STUDIES ON INFLUENTIAL NODE OF COMPLEX NETWORKS

Mini Project - I

Submitted in Partial Fulfilment for the degree of
BACHELOR OF TECHNOLOGY
(Computer Science and Engineering)

By
ARIJIT SAHA (510517004)
ROHIT KUMAR (510517007)
SUBHAJYOTI SAHA (510517024)
AKASH SINGH (510517012)

Under the Guidance of
PROF. SUSANTA CHAKRABORTY

INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR
HOWRAH – 711103, WEST BENGAL
DECEMBER, 2018

Signature of Students:

1.

2. Signature of Supervisor

3.

4.

Signature of Head of Department

OUTLINE

1 INTRODUCTION

Complex Networks have become an integral part of our life, without our knowledge.

Facebook ego networks, Google Plus Circles, WhatsApp group chats are just a few examples of networks that are created on a global scale.

Gene Regulatory Networks are formed in biological processes that regulate genomes in human body and can be represented as complex networks.

Influential Nodes are taken as the nodes with minimum clustering coefficient, maximum degree centrality, betweenness centrality, closeness centrality and edge contribution factor.

1.1 MOTIVATION

We can predict the pattern of spreading[1] of data given the source by finding the influential nodes in a complex network. We can identify the key nodes that if changed can affect the networks data transfer. We can identify the nodes that exhibit the most potential for spreading viruses, if the given graph pertains to a computer network.

Complex network analysis is the process of investigating Complex structures through the use of networks and graph theory. It characterises networked structures in terms of nodes and the ties, edges, or links (relationships or interactions) that connect them. This Analysis is done by creating a mathematical model of the Complex Network Graph[2].

Complex Network Analysis can be used in real life for the following:

1. Finding Political Preferences of the general Public
2. Social Score of a person on a network such as Facebook, Orkut, Twitter
3. Finding Key Targets in terrorist Organisations
4. Finding Clusters among Social Networks and draw inferences regarding location and psychology[3]
5. Gene Regulatory Network, to find the key genomes that cause the propagation of genetic diseases and their cure accordingly.

1.2 OBJECTIVE

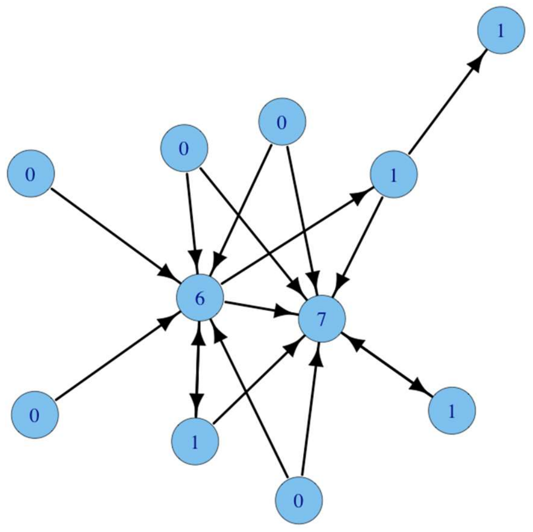
Calculation of Centrality Measures such as Clustering Coefficient, Betweenness Centrality, Closeness Centrality, Degree Centrality, Edge Contribution Factor. Identification of most influential node (central) from the given graph according to the above factors.

2 PRELIMINARIES

2.1 DEGREE CENTRALITY

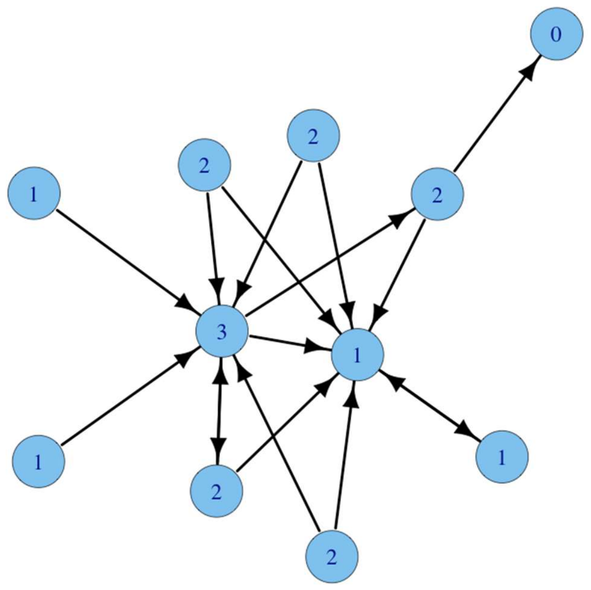
Degree Centrality[4] is defined as the total weight of the links incident upon a node divided by total weight of graph. For a directed graph, we define two degree centralities.

$Indegree\ Centrality(v) = \frac{\sum e_{iv}}{W(G)}$, where, $\sum e_{iv}$ is the sum of all weights of edges incoming on the vertex v and $W(G)$ is the total weight of the graph.



$Indegree\ Counts\ of\ Nodes$

$Outdegree\ Centrality(v) = \frac{\sum e_{ov}}{W(G)}$, where, $\sum e_{ov}$ is the sum of all weights of edges originating from the vertex v and $W(G)$ is the total weight of the graph.



Outdegree Counts of Nodes

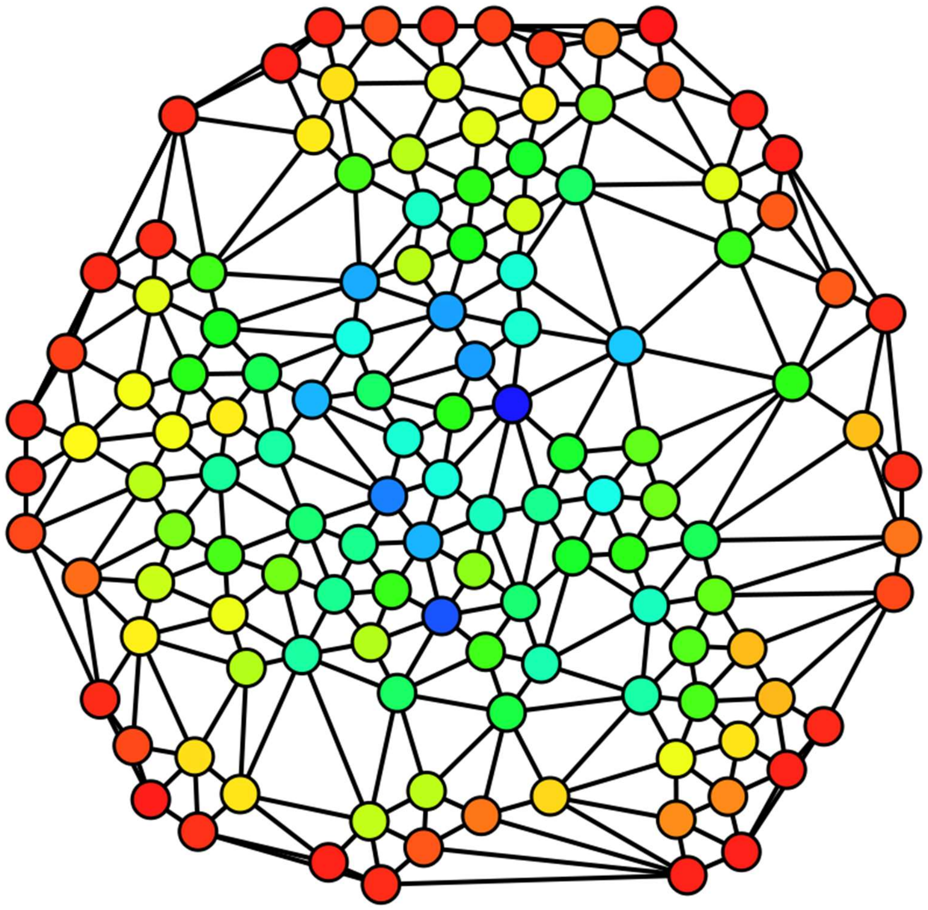
Dividing the degree counts by the total weight of the graph gives the respective Indegree Centralities and Outdegree Centralities.

2.2 BETWEENNESS CENTRALITY

Betweenness centrality[5] quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.

$Betweenness\ centrality(v) = \sum_{s \neq v \neq d} \frac{\sigma_{sd}(v)}{\sigma_{sd}}$, where σ_{sd} is the no of paths from s to t and $\sigma_{sd}(v)$ is the no of these paths v appears in.

Betweenness Centrality of nodes represented by colour, red representing low and blue representing high.



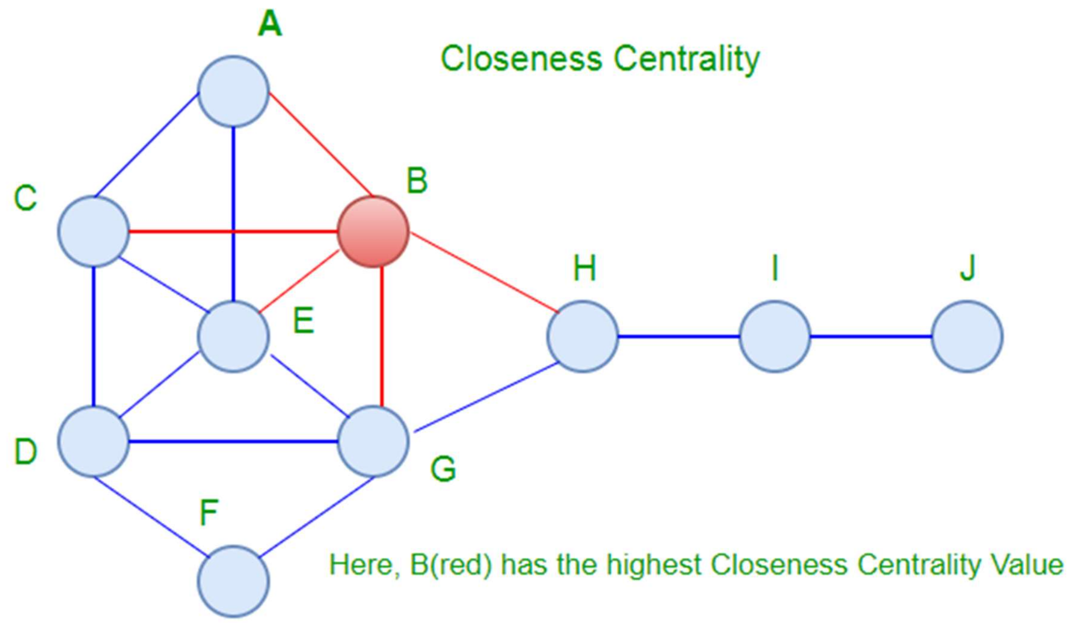
2.3 CLOSENESS CENTRALITY

The closeness centrality[6] of a node is taken as the inverse of farness, which is defined as the sum of all path lengths from that node to all other nodes of the graph.

Closeness is defined as the reciprocal of the farness, that is:

$$Closeness\ Centrality(v) = \frac{1}{\sum_u d(v,u)}$$

Where $d(u,v)$ is the distance between the node u and v.



2.4 EDGE CONTRIBUTION FACTOR

Edge Contribution Factor[7] of a node is defined as the sum of weights of all neighbour edges and neighbour to neighbour edges of that node divided by total weight of the graph.

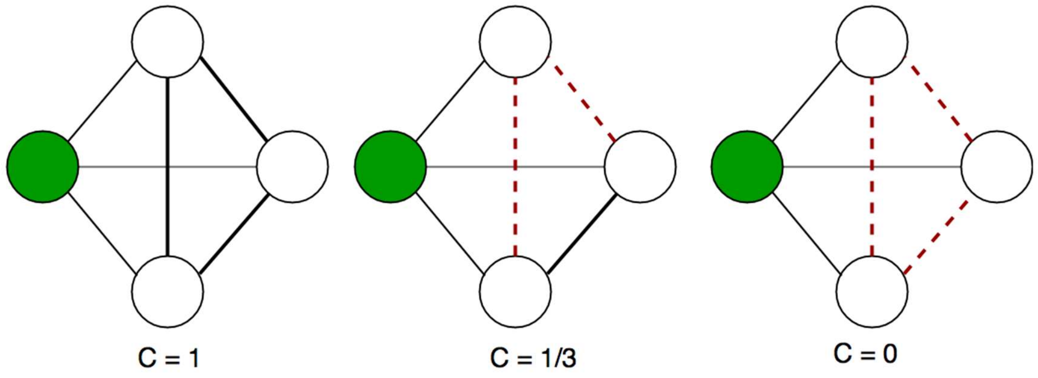
$$Edge\ Contribution\ Factor(v) = \frac{Sum\ of\ Neighbour\ edge\ weights\ and\ 2nd\ neighbour\ edge\ weights}{Total\ Weight\ of\ graph}$$

2.5 CLUSTERING COEFFICIENT

In a complex network, clustering coefficient[8] of a node is a measure of the degree to which its neighbour nodes are interconnected.

$Clustering\ Coefficient(v)$ is defined as

$$\frac{No\ of\ Edges\ between\ neighbours\ of\ v}{Total\ no\ of\ edges\ possible\ between\ neighbours\ of\ v}$$



3 OUR APPROACH

We have calculated Closeness Centrality and Betweenness Centrality using their mathematical definition with the help of Dijkstra's Algorithm to find out all shortest paths[9].

For Edge Contribution Factor and Degree Centralities, both Indegree and Outdegree, we have calculated using simple traversal though all nodes and summing their Indegree Count and Outdegree Count.

For Clustering Coefficient, we take a simple ratio of all neighbour edges with total possible directed edges between the neighbours.

3.1 CODE

```
#include<bits/stdc++.h>
using namespace std;

#define MAX 200

void edgeContribution(vector<vector<pair<int,int> >
>&,vector<vector<pair<int,int> > >&,int,vector<double>&);
void clusteringCoefficient(vector<vector<pair<int,int> >
>&,vector<vector<pair<int,int> > >&,int,vector<double>&);
void degreeCentrality(vector<vector<pair<int,int> >
>&,int,vector<double>&,vector<double> &);
void CCnBC(vector<vector<pair<int,int> >
>&,int,vector<double>&,vector<double>&);
int updatePass(vector<int>&,int,vector<int>&,int,vector<int>&);
int dijkstra(vector<vector<pair<int,int> > > &,int,int,vector<int>&);
void voting(vector<vector<pair<int,int> >
>&,vector<vector<pair<int,int> > >&,int);

int main()
{
	int N,w;
	do
	{
		cout<<"Enter number of Nodes (Maximum "<<MAX<<") : ";
		cin>>N;
		if((N>MAX)||(N<1))
			cout<<"Please Enter Valid Number Of Nodes!\n";
	}while((N>MAX)||(N<1));
	vector<vector<pair<int,int> > > outAdj(N),inAdj(N);
	printf("Enter the weights of each edge as an Adjacency Matrix
:\n");
	for(int i=0;i<N;++i)
		for(int j=0;j<N;++j)
		{
			cin>>w;
			if(w!=0)

	outAdj[i].push_back(make_pair(j,w)),inAdj[j].push_back(make_pair
(i,w));
		}
	voting(outAdj,inAdj,N);
	return 0;
}
void edgeContribution(vector<vector<pair<int,int> > >
&outA,vector<vector<pair<int,int> > > &inA,int V,vector<double>
&edgeCon)
{
	vector<int> nearWeightSum(V,0);
	int totalWeight=0;
	for(int i=0;i<V;++i)
	{
		map<pair<int,int>,bool > done;
		for(int j=0;j<outA[i].size();++j)
```

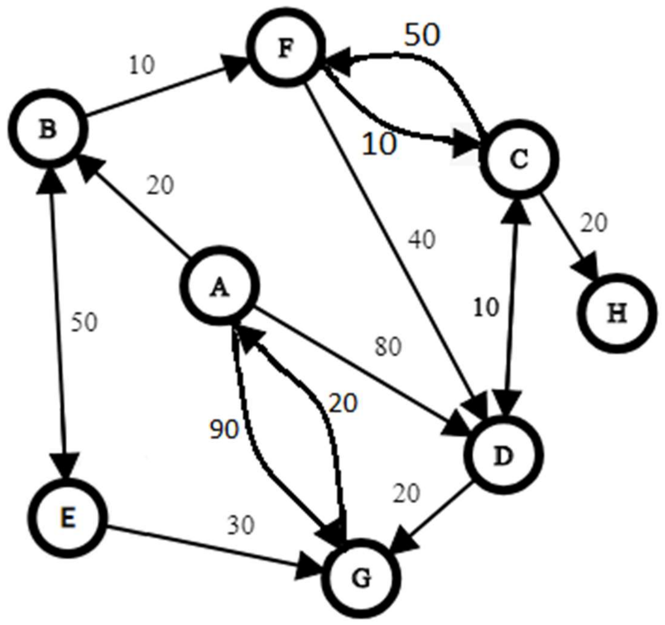
```
                {
                        totalWeight+=outA[i][j].second;
                        nearWeightSum[i]+=outA[i][j].second;
                        done[make_pair(i,outA[i][j].first)]=true;
                        int x=outA[i][j].first;
                        for(int k=0;k<outA[x].size();++k)
                        {

        if(done.find(make_pair(x,outA[x][k].first))==done.end())

        nearWeightSum[i]+=outA[x][k].second;
                                done[make_pair(x,outA[x][k].first)]=true;
                        }
                        for(int k=0;k<inA[x].size();++k)
                        {

        if(done.find(make_pair(inA[x][k].first,x))==done.end())
                                        nearWeightSum[i]+=inA[x][k].second;
                                done[make_pair(inA[x][k].first,x)]=true;
                        }
                }
                for(int j=0;j<inA[i].size();++j)
                {

        if(done.find(make_pair(inA[i][j].first,i))==done.end())
                                nearWeightSum[i]+=inA[i][j].second;
                        done[make_pair(inA[i][j].first,i)]=true;
                        int x=inA[i][j].first;
                        for(int k=0;k<outA[x].size();++k)
                        {

        if(done.find(make_pair(x,outA[x][k].first))==done.end())

        nearWeightSum[i]+=outA[x][k].second;
                                done[make_pair(x,outA[x][k].first)]=true;
                        }
                        for(int k=0;k<inA[x].size();++k)
                        {

        if(done.find(make_pair(inA[x][k].first,x))==done.end())
                                        nearWeightSum[i]+=inA[x][k].second;
                                done[make_pair(inA[x][k].first,x)]=true;
                        }
                }
        }
        cout<<"Edge Contribution Factor of Each Node is as Follows :\n";
        for(int i=0;i<V;++i)
        {
                edgeCon[i]=nearWeightSum[i]/(double)totalWeight;
                cout<<"Node "<<i+1<<" : "<<edgeCon[i]<<endl;
        }
}
void clusteringCoefficient(vector<vector<pair<int,int> > >
&outA,vector<vector<pair<int,int> > > &inA,int V,vector<double>
&clusterCoeff)
{
        vector<int> connection(V,0),neighbours(V,0);
        for(int i=0;i<V;++i)
```

```
	{
		set<int> neigh;
		map<pair<int,int>,bool> done;
		for(int j=0;j<outA[i].size();++j)
			neigh.insert(outA[i][j].first);
		for(int j=0;j<inA[i].size();++j)
			neigh.insert(inA[i][j].first);
		for(int j=0;j<outA[i].size();++j)
		{
			int x=outA[i][j].first;
			for(int k=0;k<outA[x].size();++k)
			{

	if((neigh.find(outA[x][k].first)!=neigh.end())&&(!done[make_pair
(x,outA[x][k].first)]))
						connection[i]++;
					done[make_pair(x,outA[x][k].first)]=true;
			}
			for(int k=0;k<inA[x].size();++k)
			{

	if((neigh.find(inA[x][k].first)!=neigh.end())&&(!done[make_pair(
inA[x][k].first,x)]))
						connection[i]++;
					done[make_pair(inA[x][k].first,x)]=true;
			}
		}
		for(int j=0;j<inA[i].size();++j)
		{
			int x=inA[i][j].first;
			for(int k=0;k<outA[x].size();++k)
			{

	if((neigh.find(outA[x][k].first)!=neigh.end())&&(!done[make_pair
(x,outA[x][k].first)]))
						connection[i]++;
					done[make_pair(x,outA[x][k].first)]=true;
			}
			for(int k=0;k<inA[x].size();++k)
			{

	if((neigh.find(inA[x][k].first)!=neigh.end())&&(!done[make_pair(
inA[x][k].first,x)]))
						connection[i]++;
					done[make_pair(inA[x][k].first,x)]=true;
			}
		}
		neighbours[i]=neigh.size();
	}
	cout<<"Clustering Coefficient of Each Node is as Follows :\n";
	for(int i=0;i<V;++i)
	{
		if(neighbours[i]<=1)
			clusterCoeff[i]=neighbours[i];
		else

	clusterCoeff[i]=connection[i]/(double)((neighbours[i])*(neighbou
rs[i]-1));
```

```
                cout<<"Node "<<i+1<<" : "<<clusterCoeff[i]<<endl;
        }
}
void degreeCentrality(vector<vector<pair<int,int> > > &outA,int
V,vector<double> &inDegree,vector<double> &outDegree)
{
        vector<int> inSum(V,0),outSum(V,0);
        int totalWeight=0;
        for(int i=0;i<V;++i)
        {
                for(int j=0;j<outA[i].size();++j)
                {
                        outSum[i]+=outA[i][j].second;
                        inSum[outA[i][j].first]+=outA[i][j].second;
                        totalWeight+=outA[i][j].second;
                }
        }
        cout<<"Indegree Centrality of Each Node is as Follows :\n";
        for(int i=0;i<V;++i)
        {
                inDegree[i]=inSum[i]/(double)totalWeight;
                cout<<"Node "<<i+1<<" : "<<inDegree[i]<<endl;
        }
        cout<<"Outdegree Centrality of Each Node is as Follows :\n";
        for(int i=0;i<V;++i)
        {
                outDegree[i]=outSum[i]/(double)totalWeight;
                cout<<"Node "<<i+1<<" : "<<outDegree[i]<<endl;
        }
}
void CCnBC(vector<vector<pair<int,int> > > &outA,int V,vector<double>
&closeCent,vector<double> &betCent)
{
        vector<int> pass(V,0);
        cout<<"Closeness Centrality of Each Node is as Follows :\n";
        for(int i=0;i<V;++i)
        {
                int farness=dijkstra(outA,V,i,pass);
                if(farness==INT_MAX)
                        closeCent[i]=0;
                else
                        closeCent[i]=1/(double)farness;
                cout<<"Node "<<i+1<<" : "<<closeCent[i]<<endl;
        }
        cout<<"Betweenness Centrality of Each Node is as Follows :\n";
        for(int i=0;i<V;++i)
        {
                betCent[i]=(pass[i]*2)/(double)((V-1)*(V-2));
                cout<<"Node "<<i+1<<" : "<<betCent[i]<<endl;
        }
}
int updatePass(vector<int> &distance,int V,vector<int> &parent,int
source,vector<int> &pass)
{
        for(int i=0;i<V;i++)
        {
                if((i!=source)&&(distance[i]<INT_MAX))
                {
                        for(int j=i;parent[j]!=-1;j=parent[j])
                                if(parent[parent[j]]!=-1)
                                        pass[parent[j]]++;
                }
        }
}
int dijkstra(vector<vector<pair<int,int> > > &outA,int V,int
source,vector<int> &pass)
{
        priority_queue<pair<int,int>,vector<pair<int,int>
>,greater<pair<int,int> > > pq;
        vector<int> distance(V,INT_MAX),parent(V,-1);
        distance[source]=0;
        pq.push(make_pair(0,source));
        while(!pq.empty())
        {
                int u=pq.top().second;
                pq.pop();
                for(int v=0;v<outA[u].size();v++)
                {

        if(distance[outA[u][v].first]>distance[u]+outA[u][v].second)
                        {

        distance[outA[u][v].first]=distance[u]+outA[u][v].second;

        pq.push(make_pair(distance[outA[u][v].first],outA[u][v].first));
                                parent[outA[u][v].first]=u;
                        }
                }
        }
        int sumDistance=0;
        for(int i=0;i<V;++i)
        {
                if(distance[i]==INT_MAX)
                {
                        sumDistance=INT_MAX;
                        break;
                }
                else
                        sumDistance+=distance[i];
        }
        updatePass(distance,V,parent,source,pass);
        return sumDistance;
}
void voting(vector<vector<pair<int,int> > >
&outA,vector<vector<pair<int,int> > > &inA,int V)
{
        vector<double>
closenessCent(V),ECF(V),clusterCoeff(V),BC(V),inDegree(V),outDegree(V);
        clusteringCoefficient(inA,outA,V,clusterCoeff);
        edgeContribution(outA,inA,V,ECF);
        degreeCentrality(outA,V,inDegree,outDegree);
        CCnBC(outA,V,closenessCent,BC);
        vector<double> sum(V);
        double
maxCloCent=closenessCent[0],maxECF=ECF[0],minCluCof=clusterCoeff[0],max
BC=BC[0],maxInDegree=inDegree[0],maxOutDegree=outDegree[0];
```

```
        vector<int> vote(V,0);
        for(int i=1;i<V;++i)
        {
                minCluCof=min(minCluCof,clusterCoeff[i]);
                maxECF=max(maxECF,ECF[i]);
                maxInDegree=max(maxInDegree,inDegree[i]);
                maxOutDegree=max(maxOutDegree,outDegree[i]);
                maxCloCent=max(maxCloCent,closenessCent[i]);
                maxBC=max(maxBC,BC[i]);
        }
        for(int i=0;i<V;++i)
        {
                if(minCluCof==clusterCoeff[i])
                        ++vote[i];
                if(maxECF==ECF[i])
                        ++vote[i];
                if(maxInDegree==inDegree[i])
                        ++vote[i];
                if(maxOutDegree==outDegree[i])
                        ++vote[i];
                if(maxCloCent==closenessCent[i])
                        ++vote[i];
                if(maxBC==BC[i])
                        ++vote[i];
                sum[i]=(double)1-
clusterCoeff[i]+ECF[i]+closenessCent[i]+inDegree[i]+outDegree[i]+BC[i];
        }
        for(int i=0;i<V;++i)
                cout<<vote[i]<<' ';
        cout<<endl;
        int maxVote=0,maxNode=0;
        for(int i=0;i<V;++i)
        {
                if(vote[i]>maxVote)
                {
                        maxNode=i;
                        maxVote=vote[i];
                }
                else if(vote[i]==maxVote)
                {
                        if(sum[i]>sum[maxNode])
                                maxNode=i;
                }
        }
        cout<<"Most Influential Node after voting is Node
"<<maxNode+1<<".\n";
}
```

4 AN EXAMPLE



Node	Edge Contribution Factor	Clustering Coefficient	Indegree Centrality	Outdegree Centrality	Closeness Centrality (10^{-3})	Betweenness Centrality
A	0.84	0.167	0.03	0.37	2.94	0.57
B	0.88	0	0.13	0.11	3.70	0.85
C	0.49	0.167	0.03	0.15	2.85	0.90
D	0.80	0.33	0.25	0.05	3.03	0.52
E	0.56	0	0.09	0.15	2.32	0
F	0.72	0.33	0.11	0.09	2.70	0.71
G	0.82	0.167	0.27	0.03	2.43	0.52
H	0.19	1	0.03	0	0	0

After Voting on the factors as our algorithm in Our Approach Section Suggests, we get the most influential node to be node B with 3 votes.

5 DATASET

We have taken a free dataset from Wiki-Vote which is a directed weighted network.

The network contains all the Wikipedia Voting data from the inception of Wikipedia till January 2008 that consist of 1235 elections.

Nodes in the network represent Wikipedia users and a directed edge from node *i* to node *j* with weight *w* represents that user *i* voted for user *j, w* times.

The Graph Contains 7066 nodes (voters and candidates) and has in total 103663 votes.

6 RESULT OF DATASET

Minimum (non-zero) Clustering Coefficient: 0.140936

Maximum Closeness Centrality: 0

Maximum Betweenness Centrality: 0.000331

Maximum Outdegree Centrality: 0.003667

Maximum Indegree Centrality: 0.000042

Maximum Edge Contribution Factor: 0.000729

Key Node: Node no 3

7 FUTURE SCOPE

In the future, our aim is to be able to assimilate the temporal data associated with each node such that we can find out at which point of time a node was influential (or central) in this network.

By using the above data, and concepts of Machine Learning, we also aim to be able to predict the shift in Centrality of the Network as time passes.

8 REFERENCES

1. Sabidussi, Gert. (1966). The Centrality Index of a Graph. Psychometrika. 31. 581-603. 10.1007/BF02289527.
2. David A. Bader and Kamesh Madduri. Parallel algorithms for evaluating centrality indices in real-world networks. In ICPP, pages 539–550. IEEE Computer Society, 2006.
3. P. Hage and F. Harary. Eccentricity and centrality in networks. Social Networks, 17:57–63, 1995
4. Nieminen, Juhani. (1974). On centrality in a graph. Scandinavian journal of psychology. 15. 332-6. 10.1111/j.1467-9450.1974.tb00598.x.
5. L. C. Freeman. A set of measures of centrality based on betweenness. Sociometry, 40:35–41, 1977.

6. S. Murai, "Theoretically and Empirically High Quality Estimation of Closeness Centrality," 2017 IEEE International Conference on Data Mining (ICDM), New Orleans, LA, 2017, pp. 985-990. doi: 10.1109/ICDM.2017.126
7. S. Muhuri, S. Chakraborty and S. K. Setua, "An Edge Contribution-Based Approach to Identify Influential Nodes from Online Social Networks," 2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS), Gwalior, 2016, pp. 155-160.
8. M. Li and C. O'Riordan, "The effect of clustering coefficient and node degree on the robustness of cooperation," 2013 IEEE Congress on Evolutionary Computation, Cancun, 2013, pp. 2833-2839. doi: 10.1109/CEC.2013.6557913
9. Ulrik Brandes, “A faster algorithm for betweenness centrality”, Journal of Mathematical Sociology, 25(2):163–177, 2001.