

## Using SAS to Calculate Betweenness Centrality

---

**Alan R. Ellis**

*Research Associate and Fellow  
Cecil G. Sheps Center for Health Services Research  
University of North Carolina at Chapel Hill*

### **Abstract**

Betweenness centrality is a useful measure of an actor's importance in a social network. The SAS PROC IML module presented in this paper facilitates the calculation of betweenness centrality by social scientists by making it possible to run a faster algorithm for betweenness centrality using popular statistical software. The algorithm and module could be extended in order to calculate betweenness centrality for weighted graphs or to calculate other network measures that are based on geodesics, such as closeness centrality, graph centrality, or radiality.

**Acknowledgments:** This research was supported by grant 1-R03-MH073728-01A1 from the National Institute of Mental Health.

*Please address correspondence to Alan R. Ellis at 725 MLK Blvd. CB 7590, Chapel Hill, NC 27599-7590. Phone: (919) 966-2340, Fax: (919) 966-1634, Email: are@unc.edu.*

## Introduction

In analyzing data on social networks, researchers are often interested in *betweenness centrality*, which is one measure of an actor's importance in a network. Betweenness centrality reflects the extent to which an actor lies on geodesics between others in the network. Depending on the context, betweenness centrality might indicate the degree of power, control, or stress experienced by an actor in the course of network interactions (Freeman, 1977). The idea was introduced by Bavelas (1948, cited in Freeman, 1977), and the measure was defined by Freeman (1977):

$$C_B(p_k) = \sum_{i=1}^{j-1} \sum_{j=1}^n \frac{g_{ij}(p_k)}{g_{ij}}, i \neq j \neq k \quad (1)$$

where  $p_k$  is a point on the graph,  $n$  is the total number of vertices,  $i$  and  $j$  index vertices on the graph other than  $p_k$ ,  $g_{ij}$  is the number of geodesics between a pair of vertices, and  $g_{ij}(p_k)$  is the number of such geodesics that include  $p_k$ .  $C_B$ , then, is the proportion of geodesics between others in the network on which actor  $p_k$  lies. As defined by Freeman, betweenness centrality is normalized by dividing by its maximum possible value, which is the number of vertex pairs excluding  $p_k$ :

$$C'_B(p_k) = \frac{C_B(p_k)}{(n-1)(n-2)/2} \quad (2)$$

The normalized version of betweenness centrality ranges from 0 to 1 and allows comparisons between networks.

There are two significant barriers to the use of these measures of betweenness centrality by social scientists: (1) the need for specialized network analysis software and (2) the space and time typically required for processing. A related measure that requires less computation is egocentric network centrality (i.e., centrality of an actor within its first-order zone), which may predict betweenness centrality fairly well. Marsden (2002) found correlations ranging from .83 to .99 in an analysis of network data from a variety of studies with network sizes from 14 to

217. More recently, Everett and Borgatti (2005) found average correlations ranging from .85 to .99 in simulations with 25-500 actors and network density ranging from .1 to .6. However, a correlation of .83 implies that in the worst case tested, 31% ( $=1-.83^2$ ) of the variance in betweenness centrality was not explained by the egocentric measure.

Furthermore, Marsden identified scenarios in which the egocentric measure does a poor job of predicting betweenness centrality for a particular actor. (These involved the index actor having alters with extremely high or extremely low centrality in their own first-order zones.) Rather than perform analyses that are subject to this type of error, it is preferable to calculate the standard measure of betweenness centrality if data on the full network are available. This paper diminishes the barriers to calculating betweenness centrality by introducing a SAS Interactive Matrix Language (IML; SAS Institute, 2007) module that implements the faster algorithm for betweenness centrality recently developed by Brandes (2001).

## A SAS PROC IML module to calculate betweenness centrality

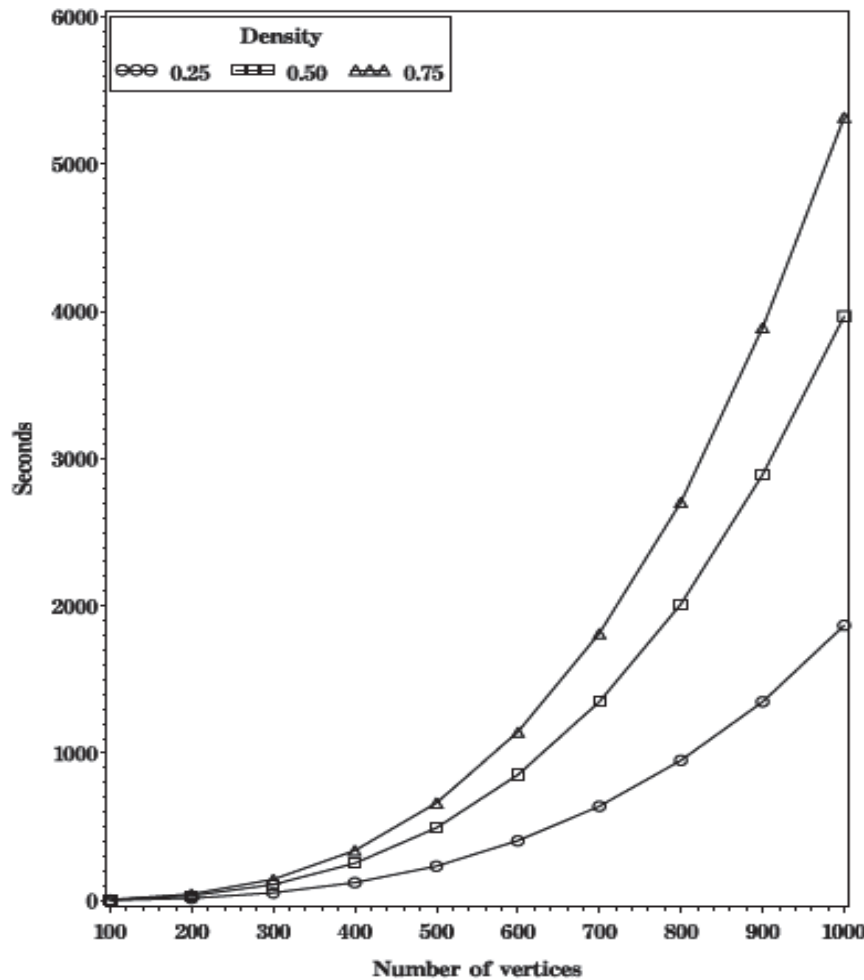
Brandes' (2001) strategy is to count and store a network's geodesics more quickly using network traversal algorithms instead of matrix multiplications. The author implemented Brandes' algorithm for unweighted graphs using a SAS PROC IML module and supporting macros (Appendix 1; also available from the author's website, <http://www.unc.edu/~arellis>). The supporting macros use row vectors to implement stacks and queues to store information obtained during network traversal. Stacks are "last-in, first-out" storage mechanisms into which a unit of information can be "pushed" and out of which a unit of information can be "popped". Queues are "first-in, first-out" storage mechanisms into which a unit of information can be "enqueued" and out of which a unit of information can be "dequeued". The PROC IML module expects as input a square matrix (which should be symmetric if the graph is undirected), a dummy (0-1) variable to indicate whether the graph is directed, and another dummy variable to indicate

whether betweenness centrality should be normalized. Based on this input, the module traverses the network and returns a column vector which contains, for each vertex in the graph, the requested form of betweenness centrality.

**Performance on simulated networks**

The module was used to calculate betweenness centrality for 30 simulated networks with size ranging from 100-1000

vertices (100, 200, ..., 1000) and with densities of .25, .50, and .75. The module was run on a POWER5+ processor server running AIX UNIX. The maximum amount of memory used was 47 MB. Figure 1 shows the amount of processing time required as a function of network size and density. For the network with 1000 vertices and a density of .75, the processing time was 5,318 seconds (approximately 89 minutes).



**Figure 1. Processing time as a function of network size and density**

Given current computer technology, the memory required for running the module is negligible even for a network with 1000 vertices. Therefore, processing time is more important than memory as a potential barrier to data analysis. According to Brandes (2001), processing time should be on the order of  $n*m$ ,

where  $n$  is network size and  $m$  is the number of links in the network, equal to  $p*n(n-1)/2$  where  $p$  is the network density (Scott, 2000). For the simulated networks, processing time was indeed on the order of  $n*m$ . This was verified with a linear regression model that predicted processing time as a function of  $n^3*p$  ( $R^2=.996$ ; other

*results not shown*). This means that (1) for networks of a given density, the processing time required is roughly proportional to  $n^3$ , and (2) for networks of a given size, the processing time required is roughly proportional to  $p$ .

### Conclusion

Betweenness centrality is a useful measure of an actor's importance in a social network. The SAS PROC IML module presented in this paper facilitates the calculation of betweenness centrality by social scientists by making it

possible to run Brandes' (2001) faster algorithm for betweenness centrality using popular statistical software. As noted by Brandes (2001), his algorithm, and therefore the module presented here, could be extended in order to calculate betweenness centrality for weighted graphs or to calculate other network measures that are based on geodesics such as closeness centrality (Sabidussi, 1966), graph centrality (Hage & Harary, 1995), or radiality (Valente & Foreman, 1998).

### References

- Bavelas, A. (1948). A mathematical model for group structure. *Applied Anthropology*, 7, 16-30.
- Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2): 163-177.
- Everett, M. & Borgatti, S. P. (2005). Ego network betweenness. *Social Networks*, 27, 31-38.
- Freeman, L.C. (1977). A set of centrality measures based on betweenness. *Sociometry*, 40, 35-41.
- Hage, P. & Harary, F. (1995). Eccentricity and centrality in networks. *Social Networks*, 17, 57-63.
- Marsden, P. V. (2002). Egocentric and sociocentric measures of network centrality. *Social Networks*, 24, 407-422.
- SAS Institute. (2007). SAS OnlineDoc 9.1.3. Retrieved January 30, 2008, from <http://support.sas.com/onlinedoc/913/docMainpage.jsp>
- Sabidussi, G. (1966). The centrality index of a graph. *Psychometrika*, 31, 581-603.
- Scott, J., (2000). *Social network analysis*. (2<sup>nd</sup> ed.). Thousand Oaks, CA: SAGE.
- Valente, T. W. & Foreman, R. K. (1998). Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network. *Social Networks*, 20(1), 89-105.

## Appendix 1. SAS PROC IML module and supporting macros

```

/*      bcent.sas

2007/09/12

[Alan R. Ellis, MSW, http://www.unc.edu/~arellis; emial: are@unc.edu]

IML modules and supporting SAS macros to calculate betweenness centrality
Uses algorithm by Ulrik Brandes
Brandes, Ulrik. (2001). A faster algorithm for betweenness centrality.
Journal of Mathematical Sociology, 25(2), 163-177.

returns a vector of betweenness centrality values

Usage:  %include bcent.sas;
        bc=bcent(m,directed,normalize)
        m = matrix for which betweenness centrality is to be computed
        directed = 0 if relationships are undirected, 1 if directed
        normalize = 0 if raw centrality is desired, 1 if normalized value is desired

This module creates the following macros: push, pop, enq, deq, isempty

Input matrix should be square and, if relationships are undirected, should be symmetric.

2006/04/11  original version
2006/05/04  changed code so input matrix would not be modified
2007/08/31  revised comments
2007/09/12  changed name of input matrix and disabled error checking code that modified input matrix
*/

/* push values onto a stack - i.e., insert into column 1 of a row vector */
%macro push(s,val);
  if ncol(&s)=0 then                                /* if empty then start with value */
    &s=&val;
  else
    &s=insert(&s,&val,0,1);                          /* otherwise insert value */
%mend;

/* pop value off a stack - i.e., remove from column 1 of a row vector */
%macro pop(s,val);
  if ncol(&s)=0 then do;                            /* if empty then return undefined value */
    free &val;
    end;
  else do;
    &val=&s[1];
    &s=remove(&s,1);
  end;
%mend;

/* enqueue a value - i.e., insert it at the end of a row vector */
/* adapted from "push" macro - simply changed location of insertion */
%macro enq(s,val);
  if ncol(&s)=0 then                                /* if empty then start with value */
    &s=&val;
  else
    &s=insert(&s,&val,0,1+ncol(&s)); /* otherwise insert value at end of queue */
%mend;

/* dequeue a value - i.e., remove it from column 1 of a row vector */
%macro deq(q,val);
  %pop(&q,&val);
%mend;

/* function to determine whether a stack or queue (i.e., row vector) is empty */
%macro isempty(sq);
  %str((ncol(&sq)=0))
%mend;

/* function to calculate Betweenness Centrality using Brandes' algorithm */
start bcent(m,directed,normalize);
  nvert=nrow(m);                                  /* # vertices in network */

  /* check input */

```

```

err=0;
if ((directed^=0) & (directed^=1) & (normalize^=0) & (normalize^=1)) then
  err=1;

/* The following lines could be enabled in order to check input further. Note that the input
   matrix is modified. */

/*
else if (nvert^=ncol(m)) then
  err=1;
else do;
  tm=m`;
  if ((directed=0) & any(m^=tm)) then err=1;
end;
*/

if err=1 then do;
  file log;
  put 'ERROR: invalid parameter values for bcent(). Returning -1.';
  put 'USAGE: bcent(m,directed,normalize);';
  put '      <m>          : square matrix (symmetric if graph is undirected)';
  put '      <directed> : 1 if graph is directed, 0 otherwise';
  put '      <normalize>: 1 if result should be normalized, 0 otherwise';
  return(-1);
end;
cb=j(nvert,1,0);          /* betweenness centrality of each vertex starts at zero */

do s=1 to nvert;
  free stack;          /* start with empty stack - just being explicit */
  p=j(nvert,nvert,0); /* create empty list of predecessors for each vertex */

  sigma=j(nvert,1,0); /* count # geodesics each vertex is on: initially zero, */
  sigma[s]=1;         /* except one for current vertex */

  d=j(nvert,1,-1);   /* vector of -1 values, except zero for current vertex */
  d[s]=0;            /* d appears to measure depth */

  free queue;        /* start with empty queue - just being explicit */
  %enq(queue,s);     /* add current vertex to queue */

  do while (%isempty(queue)^=1); /* while queue is not empty */
    %deq(queue,v);     /* de-queue a vertex number into v */
    %push(stack,v);   /* and also push it onto the stack */

    /* loop through each neighbor w-sub-j of v */
    w=loc(m[v,]);     /* loop through vertices w where m(v,w) is nonzero */
    /* i.e., there is a path from v to w */
    do j=1 to ncol(w);
      /* w-sub-j found for the first time? */
      if d[w[j]]<0 then do;
        %enq(queue,w[j]);
        d[w[j]]=d[v]+1;
      end;

      /* shortest path to w via v? */
      if d[w[j]]=d[v]+1 then do;
        sigma[w[j]]=sigma[w[j]]+sigma[v];
        p[w[j],v]=1; /* add v to w's list of vertices */
      end;
    end;
  end;

  delta=j(nvert,1,0); /* initialize delta to zero for each vertex */

  /* stack returns vertices in order of non-increasing distance from vertex s */

  do while (%isempty(stack)^=1); /* while stack is not empty */
    %pop(stack,ww); /* use double-w; this is distinct from */
    /* the w used for neighbors-to-v above */

    vv=loc(p[ww,]); /* indices of vertices on ww's list of predecessors */
    /* use vv - this is a new v, too */
  end;
end;

```

## CONNECTIONS

Using SAS to Calculate Betweenness Centrality

```
do k=1 to ncol(vv);
  delta[vv[k]]=delta[vv[k]]+(sigma[vv[k]]/sigma[ww])*(1+delta[ww]);
end;
if ww ^= s then cb[ww]=cb[ww]+delta[ww];
end;
if (directed=0) then cb=cb/2; /* if undirected then divide by 2 */
if (normalize=1) then do;
  if (directed=0) then cb=2*cb/(nvert-1)/(nvert-2); /* normalize by maximum possible centrality */
  else cb=cb/(nvert-1)/(nvert-2);
end;
return(cb);
finish;
```