Complex Network - Assignment 2

NGUYEN T. Hoang - SID: 15M54097

Fall 2015, W833 Mon. Period 5-6

Due date: 2016/02/01

Problem

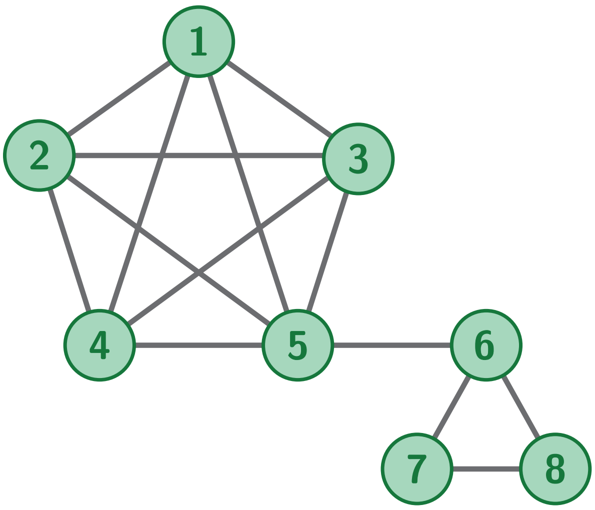
Given a network as in Figure 1. Complete these following tasks:

1. Compute eigenvector centrality and betweenness centrality of each vertex in the given network.
2. Find eigenvectors of matrices, construct the Laplacian and the modularity matrix for this small network:
   (a) Find the eigenvector of the Laplacian corresponding to the second smallest eigenvalue and hence perform a spectral bisection of the network into two equally sized parts.
   (b) Find the eigenvector of the modularity matrix corresponding to the largest eigenvalue and hence divide the network into two communitites.
3. Explain quantitatively why "*your friends have more friends than you do*" in the configuration model.
4. Write examples of parameters β and γ of SIR model when:
   (a) There is an epidemic.
   (b) There is no epidemic.

Answer

In this assignment, I will use SNAP.PY, a network analysis developed by Stanford University, as a computational tool. All illustrations (figures, graphs ...) are drawn using D3 Javascript Framework.

Question 1: Compute eigenvector centrality and betweenness centrality of each vertex.



$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 1: Simple network and its corresponding adjacency matrix.

Eigenvector centrality is given by the formula:

$$x_i^{(t)} = \Sigma_{j \neq i} A_{ij} \times x_i^{(t)}$$

Rewrite in final matrix form:

$$A\boldsymbol{x} = k_1 \boldsymbol{x}$$

where $\boldsymbol{x}$ is a vector storing score of all nodes, and k_1 is the largest eigenvalue of the adjacency matrix A.

Listing 1: Eigenvector centrality computation with SNAP.PY

```
# Extracted from UnweightedUndirectedGraph class - File: cn_a1_p1.py
...
import snap as sn
self._graph = sn.LoadEdgeList(sn.PUNGraph, edge_list, 0, 1, '␣')
...
# Compute eigenvector centrality and store to a hash table.
def EigenvectorCentrality(self):
    # Create a hash map: Int -> Float
    NIdEigenH = sn.TIntFltH()
    sn.GetEigenVector(self._graph, NIdEigenH)
    return NIdEigenH
    \label{lst:eig}
```

The vector result of Listing **??** is shown as follow:

$$\begin{pmatrix} 0.437 & 0.437 & 0.437 & 0.437 & 0.464 & 0.136 & 0.044 & 0.044 \end{pmatrix}$$

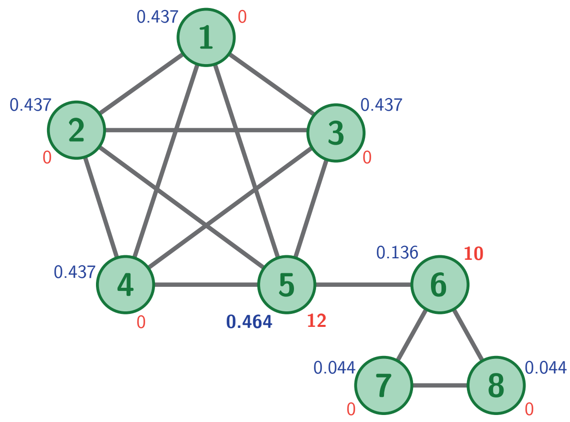


Figure 2: The given network with eigenvector centrality (blue) and betweenness centrality (red)

Figure 2 shows the result in the graph. Eigenvector centrality of each vertex is shown by a blue decimal number next to it. As we can see, vertex number 5 has the highest eigenvector centrality means that vertex number 5 is the most *central* vertex. By looking at the graph, we can intuitively understand this fact.

Betweenness centrality is another metric to measure how important a vertex is within the network. Different from other metric, betweenness centrality measure how important a vertex is in the information flow between other vertices. In [**?**], the author defines betweenness centrality x_i of vertex i as follow:

$$x_i = \sum_{st} \frac{n^i_{st}}{g_{st}}$$

where n^i_{st} is the number of geodesic paths between vertex s and vertex t that go through i, and g_{st} is the total number of geodesic paths between s and t. Besides the normal betweenness centrality, in [**?**] the author also mentioned 2 other types of betweenness: *flow betweenness* and *random walk betweenness*. However, in this assignment, I will only compute the standard betweenness for the given network.

Listing 2: Eigenvector centrality computation with SNAP.PY

```
# Extracted from UnweightedUndirectedGraph class - File: cn_a1_p1.py
...
import snap as sn
self._graph = sn.LoadEdgeList(sn.PUNGraph, edge_list, 0, 1, '␣')
...
# Compute eigenvector centrality and store to a hash table.
def EigenvectorCentrality(self):
    # Create a hash map: Int -> Float
    NIdEigenH = sn.TIntFltH()
    sn.GetEigenVector(self._graph, NIdEigenH)
    return NIdEigenH
    \label{lst:eig}
```