, we look at networks sizes ranging from 5 to 100 in steps of 5, and keep track of the time needed to draw a graph over 10 iterations per network size.

The results are shown in Figure 8. The figure indicates that average time increases quadratically with network size, although more strongly with constant density than with constant degree. Closer analysis (not reported) of the running time of the different components of the program reveals that the computation of coordinates using MDS (*after* computation of distances) is the most time-consuming step in the procedure.

6 Discussion

In this paper we have demonstrated how network data can be visualized in Stata, using the built-in techniques for multidimensional scaling and graphics. This method often produces useful results, although not always for all networks. Moreover, for larger networks the time needed to compute vertex coordinates can become a nuisance, especially when several trials are needed to achieve a

[2]The placement of labels outside the plotregion is part of the default behavior of `twoway scatter`, which is used by `netplot`. This can be easily adjusted afterwards.

[3]The actual simulation was conducted in Mata. The function in which the Albert-Barabasi algorithm is implemented is part of a larger library of functions for network analysis under development by the author.

[4]The tests were run in Stata 11 SE on a PC with a 2.66GHz dual core processor and 1Gb of memory, and running the Microsoft Windows XP 32-bit operating system.

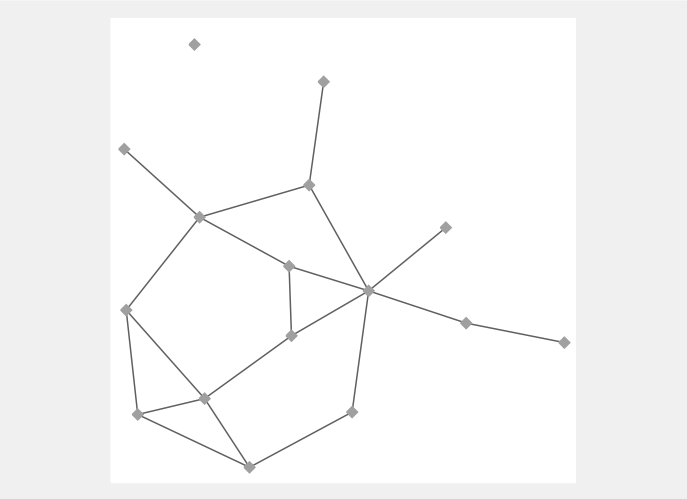

Figure 3: Example: Marital relations between Florentine families with vertex placement by MDS

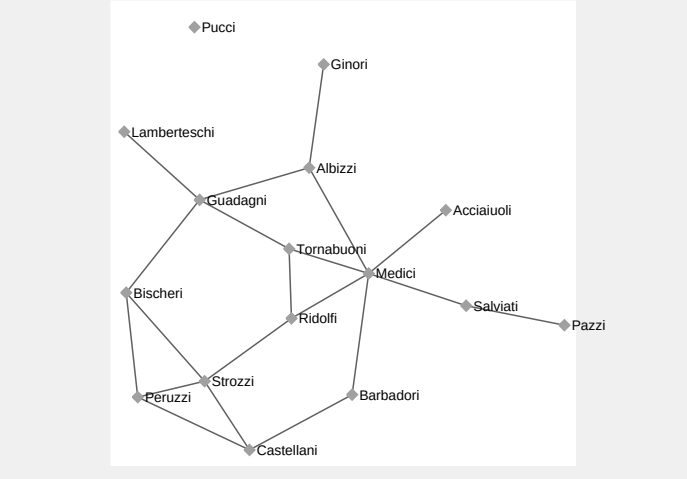

Figure 4: Example: Marital relations between Florentine families with vertex placement by MDS and labels added

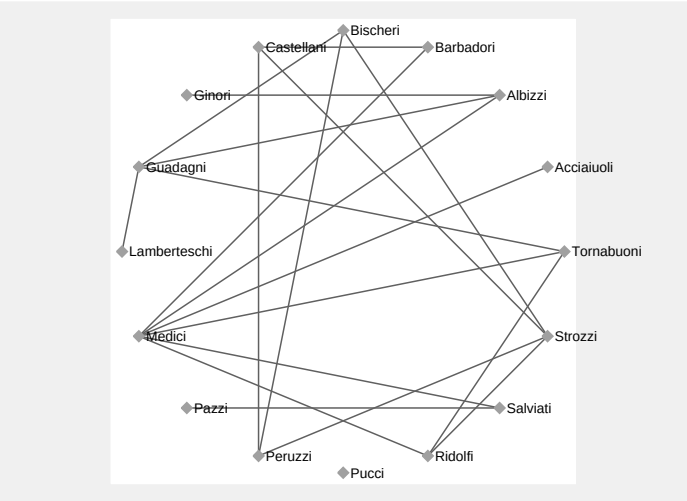

Figure 5: Example: Marital relations between Florentine families with circular vertex placement and labels

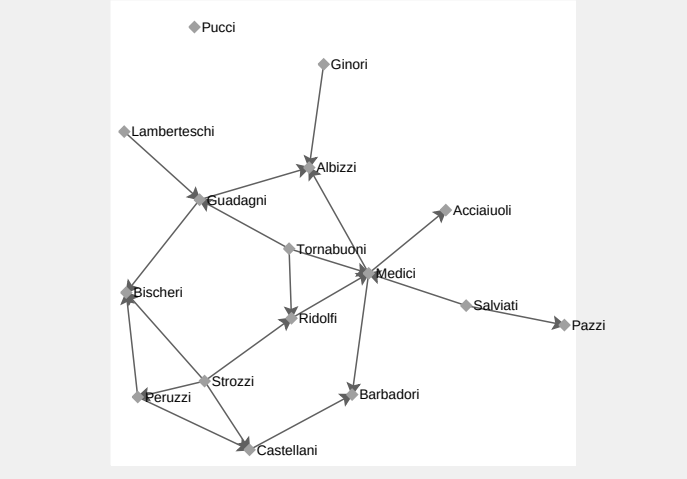

Figure 6: Example: Marital relations between Florentine families as directed relations with vertex placement by MDS and labels

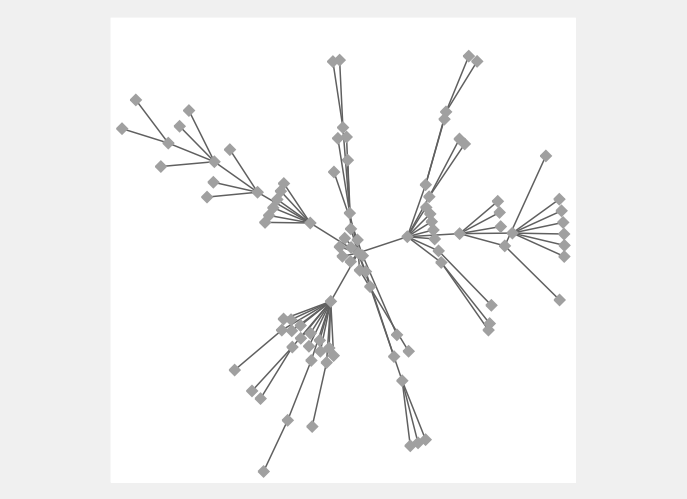

Figure 7: Example: A simulated network of 100 vertices with vertex placement by MDS

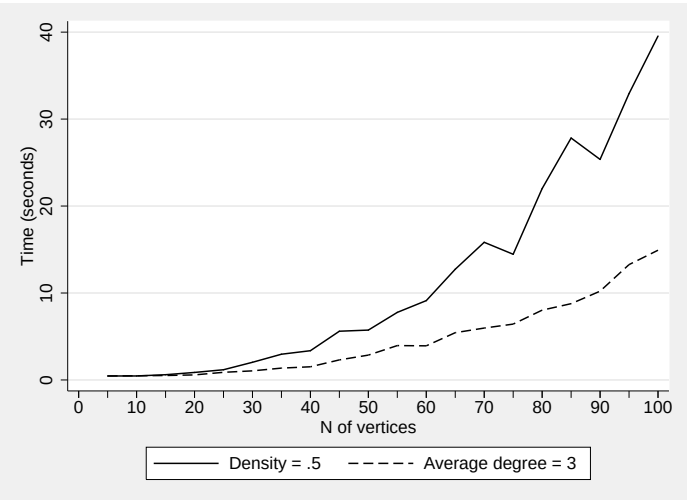

Figure 8: Average computation time by network size

satisfying result. As a workaround for this problem, the number of iterations may be limited using the `iterations()` option.

Visual results could likely be improved by using different vertex placement algorithms than MDS. Good candidates are the often used "spring embedding" algorithms by Kamada and Kawai (1989) and Fruchterman and Reingold (1991). Given the program architecture of `netplot`, these methods could be added relatively easily, and implementing them would be an obvious target for future development. It is, however, not clear how the Kamada-Kawai and Fruchterman-Reingold algorithms compare to MDS in terms of computation time.

Another approach to improving efficiency is to use more efficient methods for computing distances in the network. The simple approach currently implemented, based on repeated matrix squaring, computes quite some unneeded information in the process. More efficient algorithms for computing shortest paths exist (i.e., Cormen et al., 2001) and might be implemented in the future.

The introduction of Mata with Stata 9 has made matrix programming more effective and more accessible for the average user. This opens up further possibilities for the development of SNA methods in Stata. Especially the fact that Mata can be used interactively makes it easier to use the alternative data structures representing networks common in SNA. The quickly growing interest in social networks in and outside the social sciences certainly justifies the further development of SNA methods for Stata.

References

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

Bagatelj, V. and A. Mrvar (2009). Pajek.

Borgatti, S. P., M. G. Everett, and L. C. Freeman (1999). Ucinet 5.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001). *Introduction to Algorithm Design* (Second ed.). Cambridge, MA: MIT Press.

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

Freeman, L. C. (2000). Visualizing social networks. *Journal of Social Structure 1*(1).

Fruchterman, T. M. J. and E. M. Reingold (1991). Graph drawing by force-directed placement. *Software-Practice and Experience 21*(1 1), 1129–1164.

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

Laumann, E. O. and L. Guttman (1966). The relative associational contiguity of occupations in an urban setting. *American Sociological Review 31*(2), 169–178.

Newman, M., A.-L. Barabási, and D. Watts (Eds.) (2006). *The Structure and Dynamics of Networks*. Princeton, NJ: Princeton University Press.

Padgett, J. F. and C. K. Ansell (1993). Robust action and the rise of the medici, 1400-1434. *The American Journal of Sociology 98*(6), 1259–1319.

Scott, J. (2000). *Social Network Analysis: A Handbook* (second ed.). London: Sage.

Wasserman, S. and K. Faust (1994). *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.