

Implementation of Social Network Algorithm

Project Report submitted in fulfillment of the requirement for the
degree of

Bachelor of Technology.

in

Computer Science & Engineering

under the Supervision of

Mr. Shailendra Shukla

By

Gunjan Goyal (111452)

to



Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled “Implementation of Social Network Algorithm”, submitted by Gunjan Goyal (111452) in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: May 15, 2015

Mr. Shailendra Shukla

Assistant Professor

Acknowledgement

On the submission of my report on “Implementation of Social Network Algorithm”, I would like to extend my gratitude and sincere thanks to my supervisor Mr. Shailendra Shukla, Department of Computer Science and Engineering for his constant motivation and support during the course of my work in the last 6 months. I truly appreciate and value his guidance and encouragement from the beginning to the end of this project. I am indebted to him for helping me shape the problem and providing insights towards the solution.

Above all, i would like to thank all my friends whose direct and indirect support helped me complete my project. The project would have been impossible without their perpetual moral support.

Date: May 15, 2014

Gunjan Goyal

CONTENTS

Certificate.....	2
Acknowledgement.....	3
List of Figures.....	6
List of Tables.....	7
Abstract.....	8

Chapter-1- Social Networks

Introduction.....	9
Applications.....	10

Chapter-2- Centrality

Introduction.....	12
Limitation.....	12
Types of Centrality.....	13
Degree Centrality.....	13
Closeness Centrality.....	14
Betweenness Centrality.....	15
Ego Network.....	16
Calculation of Ego Betweenness Centrality.....	16
Ego Degree and Ego Closeness.....	17

Chapter-3- Community Detection

Introduction.....	17
Girvan Newman Algorithm.....	18
Algorithm.....	18
Complexity.....	19
Bubble Rap Algorithm.....	20
Implementation.....	20

Assumptions.....	21
Working.....	21
Pseudocode.....	22

Chapter-5- Implementation

Coding.....	23
Graph Used.....	23
Text File For Graph Used.....	24
Result.....	25
Betweenness, Ego betweenness, Closeness, Degree of nodes.....	25
Degree Centrality	
Code.....	26
Output.....	30
Closeness Centrality	
Code.....	31
Output.....	42
Betweenness Centrality	
Code.....	43
Conclusion.....	50
References.....	51

List of Figures

S.No.	Title	Page No.
1.	Illustration of Bubble Rap Algorithm	20
2.	Graph Representation.	23

List of Tables

S.No.	Title	Page No.
1.	Betweenness, Ego Betweenness, Degree, Closeness of the nodes.....	24

Abstract

Message delivery in sparse Mobile Ad hoc Networks (MANETs) is difficult due to the fact that the network graph is rarely connected. A key challenge is to find a route that can provide good delivery performance and low end-to-end delay in a disconnected network graph where nodes may move freely.

In this project we seek to improve our understanding of human mobility in terms of social structures, and to use these structures in the design of forwarding algorithms for Pocket Switched Networks (PSNs). Taking human mobility traces from the real world, we discover that human interaction is heterogeneous both in terms of centrality (popular individuals) and communities.

The social network algorithms are used to detect centrality and community in a graph. We also show how these algorithms can be implemented in a distributed way, which demonstrates that it is applicable in the decentralized environment of PSNs.

CHAPTER 1

SOCIAL NETWORKS

1. INTRODUCTION

A social network is a social structure made up of a set of social actors (such as individuals or organizations) and a set of the ties between these actors. The social network perspective provides a set of methods for analyzing the structure of whole social entities as well as a variety of theories explaining the patterns observed in these structures. The study of these structures uses social network analysis to identify local and global patterns, locate influential entities, and examine network dynamics.

Social networks and the analysis of them is an inherently interdisciplinary academic field which emerged from social psychology, sociology, statistics, and graph theory.

The social network is a theoretical construct useful in the social sciences to study relationships between individuals, groups, organizations, or entire societies. The term is used to describe a social structure determined by such interactions. The ties through which any given social unit connects represent the convergence of the various social contacts of that unit. This theoretical approach is, necessarily, relational. An axiom of the social network approach to understanding social interaction is that social phenomena should be primarily conceived and investigated through the properties of relations between and within units, instead of the properties of these units themselves.

In general, social networks are self-organizing, emergent, and complex, such that a globally coherent pattern appears from the local interaction of the elements that make up the system. These patterns become more apparent as network size increases.

2. Social Network Analysis

Social network analysis (SNA) is the use of network theory to analyse social networks. Social network analysis views relationships in terms of network theory, consisting of *nodes*, representing individual actors within the network, and *ties* which represent relationships between the individuals, such as friendship, kinship, organizations and sexual relationships. These networks are often depicted in a social network diagram, where nodes are represented as points and ties are represented as lines.

Social network analysis has emerged as a key technique in modern sociology. It has also gained a significant following in anthropology, biology, communication studies, economics, geography, history, information science, organizational studies, political science, social psychology, development studies, and sociolinguistics.

3. APPLICATION

The main application of Social Network Algorithm is Delay Torrent Networking. Delay-tolerant networking (DTN) is an approach to computer network architecture that seeks to address the technical issues in heterogeneous networks that may lack continuous network connectivity. DTN routing algorithms provide forwarding by building and updating routing tables whenever mobility occurs. Since mobility is often unpredictable, and topology structure is highly dynamic. Rather than exchange much control traffic to create unreliable routing structures, which may only capture the “noise” of the network, it is preferred to search for some characteristics of the network which are less volatile than mobility.

Pocket Switched Network is a type of Delay Torrent Network. The forwarding algorithm used in PSN uses human mobility traces for data forwarding without network infrastructure which is less volatile than mobility. Example of such algorithm is Bubble Rap Algorithm which enhances delivery performance using two metrics- centrality and community.

CHAPTER 2

CENTRALITY

1. INTRODUCTION

Centrality refers to indicators which identify the most important vertices within a graph. The centrality of a node in a network is a measure of the structural importance of the node. A central node, typically, has a stronger capability of connecting other network members. There are several ways to measure centrality. The three most widely used centrality measures are Freeman's degree, closeness, and betweenness measures

2. LIMITATION OF CENTRALITY

Centrality indices have two important limitations, one obvious and the other subtle. The obvious limitation is that a centrality which is optimal for one application is often sub-optimal for a different application. Indeed, if this were not so, we would not need so many different centralities.

The more subtle limitation is the commonly held fallacy that vertex centrality indicates the relative importance of vertices. Centrality indices are explicitly designed to produce a ranking which allows indication of the most important vertices. This they do well, under the limitation just noted. The error is two-fold. Firstly, a ranking only orders vertices by importance, it does not quantify the difference in importance between different levels of the ranking. Secondly, and more importantly, the features which (correctly) identify the most important vertices in a given network/application do not generalize to the remaining vertices. The rankings are meaningless for the vast majority of network nodes. This explains why, for example, only the first few results of a Google image search appear in a reasonable order.

While the failure of centrality indices to generalize to the rest of the network may at first seem counter-intuitive, it follows directly from the above definitions. Complex networks have heterogeneous topology. To the extent that the optimal measure depends on the network structure of the most important vertices, a measure which is optimal for such vertices is sub-optimal for the remainder of the network.

3. TYPES OF CENTRALITY

3.1 DEGREE CENTRALITY

‘Degree’ centrality is measured as the number of direct ties that involve a given node. A node with high degree centrality maintains contacts with numerous other network nodes. Such nodes can be seen as popular nodes with large numbers of links to others. As such, a central node occupies a structural position (network location) that may act as a conduit for information exchange. Degree centrality for a given node p_i is calculated as:

$$C_D(p_i) = \sum_{k=1}^N a(p_i, p_k)$$

[1]

where $a(p_i, p_k) = 1$ if a direct link exists between p_i and p_k and $i \neq k$.

3.2 CLOSENESS CENTRALITY

‘Closeness’ centrality measures the reciprocal of the mean geodesic distance $d(p_i, p_k)$, which is the shortest path between a node p_i and all other reachable nodes. Closeness centrality can be regarded as a measure of how long it will take information to spread from a given node to other nodes in the network. Closeness centrality for a given node is calculated as:

$$C_C(p_i) = \frac{N - 1}{\sum_{k=1}^N d(p_i, p_k)}$$

[1]

where N is the number of nodes in the network and $i \neq k$.

3.3 BETWEENNESS CENTRALITY

‘Betweenness’ centrality measures the extent to which a node lies on the paths linking other nodes. Betweenness centrality can be regarded as a measure of the extent to which a node has control over information flowing between others. A node with a high betweenness centrality has a capacity to facilitate interactions between the nodes that it links. In our case it can be regarded as how well a node can facilitate communication to other nodes in

the network. Betweenness centrality is calculated as:

$$C_B(p_i) = \sum_{j=1}^N \sum_{k=1}^{j-1} \frac{g_{jk}(p_i)}{g_{jk}}$$

[1]

where g_{jk} is the total number of geodesic paths linking p_j and p_k , and $g_{jk}(p_i)$ is the number of those geodesic paths that include p_i .

3.4 EGO NETWORK

Ego networks are the networks consisting of a single actor (ego) together with the actors they are connected to (alters) and all the links among those alters. These networks are also known as the neighborhood networks or first order neighborhoods of ego.

The attraction of ego networks is the ease of collecting data compared with collecting data on whole networks. Information on the alters (including how they are connected), is usually obtained entirely from ego.

2.4.1 CALCULATION OF EGO BETWEENNESS CENTRALITY

Betweenness centrality is calculated using an ego network representation of the nodes with which the ego node has come into contact. Mathematically, node contacts can be represented by an adjacency matrix A , which is an $n \times n$ symmetric matrix, where n is the number of contacts a given node has encountered. The adjacency matrix has elements:

$$[6] \quad A_{ij} = \begin{cases} 1 & \text{if there is a contact between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

The Ego Betweenness Centrality is the value of the Betweenness Centrality computed using only the nodes and the links in the ego-network of a node. The computation of the Ego Betweenness Centrality for an undirected graph has been proposed in. Given A_n the adjacency matrix of the ego network of n , $EN(n)$, the EBC is the sum of the reciprocal values $A_n^{-2}(i; j)$ such that $A_n(i; j) = 0$, as defined in

$$[6] \quad EBC(n) = \sum_{A_n(i,j)=0, j>i} \frac{1}{A_n^2(i, j)}$$

2.4.2 Ego Degree and Ego Closeness Centrality

Degree centrality is a local property and the ego degree centrality of ego is the same as the degree of the actor in the whole network.

Closeness is about the connections from an actor to all other actors in the network and so is simply not applicable to ego networks.

CHAPTER 3

COMMUNITY DETECTION

1. INTRODUCTION

A social network consists of a set of people forming socially meaningful relationships, where prominent patterns or information flow are observed. A network is said to have community structure if the nodes of the network can be easily grouped into sets of nodes such that each set of nodes is densely connected internally.

The community is classified into two:

- **Overlapping Community-** A node belongs to different communities simultaneously.
- **Non-Overlapping Community-** A node belongs to only one community at a time.

2. Girvan-Newman Algorithm

It is the hierarchical method to detect communities in a complex network. The Girvan–Newman algorithm detects communities by progressively removing edges from the original network. The connected components of the remaining network are the communities. Instead of trying to construct a measure that tells us which edges are the most central to communities, the Girvan–Newman algorithm focuses on edges that are most likely "between" communities.

2.1 Algorithm

The algorithm's steps for community detection are summarized below

1. The betweenness of all existing edges in the network is calculated first.
2. The edge with the highest betweenness is removed.
3. The betweenness of all edges affected by the removal is recalculated.
4. Steps 2 and 3 are repeated until no edges remain.

2.2 Complexity

The fact that the only betweenness being recalculated are only the ones which are affected by the removal, may lessen the running time of the process' simulation in computers. However, the betweenness centrality must be recalculated with each step, or severe errors occur.

3 BUBBLE RAP ALGORITHM

Bubble Rap algorithm is a hybrid algorithm that selects high centrality nodes and community members of destination as relays. There is no prior information as we live in decentralized world without access to infrastructure so distributed detection of node popularity and node community and the use of these forwarding decisions are crucial.

3.1 IMPLEMENTATION

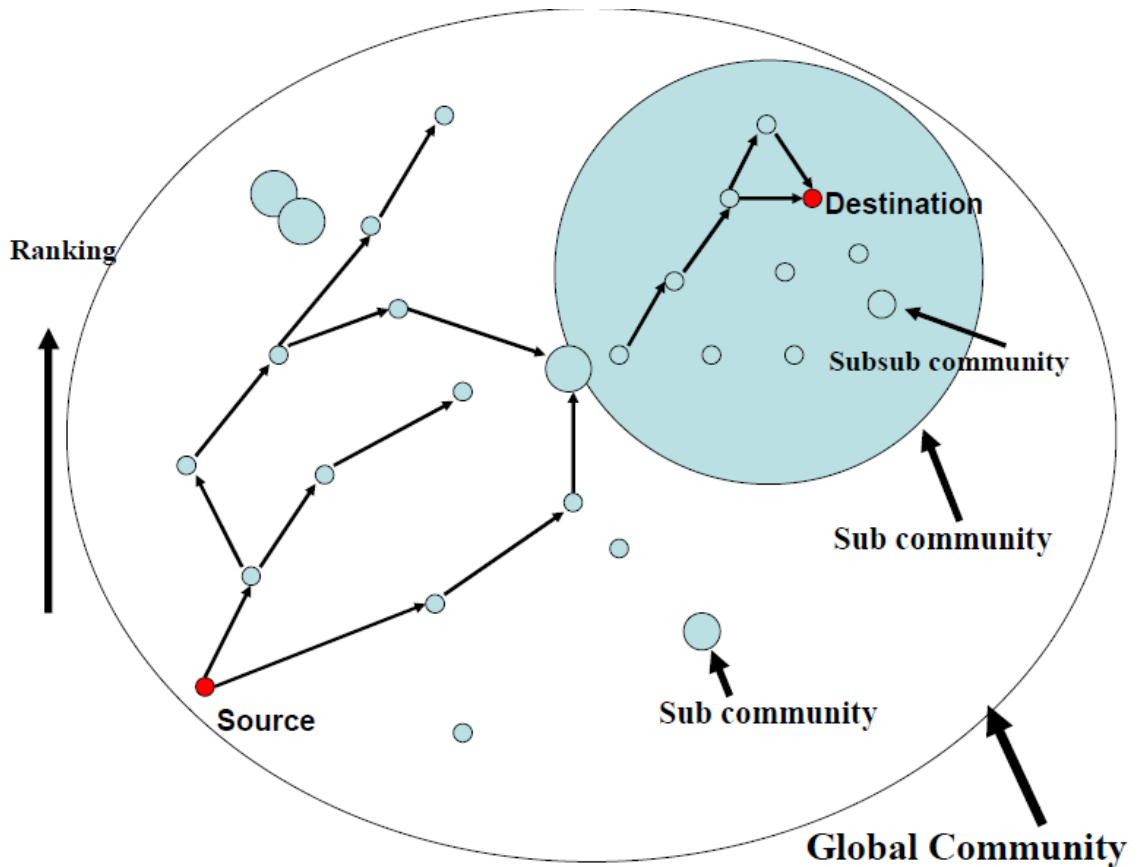


Fig 1: ILLUSTRATION OF BUBBLE RAP ALGORITHM^[1]

3.1.1. ASSUMPTIONS

1. Each node belongs to at least one community. Here we allow single node communities to exist.
2. Each node has a global ranking (i.e. global centrality) across the whole system, and also a local ranking within its local community. It may also belong to multiple communities and hence may have multiple local rankings.

3.1.2 WORKING

1. If a node has a message destined for another node, this node first *bubbles* the message up the hierarchical ranking tree using the global ranking, until it reaches a node which is in the same community as the destination node.
2. Then the local ranking system is used instead of the global ranking, and the message continues to bubble up through the local ranking tree until the destination is reached or the message expires.
3. This method does not require every node to know the ranking of all other nodes in the system, but just to be able to compare ranking with the node encountered, and to push the message using a greedy approach.
4. In order to reduce cost, we also require that whenever a message is delivered to the community, the original carrier can delete this message from its buffer to prevent further dissemination. This assumes that the community member can deliver this message.

3.2 PSEUDOCODE

Algorithm 1: BUBBLE RAP

```
begin
  foreach EncounteredNodei do
    if (LabelOf(currentNode) == LabelOf(destination)) then
      if (LabelOf(EncounteredNodei) ==
          LabelOf(destination))
          and
          (LocalRankOf(EncounteredNodei) >
           LocalRankOf(currentNode))
      then
        | EncounteredNodei.addMessageToBuffer(message)
      else
        if (LabelOf(EncounteredNodei) ==
            LabelOf(destination))
            or
            (GlobalRankOf(EncounteredNodei) >
             GlobalRankOf(currentNode))
        then
          | EncounteredNodei.addMessageToBuffer(message)
    end
  end
```

ALGORITHM 1:REFER[1]

CHAPTER 5 IMPLEMENTATION

1. CODING

The coding for all the algorithms used is done in C.

The software used is Dev-C++.

2. GRAPH USED:^[2]

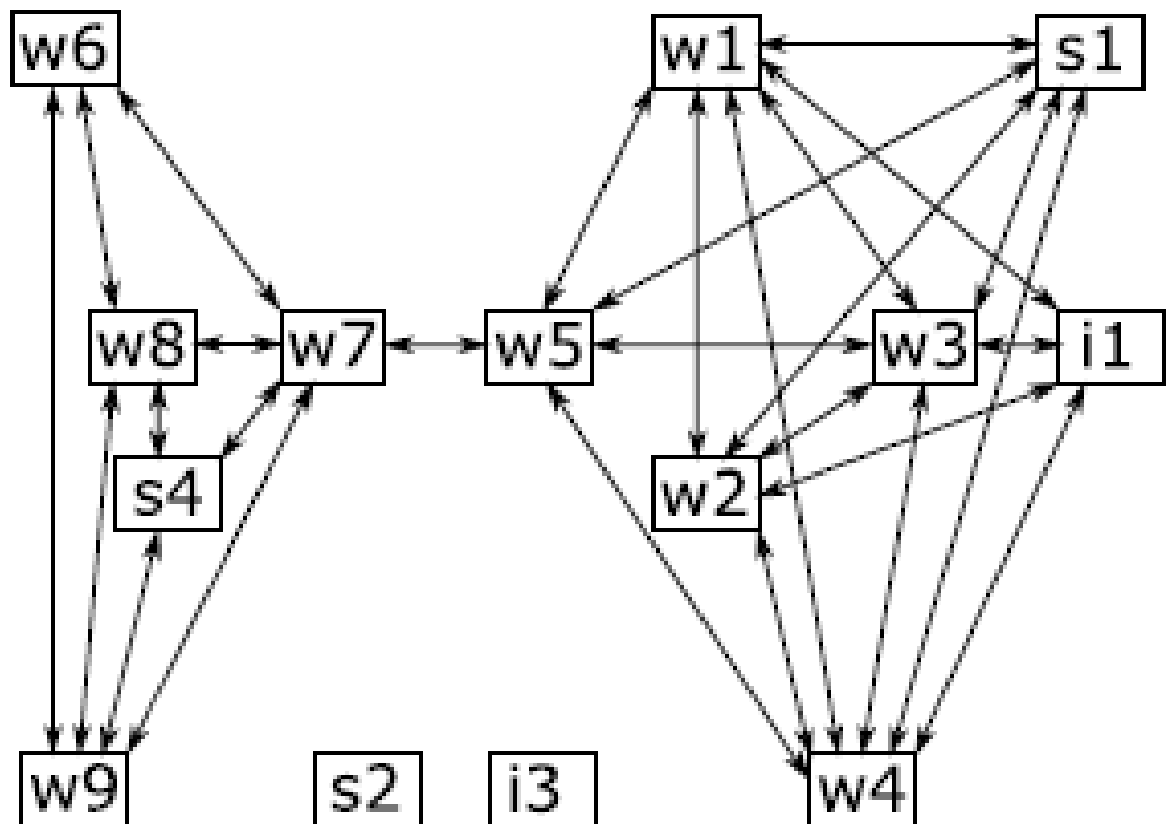
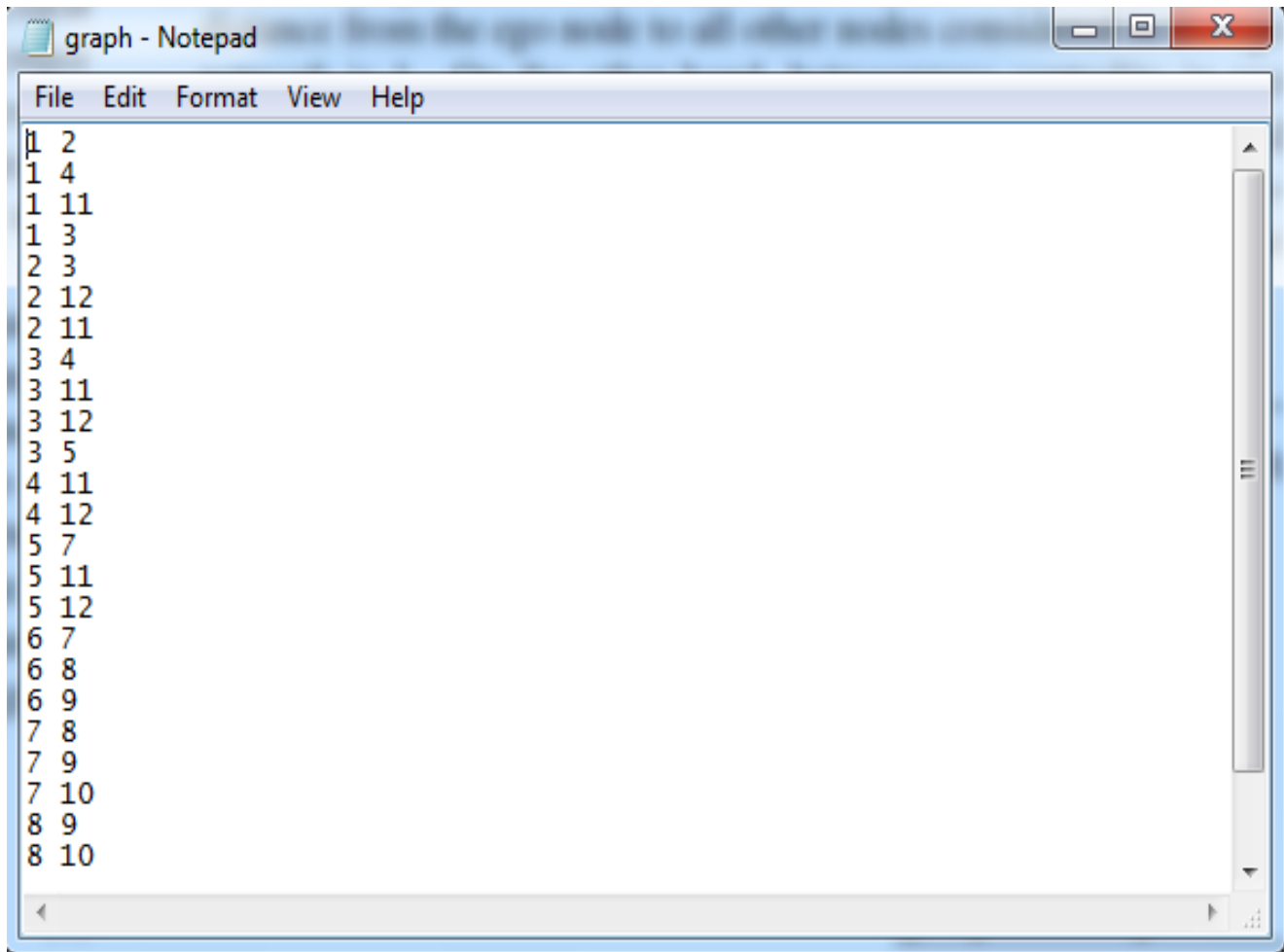


Fig 2

3. THE TEXT FILE FOR THE GRAPH



```
graph - Notepad
File Edit Format View Help
1 2
1 4
1 11
1 3
2 3
2 12
2 11
3 4
3 11
3 12
3 5
4 11
4 12
5 7
5 11
5 12
6 7
6 8
6 9
7 8
7 9
7 10
8 9
8 10
```


4.RESULT

3.1DIFFERENT BETWEENNESS AND EGO BETWEENNESS OF ALL THE NODES

Node	Sociocentric Betweenness	Degree	Closeness	Egocentric Betweenness
w1	3.75	2	0.52	0.83
w2	0.25	2	0.24	0.25
w3	3.75	3	0.17	0.83
w4	3.75	2	0.12	0.83
w5	30	2	0.10	4
w6	0	1	0.08	0
w7	28.33	2	0.07	4.33
w8	0.33	2	0.65	0.33
w9	0.33	2	0.06	0.33
s1	1.5	1	0.05	0.25
s2	0	2	0.04	0
s4	0	2	0.04	0
i1	0	0	0	0
i2	0	0	0	0

4.1 Degree Centrality

4.1.1 Code for Degree Centrality

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int count_occur(int a[], int num_elements, int value);
void print_array(int a[], int num_elements);

main()
{
char ch, file_name[25];
FILE *fp;

printf("Enter the name of file you wish to see\n");
gets(file_name);

fp = fopen(file_name,"r"); // read mode

int integers[100];

int i=0,j,k=0,length,n;
int num;
int num_occ, value;

while(fscanf(fp, "%d", &num) > 0) {
integers[i] = num;
```

```

i++;
}

while(k!=i+1){
length=k;
k++;
}

printf("\n%d",length);

for(value=1; value<8; value++)
{
num_occ = count_occur(integers, length, value);
printf("The value %d was found %d times.\n", value, num_occ);
printf("The Degree Centrality Of %d is %d.\n",value,num_occ/2);

}

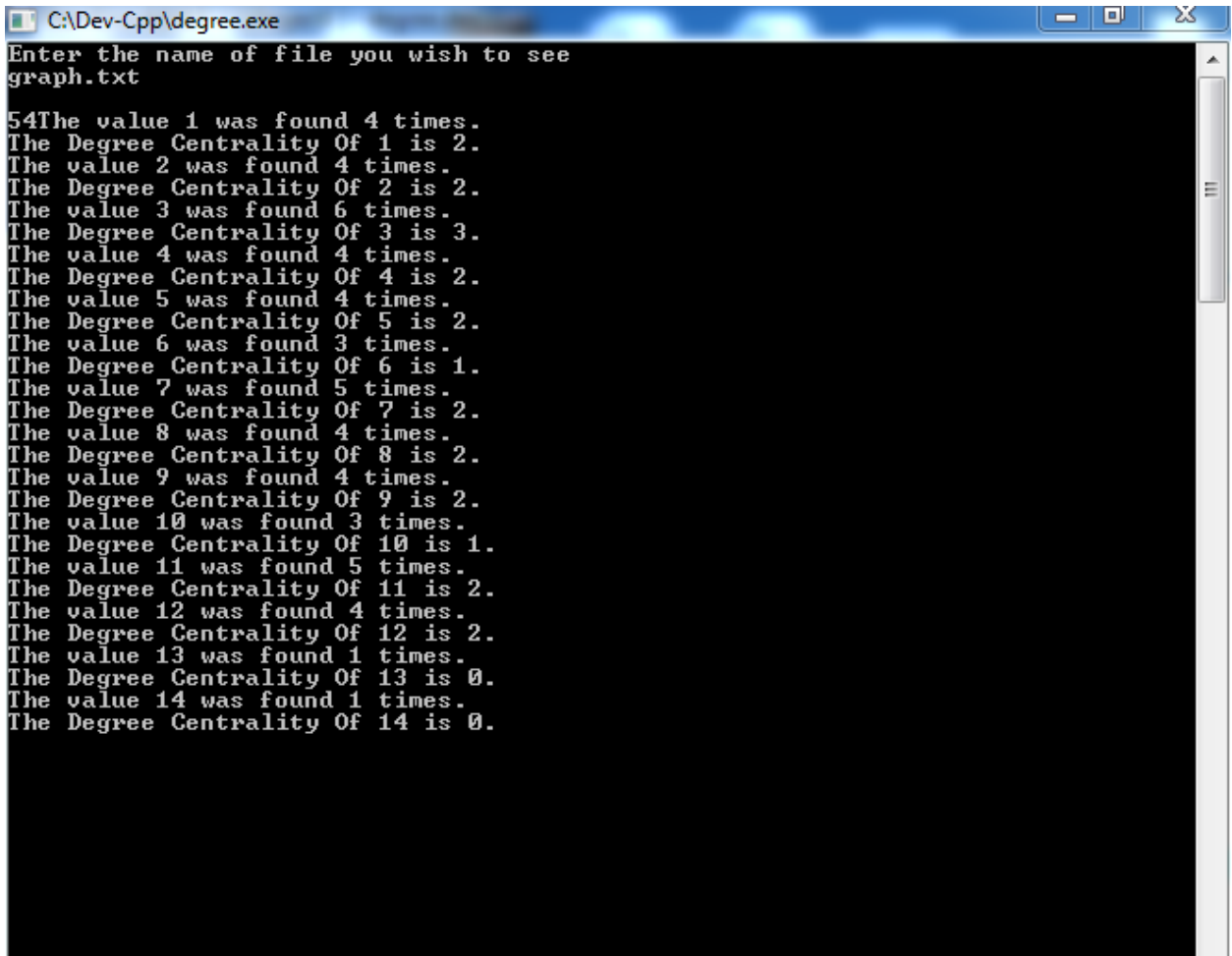
fclose(fp);
getch();
//return 0;
}

int count_occur(int a[], int num_elements, int value)
/* checks array a for number of occurrences of value */
{
int i, count=0;
for (i=0; i<num_elements; i++)
{
if (a[i] == value)

```

```
{  
++count; /* it was found */  
}  
}  
return(count);  
}
```

4.1.2 Output for Degree Centrality



```
C:\Dev-Cpp\degree.exe
Enter the name of file you wish to see
graph.txt

54The value 1 was found 4 times.
The Degree Centrality Of 1 is 2.
The value 2 was found 4 times.
The Degree Centrality Of 2 is 2.
The value 3 was found 6 times.
The Degree Centrality Of 3 is 3.
The value 4 was found 4 times.
The Degree Centrality Of 4 is 2.
The value 5 was found 4 times.
The Degree Centrality Of 5 is 2.
The value 6 was found 3 times.
The Degree Centrality Of 6 is 1.
The value 7 was found 5 times.
The Degree Centrality Of 7 is 2.
The value 8 was found 4 times.
The Degree Centrality Of 8 is 2.
The value 9 was found 4 times.
The Degree Centrality Of 9 is 2.
The value 10 was found 3 times.
The Degree Centrality Of 10 is 1.
The value 11 was found 5 times.
The Degree Centrality Of 11 is 2.
The value 12 was found 4 times.
The Degree Centrality Of 12 is 2.
The value 13 was found 1 times.
The Degree Centrality Of 13 is 0.
The value 14 was found 1 times.
The Degree Centrality Of 14 is 0.
```

4.2 Closeness Centrality

4.2.1 Code for Closeness Centrality

```
#include<stdio.h>
#include<conio.h>
#define infinity 999

void dij(int n,int v,int cost[20][20],int dist[])
{
    int i,u,count,w,flag[10],min;
    for(i=1;i<=n;i++)
        flag[i]=0,dist[i]=cost[v][i];
    count=2;
    while(count<=n)
    {
        min=99;
        for(w=1;w<=n;w++)
            if(dist[w]<min && !flag[w])
                min=dist[w],u=w;
        flag[u]=1;
        count++;
        for(w=1;w<=n;w++)
            if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
                dist[w]=dist[u]+cost[u][w];
    }
}

int main()
{
    int edges,i,j,tail,head,gtype,adj[20][20],n,k;
    //clrscr();
```

```

int number;
int m=0;
double close=0.0;
int sum=0;
int max=0;
int dist1[20],dist2[20],dist3[20],dist4[20],dist5[20],dist6[20],dist7[20],dist8[20],dist9[20],
dist10[20],dist11[20],dist12[20],dist13[20],dist14[20];
FILE* in_file = fopen("graph.txt", "r"); // read only
int arr[100][2];
int count=0;
int counter=0;

if (! in_file ) // equivalent to saying if ( in_file == NULL )
{
printf("oops, file can't be read\n");
exit(-1);
}

// attempt to read the next line and store
// the value in the "number" variable

while ( fscanf(in_file, "%d", &number ) == 1 )
{

if(max<number)
{
max=number;
}
if(count>1)
count=0;
arr[counter/2][count]=number;
}

```

```

        count++;
        counter++;

    }

    m=max;
    n=max;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
adj[i][j]=0;
}
//printf("\n");
}
//printf("%d",n);
//printf("%d max",max);

for(i=1;i<=n;i++)
{
//    printf("%d",n);
for(j=1;j<=n;j++)
{
//    printf("%d",n);
adj[i][j]=0; //Initialize the array of n*n size with 0
}
}

//printf("%d",n);

edges=n*(n-1)/2; //maximum no. of edges in undirected

```



```

//printf("%d",n);
//printf("%d",max);

for(i=0;i<counter/2;i++)
    {
        for(j=0;j<2;j++)
            {
//                printf("%d ",arr[i][j]);
                if(j==0)
                    tail=arr[i][j];
                else
                    head=arr[i][j];
            }
//        printf("\n");
//        printf("tail %d",tail);
//        printf("tail %d",n);
                if(tail>n||head>n||tail<0||head<0)
                    {
printf("Invalid edge!\n");
break;
i--;
}
                else
                    {
adj[tail][head]=1;

adj[head][tail]=1;

```

```

    }
        }
printf("\nAdjacency Matrix \n\n");
printf("%d\n\n",n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("\t%d",adj[i][j]);

if(adj[i][j]==0)
    adj[i][j]=infinity;

}
printf("\n");
}

dij(n,1,adj,dist1);
printf("\n Shortest path for 1:\n");
for(i=2;i<=n;i++){
    if(dist1[i]==999)

        dist1[i]=0;
    sum+=dist1[i];
if(i!=1)
    printf("%d->%d,adj=%d\n",1,i,dist1[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 1 is %f",close);

```

```

dij(n,2,adj,dist2);
printf("\n Shortest path for 2:\n");
for(i=1;i<=n;i++){
    if(dist2[i]==999)
        dist2[i]=0;
    sum+=dist2[i];
if(i!=2)
    printf("%d->%d,adj=%d\n",2,i,dist2[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 2 is %f",close);

```

```

dij(n,3,adj,dist3);
printf("\n Shortest path for 3:\n");
for(i=1;i<=n;i++){
    if(dist3[i]==999)

        dist3[i]=0;
    sum+=dist3[i];
if(i!=3)
    printf("%d->%d,adj=%d\n",3,i,dist3[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 3 is %f",close);

```

```

dij(n,4,adj,dist4);
printf("\n Shortest path for 4:\n");
for(i=1;i<=n;i++){

```

```

        if(dist4[i]==999)

                dist4[i]=0;
        sum+=dist4[i];
if(i!=4)
    printf("%d->%d,adj=%d\n",4,i,dist4[i]);
}
close=(float)(m-1)/sum;
printf("\n\ncloseness centrality of 4 is %f",close);

```

```

dij(n,5,adj,dist5);
printf("\n Shortest path for 5:\n");
for(i=1;i<=n;i++){
        if(dist5[i]==999)
                dist5[i]=0;
        sum+=dist5[i];
if(i!=5)
    printf("%d->%d,adj=%d\n",5,i,dist5[i]);
}
close=(float)(m-1)/sum;
printf("\n\ncloseness centrality of 5 is %f",close);

```

```

dij(n,6,adj,dist6);
printf("\n Shortest path for 6:\n");
for(i=1;i<=n;i++){
        if(dist6[i]==999)

                dist6[i]=0;
        sum+=dist6[i];
if(i!=6)

```

```

    printf("%d->%d,adj=%d\n",6,i,dist6[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 6 is %f",close);

dij(n,7,adj,dist7);
printf("\n Shortest path for 7:\n");
for(i=1;i<=n;i++){
    if(dist7[i]==999)
        dist7[i]=0;
    sum+=dist7[i];
if(i!=7)
    printf("%d->%d,adj=%d\n",7,i,dist7[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 7 is %f",close);

dij(n,8,adj,dist8);
printf("\n\n Shortest path for 8:\n");
for(i=1;i<=n;i++){
    if(dist8[i]==999)

        dist8[i]=0;
    sum+=dist8[i];
if(i!=8)
    printf("%d->%d,adj=%d\n",8,i,dist8[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 8 is %f",close);

```

```

dij(n,9,adj,dist9);
printf("\n\n Shortest path for 9:\n");
for(i=1;i<=n;i++){
    if(dist9[i]==999)

        dist9[i]=0;
        sum+=dist9[i];
if(i!=9)
    printf("%d->%d,adj=%d\n",9,i,dist9[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 9 is %f",close);

```

```

dij(n,10,adj,dist10);
printf("\n Shortest path for 10:\n");
for(i=1;i<=n;i++){
    if(dist10[i]==999)

        dist10[i]=0;
        sum+=dist10[i];
if(i!=10)
    printf("%d->%d,adj=%d\n",10,i,dist10[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 10 is %f",close);

```

```

dij(n,11,adj,dist11);
printf("\n Shortest path for 11:\n");
for(i=1;i<=n;i++){

```

```

        if(dist11[i]==999)

                dist11[i]=0;
        sum+=dist11[i];
if(i!=11)
    printf("%d->%d,adj=%d\n",11,i,dist11[i]);
}
close=(float)(m-1)/sum;
printf("\ncloseness centrality of 11 is %f",close);

dij(n,12,adj,dist12);
printf("\n Shortest path for 12:\n");
for(i=1;i<=n;i++){
        if(dist12[i]==999)

                dist12[i]=0;
        sum+=dist12[i];
if(i!=12)
    printf("%d->%d,adj=%d\n",12,i,dist12[i]);
}
close=(float)(m-1)/sum;
printf("\n\ncloseness centrality of 12 is %f",close);
printf("\n\ncloseness centrality of 13 is 0");
printf("\n\ncloseness centrality of 14 is 0");

getch();
return 0;
}

dij(n,1,adj,dist);
printf("\n Shortest path:\n");

```

```
for(i=2;i<=n;i++){
    if(dist[i]==999)

        dist[i]=0;
    sum+=dist[i];
if(i!=1)
    printf("%d->%d,adj=%d\n",1,i,dist[i]);
}
close=(float)(m-1)/sum;
printf("\n%f",close);
```


4.2.2 Output for Closeness Centrality

```
C:\Dev-Cpp\closeness2.exe
Adjacency Matrix
14
    0    1    1    1    0    0    0    0    0
0    1    0    0    0    0    0    0    0    0
0    1    0    1    0    0    0    0    0    0
0    1    1    0    1    1    0    0    0    0
0    1    1    0    0    0    0    0    0    0
0    1    0    1    0    0    0    0    0    0
0    0    0    1    0    0    0    1    0    0
0    0    0    0    0    0    0    1    1    1
0    0    0    0    0    0    1    1    0    1
1    0    0    0    0    0    0    1    1    0
1    0    0    0    0    0    0    1    1    1
1    0    0    0    0    0    0    1    1    0
0    0    0    0    0    0    0    0    1    1
0    1    1    1    1    1    1    0    0    0
0    0    0    0    0    0    1    0    0    0
0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0

Shortest path for 1:
1->2,adj=1
1->3,adj=1
1->4,adj=1
1->5,adj=2
1->6,adj=4
1->7,adj=3
1->8,adj=4
1->9,adj=4
1->10,adj=4
1->11,adj=1
1->12,adj=0
1->13,adj=0
1->14,adj=0

closeness centrality of 1 is 0.520000
Shortest path for 2:
2->1,adj=1
2->3,adj=1
2->4,adj=2
2->5,adj=2
2->6,adj=4
2->7,adj=3
2->8,adj=4
```

```
C:\Dev-Cpp\closeness2.exe
closeness centrality of 9 is 0.058036
Shortest path for 10:
10->1,adj=4
10->2,adj=4
10->3,adj=3
10->4,adj=4
10->5,adj=2
10->6,adj=2
10->7,adj=1
10->8,adj=1
10->9,adj=1
10->11,adj=3
10->12,adj=0
10->13,adj=0
10->14,adj=0

closeness centrality of 10 is 0.051793
Shortest path for 11:
11->1,adj=1
11->2,adj=1
11->3,adj=1
11->4,adj=1
11->5,adj=1
11->6,adj=3
```

4.3 Code for Betweenness centrality

```
// Program for Floyd Warshall Algorithm
#include<stdio.h>

/* Define Infinite as a large enough value. This value will be used
   for vertices not connected to each other */
#define infinity 999

// A function to print the solution matrix

void printSolution(int dist[20][20]);

// Solves the all-pairs shortest path problem using Floyd Warshall algorithm

void floydWarshell (int graph[20][20])
{

/* dist[][] will be the output matrix that will finally have the shortest
   distances between every pair of vertices */
   int dist[20][20], i=0, j=0, k=0;

/* Initialize the solution matrix same as input graph matrix. Or
   we can say the initial values of shortest distances are based
   on shortest paths considering no intermediate vertex. */

   for (i = 0; i < 14; i++)
       for (j = 0; j < 14; j++)
           dist[i][j] = graph[i][j];
```

```

/* Add all vertices one by one to the set of intermediate vertices.
---> Before start of a iteration, we have shortest distances between all
pairs of vertices such that the shortest distances consider only the
vertices in set {0, 1, 2, .. k-1} as intermediate vertices.
----> After the end of a iteration, vertex no. k is added to the setof
intermediate vertices and the set becomes {0, 1, 2, .. k} */

```

```

for (k = 0; k < 14; k++)
{
    // Pick all vertices as source one by one
    for (i = 0; i < 14; i++)
    {
        // Pick all vertices as destination for the
        // above picked source
        for (j = 0; j < 14; j++)
        {
            // If vertex k is on the shortest path from
            // i to j, then update the value of dist[i][j]
            if (dist[i][k] + dist[k][j] < dist[i][j])
                dist[i][j] = dist[i][k] + dist[k][j];
        }
    }
}

// Print the shortest distance matrix
printSolution(dist);
}

/* A utility function to print solution */

```

```

void printSolution(int dist[20][20])
{
    int i=0,j=0;
    printf("\n\n");
    printf ("Following matrix shows the shortest distances"
           " between every pair of vertices \n");
    for (i = 0; i < 14; i++)
    {
        for (j = 0; j < 14; j++)
        {
            if (dist[i][j] == infinity)
                printf("\t%s", "INF");
            else
                printf ("\t%d", dist[i][j]);
        }
        printf("\n");
    }
}

```

// driver program to test above function

```

int main()
{   int edges,i,j,tail,head,gtype,adj[20][20],n,k;
//clrscr();
    int number;
    int m=0;
    double close=0.0;
    int sum=0;
int max=0;

    FILE* in_file = fopen("graph.txt", "r"); // read only
    int arr[100][2];

```

```

        int count=0;
        int counter=0;

if (! in_file ) // equivalent to saying if ( in_file == NULL )      {
    printf("oops, file can't be read\n");
    exit(-1);
}

// attempt to read the next line and store
// the value in the "number" variable

while ( fscanf(in_file, "%d", &number ) == 1 )
{

    if(max<number)
    {
        max=number;
        }
        if(count>1)
        count=0;
        arr[counter/2][count]=number;
        count++;
        counter++;

    }

    m=max;
    n=max;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)

```

```

{
adj[i][j]=0;
}
//printf("\n");
}
//printf("%d",n);
//printf("%d max",max);

for(i=1;i<=n;i++)
{
//    printf("%d",n);
for(j=1;j<=n;j++)
{
//    printf("%d",n);
adj[i][j]=0; //Initialize the array of n*n size with 0
}
}

//printf("%d",n);

edges=n*(n-1)/2; //maximum no. of edges in undirected

//printf("%d",n);
//printf("%d",max);

for(i=0;i<counter/2;i++)
{
    for(j=0;j<2;j++)
    {

```

```

//          printf("%d ",arr[i][j]);
        if(j==0)
            tail=arr[i][j];
        else
            head=arr[i][j];

    }

//          printf("\n");
//          printf("tail %d",tail);
//          printf("tail %d",n);
            if(tail>n||head>n||tail<0||head<0)
{
printf("Invalid edge!\n");
break;
i--;
}
else
{
adj[tail][head]=1;

adj[head][tail]=1;

}
    }

printf("\nAdjacency Matrix \n\n");
printf("%d\n\n",n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("\t%d",adj[i][j]);

```



```
if(adj[i][j]==0)
    adj[i][j]=infinity;

}
printf("\n");
\}

// Print the solution
floydWarshell(adj);
getch();
return 0;
}
```

CONCLUSION

In this project, I have analyzed different algorithm to compute Centrality and Community of different nodes in a network using a complex graph.

The algorithms used give the value of the centrality precisely and tells about the importance and popularity of the node in a network.

REFERENCES

1. BUBBLE Rap: Social-based Forwarding in Delay Tolerant Networks by Pan Hui, Jon Crowcroft, Eiko Yoneki University of Cambridge, Computer Laboratory Cambridge CB3 0FD United Kingdom-2008
2. Social Network Analysis for Routing in Disconnected Delay-Tolerant MANETs by Elizabeth Daly and Mads Haahr Distributed Systems Group, Computer Science Department, Trinity College Dublin Dublin, Ireland-2007
3. Fast exact computation of betweenness centrality in social networks by Miriam Baglioni , Filippo Geraci , Marco Pellegrini and Ernesto Lastres.
4. Ego network betweenness by Martin Everetta, Stephen P. Borgattib
5. http://en.wikipedia.org/wiki/Betweenness_centrality
6. http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm
7. Finding Communities in Dynamic Social Network Chayant Tantipathananandh and Tanya Y. Berger-Wolf
8. On the Local Approximations of Node Centrality in Internet Router-level Topologies by Panagiotis Pantazopoulos, Merkourios Karaliopoulos, and Ioannis Stavrakakis
9. Community structure in social and biological networks by M. Girvan and M. E. J. Newman