"Topic"

Dissertation submitted for award of Degree
in
Master of Science

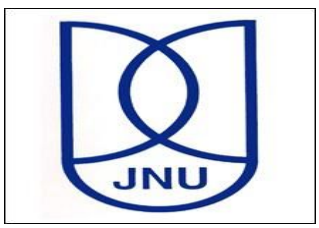
(2015-2017)

Submitted By

Nitin Kumar

(Reg. no . 71230)

JAWAHARLAL NEHRU UNIVERSITY
जवाहरलाल नेहरू विश्वविद्यालय



Under the supervision of
Dr. Andrew M. Lynn

School of Computational and Integrative Sciences
Jawaharlal Nehru University
New Delhi - 110067

Certificate

This is certify that the dissertation project entitled as **"topic"** is carried out by **Nitin kumar** at **School of Computational and Integrative Sciences**, Jawaharlal Nehru University, under my guidance. This is further certified that this dissertation has not been submitted for any other degree to any university.

I recommend that the project be considered for evaluation leading to the award of the M.Sc in Computational biology .

Signature :
Date :

Prof. Andrew M Lynn
(Supervisor & Dean)

School of Computational and
Integrative Sciences
Jawaharlal Nehru University
New Delhi- 110067

Acknowledgment

The completion of this study could not have been possible without my supervisor and Dean **Dr. Andrew M Lynn** . I would like to give my first words of thanks and gratitude to my supervisor for his supervision, advice and sincere guidance from the first day of my project work till now.

I would like to give special thanks to our faculty members who taught me during M.Sc, providing their valuable guidance throughout the course, specially **Dr. Arnab Bhattacharjee** for positive motivation and help in gaining my programming skills and **Dr. Gajendra Pratap Singh** for helping in graph theory concepts .

I would also like to thank my classmates Vishwas, Bodhayan, Jyoti, Sher Singh for their unconditional support and specially **Shruti Gupta** for helping me with programming and everything else.

At last I wish to express my deep sense of gratitude and respect to **my family** for their unrestrained support and encouragement throughout this course.

Nitin Kumar

CONTENT

Title

Aims and Objectives

Development of a tool and a pipeline for network analysis to download data From String database and calculate the pairwise disconnectivity index for evaluating the extremely crucial, non-crucial and moderately crucial nodes/proteins within network .

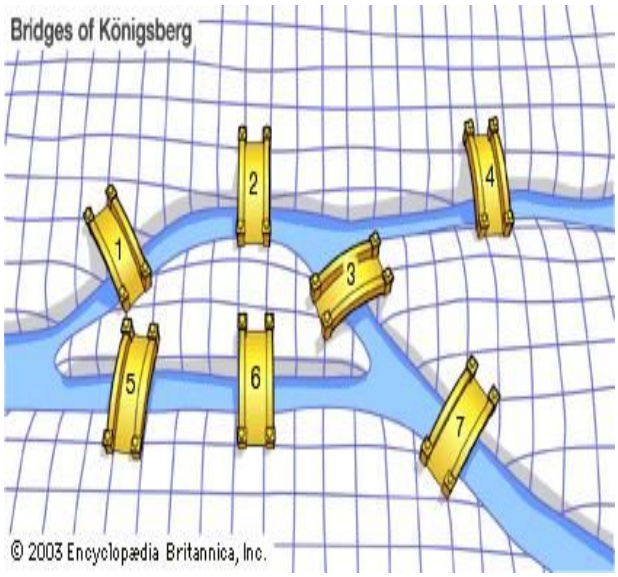
To apply the pairwise disconnectivity index pipeline on *Mycobacterium tuberculosis* from STRING Database species id : 83332 to validate and evaluate the most important,least important protein and moderately important proteins within the network through one by one knocking out each protein out of the network and checking its impact on network .

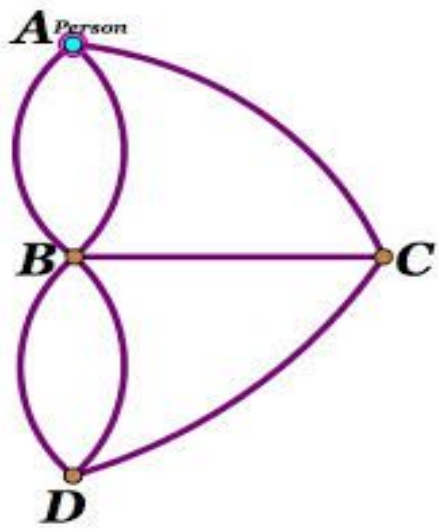
Theory

History and background of graph theory

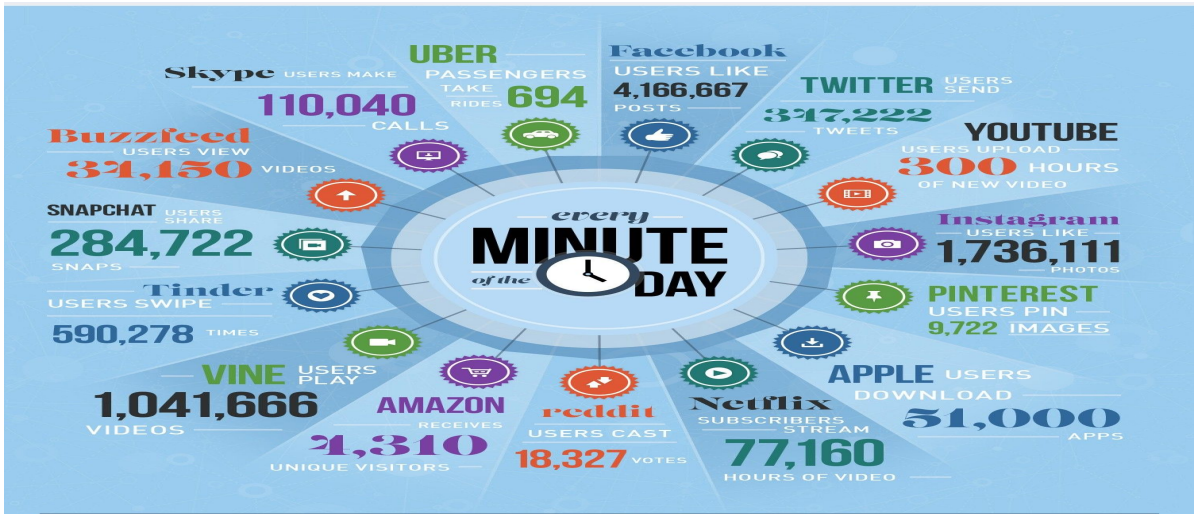
Graph theory is a branch which talks about analysis of structures,Networks .
Graph theoretic models have wide fields applications like in computer science, genomics, data-analytics and many more . Many optimization problems of practical interest can be represented by graphs. There exist several complicated optimization problems which can be solved using graph theoretical concepts because of its inherent simplicity for example :

- Most famous *konigsberg Bridge* problem which was impossible practically but Euler using graph theory gave its solution

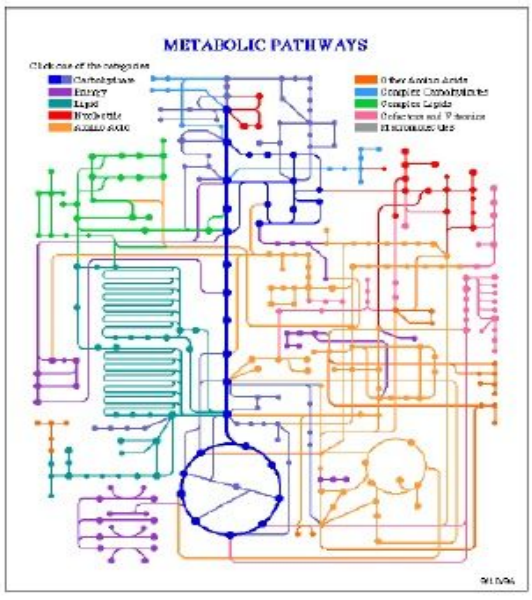




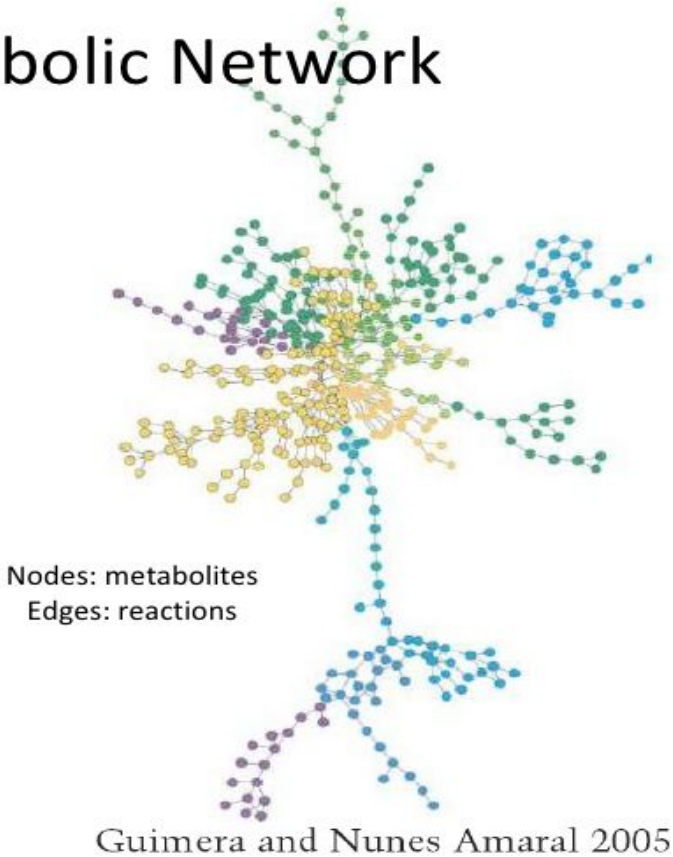
- Our today known social networks are also a example of graph theory where the data are analysed using the graph theory and bulk of data is produced each day which is also stored in form of network



- Biological network are also the best example where we apply graph theory because there exist many biological network at each step from enzyme-enzyme interaction to PPI ,neuronal network networks are everywhere Even after the genome sequencing we use graph theory to assemble the genomic data.



Kegg, Wit, Biocyc, Bigg (UCSD)



Guimera and Nunes Amaral 2005

Basics of graph theory and definitions

A graph is simply ***G = (V,E)*** consist of where V is set of vertices *(v1,v2,v3,.....,vn)* ,Vertices are often sometimes called as nodes in graph and E is the set of edges joining vertices *(e1,e2,e3,....,e4)* some time known as arcs.
Graphs are divided in two major classes

- Directed graph (if the set of edges are ordered hence having directions)

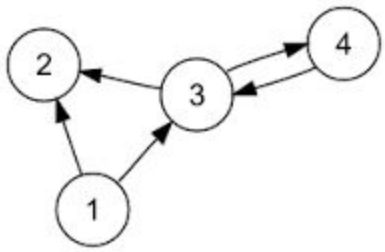


Figure 1

- Undirected graph (if the set of edges are unordered , do not have any directions)

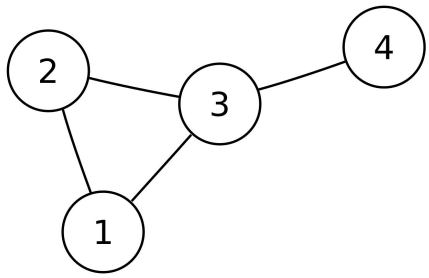


Figure 2

In biological systems we often treat our networks as directed network example Regulatory Network , Metabolic Network are modelled as directed graphs because there is natural direction associated with network but PPI (protein-protein interactions) are modelled as undirected graph hence no direction associated with it

Degree

Degree of node/vertex *v* in graph is number of lines/edges incident on the vertice *v* . let say degree of a node is 2 hence there is two edges incident on the node

So for example degree of node in *Figure 2*
vertex 1 degree 2
vertex 2 degree 2
Vertex 3 degree 3

Vertex 4 degree 1

But in case of directed graph we consider indegree (incoming edges) and out degree (outgoing edges) so figure 1 (indegree : arrow showing inward outgoing arrow showing outward)

Vertex 1 indegree 0 outdegree 2
Vertex 2 indegree 2 outdegree 0
Vertex 3 indegree 2 outdegree 2
Vertex 4 indegree 1 outdegree 1

Path

Path of a graph is the sequence of edges which connect any node *vi* to node *vj .* the *geodesic distance* in a graph is the distance from *vi* to *vj* is the shortest distance between two pair of nodes and if vi and vj are disconnected it will be infinity else a finite value

Example in *Figure 2* if *vi = 1* and *vj = 4* the path length is 2 (1 -> 2 -> 4)

Average path length

Average path length in network defined as the average number of steps along the geodesic path/shortest path for all possible pairs of network nodes . It tells about the efficiency of information or mass transport on a network.

Example Average path length in *Figure 2* is 1.333

Diameter of graph

Longest distance between any pair of vertices. To find calculate the **diameter of a graph**, first find the shortest path between each pair of vertices. The longest length among the shortest path is the **diameter** of the **graph**

For example diameter of the *figure 2* is 2 longest among all shortest paths

Clustering coefficient

A clustering coefficient tells about the degree to which extent nodes in a graph tend to cluster together . clustering coefficient for a node *vi* of degree *k* is given by where e is the number of edges between immediate neighbours

$$c(vi) = 2e/k(k - 1)$$

Degree distribution

It is the distribution of degrees over all network it can be used to infer the kind of network example in a random network the degree distribution is gaussian function .it also helps to know how many no of nodes have high degree and how many are least connected

Scale free network follow power law in their degree distribution and most of the biological network are scale free network

Shortest path

Shortest path is the least number of nodes or least sum of weights in weighted graph between any pair of nodes

Example in *Figure 2* to reach node 1 to node 3 one can take (1 -> 2 -> 3) or (1 -> 3) where the second one will be a shortest path between pair of nodes

Neighbours

Neighbour of the vertex are the immediate nodes that are connected to any node

Example in *Figure 2* neighbour of node 2 are node 1 and node 3

Functional motifs

Functional motifs are subgraphs of a graph that repeat themselves in a network or among various networks. Each of these sub-graphs, characterised by a particular pattern or interactions between vertices, may reflect a framework in which particular functions are achieved . motifs are of notable importance largely because they sometime may reflect functional properties of a network

Modularity

It is the measure the strength to divide network into modules (clusters or communities). Networks with high modularity have dense connections between the nodes within its modules but weak connections with nodes in different modules.
it is often used to detect community structure in networks. However, it has been shown that modularity suffers a resolution limit and, therefore, it is unable to detect small communities. Biological networks exhibit high degree of modularity within them

Centrality

It helps to identify the most important vertices in a network is .used for analysis of network

- Degree centrality

Can be interpreted as the immediate risk of a node for catching something is flowing through the network . In the directed network , indegree and outdegree are taken into consideration.

- Closeness centrality

The average length of the shortest path between the node and the all of other nodes in the network. Thus the more central a node is, the closer it is to all other nodes.

- Betweenness centrality

Measure of a vertex within a graph . Betweenness centrality quantifies the number of times a node or vertex acts as a bridge between the shortest path between two other nodes in a graph

Giant component

A connected component have set of nodes where each node is connected to each noda via an edge between them no node is connected outside the connected component so the largest component among all existing component in a network is termed as Giant component

Biological network

graph theory have provided new view on the topological design of the real-world networks . biological networks exhibit small-world properties: They are surprisingly compact their diameter is too small and increasing clustering features also show a scale-free topology and follow the power-law type of the degree distribution: most of the components exhibit one or two connections with other nodes , but few are involved in dozens and function as hubs, so they provide networks with high robustness against random failures and removal of nodes

Graph-theoretical characteristics have been explored for the analysis of biological networks There is a great need for approaches that are capable of quantitatively evaluating the importance of individual components in biological systems. Centrality provides a valuable method for the structural analysis of biological networks. It allows to identify key elements within networks . approaches such as the degree of a vertex help to find important node which directly control many other molecules/genes, but it fail to tell about the key regulators which are capable of affecting other molecules/genes in an indirect fashion. Parameters as closeness and betweenness centrality, considers both local and distant connections in a network .

Pairwise Disconnectivity index

The pairwise disconnectivity index tells how crucial an individual element is for sustaining the communication between connected pairs of vertices in a network that is displayed in a directed graph or network . That be a vertex (atom, molecules, genes, enzyme), an edge (reactions, interactions, reaction direction), as well can be as a group of vertices and/or edges. The index can be seen as a measure of topological importance or of no use in the regulatory paths which connect different parts of any network and as a measure of sensitivity of this network to the presence (absence) of each individual element. It is been applied to the analysis of several regulatory networks from various organisms and interactions. The importance of an individual vertex or edge for the coherence of the network is determined by the particular position of the given element in the whole network.

It helps to determine the effect on removing each element that can be vertex or edge from a network. The most valuable thing about this approach is its ability in systematically

analyzing the role of every element, as well as groups of elements, in a regulatory or biological network.

***pairwise disconnectivity index of vertex v* is *Dis*(*v*) is the fraction of those initially connected pairs of vertices in a network which becomes disconnected if vertex *v* is removed from the network**

$$Dis(v) = (No - Nv)/No$$

Where *No* is number of ordered pairs in the network that are at least joined by a single path *Nv* is the numbers of ordered pairs left after removal of the vertex *V* . *No* > 0 because there there exist at least one edge between the pair of nodes .

communication mainly depends on vertex *v*. In case the removal of vertex *v* destroys all communication in a network resulting in ***Dis*(*v*) = 1** then its a crucial node for network . In contrast, *Dis*(*v*) = 0 refers to a non-crucial vertex which is not connected to any other vertex in a network.Vertex removal is a strong measure in a network because it one by one knocks out all incoming and outgoing edges of that vertex.

Also we can think about the network by selectively knocking out a particular edge. in a regulatory network when all components are yet present, but due to a mutation in one of it some of the reactions are specifically disabled or interrupted while others are still working. That is important while considering the fact that edges are evenly important and one must look at the behavior or the network after knocking out the edge

***pairwise disconnectivity index of an edge*, *Dis*(e) is**

$$Dis(e) = (No - Ne)/No$$

also N_0 is the number of ordered pairs of vertices connected by at least one of directed path in the graph. N_e is the number of pairs left removing edge *e* from the network. It also ranges between 0 ≤ *Dis*(e) ≤ 1.

All of major functional breakdowns of a network can't be explained as due to failure of one single element either vertex or edge , but the dysfunction of subset of vertices or edges may malfunctioning of this subset may disrupt a significant number of communication within network because paths may be destroyed .

Why *pairwise disconnectivity index* not *centrality*

In the Analysis of regulatory networks there are two major disadvantages of betweenness centrality.

- Shortest paths are supposed to be the most important ones, which is a misleading. The importance of a path is not because of length, but rather by the integral efficiency of it . Longer paths can also be faster and more efficient than shorter paths. For instance, in networks, the initiation of transcription and translation is achieved specific factors. This increases the length of the corresponding paths, but drastically improves the efficiency and specificity of these processes.

- Betweenness centrality can only be applied to vertices that are between other ones. So the vertices having either zero incoming or outgoing degree, are not considered in it . This excludes many extracellular ligands, receptors, target molecules and genes from the analysis .those components, however, directly respond to input-output functionality of the network and are of key significance. betweenness centrality, that are attributed with zero values fail to reflect the individual connectedness of such input/output elements within the whole network.

Therefore concept of the pairwise disconnectivity index as new topological metriccan evaluate alternative though longer paths also can be used to characterize the topological significance of all individual nodes in biological regulatory network. This might be similar to to *vertex-connectivity* or *edge-connectivity* used in graph theory .but does not only focuses on it

Database

STRING Database

STRING (http://www.string-db.org) is a the database of known and predicted biological protein-protein interactions. These interactions can be direct (physical) and indirect (functional) associations between the nodes .Information is contained from various sources that's including experimental repositories , computational prediction methods and public text collections. Each interaction is associated to a combined confidence score that accounts for many useful information . Database currently cover 9'643'763 proteins from 2'031 organisms.Interactions in database are derived from five main sources:

1. Genomics context prediction
2. High-throughput lab experiment
3. Conserved co-expression
4. Automated Text-Mining
5. Previous Knowledge in Databases

Scoring In STRING, each protein-protein interaction is annotated with one or more 'scores'. Importantly, these scores do **not** indicate the strength or the specificity of the interaction. Instead, they are indicators of **confidence**, i.e. how likely STRING judges an interaction to be true, given the available evidence. All scores rank from 0 to 1, with 1 being the highest possible confidence. A score of 0.5 would indicate that roughly every second interaction might a false positive.

Materials and Methods

Python

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.
Currently available as python 2.7 and python 3 .freely publicly available under osl (open source licensing) one can download and install it
Here scripts scripts which are written all are in python and compatible with syntax of both versions of python 2.7 and python 3

R

R is an open source programming language for statistical computing and graphics **.**Freely publicly available under osl (open source licensing) one can download and install it
Script which is used to download data from string database is coded in R .

Python-igraph

Igraph is a library of collection for creating and manipulating graphs and analyzing networks. Which offers many in built functions function for analysis and data parsing in many formats .
It is written in C and also used with Python and R package
Most of the scripts are written with python igraph here in this work

Matplotlib

Matplotlib is a Python 2D plotting library it produces publication quality figures in a variety of formats and interactive environments across platforms. Matplotlib can be used in Python scripts. For plotting matplotlib is used

Bioconductor

Bioconductor is an open source, open development software project that provide tools for the analysis and comprehension of high-throughput genomic data. It is based on the R programming language. A large number of metadata packages provide pathway, organism, microarray and other annotations.The Bioconductor project started in 2001 and is overseen by a core team, based primarily at Roswell Park Cancer Institute.

STRING db - Bioconductor

STRINGdb R package facilitate users in accessing the STRING database from R.

One can download the string data using this module for any specific organism also it offers many functionality along with data download from STRING Database it also comes with many builtin function .

Methodology

Installation of packages

Python- Igraph

One can download the package form http://igraph.org/python/ from their official website
Also in a unix , linux machine one can access their application terminal or if ubuntu/debian can be done by

$ sudo apt-get install python-igraph

Igraph can be installed using pip or easy_instal for python :

$ pip install python-igraph

For offline installation after downloading package from
https://github.com/igraph/python-igraph can do on terminal

$ sudo python setup.py install

Matplotlib

On unix linux machine on can simply do
Debian /ubuntu

$sudo apt-get install python-matplotlib

Fedora/Redhat

$ sudo yum install python-matplotlib

can be installed using pip or easy_instal for python :

$ python -m pip install matplotlib

For offline installation from file https://matplotlib.org/faq/installing_faq.html#install-from-git can go to this link and

> git clone git@github.com:matplotlib/matplotlib.git
> cd matplotlib
> python setup.py install

STRING db - Bioconductor

One can access their official website to download zip file and refer to installation manual
http://bioconductor.org/packages/release/bioc/html/STRINGdb.html

$ R
>source("https://bioconductor.org/biocLite.R")
biocLite("STRINGdb")

Steps :

- First the bioconductor stringdb library r was installed in machine which was about to be used for downloading data from string database .

- A script was developed using string db in r using string db library and data for ***mycobacterium tuberculosis*** was downloaded using the script (specie id : 83332)

- Data comes into two zipped files as '83332__proteins_link.tsv.gz' , '83332__protein.tsv.gz' where both were unzipped using **$ gunzip -k filename** where the 1st one is the interaction network and the later one is an annotation file .

- We wanted to convert the interaction network into a weighted edgelist format therefore an another script to parse the string data and convert it into a weighted edgelist format was written . where 1st two columns are nodes and 3rd column is score which is have used was used as edge weights

- To evaluate the pairwise disconnectivity script was developed to calculate the index where it takes edgelist as input which we have got in last step so to evaluate pairwise disconnectivity we removed one node

  $$(\textit{total no. of pairs in edgelist} - \textit{pairs left after removal of node})/\textit{total no. of pairs}$$

- So it gives the index value between 0 and 1 this was done iteratively and one by one each node is removed and index was calculated to check the impact on network .nodes on whose removal index value was 1 assigned as important and crucial nodes and nodes whose removal gave index value 0 were assigned as non crucial nodes remaining nodes which fall in between values were assigned as moderately important nodes .

- These were stored in a file with 1st column as index value and second column as the nodes whose removal caused it .

- This result file was mapped onto the annotation file using another developed script which returned the names nodes and their functions also their preferred name from the annotation file so now we know which node having major impact on network and in biological system of ***mycobacterium tuberculosis*** **.**

- But the result were not satisfactory using umber of ordered pairs may be because of highly connectedness of the biological network so another script was developed similar for calculating the disconnectivity index but this time using disjoint paths between nodes .

  a= *disjoint paths between all pairs of node*
  b= *disjoint paths left after removal of a node*

$$(a - b)/a$$

- This was done for all pairs of nodes and the index are calculated and stored in a file with 1st column as the index next the corresponding node whose deletion caused it .

- Similarly it was also mapped to distinguish between important ,least important and moderately important nodes onto the annotation file and now we have the result where we can say these number of nodes are important for the network

- Later the edgelist generated were further taken for more downstream analysis using the tool.py for evaluation of other other parameters (mentioned under scripts)

which of the target proteins in the ***mycobacterium tuberculosis*** can be suitable drug designing or essential for it to survive .

To automate the whole process and increase the efficiency a bash script was developed Which convert the whole process into a pipe line

Pipeline

A bash script was written to pipeline the whole process . ***" $ x.sh"*** script was converted into a executable file using ***$ chmod +x x.sh*** .

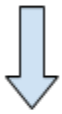
-------->>>>>>>>>>>>>>>>>>>>>> $ ./x.sh <<<<<<<<<<<<<<<<<<<<<<----------

Makes a "out" directory to save all result and files

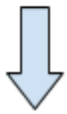
"string_data_download .r " this download data from takes species id as input from user

Zip files were extracted

"stringdb_parse.py " parses Interaction file from the data and converts it into a weighted edgelist using and scores were used as edge weight



"Pairwise_dis_index.py" calculated the pairwise wise disconnectivity index by taking edgelist as input and output file with index and node which was deleted



"Pairwise_discon_vdisjoint_paths.py" evaluated the index using disjoint paths takes edgelist as the input and returns the output file as index and node which was deleted

"Mapping_on_annotation_file.py" maps node onto annotation file and tells which node id corresponds to which protein in network. Returns the file 3 output file Result_crucialnodes.txt ,Result_noncrucialnodes ,Result_moderaltycrucialnodes.txt

String _data_download.r

```
library(STRINGdb)
get_STRING_species(version="10", species_name=NULL)
cat("Enter ID: ")
a <- readLines(con="stdin", 1)
cat(a,"\n")
a <- as.numeric(a)
#n <- as.integer(readline(prompt = "Enter an species id : "))
string_db <- STRINGdb$new( version="10", species=a  ,score_threshold=0,
input_directory="./out" )
string_db
```

This imports the stringdb library and prints the species id list and then allow user to give input the species whose data they want to download and saves it into a “out” directory

$ gunzip -k *.gz

This will unzip the gz files downloaded from the string database .

Stringdb_parse.py

```
import sys
foin =open(sys.argv[1] , 'r')
fout = open(sys.argv[2],'w')
lines = foin.readlines()[1:]
for line in lines:
    word = line.split()
    fout.write(str(word[0])+'\t'+str(word[1])+'\t'+str(word[-1])+'\n')
```

This parses the string data into a weighted edgelist format where the scores are used weights . 1st it takes the string data as an 1st argument splits it on the base tab and and writes the 1st 2nd and last columns into a output file .

Pairwise_dis_index.py

```
from igraph import *
import sys

print ('usage: inputfile output_filename')

fout=open(sys.argv[2],'w')
```

```
g_s=Graph.Read_Ncol(sys.argv[1],   names=True,   weights="if_present"   ,
directed=True)
print ('No. of nodes in given file ', len(g_s.vs))

pair=[]
crucial=0
noncrucial=0
counter =0
for i in range(len(g_s.vs)):
      counter +=1
      g = g_s.copy()
      a= len(g.es)
      dlt=g.delete_vertices(i)
      b= len(g.es)
      c= float(a-b)/float(a)
      pair.append(c)
      fout.write(str(c)+"\t"+str(counter)+'\n')
      if c == 1:
      crucial +=1
      if c == 0:
      noncrucial +=1
print "crucial nodes" ,crucial
print "non crucial nodes " ,noncrucial
```

This takes the edgelist and reads it as graph using igraph library and delete each node one by one from the graph . number of edges are number of ordered pairs having at least one edge among them then iteratively one by one knocking out each node and then evaluating the index this stores it into a file with index and the deleted node id .
If the index == 1 it marks it as a crucial node and if index == 0 puts them into non crucial node and at last print the number of crucial and non crucial nodes .

Pairwise_discon_vdisjoint_paths.py

```
import matplotlib.pyplot as plt
from igraph import *
import sys

print ('usage: inputfile output_filename')

n=[]
n_removed=[]

fout=open(sys.argv[2],'w')

g=Graph.Read_Ncol(sys.argv[1],    names=True,    weights="if_present"    ,
```

```
directed=True)
print ('No. of nodes in given file ', len(g.vs))
vs=[]
for i in range(len(g.vs)):
      vs.append(i)

for i in range(len(vs)-1):
      start=vs[i]
      stop =vs[i+1]
      no=g.vertex_disjoint_paths(source=start,  target=stop,  checks=True,
neighbors="ignore")
      n.append(no)

i = 0
deleted =[]
while i == 0:
      st=0
      sp=1
      deleted.append(((str(g.vs[1]).split())[-1])[1:-3])
      dlt=g.delete_vertices(st)
      if len(g.vs) == 1:
      break
      n1=g.vertex_disjoint_paths(source=st,    target=sp,    checks=True,
neighbors="ignore")
      n_removed.append(n1)

pairwise = []
crucial=0
noncrucial=0
for i in range(len(n)-1):
      if n[i] > 0:
      index=(n[i]-n_removed[i])/float(n[i])
      fout.write(str(index)+'\t'+str( deleted[i])+'\n')
      #print index , deleted[i]
      pairwise.append(index)
      if index == 1:
            crucial +=1
      if index == 0:
            noncrucial +=1
print "crucial nodes" ,crucial
print "non crucial nodes " ,noncrucial
```

This takes the edgelist and reads it as graph using igraph library and evaluates the disjoint paths between all pairs of nodes and delete each node one by one from the graph and evaluates the number of disjoint paths this stores it into a file with index and the deleted node id . If the index == 1 it marks it as a crucial node and if index == 0 puts them into non

crucial node and at last print the number of crucial and non crucial nodes .this also makes sure that the number of disjoint path are more than 1

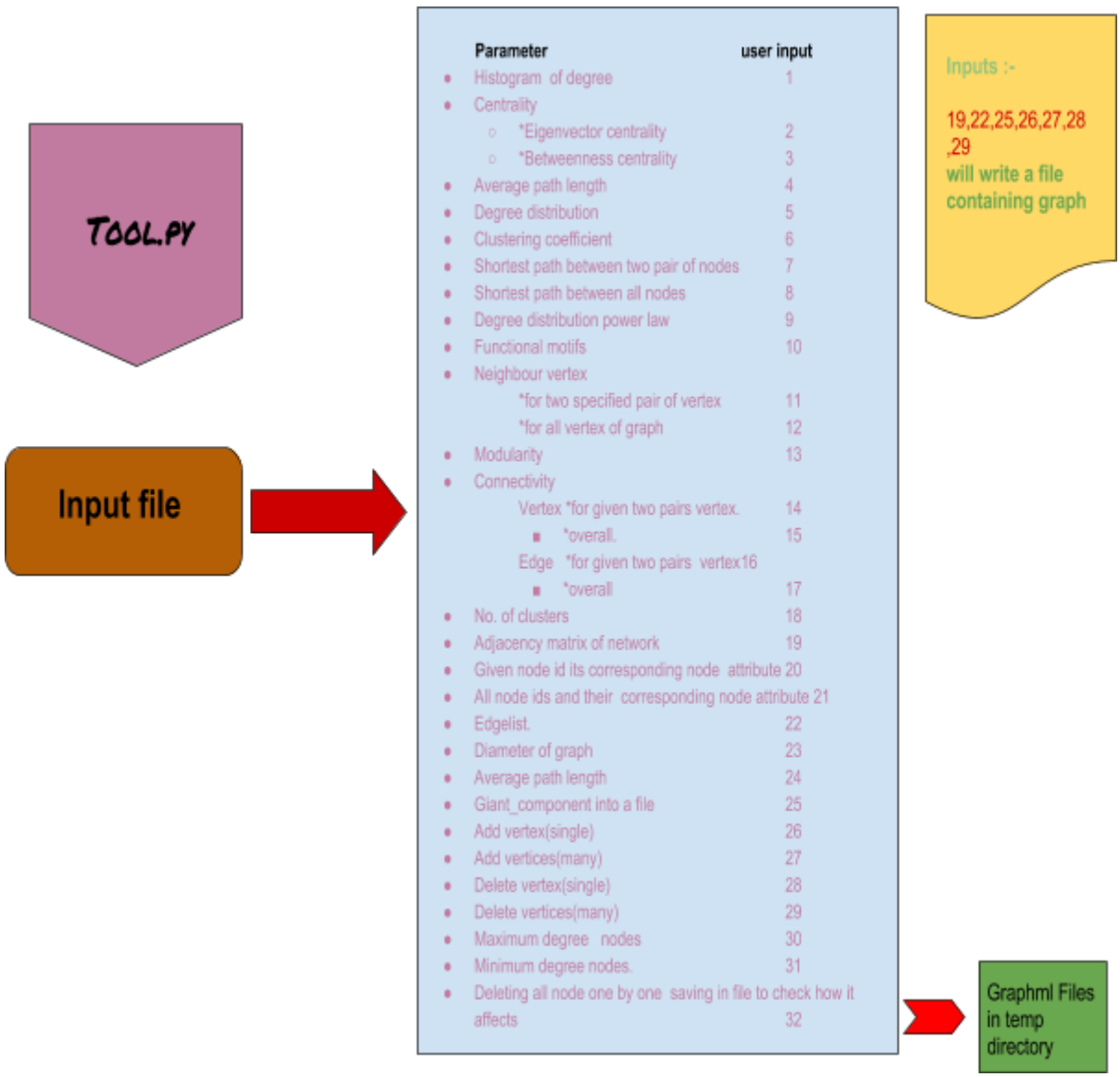
mapping _on_annotation_file.py

```
import sys
file1 = open(sys.argv[1],'r')
file2 = open(sys.argv[2],'r')
result1= open('Result_crucialnodes.txt','w')
result2= open('Result_noncrucialnodes.txt','w')
result3 =open('Result_moderatenodes.txt','w')

for line1 in file1:
      word1 = line1.split()
      file2.seek(0)
      for line2 in file2:
      word2 = line2.split()
      if word2[1] == word1[1]:
            if word1[0] == '1.0':

result1.write(word1[0]+'\t'+word1[1]+'\t'+word2[1]+'\t'+word2[5]+'\t'+wo
rd2[-1]+'\n')
            if word1[0] == '0.0':

result2.write(word1[0]+'\t'+word1[1]+'\t'+word2[1]+'\t'+word2[5]+'\t'+wo
rd2[-1]+'\n')
            if word1[0] > '0.0' and word1[0] < '1.0' :

result3.write(word1[0]+'\t'+word1[1]+'\t'+word2[1]+'\t'+word2[5]+'\t'+wo
rd2[-1]+'\n')
```

This takes result file from the previous step file containing the index value and node id and splits the file on base of tab and matches the value of column 2 on the the annotation file and if match is true it writes out the preferred name of the node function of the protein and index value into irrespective files where 1st file will be crucial nodes and all element in 1st column should be equal to 1.0 .second file where the 1st column should be 0.0 file of non crucial nodes .and last file where values between 0 and 1 will be placed and annotated as moderately crucial nodes

Network analysis Tool

A python script was developed using python-igraph library which can be used as network analysis tool script offers evaluation of many parameters and reads the user friendly **“Tool.py” .** it reads the file containing network in many different formats as “Graphml file “ , “Adjacency matrix” , “ Edgelist format “ , ” Weighted Edgelist “ , “ Lgl format “ also generates a Random network to benchmark your data and compare it against random dataset



```
import matplotlib.pyplot as plt
from igraph import *
import sys
import os

#for plotting tool and file must be in a same location


l = os.listdir(os.getcwd())
if 'temp' not in l:
    os.mkdir('temp')
else:
    pass


if sys.argv[1] == True or  '-h' or '-H' or '--help' :
    print ('    usage -format [filename]'+'\n')
      print ('      format: adjacency matrix -adj   edgelist -edgelist
graphml -graphml  lgl -lgl  random network - random' +'\n' )
else:
    sys.argv[1] == None or False
    print ('    usage -format [filename]')


if sys.argv[1] == '-adj':
    g=Graph.Read_Adjacency(sys.argv[2])
    print ('No. of nodes in given file ', len(g.vs))
    layout = g.layout('kk')
    plot(g,layout = layout)


elif sys.argv[1] == '-edgelist':
       g=Graph.Read_Ncol(sys.argv[2],  names=True,  weights="if_present",
directed=True)
    print ('No. of nodes in given file ', len(g.vs))
    c = [ ]
    for i in range(len(g.vs)):
       c.append(str(i))
    g.vs['label'] = c
    layout = g.layout('kk')
    plot(g,layout = layout)


elif sys.argv[1] == '-graphml':
```

```
    g=Graph.Read_GraphML(sys.argv[2])
    print ('No. of nodes in given file ', len(g.vs))
    layout = g.layout('kk')
    plot(g ,layout = layout)


elif sys.argv[1] == '-adj':
    g=Graph.Read_(sys.argv[2])
    print ('No. of nodes in given file ', len(g.vs))
    layout = g.layout_sphere('kk')
    plot(g,layout = layout)


elif sys.argv[1] == '-lgl':
    g=Graph.Read_Lgl,(sys.argv[2])
    print ('No. of nodes in given file ', len(g.vs))
    layout = g.layout('kk')
    plot(g,layout = layout)


elif sys.argv[1] == '-random':
    num = input("Enter the number of nodes ")
    g = Graph.GRG(num,0.1)
    c = [ ]
    for i in range(num):
       c.append(str(i))
    g.vs['label'] = c
    a = [ ]
    a = g.degree()
    layout = g.layout('kk')
    plot(g,target='graph_with_nodes'+str(num)+'.png' , layout = layout)



print ('\n'+'    =========== parameters covered ============')
print                                                          ('
Histogram.....................................................
.(1)')
print         ('                                  Centrality         :
')
print ('      *Eigenvector  centrality.....................................
(2)')
print ('                                                    *Betweenness
centrality.....................................(3)')
print         ('                                  Average          path
length.....................................................(4)')
print            ('                                                  Degree
distribution.....................................................(5)')
```

```
print          ('                                                Clustering
coefficient.....................................................(6)')
print     ('                     Shortest     path     between     two
nodes.............................(7)')
print     ('                     Shortest     path     between     all
nodes..............................(8)')
print      ('                        Degree      distribution      power
law.................................(9)')
print          ('                                                Functional
motifs.....................................................(10)')
print       ('                               Neighbour       vertex       :
.     ...       ')
print ('                *           for           two           specified
vertes.................................(11)')
print ('             *         for          all          vertex          of
graph.....................................(12)')
print                                                                  ('
Modularity.....................................................................
...(13)')
print ('    connectivity : '
.....       )
print ('     vertex *for given two vertex.....(14)')
print ('
*overall.....................................(15)')
print ('     Edge   *for given two vertex.....(16)')
print ('
*overall.....................................(17)')
print          ('                                          No.          of
clusters.....................................................(18)')
print           ('                                                Adjacency
matrix.....................................................(19)')
print   ('           given   node   id   its   EC   no(for   enzyme   data
file)...........(20)')
print   ('           EC   no   for   all   node   ids(for   enzyme   data
file).............(21)')
print                                                                  ('
Edgelist.......................................................................
....(22)')
print                                                                  ('
Diameter.......................................................................
..(23)')
print         ('                                      Average         path
length.....................................(24)')
print                                                                  ('
giant_component.....................................................(25)'
)
print            ('                                                    add
vertex(single).....................................................(26)')
```

```
print            ('                                                    add
vertices(many)..................................................(27)')
print           ('                                                   delete
vertex(single)..................................................(28)')
print       ('                              delete       vertices(many)....
.....................................(29)')
print      ('                            maximum      degree      nodes....
.................................(30)')
print      ('                            minimum      degree      nodes....
.................................(31)')
print    ('               Deleting    all    nodes    saving    in    file
.............................(32)'+'\n')


user = int(input('Type the no of function wanted: '))



if user == 1 :
    b = g.degree_distribution(bin_width=1)
    print ('degree histogram',b)
    a = [ ]
    a = g.degree()
    plt.hist(a, bins=range(0,(max(a))),normed=0, facecolor='b')
    plt.xlabel('Number of Nodes')
    plt.ylabel('Degree')
    plt.title('Degree Distribution')
    plt.show()
    print('figure is saved ')
    plt.savefig("hist.png")


if user == 5 :
    c=[]
    for i in range(len(g.vs)):
       c.append(str(i))
    g.vs['label'] = c
    a = [ ]
    a = g.degree()
    degree=open('degree_distribution.txt','w')
    for deg in range(len(c)):
                           degree.write('degree                      vertex
id'+'\t'+str(a[deg])+'\t'+str(c[deg])+'\n')
       print ('degree distribution (degree, vertex id) file is saved
degree_distribution.txt' )

if user == 7 :
     start = input("Enter the name(which is integer here) of the node
```

```
from which the shortest path is to be calculated : ")
     end = input("Enter the name(which is integer here) of the node to
which the shortest path is to be calculated : ")
      print   ('shortest path', g.get_all_shortest_paths(start, to=end,
weights=None, mode=OUT))
if user == 8 :
     print  ('shortest path', g.shortest_paths(source=None, target=None,
weights=None, mode=ALL))
if user == 6 :
                              print       ('clustering       coefficient',
g.transitivity_undirected(mode='nan'))

if user == 3 :
       startc  =  input("Enter  the  start  node  from  which  betweenness
centrality to be calculated : ")
     endc = input("Enter the end node till which betweenness centrality
to be calculated :  ")
    print ('betweenness centrality', g.betweenness(startc,endc))
if user == 2 :
                                                print           ('eigenvector
centrality',g.eigenvector_centrality(directed=True,scale=True,weights=No
ne,return_eigenvalue=False))
if user ==  9 :

deg=g.Static_Power_Law(len(g.vs),10,exponent_out=2,exponent_in=-1,loops=
False,multiple=False,finite_size_correction=True)
    plot(deg)
    print ('Degree distribution(power law)', deg)
if user == 4 :
                             print       ('average       path       length',
g.average_path_length(directed=False,unconn=True))
if user == 10 :
                  print    ('    no.    of    functional    motif    ',
g.motifs_randesu_no(size=3,cut_prob=None,))
if user == 12 :
                   print    ('neighbourhood    vertice    for    each',
g.neighborhood(vertices=None,order=1,mode=ALL))
if user == 11 :
     neighbors=input('type the vertics to whom neighbour vertex needed:
')
               print   ('neighbour   vertices   to   gien   vertices',
g.neighbors(neighbors,mode=ALL))
if user == 13 :
    membership=a
    print ('modularity', g.modularity(membership,weights=None))
if user == 14 :
    x=input('connectivity from: ')
    y=input('connectivity to : ')
```

```
            print      ('connectivity   from   one   node   to   another',
g.vertex_connectivity(source=x,target=y,checks=True,neighbors="error"))
if user == 15 :
                    print          ('vertex     connectivity     overall',
g.vertex_connectivity(source=-1,target=-1,checks=True,neighbors="error")
)
if user == 16 :
    source=input('source vertex : ')
    target=input('target vertex : ')
                             print       ('edge       connectivity       ',
g.edge_connectivity(source=source,target=target,checks=True))
if user == 17 :
                                print       ('overall       connectivity',
g.edge_connectivity(source=-1,target=-1,checks=True))
if user == 19 :
    g.get_adjacency(type=GET_ADJACENCY_BOTH,eids=True)
    g.write_adjacency('matrix.txt')
    print ('adjacency matrix saved in file ')

if user == 18 :
    print ('no of clusters of the graph', len(g.clusters(mode=STRONG)))
if user == 20 :
    node=input('type the node id : ')
    print (g.vs[node])
if user == 21 :
    for i in range(len(g.vs)):
       print (i, g.vs[i])
if user == 22 :
    edgelist=g.get_adjedgelist(mode=OUT)
    fout=open("edgelist.txt",'w')
    fout.write(str(edgelist))
    print ("edgelist saved")
    fout.close()
if user == 23 :
    print (g.diameter(directed=True, unconn=True, weights=None))

if user == 24 :
    print (g.average_path_length(directed=False,unconn=True))
if user == 25 :
                             print       (g.clusters(       ).giant       (
).write_graphml("gaint_component.graphml"))
    print ('gaint component is saved in graphml file')

if user == 26:
    n=input(' single vertex to be added attribute:  ')
    g.add_vertex(n)
    print (g.write_graphml("new_added_single_node.graphml"))
    print ('new added vertice is saved in graphml file ')
```

```
if user == 27:
    n=input('the number of vertices to be added, :  ')
    g.add_vertices(n)
    g.write_graphml("new_added_many_node.graphml")
    print ('new added vertice is saved in graphml file ')

if user == 28:
    dlt=input('id of vertices to be deleted:  ')
    g.delete_vertices(dlt)
    g.write_graphml("new_deleted__node.graphml")
    print ('Deleted vertex FILE is graphml file ')

if user == 29:
    s =raw_input('ids of vertices to be deleted saprated by space:  ')
    dlt_list=list(map(int, s.split()))
    g.delete_vertices(dlt_list)
    g.write_graphml("new_deleted__node.graphml")
    print ('Deleted vertex FILE is graphml file ')


if user == 30:
    a=g.degree()
    print ('node with max degree' , max(a), g.vs(max(a)))

if user == 31:
    a=g.degree()
    print ('node with max degree', min(a), g.vs(max(a)))

if user == 32:
    num=0
    j=0
    while num == 0:
       g.delete_vertices(0)
       if len(g.vs) ==1:
             break

g.write_edgelist(str(os.getcwd())+"/temp/deleted_node_"+str(j)+".txt")
       print ( 'files complete and saved after deleting node :- ' ,
str(j))
       j +=1
```

$ python tool.py -h **#**Print the usage of the tool .
Input files are passed as the system argument from where it reads the graph and stores into its memory and plots it , later it prints the various option parameters and pruning method by which one can analyse and calculate the network also every plots it saves into the current directory For option 32 where one tries to delete the node one by one in a network for further

analysis it saves into a temp folder . parameters are given in form of list user is just required to input the function name .
Results were later analysed based on their index value and role in Network .

Validation across species

Pipeline was used to analyse followed datasets also :

Species ID	**ORGANISM**
83332	*Mycobacterium tuberculosis H37Rv*
405566	*Lactobacillus helveticus DPC 4571*
941770	*Lactobacillus fructivorans KCTC 3543*
1050720	*Agrobacterium tumefaciens F2*
1129369	*Mycoplasma hyorhinis*
1069533	*Streptococcus infantarius*

Results and Discussion

The study was carried out and specie Id. 83332 , ***Mycobacterium tuberculosis H37Rv***
Was analysed using the pairwise disconnectivity pipeline which returned

Total number of nodes in network : 3967

Crucial nodes	Index value 1.0	1422
Moderate nodes	Index value 0 > 1.0	712
Non crucial nodes	Index value 0.0	409

Also mapped against annotation file resulting into followed results

```
1.0    83332.Rv0553    83332.Rv0553    O-succinylbenzoate    menC
1.0    83332.Rv0697    83332.Rv0697    dehydrogenase    MT0724
1.0    83332.Rv0778    83332.Rv0778    cytochrome    cyp126
1.0    83332.Rv0859    83332.Rv0859    acetyl-CoA    fadA
1.0    83332.Rv0974c    83332.Rv0974c    acetyl-/propionyl-CoA    accD2
1.0    83332.Rv1123c    83332.Rv1123c    peroxidase    bpoB
1.0    83332.Rv1140    83332.Rv1140    hypothetical    MT1173
1.0    83332.Rv1144    83332.Rv1144    short-chain    Rv1144
1.0    83332.Rv1248c    83332.Rv1248c    alpha-ketoglutarate    kgd
1.0    83332.Rv1279    83332.Rv1279    dehydrogenase    Rv1279
1.0    83332.Rv1350    83332.Rv1350    3-ketoacyl-ACP    fabG2
1.0    83332.Rv1484    83332.Rv1484    enoyl-ACP    inhA
1.0    83332.Rv1533    83332.Rv1533    hypothetical    Rv1533
1.0    83332.Rv1715    83332.Rv1715    3-hydroxybutyryl-CoA    fadB3
```

Figure : Truncated view of crucial node result (columns showing index value ,deleted protein and corresponding annotated protein respectively)

```
0.964912280702    83332.Rv0551c    83332.Rv0551c    acyl-CoA    fadD8
0.916666666667    83332.Rv0554    83332.Rv0554    bromide    bpoC
0.5    83332.Rv0562    83332.Rv0562    polyprenyl    grcC1
0.833333333333    83332.Rv0636    83332.Rv0636    (3R)-hydroxyacyl-ACP    hadB
0.75    83332.Rv0904c    83332.Rv0904c    acetyl-CoA    accD3
0.857142857143    83332.Rv1013    83332.Rv1013    long-chain-fatty-acid--CoA    pks16
0.5    83332.Rv1070c    83332.Rv1070c    enoyl-CoA    echA8
0.8    83332.Rv1106c    83332.Rv1106c    cholesterol    Rv1106c
0.833333333333    83332.Rv1193    83332.Rv1193    acyl-CoA    fadD36
```

0.5	83332.Rv1427c	83332.Rv1427c	acyl-CoA	fadD12
0.6	83332.Rv1472	83332.Rv1472	enoyl-CoA	echA12
0.5	83332.Rv1483	83332.Rv1483	3-oxoacyl-ACP	fabG
0.666666666667	83332.Rv1521	83332.Rv1521	acyl-CoA	fadD25
0.5	83332.Rv1532c	83332.Rv1532c	hypothetical	MT1583
0.857142857143	83332.Rv1661	83332.Rv1661	polyketide	pks7

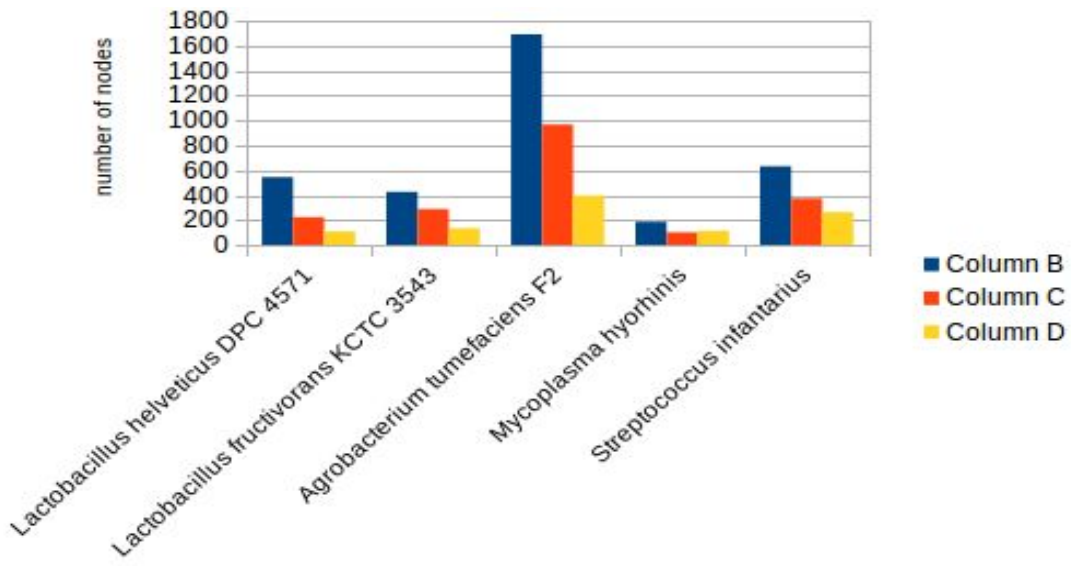
Figure : Truncated view of moderate valued nodes result (columns showing index value ,deleted protein and corresponding annotated protein respectively)

0.0	83332.Rv0658c	83332.Rv0658c	hypothetical	MT0687
0.0	83332.Rv0672	83332.Rv0672	acyl-CoA	fadE8
0.0	83332.Rv0675	83332.Rv0675	enoyl-CoA	echA5
0.0	83332.Rv0764c	83332.Rv0764c	cytochrome	cyp51
0.0	83332.Rv0766c	83332.Rv0766c	cytochrome	cyp123
0.0	83332.Rv0768	83332.Rv0768	aldehyde	aldA
0.0	83332.Rv0860	83332.Rv0860	fatty	fadB
0.0	83332.Rv0905	83332.Rv0905	enoyl-CoA	echA6
0.0	83332.Rv0906	83332.Rv0906	hypothetical	Rv0906
0.0	83332.Rv0914c	83332.Rv0914c	acetyl-CoA	MT0939
0.0	83332.Rv0945	83332.Rv0945	short-chain	Rv0945
0.0	83332.Rv0971c	83332.Rv0971c	enoyl-CoA	echA7
0.0	83332.Rv0972c	83332.Rv0972c	acyl-CoA	fadE12

Figure : Truncated view of non-crucial nodes result (columns showing index value ,deleted protein and corresponding annotated protein respectively)

Similarly for other species

	Lactobacillus helveticus DPC 4571	*Lactobacillus fructivorans KCTC 3543*	*Agrobacterium tumefaciens F2*	*Mycoplasma hyorhinis*	*Streptococcus infantarius*
Crucial nodes	545	427	1690	186	632
Moderate nodes	225	289	967	103	373
Non-crucial nodes	109	134	396	115	264



Further Downstream analysis using tool.py

Mycobacterium tuberculosis H37Rv

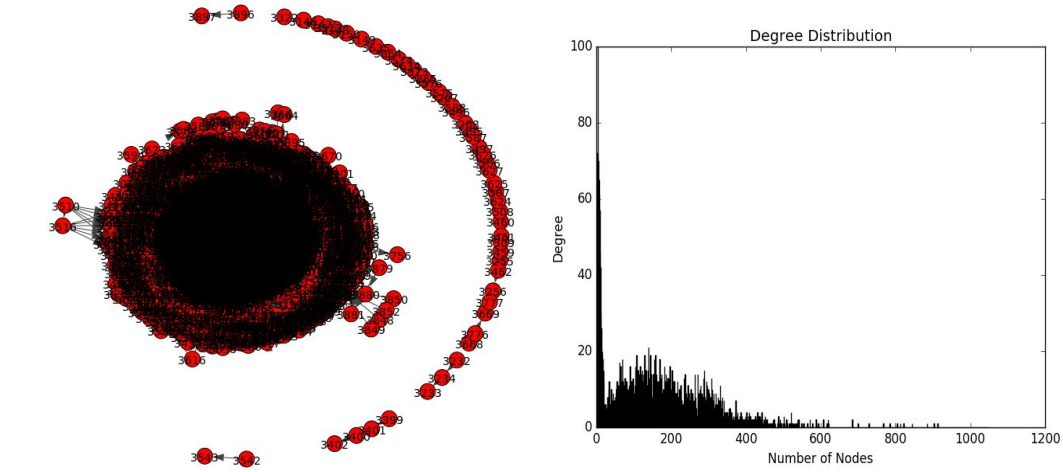


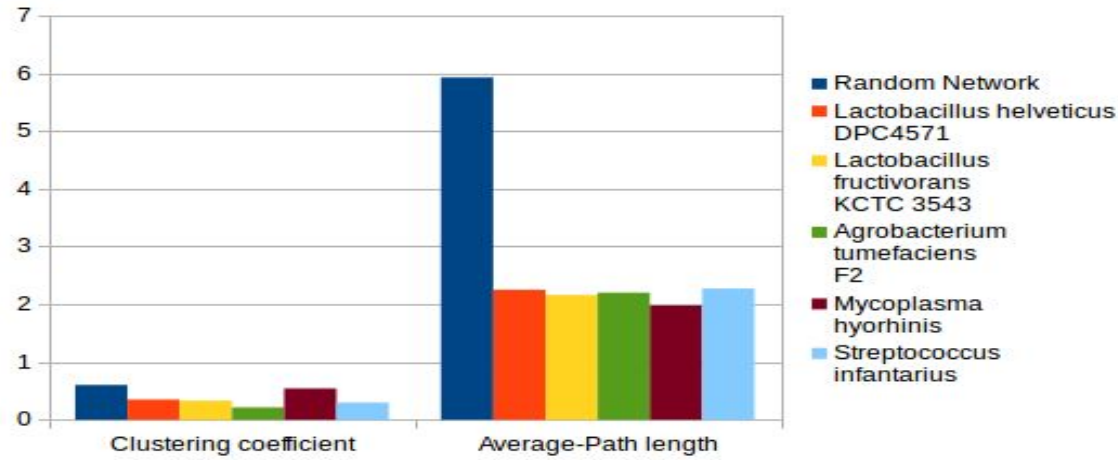
Fig : plot and degree distribution

Clustering coefficient : 0.24846675631890996 >>>> function 6
Average path length : 2.326801269355572 >>>> function 4
Number of functional motif : 69397903 >>>> function 10
Diameter : 12 >>>> function 23
Modularity : 0.00219002235829123 >>>> function 13
Number of of clusters of the graph', 3967 >>>> function 18

Also for other species

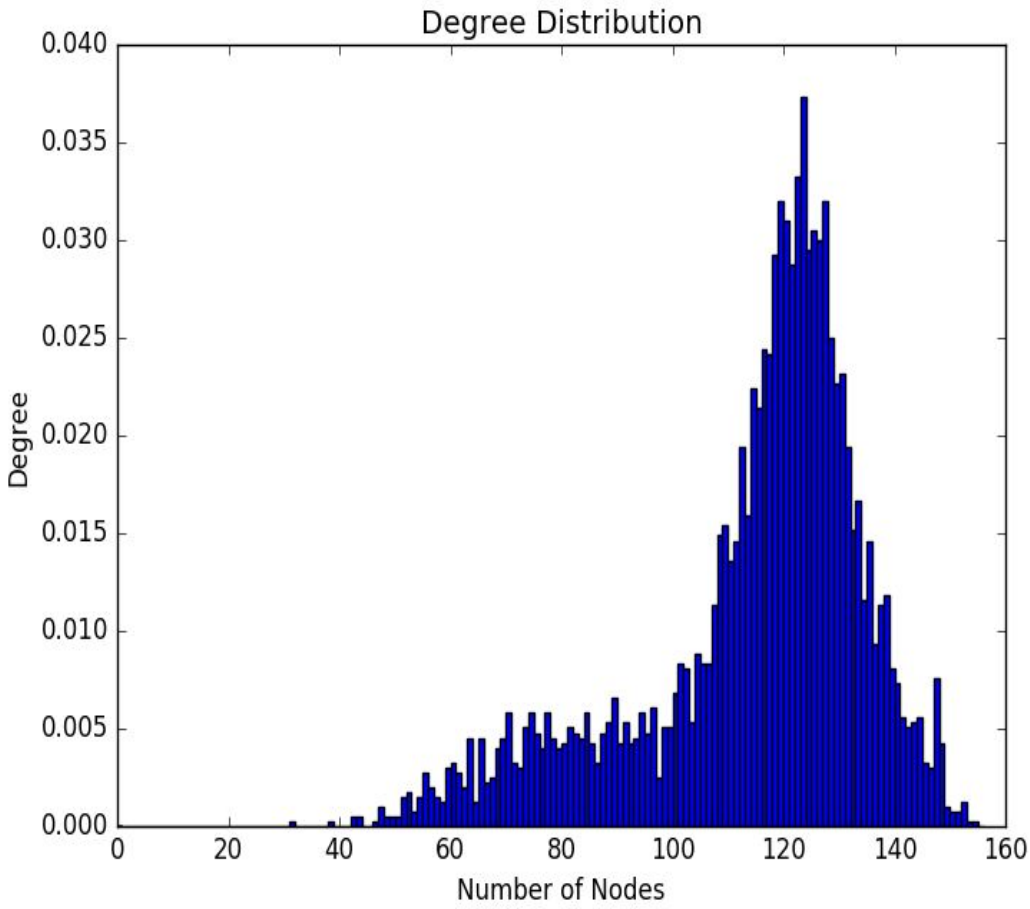
Function used from tool .py	Parameter	*Lactobacillus helveticus DPC 4571*	*Lactobacillus fructivorans KCTC 3543*	*Agrobacterium tumefaciens F2*	*Mycoplasma hyorhinis*	*Streptococcus infantarius*
6	Clustering coefficient	0.3508520122	0.330410800	0.213433	0.54137795	0.2984035

4	Average path length	2.25065919295	2.16010520	2.2008772147	1.97746338	2.2720507
10	Number of functional motif	13907503	9254957	126324842	4035184	13539239

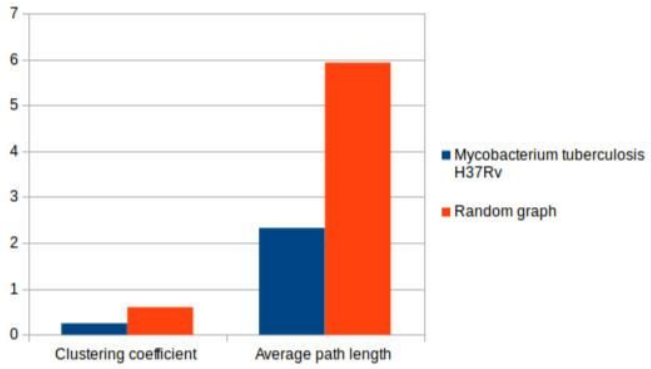


Random network

- Nodes : 3967
- Clustering coefficient : 0.602889037
- Average path length : 5.926425537156



Random graph & *Mycobacterium tuberculosis H37Rv*



Conclusions

Study was carried out for evaluation of important protein in the different organism showing large number of nodes connected together in a cluster adding onto robustness of the network networks behaved as **Barabási–Albert (BA) model** showing high clustering coefficient and shorter average path length when compared to random network of same number of nodes .
In all the organism chosen as dataset showed some common property as high clustering coeff. And shorter average path length revealing that most of the nodes are clustered together and knocking out large number of nodes will disrupt the network
Also there also a giant component which exists within the network containing high number of nodes.

Also important nodes out of the network were identified using the disconnectivity index and further analysed through Tool.py using python igraph library for the data downloaded from string database using pipeline

Recommendations and possibilities

After the evaluation disconnectivity index of the network out of large numbers of proteins we come to know only those which are essential once for organisms survival knocking out these can disrupt the whole network these target proteins can be used as the potential drug target .

Further for more downstream analysis one can go through the pharmacophore modeling using these target protein which will also lowers the probable sample size for drug designing via docking , Also the results can also validated via wet lab experimentation

These nodes Can act as drug target

- **Can tell about biology of organism ,**
- **System level study can be predicted**
- **Treated as vaccine candidates**
- **Also search for sample targets decreased from thousands to few hundred only**
- **Currently only 32 functions are there in tool more can be added**

Scripts

String_data_download.r

This uses bioconductor STRINGdb to download the data from String database of user specified organism id .for test and validation purpose have been used against *Mycobacterium tuberculosis* with species id :83332 .

It returns two .gz files one annotation file and one links file with interaction .

Stringdb_parse.py

This parse the data downloaded from the previous script after unzipping links file

It returns the data file as a weighted-edgelist format with first two columns as nodes on edges and last columns it uses score as edge weight

Pairwise_dis_index.py

This takes the string data which is parsed and set into edgelist format using previous script

And returns a file with one column as the index value after deletion of one node and second columns as the node which was deleted for which the pairwise disconnectivity index is calculated

Pairwise_discon_vdisjoint_paths.py
This similarly takes the output file from the string parse data and calculate the pairwise disconnectivity index but it uses vertex disjoint path to calculate the index value.

Mapping_on_annotation_file.py
This mapps the output files from scripts which calculate disconnectivity index onto the annotation file which comes through the string_data_donwload.r script to tell about the nodes which they were for more biological significance of it .it returns the files with columns as

- index score
- node
- Annotations
- Preferred biological of the node

Into 3 separate file with crucial nodes ,non-crucial nodes and then the moderately crucial nodes

X.sh
It is the bash script which automates the whole process from downloading data to parsing , calculating the pairwise disconnectivity index and then mapping it over the annotation file to extract out meaningful information it returns 3 files with crucial nodes ,non-crucial nodes and then the moderately crucial nodes

Tool.py
This can be used as tool its accepts network file in major formats as edgelist, graphml file ,adjacency matrix ,lgl,and for benchmarking random network also generated to compare data against random dataset
One can use this script to evaluate many parameters or to analyse data for many purpose

- Histogram of degree
- Centrality
    - *Eigenvector centrality
    - *Betweenness centrality
- Average path length
- Degree distribution
- Clustering coefficient

- Shortest path between two pair of nodes
- Shortest path between all nodes
- Degree distribution power law
- Functional motifs
- Neighbour vertex
- *for two specified pair of vertex
- *for all vertex of graph
- Modularity
- Connectivity
    - Vertex *for given two pairs vertex.
        - *overall.
    - Edge *for given two pairs vertex
        - *overall
- No. of clusters
- Adjacency matrix of network
- Given node id its corresponding node attribute
- All node ids and their corresponding node attribute
- Edgelist.
- Diameter of graph
- Average path length
- Giant_component into a file
- Add vertex(single)
- Add vertices(many)
- Delete vertex(single)
- Delete vertices(many)
- Maximum degree nodes
- Minimum degree nodes.
- Deleting all node one by one saving in file to check how it affects

Usage

One can use the **x.sh script** which is a bash script by simply changing it into an executable file

$ chmod +x x.sh

$ ./x.sh

This is a pipeline which initialise the process and input from one script is passed to other script and at the end it will return the 3 files one with all crucial nodes ,non-crucial nodes ,moderately crucial nodes mapped onto the annotation file .
This will first download the string data by taking species id from user as input

Than the data will be unzipped and parsed on for the scripts which evaluates the pairwise disconnectivity index and similar using disjoint path return result files in a directory named out .

If needed one can also use the scripts as stand alone on the dataset they have

String_data_download.r
This will print the species and their corresponding ids .user can choose the organism they whose data they want , then it will ask for an user input where you are supposed put species id and data will be downloaded in out folder

$ Rscript String_data_download.r

Stringdb_parse.py
This will parse string data and convert it into edgelist format

$ python Stringdb_parse.py string data links file output filename

Pairwise_dis_index.py
This will calculate the pairwise disconnectivity index this takes edgelist as input and returns the file with score and its corresponding node on its deletion from network

$ python pairwise_dis_index.py edgelist output filename

Pairwise_discon_vdisjoint_paths.py
This will calculate the pairwise disconnectivity index using disjoint paths this takes edgelist as input and returns the output file with score of the corresponding node on its deletion from network

$ python pairwise_discon_vdisjoint_paths.py edgelist output name

Mapping_on_annotation_file.py
This mapps the output files from with pairwise disconnectivity score index onto the annotation file returns the 3 file as Result_crucialnodes.txt , Result_noncrucialnodes.txt ,Result_moderatenodes.txt

$ python mapping_on_annotation_file.py file with index annotation file

Tool.py
It accepts files in different formats also allows to calculate many more other parameters from the network

$ python tool.py -h

$ python -format [filename]

Then it simply asks the user input for analysis part where one can simply by entering integers after this can get the desired result . script is so much user friendly at each steps it will ask the user input (wherever needed) .
For example if one wants **the random graph** to benchmark their dataset this can be simply done by

$ python tool.py - random

Then it will ask the number of nodes you want in your network and returns the result in form of graph and then takes to next step of analysis

References

[1] al. FAe (2013). "STRING v9.1: protein-protein interaction networks, with increased coverage and integration." ***Nucleic Acids Research (Database issue)*****, 41**

[2] Kardam, Yogita Singh. ***METAHEURISTIC TECHNIQUES FOR SOLVING OPTIMIZATION PROBLEMS IN GRAPH THEORY.*** **Diss. Dayalbagh Educational Institute, 2014.**

[3] Potapov, Anatolij P., Björn Goemann, and Edgar Wingender. "The pairwise disconnectivity index as a new metric for the topological analysis of regulatory networks." ***BMC bioinformatics*** **9.1 (2008): 227.**

[4] Hunter, John D. "Matplotlib: A 2D graphics environment." ***Computing In Science & Engineering*** **9.3 (2007): 90-95.**

[5] Csardi G, Nepusz T: The igraph software package for complex network research, InterJournal, Complex Systems 1695. 2006. http://igraph.org
Corresponding BibTeX entry:

[6]Mason, Oliver, and Mark Verwoerd. "Graph theory and networks in biology." ***IET systems biology*** **1.2 (2007): 89-119.**

[7] de Silva, Eric, and Michael PH Stumpf. "Complex networks and simple models in biology." ***Journal of the Royal Society Interface*** **2.5 (2005): 419-430.**

[8] Barabási, Albert-László, and Réka Albert. "Emergence of scaling in random networks." ***science*** **286.5439 (1999): 509-512.**

BOOKS

Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science (Prentice Hall Series in Automatic Computation), Prentice-Hall, Inc., Upper Saddle River, NJ, 1974

Mark Newman , Albert-Laszlo Barabasi , Duncan J. Watts, The Structure and Dynamics of Networks: (Princeton Studies in Complexity), Princeton University Press, Princeton, NJ, 2006

Harary, F. (1969). *Graph theory*. Reading, Mass: Addison-Wesley Pub. Co.