CS6385 ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

IMPLEMENTATION OF NAGAMOCHI IBARAKI ALGORITHM TO FIND THE MINIMUM CUT AND NUMBER OF CRITICAL EDGES

SUBMITTED BY

KAUSHIK SIVAKUMAR

kxs104820@utdallas.edu

2021108266

TABLE OF CONTENTS

OBJECTIVE

Based on the number of nodes (n) and Number of Edges (m) taken from the user,

- To implement Nagamochi-Ibaraki algorithm and find a minimum cut in an undirected graph with reasonable running time.
- To find the edge connectivity and critical edges for various values of m and n
- To express the edge connectivity as a function of the average degree of the graph.
- To express the number of critical edges as a function of the average degree of the graph.

ALGORITHM USED TO FIND MINIMUM CUT:

Nagamochi-Ibaraki Algorithm

Nagamochi and Ibaraki developed an algorithm to find the minimum cut in an entire graph by using the nodes and the connectivity between them.

The edge connectivity of a graph G is denoted by $\lambda\ (G)$.If x,y are two different nodes then $\lambda\ (x,y)$.denote the edge-connectivity between the node pairs.

Description of the algorithm:

- A node is picked and the node list of all the nodes that are connected to it is obtained. From this list the node with the maximum adjacency to the previous node or node pairs are picked and are ordered in that manner. This ordering is known as MA ordering or Maximum Adjacency ordering.
- From the obtained maximum adjacency, last two nodes are picked.
- The connectivity on the reduced graph (Gxy) is caluculated.

- Overall connectivity of the random graph is:
  λ (G)= min { λ (X, Y), λ (G_{XY})}
- We can find a pair of XY nodes as follows. It is obtained by Maximum Adjacency ordering.

An MA ordering $V_1,\ldots,V_n$ of the nodes is generated recursively by the following algorithm:

1. Take any random node V_1

2.Once $V_1,\ldots,V_i$ is already chosen, take a node for V_{i+1} that has the maximum number of edges connecting it with the set $\{V_1,\ldots,V_i\}$.

Nagamochi and Ibaraki proved that this ordering has the following property:

$$\lambda\,(V_{n-1},V_n) = d(V_n)$$

where $d(V_n)$ denotes the degree of the node V_n.

i.e. the connectivity between the last two pairs of nodes is equal to the degree of the last node.

By repeating this algorithm recursively we can then obtain the minimum cut for the given graph.

- The average degree of the graph is obtained by dividing the maximum edges by the number of nodes

$$\text{Average Degree} = 2 * (\text{maxedges} / \text{Noofnodes})$$

- A critical edge is an edge in the graph such that by removing it the graph connectivity is reduced

$$\lambda (G - e) < \lambda (G)$$

Where $\lambda (G - e)$ Connectivity of graph with edge e deleted

PSEUDO CODE

1. *Read values of nodes(n) and edges(m) from the user*
2. *Create graph G with n nodes and m edges*
3. *Using the graph created find the minimum cut of the graph using nagamochi()*
   i.e.,
4. *The minimum cut is 0 if graph is disconnected.*
5. *While number of nodes > 2*
6. *If G is connected then Maximum Adjacency Ordering of v1, v2, …, vn of G is created.*
7. *Edge connectivity is calculated as degree of vn.*
8. *Nodes vn-1 and vn are contracted into vn and vn is added back.*
9. *$\lambda(G)$ is updated each time 2 nodes in MAO are contracted. If edge connectivity is smaller, the minimum of both the values is taken.*
10. *Minimum cut is, the minimum connectivity calculated so far.*
11. *Critical edge is calculated using cedges()i.e.,*
12. *Make a random edge e of the graph 0.*
13. *Make the necessary changes and calculate the minimum cut for new graph.*
14. *If $\lambda(G\text{-}e) < \lambda(G)$, update number of critical edges C(G).*
15. *Repeat procedure for all the edges and find total number of critical edges*

CODE IMPLEMENTATION: The program is implemented in C++ on the Windows operating system. The program is created, compiled and tested using Microsoft Visual Studio 2010. The bar graph is generated using Microsoft Office Excel and the network topology is Generated using Graphviz.

The implementation takes as input the number of nodes N(25,50,75) and the number of value of edges m. The experiment was repeated for various values of n and m and edge connectivity ,degree ,critical edges was calculated for each experiment

Multiple method in the source code help realize the goal of the project. These methods with
their purpose are enumerated below.

void main()

*This function gets the value of nodes n and edges m from the user and calls

other methods to realize the goal of this project

void graphgenerator(int, int);

*This function uses the Valus no.of nodes n and edges m to generate the graph.

int nagamochi (int);

*The function is used to do the Nagarmochi Ibaraki Algorithm to find the

minimum cut

void graphgenerator(int, int);

*This Function is used to used to write a dot file to draw the graph using graphviz.

int cedges(int , int);

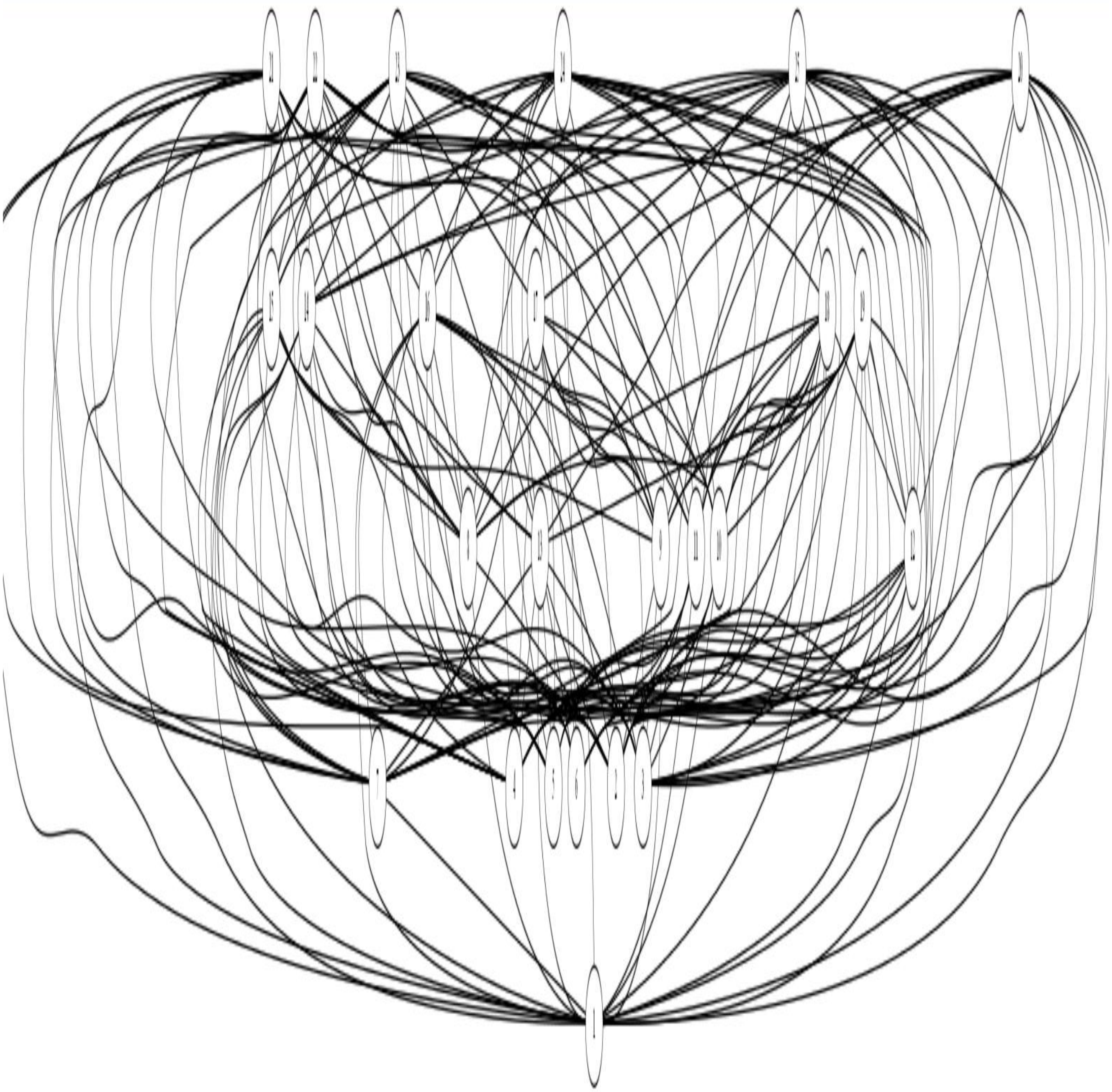
*This function the vale of lambda and no. of nodes and finds the total number of critical edges

***Note**: The Source Code is in the appendix

EXPERIMENTAL RESULTS

SAMPLE GRAPH

FOR NODES:25
EDGES:200



EXPERIMENT I

FOR N=25

S:NO	EDGES M	D	EDGE CONNECTIVITY	CRITICAL EDGES
1.	50	4	2	2
2.	75	6	3	8
3.	100	8	4	8
4.	150	12	7	7
5.	200	16	10	15

EXPERIMENT II

FOR N=50

S:NO	EDGES M	D	EDGE CONNECTIVITY	CRITICAL EDGES
1.	150	6	3	7
2.	400	16	8	11
3.	800	32	20	30
4.	1200	48	42	42

EXPERIMMENT III

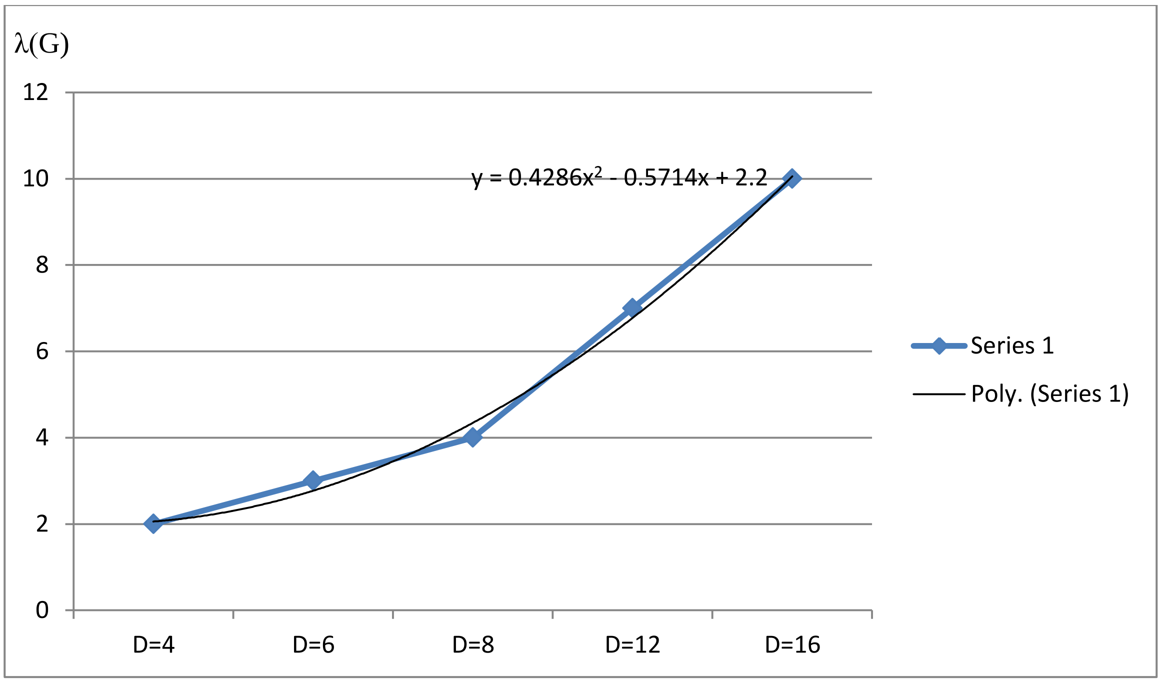
FOR N=75

S:NO	EDGES M	D	EDGE CONNECTIVITY	CRITICAL EDGES
1.	150	4	2	4
2.	675	18	9	9
3.	975	26	14	14
4.	1200	32	18	20
5.	2000	53	35	42

GRAPHS

1) N=25

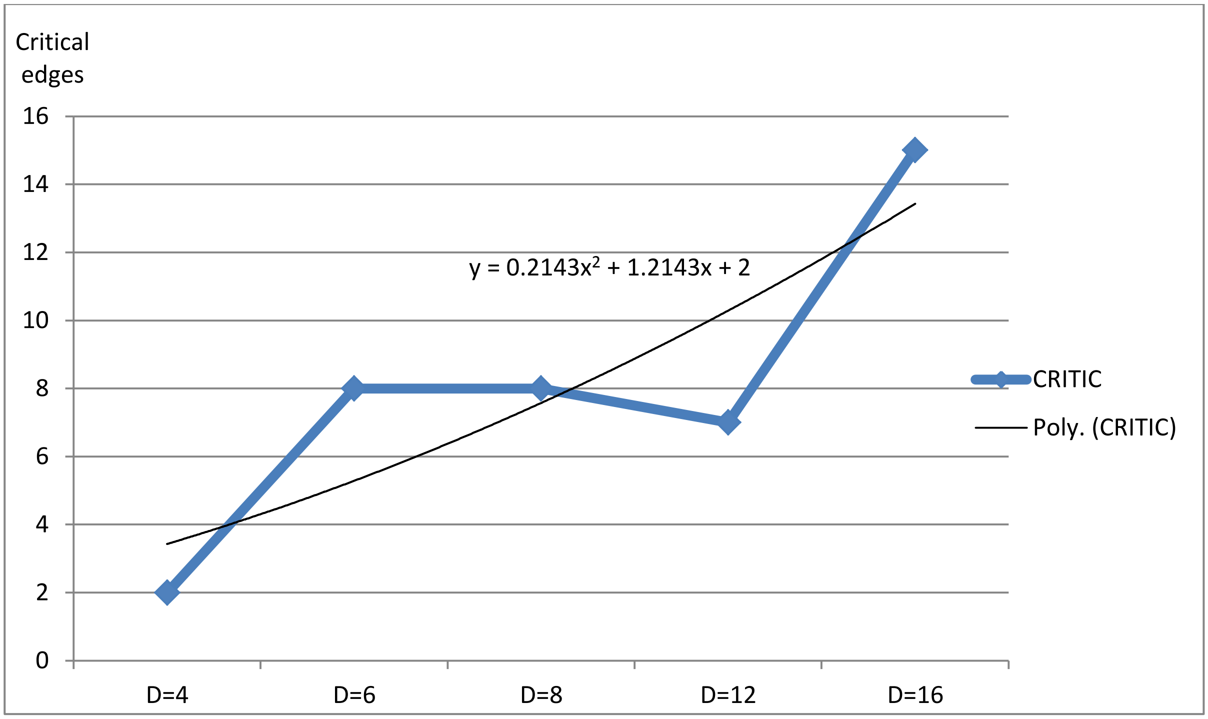
a) λ(G) vs Average degree d



$y = 0.4286x^2 - 0.5714x + 2.2$

where y=λ(G) and x=d

b)Critical Edges C(G) vs Average degree d

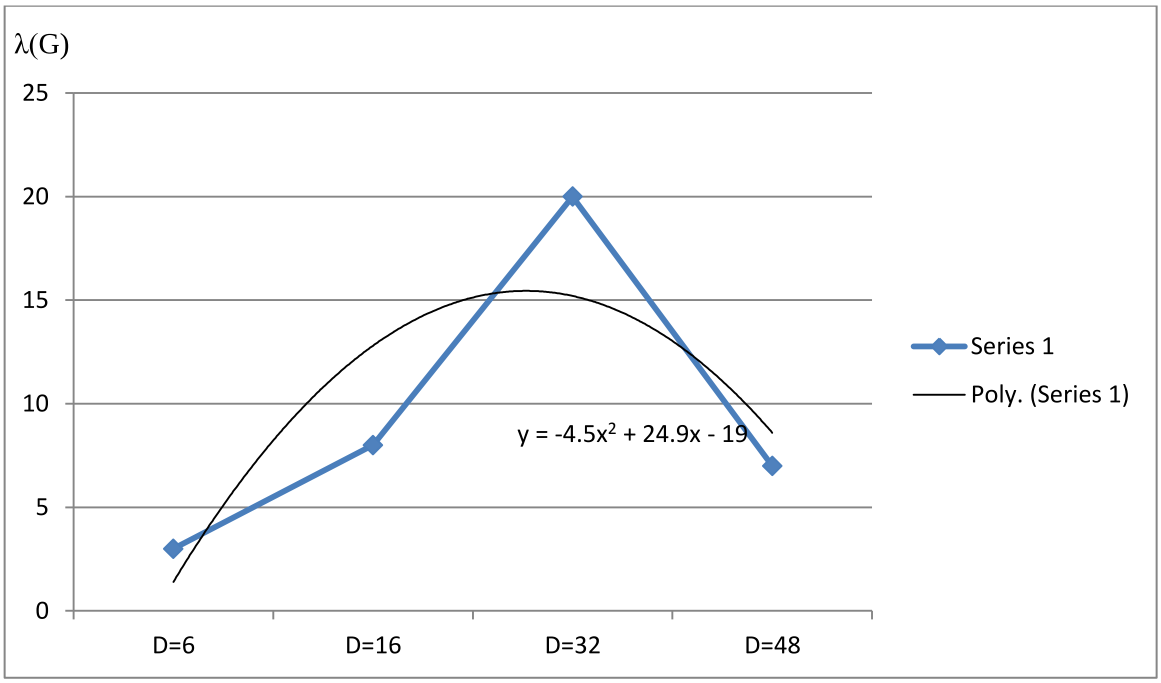


$$\mathbf{y = 0.2143x^2 + 1.2143x + 2}$$

where y=C(G) and x=d

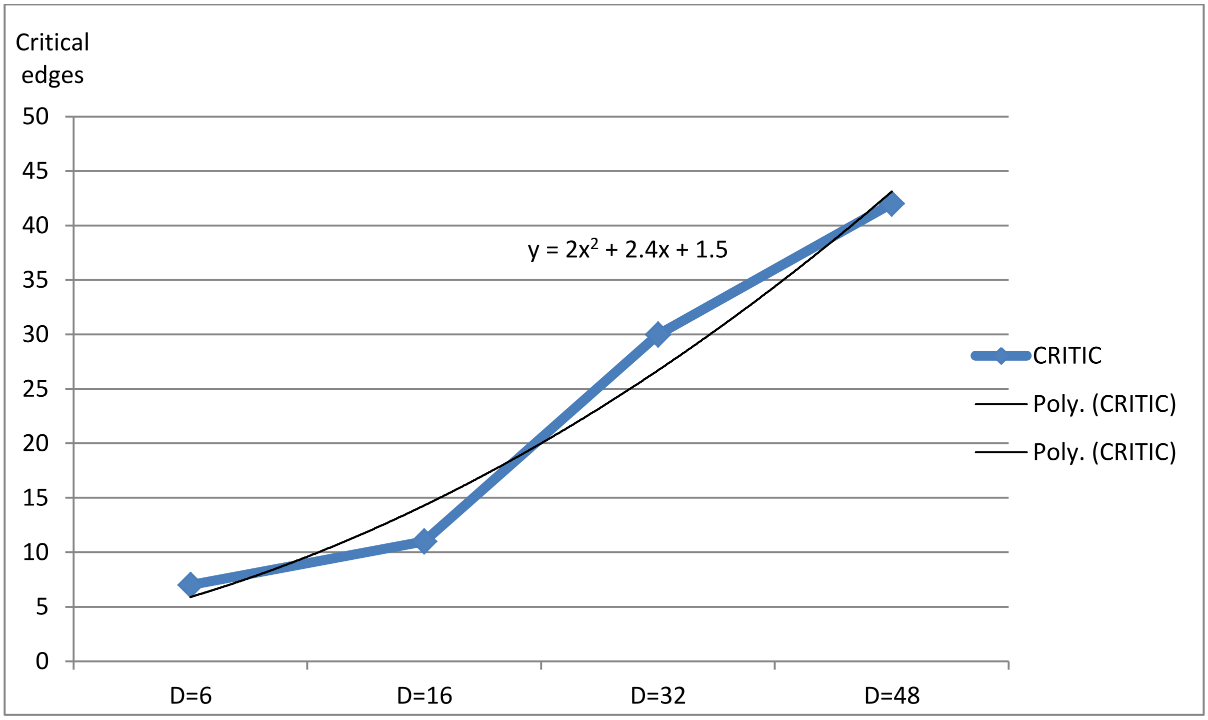
2) N=50

a) λ(G) vs Average degree d



$y = -4.5x^2 + 24.9x - 19$
where y=λ(G) and x=d

b)Critical Edges C(G) vs Average degree d

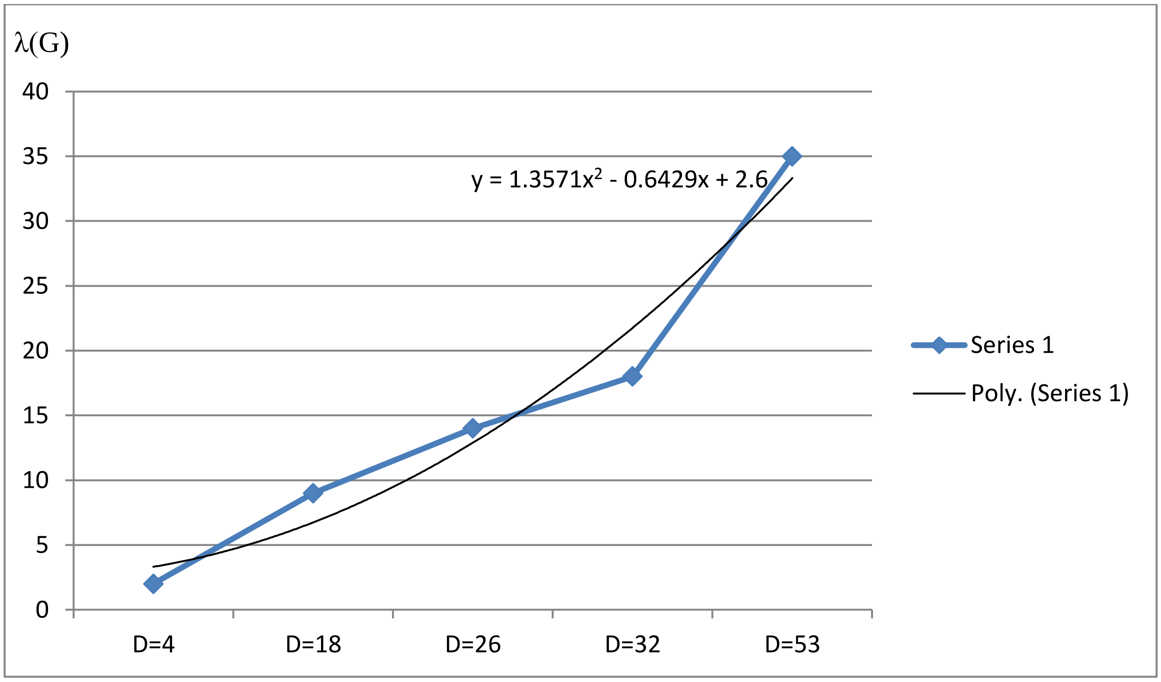


$$\mathbf{y = 2x^2 + 2.4x + 1.5}$$

where y=C(G) and x=d

3) N=75

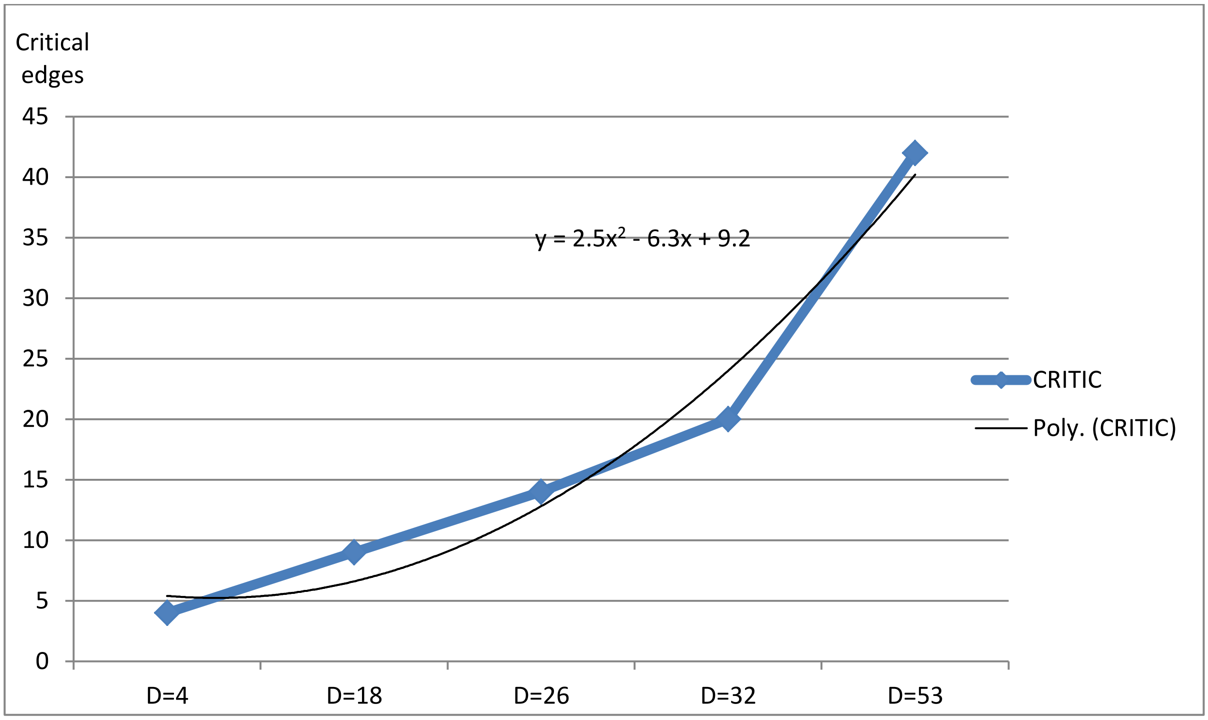
a) λ(G) vs Average degree d



$y = 1.3571x^2 - 0.6429x + 2.6$

where $y=\lambda(G)$ and $x=d$

b)Critical Edges C(G) vs Average degree d



$$\mathbf{y = 2.5x^2 - 6.3x + 9.2}$$

where y=C(G) and x=d

OBSERVATION AND CONCLUSION

- ✓ *For any number of nodes, the degree increases as the number of edges increases.*
- ✓ *For any number of nodes ,the edge connectivity increases as the number of edges increases.*

APPENDIX

```
//*********************ATNPROJECT2****************************
#include<stdio.h>
#include<iostream>
#include<math.h>
#include<conio.h>
#include<stdlib.h>
using namespace std;
void graphgen(int,int);
void graphgenerator(int, int);
int nagamochi (int);
int cedges(int , int);
int adj[100][100];
int nagibamatrix[100][100];
int cpy[100][100];
int edgeexist[100][100];
int d[100];
//__________FUNCTION TO DRAW A GRAPH_____________________
void graphgen(int i, int j)
{
	FILE* graph;
	graph=fopen("graph.dot","w");
	fprintf(graph,"digraph G {\n");
	fprintf(graph,"%d -> %d [dir=none];\n",i,j);
	fprintf(graph,"}");
	fclose(graph);
}
//_______NAGAMOCHI IBARAKI ALGORITHM TO FIND LAMDA_______
int nagamochi(int tot)
{
int max = 0,n=0,l,u,t,lamda=0,start, choosenode, prev1, prev2;
int madj[100];
while (tot > 2 )
{
start = 0;
for (l=0;l<tot;l++)
madj[l]=0;
choosenode = rand()%tot;//choosing a random node
madj[start] = choosenode;
```

```
for (l=1;l<tot;l++)
{
max = 0;
for (u=0;u<tot; u++)
{
if (adj[choosenode][u] > max)
{
max = adj[choosenode][u];
n= u;
}
}
madj[++start]= n;
for (u=0;u<tot;u++)
{
adj[choosenode][u] += adj[n][u];
adj[u][choosenode] += adj[u][n];
adj[choosenode][choosenode] = 0;
}
for (u=0;u<tot;u++)
{
adj[n][u]=adj[u][n]=0;
}
}
if ((!lamda) || (lamda >= d[madj[tot-1]]))
{
lamda = d[madj[tot-1]];
}
prev1 = madj[tot-1];
prev2 = madj[tot-2];
for (l=0; l < tot; l++)
{
nagibamatrix[prev2][l] += nagibamatrix[prev1][l];
nagibamatrix[l][prev2] += nagibamatrix[l][prev1];
nagibamatrix[prev2][prev2] = 0;
nagibamatrix[prev1][l] = nagibamatrix[tot-1][l];
nagibamatrix[l][prev1] = nagibamatrix[l][tot-1];
}
tot = tot-1;
for (l=0; l<tot; l++)
for (t=0; t <tot; t++)
```

```
adj[l][t]=nagibamatrix[l][t];
}
return (lamda);
}
//________________FUNCTION TO CREATE GRAPH____________________
void graphgenerator(int n, int maxedges)
{
int i,j,p,q,x;
int cnt = maxedges;
while(cnt>0)
{
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if ((i == j))
{
adj[i][j] = adj[j][i] = 0;
edgeexist[i][j] = edgeexist[j][i] = 1;
}
else
{
if ((edgeexist[i][j] == 0) && (edgeexist[j][i] == 0))
{
if ((adj[i][j] == 0) && (adj[j][i] == 0))
{
x = (rand()%1);
if (x = 1)
{
adj[i][j] = adj[j][i] = 1;
cnt--;
d[i]=d[i]+1;
d[j]=d[j]+1;
edgeexist[i][j] = edgeexist[j][i] = 1;
graphgen(i,j);
if (cnt <= 0)
break;
}
else
{
```

```
adj[i][j] = adj[j][i] = 0;
edgeexist[i][j] = edgeexist[j][i] = 1;
}
}
}
}
}
if (cnt <= 0)
break;
}
if (cnt)
{
for (p=0;p<n;p++)
for (q=0;q<n;q++)
edgeexist[p][q]= 0;
}
}
}



//_____FUNCTION TO COMPUTE CRITICAL EDGES_______
int cedges(int n, int l){
int cnt = 0,i,j;
int criticalarray[100];
for(i=1;i<n;i++)
{
for(j=1;j<n;j++)
{
if(adj[i][j]==1)
{adj[i][j]=0;
adj[j][i]=0;
criticalarray[i]=d[i];
criticalarray[j]=d[j];
adj[i][j]=1;
adj[j][i]=1;
}
}
}
for(i=0;i<n;i++)
```

```
{
if(nagamochi(d[i])>l)
{
cnt++;
}
}
return cnt;
}

int main()
{
int di,q,lr,h,l,criticaledges,n,p,m,b;
cout<<"\n"<<"\t"<<"\t"<<"*NAGAMOCHI-IBARAKI ALGORITHM*"<<"\n";
cout<<"\n"<<"******ENTER THE NUMBER OF NODES*******:"<<"\n";
cin>>n;
cout<<"\n"<<"******ENTER THE NUMBER OF EDGES********:"<<"\n";
cin>>m;
l = 0;
di = 2*m/n;
for (lr=0;lr<n;lr++)
{
        d[lr]=0;
}
for (p=0;p<n;p++)
for (q=0;q<n;q++)
edgeexist[p][q]= 0;
graphgenerator(n,m);
for (b=0;b<n;b++)
{
for (h=0;h<n;h++)
{
nagibamatrix[b][h]=adj[b][h];
cpy[b][h]=adj[b][h];
}
}
l = nagamochi(n);
criticaledges = cedges(n,l);
cout<<"\n "<<"No. OF EDGES"<<"\t"<<" DEGREE"<<" \t "<<"LAMBDA "<<"\t"<<" CRITICAL EDGES"<<" \n";
cout<<"\n "<<m<<" \t\t "<<di<<"\t\t  "<<l<<"\t\t   "<<criticaledges;
getch();
```

```
return 0;
}
```

REFERENCES:

Learning Modules provided by Dr. Andras Farago.
en.wikipedia.org/wiki/Connectivity_(graph_theory)