Stata Graph Library for Network Analysis

(Preliminary and Incomplete)*

Hirotaka Miura
Federal Reserve Bank of San Francisco

June 19, 2011

Abstract

Network analysis is a multidisciplinary research method that is fast becoming a popular and exciting field of study. Though a number of statistical programs possess sophisticated packages for analyzing networks, similar capabilities have yet to be made available in Stata. In an effort to motivate the use of Stata for network analysis, I design the Stata Graph Library (SGL) in Mata, which consists of algorithms that construct matrix representations of networks, compute centrality measures, and calculate clustering coefficients. Performance tests conducted between C++ and SGL implementations indicate gross inefficiencies in current SGL routines, making SGL practically infeasible to be used for large networks. The obstacles are, however, welcome challenges in the effort to spread the use of Stata as an instrument for analyzing networks, and future developments will focus on addressing computational time complexities as well as integrating additional capabilities into SGL.

Keywords: network, centrality, clustering

*Under review at the *Stata Journal*. Correspondence: Federal Reserve Bank of San Francisco, 101 Market Street, San Francisco, CA 94105. hirotaka.miura@sf.frb.org, hirotaka.miura@gmail.com. I would like to give special thanks to Phil Schumm for providing me with guidance on the structure of the program and Rense Corten for his valuable feedbacks on the program and the initial draft paper. I wish to thank one anonymous reviewer and Joseph Newton for helping me improve the program and the documentation. I am grateful for Siek et al. (2001, Boost Graph Library), Gleich (2008, MatlabBGL), Butts (2010, sna), and Csardi and Nepusz (2006, igraph) for making available their programs and source codes. I would like to express my profound gratitude to Galina Hale for introducing me to network analysis and supporting me throughout my work on this project.

1 Introduction

1.1 What is network analysis?

Network analysis is an application of network theory, which is a subfield of graph theory, and is concerned with analyzing relational data. Some questions network analysis attempts to address is how important, or how central, individual actors are in a given network, and how concentrated is the network. Example usages of network analysis include:

- Determining the importance of a web page using Google's PageRank.
- Examining communication networks in intelligence and computer security.
- Solving transportation problems that involve flow of traffic or commodities.
- Addressing the too-connected-to-fail problem in financial networks.
- Analyzing social relationships between individuals in social network analysis.

1.2 Modeling relational data

A graph model representing a network $G = (V, E)$ consists of a set of vertices V and a set of edges E. The size of set V, or the number of vertices in network G, is denoted $|V|$ and similarly the size of set E, or the number of edges in network G, is denoted $|E|$. An edge is defined as a link between two vertices i and j, not necessarily distinct, that has vertex i on one end and vertex j on the other. An edge may be directed or undirected and may also be weighted with differing edge values or have all equal edge values of one in which case the network is said to be unweighted.

There are special types of vertices and edges that standard graph algorithms cannot handle or there simply does not exist routines to accommodate such types. Thus the following types of vertices and edges are currently excluded from analysis:

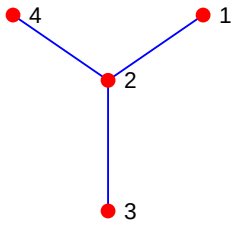
- Isolated vertex - a vertex that is not attached to any edges.
- Parallel edges - two or more edges that connect the same pair of vertices.
- Self-loop - an edge connecting vertex i to itself.
- Zero or negative weighted edge.

A variety of methods exists for capturing relational data, with the adjacency matrix and adjacency list forms being some the more widely used storage types. In Stata however, as it will be demonstrated later on, capturing relational data in a coordinate list, or an edge list, is more advantageous as it allows the user to employ Stata's built-in capabilities such as the ability to restrict the scope of the analysis by specifying `if` *expression* and `in` *range* options.

1.3 Edge list

An edge list for an undirected unweighted network is a $|E| \times 2$ matrix where each row represents an edge between vertices i and j. A directed unweighted network is defined similarly with a $|E| \times 2$ matrix capturing information on directed edges from source vertex i to target vertex j. A weighted network can be represented by adding a third column containing edge weights.

	v1	v2
1.	1	2
2.	2	3
3.	4	2



Example of an edge list and its corresponding graph (undirected and unweighted). Drawn using NETPLOT (Corten, 2011).

It is important to note that substantial modifications may be needed to arrive at an edge list from an initial dataset. The task of modifying initial data applies to datasets in both long and wide formats, as well as data in some kind of a matrix form.

2 Matrix representation

2.1 Adjacency matrix

Adjacency matrix $\mathbf{A}$ for unweighted networks is defined as a $|V| \times |V|$ matrix with A_{ij} entries being equal to one if an edge connects vertices i and j and zero otherwise. A_{ii} entries are set to zero and matrix $\mathbf{A}$ is symmetric if the network is undirected. For directed networks, rows of matrix $\mathbf{A}$ represent outgoing edges and columns represent incoming edges.[1] For weighted networks, A_{ij} entries are equal to the weight of the edge connecting vertices i and j.

2.2 Distance matrix

Distance matrix $\mathbf{D}$ is defined as a $|V| \times |V|$ matrix with D_{ij} entries being equal to the length of the shortest path between vertices i and j, where path is defined as a way of reaching vertex j starting from vertex i using a combination of edges that do not go through a particular vertex more than once. If no such path exists between vertices i and j, D_{ij} is set to missing, signifying what is sometimes referred to as an infinite path. D_{ii} is set to zero. For undirected networks, matrix $\mathbf{D}$ is symmetric.

2.3 Path matrix

Path matrix $\mathbf{P}$ is defined as a $|V| \times |V|$ matrix with P_{ij} entries being equal to the *number* of shortest paths between vertices i and j. If no paths exist between vertices i and j, P_{ij} is set to zero. P_{ii} is set to one. $\mathbf{P}$ matrix is symmetric for undirected networks.

3 Centrality measures

3.1 Degree centrality

Degree centrality measures the importance of a vertex by the number of connection the vertex has if the network is unweighted, and by the aggregate of the weights of edges connected to the vertex if the network is weighted. For an undirected network, degree centrality for vertex i is defined as

$$\frac{1}{|V|-1} \sum_{j(\neq i)} A_{ij} \tag{1}$$

[1] The convention of denoting X_{ij} entries as an edge from i to j is adopted for all matrices.

where the leading divisor is adjusted for the exclusion of the $j = i$ term. Directed networks may entail vertices having different number of incoming and outgoing edges, and thus we have out-degree and in-degree centrality. Out-degree centrality for vertex i is defined similarly to equation (1). For in-degree, we simply transpose the adjacency matrix:

$$\frac{1}{|V|-1}\sum_{j(\neq i)} A'_{ij}. \tag{2}$$

3.2 Closeness centrality

Closeness centrality provides higher centrality scores to vertices that are situated closer to members of their component, or the set of reachable vertices, by taking the inverse of the average shortest paths as a measure of proximity. That is, closeness centrality for vertex i is defined as

$$\frac{(|V|-1)}{\sum_{j(\neq i)} D_{ij}}, \tag{3}$$

which reflects how vertices with smaller average shortest path lengths receive higher centrality scores than those that are situated farther away from members of their component.

An immediate concern in computing equation (3) is how to deal with infinite distances to unreachable vertices. A common workaround is to average over only those vertices that are reachable. However, caution must be exercised as distances between vertices tend to be shorter in smaller components, possibly resulting in vertices in such components receiving higher closeness centrality scores than vertices in larger components, going against the notion that vertices in small components are less central in terms of the entire network.

3.3 Betweenness centrality

Betweenness centrality bestows larger centrality scores on vertices that lie on a higher proportion of shortest paths linking vertices other than itself. Let P_{ij} denote the number of shortest paths from vertex i to j, as defined above. Let $P_{ij}(k)$ denote the number of shortest paths from vertex i to j that vertex k lies on. Then following Anthonisse (1971) and Freeman (1977), betweenness centrality measure for vertex k is defined as

$$\sum_{ij:i\neq j,k\notin ij} \frac{P_{ij}(k)}{P_{ij}}. \tag{4}$$

To normalize (4), divide by $(|V|-1)(|V|-2)$, the maximum number of paths a given vertex could lie on between pairs of other vertices.[2]

3.4 Eigenvector centrality

Eigenvector centrality can provide an indication on how important a vertex is by having the property of being large if a vertex has many neighbors, important neighbors, or both. The measure first proposed by Bonacich (1972) defines the centrality of vertex i, x_i, as the sum of the centrality of its neighbors scaled by a constant. That is, for an undirected network with adjacency matrix $\mathbf{A}$,

$$x_i = \lambda^{-1}\sum_j A_{ij}x_j \tag{5}$$

which can be rewritten as

$$\lambda\mathbf{x} = \mathbf{A}\mathbf{x}. \tag{6}$$

[2] Actual implementation is carried out for undirected networks such that both $P_{ij}(k)$ and $P_{ji}(k)$ are calculated and thus the numerator need not be multiplied by two.

Vector $\mathbf{x}$ in equation (6) is then an eigenvector of adjacency matrix $\mathbf{A}$ and λ is its corresponding eigenvalue. The convention is to use the eigenvector corresponding to the dominant eigenvalue of $\mathbf{A}$. When the network is directed, the general concern is in obtaining a centrality measure based on how often a vertex is being pointed to and the importance of neighbors associated with the incoming edges. Thus with a slight modification to equation (6), eigenvector centrality is redefined as a vector $\mathbf{x}$ that satisfies

$$\lambda \mathbf{x} = \mathbf{A}'\mathbf{x} \tag{7}$$

where $\mathbf{A}'$ is the transposed adjacency matrix. As discussed in detail in (Newman, 2010, chap. 7 sec. 2), there are several shortcomings to the eigenvector centrality, such as the fact that a vertex with no incoming edges will always have centrality of zero. Furthermore, vertices with neighbors that all have zero incoming edges will also have zero centrality since the sum in equation (5) will not have any terms.

The Katz-Bonacich centrality, a variation of the eigenvector centrality, seeks to address these issues.

3.5 Katz-Bonacich centrality

The additional inclusion of a free parameter (also referred to as a decay factor) and a vector of exogenous factors into equation (7) avoids the exclusion of vertices with zero incoming edges while allowing connection values to decay over distance and is attributed to the culmination of works by Katz (1953), Bonacich (1987), and Bonacich and Lloyd (2001). The centrality measure is defined as a solution to the equation

$$\mathbf{x} = \alpha \mathbf{A}'\mathbf{x} + \boldsymbol{\beta} \tag{8}$$

where α is the free parameter and $\boldsymbol{\beta}$ is the vector of exogenous factors which can vary or be constant across vertices. For the centrality measure to converge properly, the absolute value of α must be less than the absolute value of the inverse of the dominant eigenvalue of $\mathbf{A}$. A positive α allows vertices with important neighbors to have higher status while a negative α value reduces the status.

4 Clustering coefficient

Clustering coefficient is one way of gauging how tightly connected a network is. The general idea is to consider transitive relations, that is, if vertex j is connected to vertex i, and i is connected to k, then j is also connected to k.

Global clustering coefficients provide indication on the degree of concentration of the entire network and consists of overall and average clustering coefficients. Overall clustering coefficient is equal to all observed transitive relations divided all possible transitive relations in the network. Average clustering coefficient involves applying the definition of overall clustering coefficient at the vertex level, then averaging across all the vertices.

For an undirected unweighted adjacency matrix $\mathbf{A}$, overall clustering coefficient is defined as

$$c^o(\mathbf{A}) = \frac{\sum_{i;j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}A_{jk}}{\sum_{i;j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}} \tag{9}$$

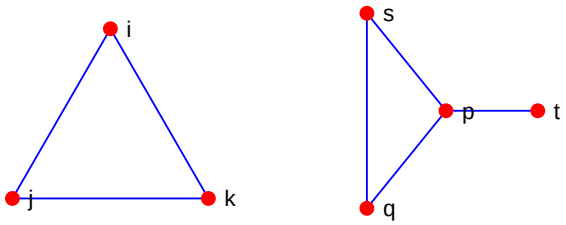
where the numerator represents the sum over i of all closed triplets in which transitivity holds, and the denominator represents the sum over i of all possible triplets. With a slight modification in notation, local clustering coefficient for vertex i is defined as

$$c_i(\mathbf{A}) = \frac{\sum_{j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}A_{jk}}{\sum_{j\neq i;k\neq j;k\neq i} A_{ji}A_{ik}} \tag{10}$$

which leads to the average clustering coefficient:

$$c^a(\mathbf{A}) = \frac{1}{|V|}\sum_i c_i(\mathbf{A}). \tag{11}$$

By convention, $c_i(\mathbf{A}) = 0$ if vertex i has zero or only one link. Since average clustering coefficient computes clustering coefficients for each vertex and then averages across all vertices, the coefficient gives more weight to low-degree vertices than does overall clustering coefficient, which takes the average over all triplets.



(left) A closed triplet. $c_i = 1$. (right) Closed and open triplets. $c_p = 1/3$. Drawn using NETPLOT (Corten, 2011).

Generalized methods exist for computing clustering coefficients. Building upon the works of Barrat et al. (2004), Opsahl and Panzarasa (2009) propose a set of measures consisting of four types of coefficients based on arithmetic mean, geometric mean, maximum, and minimum of triplet values. Clustering coefficients for vertex i based on weighted adjacency matrix $\mathbf{W}$ and corresponding unweighted adjacency matrix $\mathbf{A}$ are calculated as

$$c_i(\mathbf{W}) = \frac{\sum_{j\neq i;k\neq j;k\neq i} \omega A_{jk}}{\sum_{j\neq i;k\neq j;k\neq i} \omega} \tag{12}$$

where ω equals $(W_{ji} + W_{ik})/2$ for arithmetic mean, $\sqrt{W_{ji} \times W_{ik}}$ for geometric mean, $\max(W_{ji}, W_{ik})$ for maximum, and $\min(W_{ji}, W_{ik})$ for minimum.[3] For unweighted networks, $\mathbf{W} = \mathbf{A}$ and the four types of clustering coefficients are all equal. For unweighted undirected networks, the overall, local, and average clustering coefficients are equal to equations (9), (10), and (11), respectively.

5 Mata implementation

5.1 Syntax

struct structname scalar	`bfs_sp(`**E**`,` *nodots*`)`
struct structname scalar	`dijkstra_sp(`**E**`,` *nodots*`)`
real vector	`bfs_betweenness(`**E**`,` *nodots*`)`
real vector	`dijkstra_betweenness(`**E**`,` *nodots*`)`
struct structname scalar	`clustering_coefficients(`**E**`,` *nodots*`)`
struct structname scalar	`power(`**E**`,` *nodots*`,` *left*`)`
real vector	`power_katzbonacich(`**E**`,` *alpha*`,` *beta*`,` *nodots*`,` *left*`)`
real matrix	`sparse2full(`**E**`,` *initial*`)`

where

[3] Calculations for $\{W_{ki}, W_{ij}\}$ pair are also conducted.

structname:	`matrix_struct`	(structure with real matrix variables `X1` through `X6`)
E:	*real matrix* **E**	($\|E\| \times 3$ edge list)
alpha:	*real scalar alpha*	(alpha parameter for Katz-Bonacich centrality)
beta:	*real vector beta*	($\|V\| \times 1$ beta vector for Katz-Bonacich centrality)
nodots:	*string scalar nodots*	(optional indicator to suppress display of status dots)
left:	*string scalar left*	(optional indicator to transpose adjacency matrix)
initial:	*real scalar initial*	(optional initial real value of full matrix)

5.2 Description

This section assumes familiarity with Mata and matrix programming. Otherwise please see [M-0] **intro**.

Structure `matrix_struct` is already compiled as part of the Stata Graph Library (SGL). Thus, users will want to define a variable of type `struct matrix_struct scalar` in their functions in order to retrieve results from compiled functions that return objects as members of the structure. For example, here is a function that returns a distance matrix for an unweighted network that takes in an edge list matrix with optional parameters for suppressing status dots and replacing missing values:

(Lines enclosed with "`/* */`" denote comments)

```
real matrix distance_matrix(real matrix E, |string scalar nodots,
   real scalar infinity){
   /* Declare variables.  */
   struct matrix_struct scalar mystruct
   real matrix D
   /* Implement breadth-first search algorithm.  */
   mystruct=bfs_sp(E,nodots)
   /* Replace missing values.  */
   D=editmissing(mystruct.X1,infinity)
   /* Return distance matrix.  */
   return(D)
}
```

In the following descriptions, we assume that the user has declared a variable `mystruct` of type `struct matrix_struct scalar`. See [M-2] **struct**.

`bfs_sp(`**E**`,...)` returns the distance matrix, path matrix, leaf vertex matrix, and adjacency list for an unweighted network in `mystruct.X1`, `mystruct.X2`, `mystruct.X3`, and `mystruct.X4`, respectively. Dimensions of distance, path, and leaf vertex matrices are $|V| \times |V|$ and dimensions of the adjacency list are $|V| \times m$ where m denotes the largest number of outgoing edges among all vertices in the network.

`dijkstra_sp(`**E**`,...)` returns the distance and path matrices for a weighted network in `mystruct.X1` and `mystruct.X2`, respectively. Dimensions of distance and path matrices are $|V| \times |V|$.

`bfs_betweenness(`**E**`,...)` returns the betweenness centrality vector for an unweighted network. Dimensions of the returned vector are $|V| \times 1$.

`dijkstra_betweenness(`**E**`,...)` returns the betweenness centrality vector for a weighted network. Dimensions of the returned vector are $|V| \times 1$.

`clustering_coefficients(`**E**`,...)` returns the overall and average clustering coefficients in `mystruct.X1` and the local clustering coefficients in `mystruct.X2`. `mystruct.X1` contains a 2×4 matrix and `mystruct.X2` contains a $|V| \times 4$ matrix.

`power(`**E**`,...)` returns the dominant eigenvector in `mystruct.X1` and the dominant eigenvalue in `mystruct.X2` if convergence is achieved. If convergence is not achieved, Mata error code 3360 is returned in `mystruct.X1`. The returned eigenvector has dimensions $|V| \times 1$.

`power_katzbonacich(`**E**, *alpha*, *beta*`,...)` returns the Katz-Bonacich centrality vector if convergence is achieved and Mata error code 3360 if convergence is not achieved. Scalar `alpha` is the free parameter and

`beta` is the $|V| \times 1$ vector of exogenous factors. The returned Katz-Bonacich centrality vector has dimensions $|V| \times 1$.

`sparse2full(E,...)` returns the full matrix corresponding to the inputted edge list matrix **E**. The returned full matrix has dimensions $|V| \times |V|$.

nodots option specifies whether or not to display status dots. If *nodots* is set to a non-empty string, status dots are suppressed. Otherwise if *nodots* is set to "" or is not set, status dots are displayed.

left option specifies that the power method be implemented to calculate the left dominant eigenvector. If *left* is set to "" or is not set, the right dominant eigenvector is calculated.

initial option specifies initial values to fill the full matrix. If the option is not set, the default is to fill the full matrix with all missing values.

5.3 Remarks

The Stata Graph Library is compiled using Stata version 11.1.

Unless otherwise noted, all algorithms are based on those that are described in Newman (2010). Modifications are made when necessary or when additional efficiency/speed gains can be realized within the Mata environment.

Without loss in generality, all routines assume a directed network. An undirected edge can be thought of as two directed edges, one going from vertex i to j and another from j to i. Therefore, when working with an undirected network, the edge list matrix **E** should have reciprocal relations defined. If reciprocal relations are not defined, such that only the edge from vertex i to j is stored but not the edge from j to i, the edge list matrix must be redefined as

$$\mathbf{E} = \texttt{uniqrows}((((\mathbf{E}[.,1] \backslash \mathbf{E}[.,2]), (\mathbf{E}[.,2] \backslash \mathbf{E}[.,1])), (\mathbf{E}[.,3] \backslash \mathbf{E}[.,3]))) \,, \tag{13}$$

where `uniqrows()` returns sorted, unique values. See [M-5] **uniqrows( )**.

Caution must be exercised when working with undirected weighted networks, or in the process of constructing such networks. If the initial edge list matrix contains (i, j, w_1) and (j, i, w_2) where $w_1 \neq w_2$, then equation (13) would produce (i, j, w_1), (j, i, w_2), (j, i, w_1), and (i, j, w_2), resulting in parallel edges and violating one of the assumptions laid out in section 1.2. In such cases, appropriate aggregation of edge weights may need to be conducted beforehand to avoid generating parallel edges.

The third column consisting of edge weights is required. If calling `dijkstra_sp()` on an undirected network with edge weights stored as `double` fails to produce symmetric matrices, try converting edge weights to `float` data type. See [M-5] **floatround( )**.

`bfs_sp()` implements breadth-first search single-source shortest-path algorithm.

`bfs_betweenness()` makes a call to `bfs_sp()` and uses the resulting matrices to compute betweenness centrality.

`dijkstra_sp()` implements Dijkstra's single-source shortest-path algorithm. The routine first runs breadth-first search algorithm to determine the component for each source vertex, that is, the set of vertices that are reachable from the source vertex by at least one path.

`dijkstra_betweenness()` does not make a call to `dijkstra_sp()`, but instead uses Dijkstra's algorithm internally to generate information required in computing betweenness centrality for weighted networks.

`clustering_coefficients()` returns a 2×4 matrix in `mystruct.X1` with the first row corresponding to overall coefficients and the second row average coefficients. Columns one through four correspond to coefficients calculated based on arithmetic mean, geometric mean, maximum, and minimum, respectively. The $|V| \times 4$ matrix returned in `mystruct.X2` maintains the same column ordering, but contains local clustering coefficients instead. When the network is unweighted, the four calculation methods produce the same number.

`power()` and `power_katzbonacich()` routines implement the power method using the sparse matrix, or

the coordinate list data structure of edge list matrix **E**. Convergence criteria consists of two rules: The maximum number of iterations, which is set at 16,000, and the maximum relative difference, which signals convergence if `mreldif()` drops below 1e-10. See [M-5] **reldif( )**. Depending on the size and density of the network, the power method may provide faster and equally accurate results compared with using Mata's built-in linear algebra functions.

6 Stata implementation (1): SGL wrapper

6.1 Syntax

`network` *varname_source varname_target* [*if*] [*in*] `, measure(`*network_measure*`)` [`name(`*mata_mname* [`, ...`]`)` `weight(`*weightvar*`)` `label(`*var_prefix* [`, ...`]`)` `directed` `nodots` `infinity(`*real*`)` `power` `notranspose` `alpha(`*real*`)` `beta(`*beta_varlist*`)`]

where *network_measure* can be one of

`adjacency`	adjacency matrix
| `distance`	distance matrix
| `path`	path matrix
| `betweenness`	betweenness centrality
| `clustering`	local and overall/average clustering coefficients
| `eigenvector`	eigenvector centrality
| `maxalpha`	maximum free parameter alpha
| `katzbonacich`	Katz-Bonacich centrality.

varname_source and *varname_target* must either be both numeric or both string type variables. When working with an undirected network, the notion of *source* and *target* does not matter. Furthermore, reciprocal relations for undirected networks do not need to be defined when using the `network` command in Stata. Caution must still be exercised when working with weights, however, as outlined in section 5.3. *weightvar* and *beta_varlist* must be type numeric with variables of *beta_varlist* corresponding to the order of *varname_source* and *varname_target*.

`network` will return an error code of 198 and exit if parallel edges, self-loops, and/or non-positive weights are encountered. Otherwise isolated vertices, edges with missing weights (if *weightvar* specified), and vertices with missing beta exogenous factor (if *beta_varlist* specified) will automatically be excluded from the analysis sample.

`measure(`*network_measure*`)` specifies the network measure to be computed. This option is required.

`name(`*mata_mname* [`, replace`]`)` specifies the Mata matrix name to store results. `replace` requests existing Mata matrix be overwritten.

`weight(`*weightvar*`)` specifies a numeric edge weight variable. By default, edge weights are set to one for all vertices internally and thus an unweighted network is assumed.

`label(`*var_prefix* [`, replace`]`)` requests vertex labels be returned to Stata with variable names prefixed with *var_prefix* and suffixed with `_source` and `_target` for *varname_source* and *varname_target*, respectively. `replace` requests existing same-named label variables be overwritten.

`directed` specifies directed edges. By default, edges are assumed to be undirected.

`nodots` requests that status dots be suppressed.

`infinity(`*real*`)` specifies a real number to replace missing or "infinite" distances. Scalars returned in `e()` refer to matrix before replacement. `name(`*mata_mname*`)` contains distance matrix after replacement. The option only applies when generating distance matrix.

`power` requests the power method be implemented in computing eigenvector centrality, maximum alpha, and Katz-Bonacich centrality. By default, Mata's built-in linear algebra functions are used.

`notranspose` requests that the adjacency matrix not be transposed when computing eigenvector and Katz-Bonacich centralities. By default, adjacency matrix is transposed.

`alpha(`*real*`)` specifies a real number for the free parameter in Katz-Bonacich centrality calculation. By default, alpha is set to one.

`beta(`*beta_varlist*`)` specifies exogenous factors for the Katz-Bonacich centrality measure. By default, exogenous factors are set to one for all vertices.

6.2 Remarks

`network` is a wrapper for class `network_measure`, which in turn is a wrapper for functions compiled in SGL. As SGL functions are each designed to carry out specific tasks, a class wrapper is useful in providing additional organization. See [M-2] **class**.

`network` does not store results as Stata matrices. See [M-5] **st_matrix( )** on how to transfer Mata matrices from/to Stata.

The recommended approach to using Katz-Bonacich function is to first obtain maximum alpha using `maxalpha()` and then based on the returned value, specify `alpha()` option while calling `katzbonacich()`. Note that singular or near-singular matrices pose computational problems and thus `eigenvector()`, `maxalpha()`, and `katzbonacich()` functions may not converge in such cases.

`..., measure(clustering) name(`*mata_mname*`)` stores *mata_mname* matrix in Mata with columns one through four corresponding to local clustering coefficients based on arithmetic mean, geometric mean, maximum, and minimum, respectively.

6.3 Saved results

`network, measure(`*network_measure*`)`, where *network_measure* is equal to one of `adjacency`, `distance`, `path`, `betweenness`, or `katzbonacich`, saves the following in `e()`:

Scalars

`e(vertices)`	number of vertices	`e(edges)`	number of edges
`e(mean)`	`mean(mean(X)')`	`e(min)`	`min(X)`
`e(max)`	`max(X)`	`e(sum)`	`sum(X)`
`e(missing)`	`missing(X)`	`e(nonmissing)`	`nonmissing(X)`

`network, measure(clustering)` saves the following in `e()`:

Scalars

`e(vertices)`	number of vertices	`e(edges)`	number of edges

Matrices

`e(cc)`	overall and average coefficients

`network, measure(eigenvector)` saves the following in `e()`:

Scalars

`e(vertices)`	number of vertices	`e(edges)`	number of edges
`e(l)`	dominant eigenvalue	`e(mean)`	`mean(mean(X)')`
`e(min)`	`min(X)`	`e(max)`	`max(X)`
`e(sum)`	`sum(X)`	`e(missing)`	`missing(X)`
`e(nonmissing)`	`nonmissing(X)`		

`network, measure(maxalpha)` saves the following in `e()`:

Scalars

`e(vertices)`	number of vertices	`e(edges)`	number of edges
`e(alpha_max)`	maximum alpha		

In addition, all network measures return the following in `e()`:

Macros

`e(cmd)`	`network`	`e(cmdline)`	command as typed
`e(source)`	source vertex variable	`e(target)`	target vertex variable
`e(measure)`	network measure	`e(edge)`	`unweighted` or `weighted`
`e(network)`	`undirected` or `directed`		

7 Stata implementation (2): postcomputation command

7.1 Syntax

`netsummarize` *mata_exp*`, generate(`*newvar_prefix*`) statistic(`*stat_name*`)`

where *stat_name* can be one of

`mean`	`mean(mean(`*mata_exp*`)')`
`min`	`min(`*mata_exp*`)`
`max`	`max(`*mata_exp*`)`
`sum`	`sum(`*mata_exp*`)`
`missing`	`missing(`*mata_exp*`)`
`nonmissing`	`nonmissing(`*mata_exp*`)`
`rowmean`	`mean(`*mata_exp*`')'`
`rowmin`	`rowmin(`*mata_exp*`)`
`rowmax`	`rowmax(`*mata_exp*`)`
`rowsum`	`rowsum(`*mata_exp*`)`
`rowmissing`	`rowmissing(`*mata_exp*`)`
`rownonmissing`	`rownonmissing(`*mata_exp*`)`
`colmean`	`mean(`*mata_exp*`)'`
`colmin`	`colmin(`*mata_exp*`)'`
`colmax`	`colmax(`*mata_exp*`)'`
`colsum`	`colsum(`*mata_exp*`)'`
`colmissing`	`colmissing(`*mata_exp*`)'`
`colnonmissing`	`colnonmissing(`*mata_exp*`)'`

mata_exp must be a Mata matrix or a Mata expression and must evaluate to either a scalar, a $|V| \times 1$ column vector, or a $|V| \times |V|$ matrix. New variables *newvar_prefix*`_source` and *newvar_prefix*`_target` are generated for *varname_source* and *varname_target*, respectively. See [M-5] **sum( )**, [M-5] **mean( )**, [M-5] **missing( )**, and [M-5] **minmax( )**.

7.2 Remarks

Constructing network measures can be time intensive. The postcomputation command allows the user to generate multiple statistics once a network object has been created. For example, after generating a distance matrix **D**, the user can generate variables for closeness centrality with the command

```
netsummarize (rows(D)-1):/rowsum(D), generate(closeness) statistic(rowsum).
```
[4]

Specifying `rowmin` or `rowmax` would also have worked, since the expression evaluates to a column vector, as well as having set closeness centrality to a Mata vector beforehand and inserting it as *mata_exp*.

[4] When working with networks involving disconnected components, users may want to specify `(rownonmissing(D)-J(rows(D),1,1)):/rowsum(D)` as the *mata_exp*.

8 Examples

8.1 Creating an edge list using `joinby`

Here, we illustrate a method of creating an edge list using the `joinby` command and example datasets used in [D] *Data-Management Reference Manual*, `child.dta` and `parent.dta`, to generate a directed network where directed edges can represent parent vertices providing care to child vertices. Such a network is called bipartite, meaning there are two groups of vertices, parent and child, with edges lying between the two groups.

```
. use child, clear
(Data on Children)
. list
```

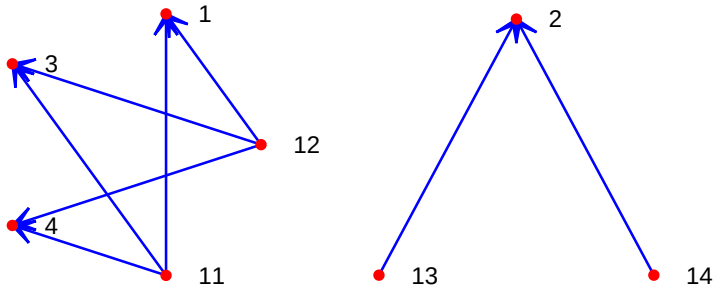
	family~d	child_id	x1	x2
1.	1025	3	11	320
2.	1025	1	12	300
3.	1025	4	10	275
4.	1026	2	13	280
5.	1027	5	15	210

```
. use parent, clear
(Data on Parents)
. list
```

	family~d	parent~d	x1	x3
1.	1030	10	39	600
2.	1025	11	20	643
3.	1025	12	27	721
4.	1026	13	30	760
5.	1026	14	26	668
6.	1030	15	32	684

```
. sort family_id
. joinby family_id using child
. list parent_id child_id
```

	parent~d	child_id
1.	12	4
2.	12	1
3.	12	3
4.	11	4
5.	11	3
6.	11	1
7.	14	2
8.	13	2



A network with directed edges from parent to child generated using `joinby`. Drawn using NETPLOT (Corten, 2011).

8.2 Workflow using PAJEK2STATA, `network`, and `netsummarize`

Next we present an example workflow using PAJEK2STATA (Corten, 2010), `network`, and `netsummarize` to convert a Pajek (Batagelj and Mrvar, 2010) .net file to Stata format then computing network centrality measures. The dataset consisting of fifteenth-century Florentine marriages is from Padgett and Ansell (1993). The network is undirected and unweighted with isolated vertex Pucci removed.

```
.   // Transfer Pajek´s .net file into Stata using Rense Corten´s
.   // pajek2stata.ado.
.   pajek2stata using florentine_marriages.net, name(X) clear replace
```

```
Vertices:                          15
Network matrix format:             Edges
Network matrix shape (r x c):      20 X 2
```

```
.   // Display edge list saved as Mata matrix.
.   mata X
```

	1	2
1	1	2
2	1	3
3	1	4
4	2	3
5	2	5
6	3	6
7	3	4
8	4	7
9	5	8
10	6	8
11	6	9
12	8	9
13	8	10
14	8	11
15	8	12
16	9	7
17	7	13
18	7	10
19	10	14
20	11	15

```
.   // Display vertex label.
.   list
```

	var1	var2
1.	1	Peruzzi
2.	2	Castellan
3.	3	Strozzi
4.	4	Bischeri
5.	5	Barbadori

6.	6	Ridolfi
7.	7	Guadagni
8.	8	Medici
9.	9	Tornabuon
10.	10	Albizzi
11.	11	Salviati
12.	12	Acciaiuol
13.	13	Lambertes
14.	14	Ginori
15.	15	Pazzi

```
. destring var1, replace
var1 has all characters numeric; replaced as byte
. // Generate new value-label.  See [M-5] st_vlexists().
. mata st_vlmodify("vertex_label",st_data(.,"var1"),st_sdata(.,"var2"))
. label list vertex_label
vertex_label:
           1 Peruzzi
           2 Castellan
           3 Strozzi
           4 Bischeri
           5 Barbadori
           6 Ridolfi
           7 Guadagni
           8 Medici
           9 Tornabuon
          10 Albizzi
          11 Salviati
          12 Acciaiuol
          13 Lambertes
          14 Ginori
          15 Pazzi
. // Drop vertex label variables and add new variables.
. drop var1 var2
. mata (void) st_addvar("double",("source","target"))
. // Add observations.
. mata st_addobs(rows(X))
. // Store edge list in Stata.
. mata st_store(.,("source","target"),X)
. // Assign value label to variables.
. label values source target vertex_label
. list
```

	source	target
1.	Peruzzi	Castellan
2.	Peruzzi	Strozzi
3.	Peruzzi	Bischeri
4.	Castellan	Strozzi
5.	Castellan	Barbadori
6.	Strozzi	Ridolfi
7.	Strozzi	Bischeri
8.	Bischeri	Guadagni
9.	Barbadori	Medici
10.	Ridolfi	Medici
11.	Ridolfi	Tornabuon
12.	Medici	Tornabuon
13.	Medici	Albizzi
14.	Medici	Salviati
15.	Medici	Acciaiuol
16.	Tornabuon	Guadagni
17.	Guadagni	Lambertes
18.	Guadagni	Albizzi

```
 19. |   Albizzi      Ginori |
 20. |  Salviati       Pazzi |
     +-----------------------+
```

```
. // Generate degree centrality.
. network source target, measure(adjacency) name(A,replace)
matrix A saved in Mata
. netsummarize A/(rows(A)-1), generate(degree) statistic(rowsum)
. // Generate closeness centrality.
. network source target, measure(distance) name(D,replace)
Breadth-first search algorithm (15 vertices)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
...............
Breadth-first search algorithm completed
matrix D saved in Mata
. netsummarize (rows(D)-1):/rowsum(D), generate(closeness) statistic(rowsum)
. // Generate betweenness centrality.
. network source target, measure(betweenness) name(b,replace)
Breadth-first search algorithm (15 vertices)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
...............
Breadth-first search algorithm completed
Betweenness centrality calculation (15 vertices)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
...............
Betweenness centrality calculation completed
matrix b saved in Mata
. netsummarize b/((rows(b)-1)*(rows(b)-2)), generate(betweenness) statistic(r
> owsum)
. // Generate eigenvector centrality.
. network source target, measure(eigenvector) name(e,replace)
matrix e saved in Mata
. netsummarize e, generate(eigenvector) statistic(rowsum)
. // Describe and list results.
. describe, full
Contains data
  obs:            20
 vars:            10
 size:         1,120 (99.9% of memory free)
```

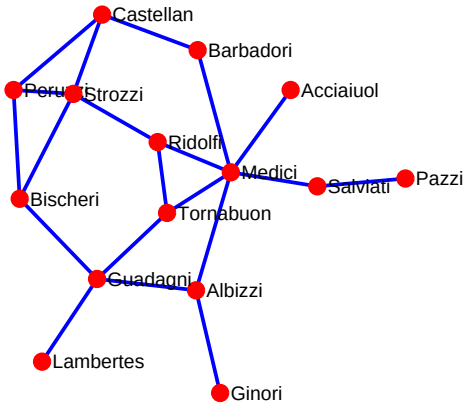
variable name	storage type	display format	value label	variable label
source	double	%10.0g	vertex_label	
target	double	%10.0g	vertex_label	
degree_source	float	%9.0g		rowsum of Mata matrix A/(rows(A)-1)
degree_target	float	%9.0g		rowsum of Mata matrix A/(rows(A)-1)
closeness_source	float	%9.0g		rowsum of Mata matrix (rows(D)-1):/rowsum(D)
closeness_target	float	%9.0g		rowsum of Mata matrix (rows(D)-1):/rowsum(D)
betweenness_source	float	%9.0g		rowsum of Mata matrix b/((rows(b)-1)*(rows(b)-2))
betweenness_target	float	%9.0g		rowsum of Mata matrix b/((rows(b)-1)*(rows(b)-2))
eigenvector_source	float	%9.0g		rowsum of Mata matrix e
eigenvector_target	float	%9.0g		rowsum of Mata matrix e

```
Sorted by:
     Note:  dataset has changed since last saved
.   list source target degree* closeness*
```

	source	target	degree~e	degree~t	closen~e	closen~t
1.	Peruzzi	Castellan	.2142857	.2142857	.368421	.3888889
2.	Peruzzi	Strozzi	.2142857	.2857143	.368421	.4375
3.	Peruzzi	Bischeri	.2142857	.2142857	.368421	.4
4.	Castellan	Strozzi	.2142857	.2857143	.3888889	.4375
5.	Castellan	Barbadori	.2142857	.1428571	.3888889	.4375
6.	Strozzi	Ridolfi	.2857143	.2142857	.4375	.5
7.	Strozzi	Bischeri	.2857143	.2142857	.4375	.4
8.	Bischeri	Guadagni	.2142857	.2857143	.4	.4666667
9.	Barbadori	Medici	.1428571	.4285714	.4375	.56
10.	Ridolfi	Medici	.2142857	.4285714	.5	.56
11.	Ridolfi	Tornabuon	.2142857	.2142857	.5	.4827586
12.	Medici	Tornabuon	.4285714	.2142857	.56	.4827586
13.	Medici	Albizzi	.4285714	.2142857	.56	.4827586
14.	Medici	Salviati	.4285714	.1428571	.56	.3888889
15.	Medici	Acciaiuol	.4285714	.0714286	.56	.368421
16.	Tornabuon	Guadagni	.2142857	.2857143	.4827586	.4666667
17.	Guadagni	Lambertes	.2857143	.0714286	.4666667	.3255814
18.	Guadagni	Albizzi	.2857143	.2142857	.4666667	.4827586
19.	Albizzi	Ginori	.2142857	.0714286	.4827586	.3333333
20.	Salviati	Pazzi	.1428571	.0714286	.3888889	.2857143

```
.   list source target betweenness* eigenvector*
```

	source	target	betwee~e	betwee~t	eigenv~e	eigenv~t
1.	Peruzzi	Castellan	.021978	.0549451	.2757304	.2590262
2.	Peruzzi	Strozzi	.021978	.1025641	.2757304	.3559805
3.	Peruzzi	Bischeri	.021978	.1043956	.2757304	.2828001
4.	Castellan	Strozzi	.0549451	.1025641	.2590262	.3559805
5.	Castellan	Barbadori	.0549451	.0934066	.2590262	.2117053
6.	Strozzi	Ridolfi	.1025641	.1135531	.3559805	.3415526
7.	Strozzi	Bischeri	.1025641	.1043956	.3559805	.2828001
8.	Bischeri	Guadagni	.1043956	.2545788	.2828001	.2891156
9.	Barbadori	Medici	.0934066	.521978	.2117053	.4303081
10.	Ridolfi	Medici	.1135531	.521978	.3415526	.4303081
11.	Ridolfi	Tornabuon	.1135531	.0915751	.3415526	.3258423
12.	Medici	Tornabuon	.521978	.0915751	.4303081	.3258423
13.	Medici	Albizzi	.521978	.2124542	.4303081	.2439561
14.	Medici	Salviati	.521978	.1428571	.4303081	.1459172
15.	Medici	Acciaiuol	.521978	0	.4303081	.1321543
16.	Tornabuon	Guadagni	.0915751	.2545788	.3258423	.2891156
17.	Guadagni	Lambertes	.2545788	0	.2891156	.0887919
18.	Guadagni	Albizzi	.2545788	.2124542	.2891156	.2439561
19.	Albizzi	Ginori	.2124542	0	.2439561	.0749227
20.	Salviati	Pazzi	.1428571	0	.1459172	.0448134



Network of Florentine marriages based on data from Padgett and Ansell (1993). Drawn using NETPLOT (Corten, 2011).

8.3 Time series analysis

The use of `if` *exp*, allows network measures to be easily generated for subsamples of data. Here we consider a country-level relational dataset and generate overall clustering coefficients for each of the reported years. Data is from publicly available International Monetary Fund Coordinated Portfolio Investment Survey (CPIS) Data, Table 8.1 - Geographic Breakdown of Portfolio Investment Assets: Equity Securities (2010). Surveys conducted from 2001 through 2009 are available. Data is provided in a matrix format with ij entries pertaining to the amount of country i's equity securities held by country j. Only information on positive investment is used and edge weights are taken to be the inverse of investment deflated using US consumer price index from the World Bank (2011). Edge weights are meant to reflect proximity between countries that have larger cross-border security holdings. The network is directed.

We focus on the overall measure of clustering coefficients, as non-respondents from the surveys are included in the network as vertices with only incoming edges, and their local clustering coefficient value of zero biases average clustering coefficients towards zero.

```
.   use imfcpis, clear
(IMF Coordinated Portfolio Investment Survey: Table 8.1 - Equity Securities)
.   describe
Contains data from imfcpis.dta
  obs:        26,160                          IMF Coordinated Portfolio Investm
> ent Survey: Table 8.1 - Equity Securities
 vars:             6                          5 May 2011 19:43
 size:     4,002,480 (99.3% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
source          str34  %34s                   Holder of equity securities, IMF
                                                CPIS
target          str95  %95s                   Issuer of equity securities, IMF
                                                CPIS
weight          float  %9.0g                  Edge weight - inverse of deflated
                                                investment in USD, author
year            float  %9.0g                  Year
investment      long   %10.0g                 Year-end holdings of equity
                                                securities in millions of
                                                nominal USD, IMF CPIS
uscpi           float  %9.0g                  US Consumer price index (2005 =
                                                100), World Bank WDI
-------------------------------------------------------------------------------
Sorted by:
.   summarize
```

```
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      source |          0
      target |          0
      weight |      26160    1.72e-07    3.07e-07   1.49e-12   1.10e-06
        year |      26160    2005.393    2.531253       2001       2009
  investment |      26160    3479.511    20176.56          1     714928
-------------+---------------------------------------------------------
       uscpi |      26160    101.4221    6.906585    90.6678   110.2466

.   // Compute clustering coefficients for each year.
.   // Suppress status dots.
.   forvalues i=2001/2009{
  2.     di ""
  3.     di "** Year: `i´ **"
  4.     network source target if year==`i´, measure(clustering) directed weigh
> t(weight) nodots
  5.     matrix list e(cc)
  6.   }

** Year: 2001 **

Clustering coefficients calculation (171 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .40161137     .35848011     .40993656      .3308401
Average     .22501606     .20278508     .22901613     .19232049

** Year: 2002 **

Clustering coefficients calculation (175 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .38291245     .33860285     .39138165     .31457621
Average     .20013265     .17977462     .20361013     .17147964

** Year: 2003 **

Clustering coefficients calculation (167 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .41680831     .36488986     .42528682     .34298863
Average     .24345751     .21968327      .2470524     .21090007

** Year: 2004 **

Clustering coefficients calculation (177 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .44158836     .38020664     .45221407     .35680374
Average     .23695718     .21101894     .24139999     .20048852

** Year: 2005 **

Clustering coefficients calculation (184 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .44308806     .37775965     .45425739     .35053421
Average     .22910225     .20174429     .23370902     .18799125

** Year: 2006 **

Clustering coefficients calculation (199 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .45589245     .39916957     .46501259     .38012479
Average     .21242901     .18523941     .21673185     .17464249

** Year: 2007 **

Clustering coefficients calculation (201 vertices)
Clustering coefficients calculation completed

e(cc)[2,4]
```

```
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .46365345     .40884779     .47235629     .38576594
Average     .21283682     .18822754     .21611827     .17932581
** Year: 2008 **
Clustering coefficients calculation (199 vertices)
Clustering coefficients calculation completed
e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall     .44361548      .3962016     .45183141     .37371945
Average     .21520551     .19244685     .21852989     .18400936
** Year: 2009 **
Clustering coefficients calculation (196 vertices)
Clustering coefficients calculation completed
e(cc)[2,4]
         Arithmetic~n  Geometric_~n       Maximum       Minimum
Overall      .4722288     .42316217     .48046887     .40051394
Average     .21779996     .19203185     .22196427     .18120866
```

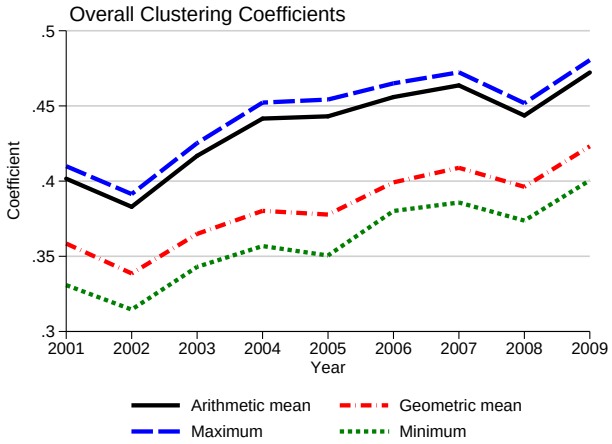


Figure based on annual networks of reporting economies from the IMF CPIS Data, Table 8.1 - Geographic Breakdown of Portfolio Investment Assets: Equity Securities.

8.4 Relational to panel dataset

In this section we demonstrate an example of a workflow that starts with a relational dataset and arrives at an unbalanced panel dataset that can be used to run regressions.

Relational data is based on the IMF CPIS dataset used in the previous example. Economic, geographic, and governance indicators for the years 2002 to 2009 are obtained from the World Bank's World Development Indicators and Worldwide Governance Indicators databases (World Bank, 2010, 2011). To facilitate the merging of the two datasets, country names in the World Bank dataset are changed to conform with that of the IMF CPIS dataset. A topic of interest may be to see what factors were important in attracting foreign investment during this time period.

For each of the years 2002 to 2009, using the IMF CPIS relational dataset we compute ineccentricity and eigenvector centrality. Ineccentricity is defined as the maximum value of incoming shortest paths and is equal to the column maximum of the distance matrix based on the weighted directed network. Since edge weight is equal to the inverse of deflated investment, smaller ineccentricity will suggest proximity to neighboring vertices. In computing eigenvector centrality, we employ a different type of edge weight instead and use real investment, as eigenvector centrality bestows larger values on more important vertices.

```
. // Load IMF CPIS dataset.
. use imfcpis, clear
```

```
(IMF Coordinated Portfolio Investment Survey: Table 8.1 - Equity Securities)
.   // Drop year 2001 - World Bank data is available from 2002 to 2009.
.   drop if year==2001
(2260 observations deleted)
.   // Compute ineccentricity.
.   forvalues i=2002/2009{
  2.     network source target if year==`i´, measure(distance) name(D`i´,replac
> e) weight(weight) directed nodots
  3.     netsummarize D`i´, generate(ineccentricity`i´) statistic(colmax)
  4.   }
Component analysis (174 vertices)
Component analysis completed
Dijkstra´s search algorithm (174 vertices)
Dijkstra´s search algorithm completed
matrix D2002 saved in Mata
Component analysis (166 vertices)
Component analysis completed
Dijkstra´s search algorithm (166 vertices)
Dijkstra´s search algorithm completed
matrix D2003 saved in Mata
Component analysis (176 vertices)
Component analysis completed
Dijkstra´s search algorithm (176 vertices)
Dijkstra´s search algorithm completed
matrix D2004 saved in Mata
Component analysis (183 vertices)
Component analysis completed
Dijkstra´s search algorithm (183 vertices)
Dijkstra´s search algorithm completed
matrix D2005 saved in Mata
Component analysis (198 vertices)
Component analysis completed
Dijkstra´s search algorithm (198 vertices)
Dijkstra´s search algorithm completed
matrix D2006 saved in Mata
Component analysis (200 vertices)
Component analysis completed
Dijkstra´s search algorithm (200 vertices)
Dijkstra´s search algorithm completed
matrix D2007 saved in Mata
Component analysis (198 vertices)
Component analysis completed
Dijkstra´s search algorithm (198 vertices)
Dijkstra´s search algorithm completed
matrix D2008 saved in Mata
Component analysis (195 vertices)
Component analysis completed
Dijkstra´s search algorithm (195 vertices)
Dijkstra´s search algorithm completed
matrix D2009 saved in Mata
.   // Transfer results to one variable per source and target.
.   generate ineccentricity_source=.
(23900 missing values generated)
.   label variable ineccentricity_source "ineccentricity"
.   generate ineccentricity_target=.
(23900 missing values generated)
.   label variable ineccentricity_target "ineccentricity"
.   forvalues i=2002/2009{
  2.     quietly replace ineccentricity_source=ineccentricity`i´_source if year
> ==`i´
  3.     quietly replace ineccentricity_target=ineccentricity`i´_target if year
> ==`i´
  4.   }
.   drop *eccentricity200*
```

```
.   // Compute eigenvector centrality.
.   // Generate and use real investment as edge weight.
.   generate realinvestment=investment/(uscpi/100)
.   forvalues i=2002/2009{
  2.     network source target if year==`i´, measure(eigenvector) name(e`i´,rep
> lace) weight(realinvestment) directed nodots
  3.     netsummarize e`i´, generate(eigenvector`i´) statistic(rowsum)
  4.   }
matrix e2002 saved in Mata
matrix e2003 saved in Mata
matrix e2004 saved in Mata
matrix e2005 saved in Mata
matrix e2006 saved in Mata
matrix e2007 saved in Mata
matrix e2008 saved in Mata
matrix e2009 saved in Mata
.   // Transfer results to one variable per source and target.
.   generate eigenvector_source=.
(23900 missing values generated)
.   label variable eigenvector_source "eigenvector"
.   generate eigenvector_target=.
(23900 missing values generated)
.   label variable eigenvector_target "eigenvector"
.   forvalues i=2002/2009{
  2.     quietly replace eigenvector_source=eigenvector`i´_source if year==`i´
  3.     quietly replace eigenvector_target=eigenvector`i´_target if year==`i´
>
  4.   }
.   drop *eigenvector200*
.   rename ineccentricity_target ineccentricity
.   rename eigenvector_target eigenvector
.   // Collapse by target country and year.
.   #d;
delimiter now ;
.     collapse (sum) weight (sum) investment (sum) realinvestment
>     (mean) uscpi (mean) ineccentricity (mean) eigenvector
>     , by(target year)
>   ;
.  #d cr
delimiter now cr
.   // Merge in World Bank data.  Keep matched observations.
.   rename target country
.   label variable country "Country name"
.   merge 1:1 country year using worldbank, keep(matched) nogenerate
    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                             1,287
    -----------------------------------------
.   // Declare data to be a panel.
.   encode country, generate(country2)
.   xtset country2 year
       panel variable:  country2 (unbalanced)
        time variable:  year, 2002 to 2009, but with gaps
                delta:  1 unit
.   label data "Country-level panel data"
.   describe
Contains data
  obs:         1,287                          Country-level panel data
 vars:            20
 size:       294,723 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
```

```
country         str95  %95s                  Country name
year            float  %9.0g                 Year
weight          double %9.0g                 (sum) weight
investment      double %10.0g                (sum) investment
realinvestment  double %9.0g                 (sum) realinvestment
uscpi           float  %9.0g                 (mean) uscpi
ineccentricity  float  %9.0g                 (mean) ineccentricity
eigenvector     float  %9.0g                 (mean) eigenvector
exrate          float  %9.0g                 DEC alternative conversion factor
                                               (LCU per US$)
gdpgrowth       float  %9.0g                 GDP growth (annual %)
inflation       float  %9.0g                 Inflation, consumer prices
                                               (annual %)
region          str26  %26s                  Region
incomegroup     str20  %20s                  Income Group
corruption      float  %9.0g                 Control of Corruption: Estimate
effectiveness   float  %9.0g                 Government Effectiveness:
                                               Estimate
stability       float  %9.0g                 Political Stability and Absence
                                               of Violence/Terrorism: Estimate
regulation      float  %9.0g                 Regulatory Quality: Estimate
ruleoflaw       float  %9.0g                 Rule of Law: Estimate
accountability  float  %9.0g                 Voice and Accountability:
                                               Estimate
country2        long   %38.0g      country2  Country name
```

```
Sorted by:  country2  year
     Note:  dataset has changed since last saved
.   summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
country	0				
year	1287	2005.625	2.278011	2002	2009
weight	1287	2.95e-06	2.52e-06	4.65e-09	.0000121
investment	1287	64439.03	218706.4	1	2383743
realinvest~t	1287	62613.31	210932.3	.9070571	2245200
uscpi	1287	101.9301	6.458229	92.10583	110.2466
ineccentri~y	1287	3.52e-07	2.52e-07	3.71e-08	1.47e-06
eigenvector	1287	.0249071	.0746418	0	.5546467
exrate	1222	506.7404	1860.914	.268788	17065.08
gdpgrowth	1214	4.225663	5.11624	-41.3	40.79159
inflation	1124	29.91726	728.86	-13.22581	24411.03
region	0				
incomegroup	0				
corruption	1260	.1205744	1.031661	-2.016047	2.466556
effectiven~s	1260	.1524394	1.006756	-2.249335	2.267191
stability	1270	.0321373	.9800003	-2.880965	1.596395
regulation	1260	.1565138	.9833775	-2.691515	1.992202
ruleoflaw	1272	.0948148	.9989472	-2.576503	1.964045
accountabi~y	1271	.0841978	.9925295	-2.290506	1.826686
country2	1287	97.04817	54.78466	1	191

9 Function and performance testing

9.1 Testing results using C++ and R

The following table provides information on what packages are used to check network measures for each type of network. Utilized packages include the Boost Graph Library (BGL) in C++ (Siek et al., 2001) and `sna`/`igraph` in R (Butts, 2010; Csardi and Nepusz, 2006). Function tests are conducted on all four types of networks (undirected/directed - unweighted/weighted) on a series of random networks with $|V|$ starting from approximately 100 and going up to 500 in increments of around 50 vertices. Maximum density, defined

as $|E|/|V|^2$ with undirected edges counted as two directed edges, is set to 0.1 in all networks.

In the last column, "L" denotes local, "A" denotes average, and "O" denotes overall clustering coefficients. "NA"s are inserted for cases in which the packages used here do not compute network measures for the particular network type or in cases where the network structure proves problematic computationally. For instance, packages differ in their ability to handle near-singular matrices. Path matrices are not tested as correct path counts are a prerequisite for accurate betweenness centrality measures.

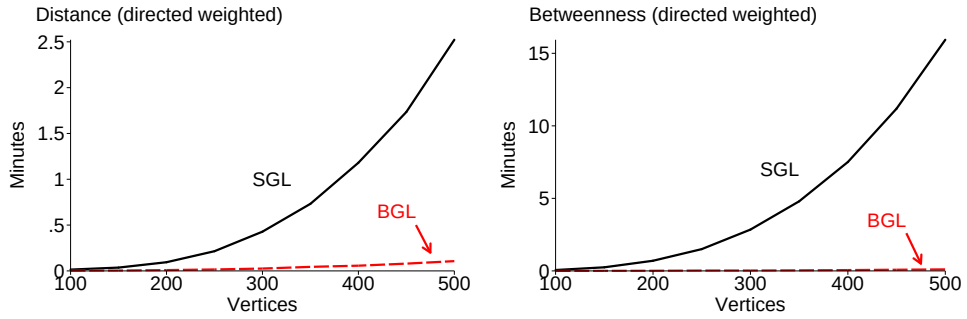
Network type	Matrices		Centralities			Clustering coefficients
	Adj.	Dist.	Betw.	Eigen.	KB	
unweighted undirected	BGL	BGL	BGL	`igraph`	`igraph`	`igraph` (L,A,O)
unweighted directed	BGL	BGL	BGL	NA	`igraph`	`sna` (O)
weighted undirected	BGL	BGL	BGL	`igraph`	`igraph`	NA
weighted directed	BGL	BGL	BGL	NA	`igraph`	NA

For tests involving BGL, function tests are deemed successful if network measures generated using SGL and BGL are equal. For tests involving R packages, the success condition is evaluated as the case when the maximum relative difference, `mreldif()`, between SGL and measures produced in R is less than 1e-7. See [M-5] **reldif( )**.

9.2 Performance tests

Time complexity is an issue, especially for weighted networks. In function tests carried out above with constant maximum density, current SGL algorithm implementation times for computing distance matrix and betweenness centrality are exponential in terms of the number of vertices. Illustrations below demonstrate algorithm completion times for both SGL and BGL routines on directed weighted networks. Similar outcomes hold for undirected weighted cases.

The difference between SGL and BGL completion times are not as stark for unweighted networks. Although SGL routines do take longer than BGL, the divergence is not to the point of what is reported from tests involving weighted networks. For a random network of around 500 vertices, computing distance matrix takes 10.451 seconds and betweenness centrality takes 27.384 seconds using SGL while corresponding numbers are 5.283 and 5.398 seconds using BGL.



Distance matrix and betweenness centrality computations based on randomly generated directed weighted networks with constant maximum density of 0.1. Tests are conducted for network sizes in increments of approximately 50 vertices from $|V| \sim 100$ to $|V| \sim 500$. SGL is run on 4-core Stata MP version 11.1 and employs Dijkstra's single-source shortest path algorithm for both computations. BGL uses Boost version 1.46.1 and implements Johnson's all-pairs shortest path algorithm (Johnson, 1977) to compute distance matrix and Brandes' algorithm (Brandes, 2001) for betweenness centrality. An adjacency list graph structure is used to represent the network in BGL. Both libraries are compiled and run on 64-bit Linux operating system.

10 Conclusion and future developments

The Stata Graph Library demonstrates the ability to employ relational data and generate network measures in Stata. As we have shown in the examples, relational data can be constructed using Stata's existing commands such as `joinby` or by using user-written commands such as Rense Corten's PAJEK2STATA. Network measures computed by `network` and returned to Stata by `netsummarize` can be merged to datasets for running regressions.

Though SGL, `network`, and `netsummarize` work in unison to deliver usable network information to the user, significant work remains to make the process more efficient and to provide additional enhancements. Areas for future development include implementing more efficient algorithms, designing algorithms for additional network measures, and optimizing SGL in Mata. It is hoped that further improvements and expansions will help facilitate the analysis of networks using Stata.

References

Anthonisse, J. M. 1971. The rush in a directed graph. Technical report, Stichting Mathematisch Centrum, Amsterdam. Technical Report BN 9/71.

Barrat, A., M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. 2004. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences* 101(11): 3747–3752.

Batagelj, V., and A. Mrvar. 2010. Pajek. http://pajek.imfm.si/doku.php?id=download.

Bonacich, P. 1972. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology 2* 113–120.

———. 1987. Power and Centrality: A Family of Measures. *The American Journal of Sociology* 92(5): 1170–1182.

Bonacich, P., and P. Lloyd. 2001. Eigenvector-like measures of centrality for asymmetric relations. *Social Networks* 23(3): 191–201.

Brandes, U. 2001. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology* 25(2): 163–177.

Butts, C. T. 2010. *sna: Tools for Social Network Analysis*. R package version 2.1-0.

Corten, R. 2010. PAJEK2STATA: Stata module to import network data in Pajek's.net format. Statistical Software Components, Boston College Department of Economics.

———. 2011. Visualization of social networks in Stata using multidimensional scaling. *Stata Journal* 11(1): 52–63.

Csardi, G., and T. Nepusz. 2006. The igraph software package for complex network research. *InterJournal* Complex Systems: 1695.

Freeman, L. C. 1977. A set of measures of centrality based on betweenness. *Sociometry* 40(1): 35–41.

Gleich, D. 2008. MatlabBGL. http://www.mathworks.com/matlabcentral/fileexchange/10922.

International Monetary Fund. 2010. Coordinated Portfolio Investment Survey (CPIS). http://www.imf.org/external/np/sta/pi/datarsl.htm.

Johnson, D. B. 1977. Efficient Algorithms for Shortest Paths in Sparse Networks. *Journal of the ACM* 24(1): 1–13.

Katz, L. 1953. A New Status Index Derived from Sociometric Analysis. *Psychometrika* 18: 39–43.

Newman, M. 2010. *Networks: An Introduction*. Oxford University Press.

Opsahl, T., and P. Panzarasa. 2009. Clustering in weighted networks. *Social Networks* 31(2): 155–163.

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.

Siek, J., L.-Q. Lee, and A. Lumsdaine. 2001. BGL. http://www.boost.org/doc/libs/.

World Bank. 2010. Worldwide Governance Indicators (WGI). http://data.worldbank.org/data-catalog/worldwide-governance-indicators.

———. 2011. World Development Indicators (WDI). http://data.worldbank.org/data-catalog/world-development-indicators.