Informe Técnico CESGA-2005-003

CalcuNetw: Calculate Measurements in Complex Networks

J.Carlos Mouriño[1], Ernesto Estrada[2], Andrés Gómez[1]
[1]CESGA. Centro de Supercomputación de Galicia
e-mail: *jmourino@cesga.es*
[2]Complex Systems Research Group, X-Rays Unit, RIAIDT
University of Santiago de Compostela
e-mail: *estrada66@yahoo.com*

Abstract: The study of complex networks has become an important area of multidisciplinary research involving physics, mathematics, biology, social sciences, informatics and other theoretical and applied sciences. The importance of this field resides in the existence of a unifying language to describe disparate real–world systems that are of great relevance in modern society. In order to study some characteristic of such networks, a program for calculating important measurements has been developed.

Key Words: Complex networks, Social Networks.

Este trabajo ha sido cofinanciado por el FSE (Fondo Social Europeo)

Identificador documento:	OT_APL_ManualEstrada_V1
Fecha:	21/11/05
Autor:	J. Carlos Mouriño
Responsable:	Andrés Gómez
Versión:	V1.1
Status	Revised

Historia de Revisiones

Versión	Autor	Descripción
11/11/05	Carlos Mouriño	Desarrollo Informe Técnico
21/11/05	Carlos Mouriño	Revisado

Table of Contents

1 Introduction

Complex networks, consisting of sets of nodes or vertices joined together in pairs by links or edges, appear frequently in various technological, social and biological scenarios. These networks include the Internet, the World Wide Web, social networks, scientific collaboration networks, lexicon or semantic networks, neural networks, food webs, metabolic networks and protein–protein interaction networks. They have been shown to share global statistical features, such as the "small world" and the "scale–free" effects, as well as the "clustering" property. The first feature is simply the fact that the average distance between nodes in the network is short and usually scales logarithmically with the total number of nodes. The second is a characteristic of several "real–world" networks in which there are many nodes with low degree and only a small number with high degree (the so–called "hubs"). The node degree is simply the number of ties a node has with other nodes. In scale–free networks, the node degree follows a power law distribution. Finally, clustering is a property of two linked nodes tat are each linked to a third node. In consequence, these three nodes form a triangle and the clustering is frequently measured by counting the number of triangles in the network.

It has been observed that not only triangles but also other subgraphs are significant in real networks. We say that a graph $G'=(V',E')$ is a subgraph of $G=(V,E)$ if $V'\subseteq V$ and $E\subseteq E$, where V are the nodes and E the edges interconnecting them of the network or graph G. The term "network motifs" designates those patterns that occur in the network far more often than in random networks with the same degree sequence. Network motifs found in technological and biological networks are small subgraphs that capture specific patters of interconnection characterizing the networks at the local level.

In order to calculate some measurements in complex networks a simple program has been developed. This application calculates some characterization measurements in a given network and compare it with a number of random networks given by the user. The measurements calculated by the program are: the Subgraph Centrality (*SC*), SC odd , SC even, Bipartivity, Network Communicability ($C(G)$) and Network communicability for Connected Nodes ().

This measurements are described as follows:

- SC: Is a new centrality measure that characterizes the participation of each node in all subgraphs in a netwrok. Smaller subgraphs are given more weight than larger ones. SC odd and SC even represent the centrality taking into account only paths of long odd or even respectively.
- Bipartivity: Is a proportion of even to total number of closed walks in the network.
- C(p,q): Measures how well communicated are two nodes in the network. The highest the number of paths connecting nodes p and q the highest value of $C(p,q)$. $C(G)$ is the mean of all the $C(p,q)$, and is the mean of all the $C(p,q)$ that are connected.

2 General information

- **Description**: Redes is a software tool for calculate complex networks measurements.
- **Copyright**: Copyright (c) CESGA. All rights reserved.
- **Authors**: J. Carlos Mouriño, Ernesto Estrada and Andrés Gómez.
- **Platforms**: Actually only is available for HP-UX platforms.
- **Current version**: v1.1.
- **Major updates of lasts versions**:

  v0.5
  - The node label is included in the Centralities output file.
  - The program checks if the input network is connected.
  - If the input network is not connected, the program extract the main connected component.

  v0.6
  - The node degree for each node is included in the output files.
  - If the input network is not connected, the program extract the main connected component and computes it.
  - Optimizations in the random networks generator that reduces the computational time have been deployed.
  - The text output has been translated to English.

  v1.0
  - A 64 bits version for major networks was also developed.
  - Optimizations in the generation of the random networks have been made.
  - Some little bugs have been corrected.

  v1.1
  - Added the description of the measurements.

3 Files provided

The program is given as a simple executable that has been developed in C, using the Lapack and Blas libraries, under an HP-UX system.

Additionally, a simple script called `count` is also provided. This script accept as parameter the network input file and count the number of nodes of the network.

section*Program execution

The syntax of the program is as follows:

```
> ./redes input_file node_number file_type
random_networks [c]
```

where:

- `input_file`: is the file containing the network description.
- `node_number`: is the number of nodes of the network. If the user doesn't know this node number can put here any number and the program will calculate the number of nodes in the input file.
- `file_type`: type of network description in the input file. This type could be:
  - *2*: edge list, one per line. This is the default option.
  - *3*: edge list with weight (not taken into account), one per line.
  - *M*: adjacency matrix, one row per line.
  - *L*: connected nodes lists, one per line.

  `random_networks`: number of random networks to be calculated. Their average will be compared with the original network. c: Write useful values to files (this parameter is optional).

If the user doesn't remember the argument list or introduces an incorrect parameter, the program will show a brief text help describing the syntax and the arguments description.

For interactive execution of the program, wall clock time and available memory issues should be taking into account. The memory consumed by the program can be approximately calculated in Mb as:

where N is the number of nodes and *size*(*double*) is the size of the double datatype in the execution machine.

4 Program output

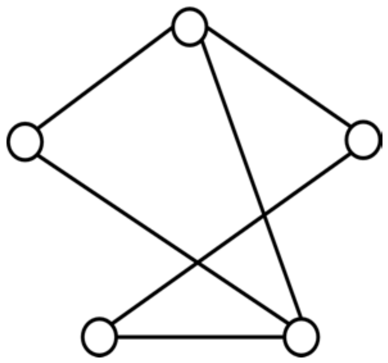
Once the program is executed, the calculated measurements are shown in the standard output and the number of random network that is being calculated is also shown. When the program finishes the average values (with the mean squared error) for all the random networks are shown.

Besides these values, the program also save to files other important information. These files are named equal that the `input`_file plus a suffix. The meaning of this suffix is as follows:

- **input_file.A** : contains the adjacency matrix of the network.
- **input_file.SC** : contains the individual values for each node of the three centrality measurements, with the name of the node, its degree and the values of the main eigenvector.
- **input_file.C** : contains the connectivity for each pair of nodes, preceded by the node names and the node degree.
- **input_file.Cconn** : contains the connectivity only for connected nodes, preceded also by the node names and the node degree.
- **input_file.R** : contains the last random network calculated as edges list of type 2.

5 Sample output

If we define the file network_sample with that simple graph



```
1        2
1        3
2        4
3        4
5        1
5        3
```

and execute with the command line:

```
redes5 network\_sample 5 2 100
```

The output of the program will be the following:

```
  Adjacency matrix stored in file: network_sample.A

     subgraph centrality          SC              = 3.07854
     subgraph centrality odd        Scodd         = 
0.319562
     subgraph centrality even      Sceven    = 2.75898
     bipartivity                           B(G)      = 
0.896197
     entrophy                              S
     = 2.55626
     communicability                    C(G)     = 2.13861
     communicability for connected nodes  Cconn(G) = 
2.53351

  Random Matrix:
     100*

    subgraph centrality               SC        = 3.02708 
O(0.0138309)
    subgraph centrality odd           SCodd     = 0.25245 
O(0.0130816)
    subgraph centrality even       Sceven   = 2.77463 
O(0.00999759)
    bipartivity                       B(G)      = 
0.916602
    entrophy                          S         = 2.55958 
O(0.00122038)
    communicability                   C(G)      = 2.10901 
O(0.0140987)
    communicability connected nodes Cconn(G) = 2.47547 
O(0.0141517)

  Random Network stored in file: network_sample.R


   -- Successful execution --

  Copytight (c) CESGA. All rights reserved.

  Thanks to:   J. Carlos Mouriño, Ernesto Estrada and 
Andrés Gómez.
```

6 Random networks

A given number of random networks is calculated for comparation issues, and the average value of the target measurements is calculated indicating also the mean squared error. As the number of random networks grows up, the computational time increases, but the mean results will improve.

The random networks are calculated taking into account some restrictions. The networks must have the same number of nodes and edges than the original network. The random networks must also be connected and the node degree must be the same that in the original network.

A simple algorithm for the generation of random nodes is presented:

```
    do
      {
        reset(a);
        initialize(degree)
        initialize(edges)
        conex[rand(k)]=1;

        do     // number of edges
          {
            i=rand(k); (or  i=node with maximun degree)
            r=rand(k);

            if    ((a[i][r]==0)    &    (degree[r]>0)    &
(degree[i]>0)
                & ((conex[i]==1) | (conex[r]==1)))
              {
                a[i][r]=1;   //connect it
                a[r][i]=1;
                degree[i]-=1;
                degree[r]-=1;
                conex[i]=1;
                conex[r]=1;
                edges--;
              }
          }
        while (edges>0 or we can't put more)
      }
    while the network is not valid (edges > 0);
```

where *edges* is the number of edges of the network, *k* the number of nodes, `degree[]` is a vector with the degree of each node and `conex[]` is a flag that is set in connected nodes. `m[][]` is the original network and `a[][]` the random network that is being calculated.

The algorithm is briefly as follows. First, after the initialization of the variables, a random node is chosen and marked as connected. After that we choose randomly a pair of nodes and we connect then if it is possible. A node can be connected to another if the remain degree of both nodes is positive, and the nodes are not already connected and one or both of them are marked as connected (for guarantee a connected network). When two nodes are chosen for be connected, the degree of both is decreased and the corresponding element of the adjacency matrix is set to one (and also the opposite, for symmetry purposes). We decrement also the number of edges left.

The process is repeated while there are edges left or we can't put any more edges. In this last case we reject the network and start again.

This algorithms generates random networks that fit the requirements of the given restrictions. There are some additional conditions in the algorithm, that are not shown, for avoid dead–locks, infinity loops, etc. For example, two nodes are not connected if the resultant degree of all connected nodes is equal to zero and there are other nodes with connection possibilities.

As there are additional conditions and some random networks are rejected, the execution time of this algorithm is not constant, and could vary.

7 Known bugs

- The maximum network size is limited by the RAM memory of the execution system. Although this limit has been widely increased with the 64 bits version. The program exits if there are not enough available memory.
- If the input network has the node's names as text labels, and the number of nodes introduced as parameter by the user is incorrect, the number of nodes given by the program could be incorrect. Use the script `count` in this case, if the input network is type 2.
- This also may occur if the labels are numerical and non continuous, and the number of nodes introduced are lower than the real number of nodes of the network. Use also the script count if the input file is type 2.
  •When the user introduces a wrong number of nodes, the calculation of the right number of nodes fails also if the input file network is type L.